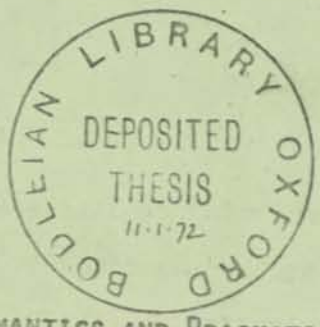


SEMANTICS AND PRAGMATICS OF THE LAMBDA-CALCULUS

by

CHRISTOPHER PETER WADSWORTH

B.A., Oxon. (1967)

Submitted for the Degree of
Doctor of Philosophy
at the
University of Oxford
September, 1971

Signature of Author C.P. Wadsworth

Programming Research Group
September 1st., 1971

ABSTRACT

In the first half of this thesis, the lambda-calculus models of Scott are analysed in detail. Both the choice of subject matter and the emphasis in its presentation are dictated by the wider aims of a developing theory of computation, of which the models studied were an early consequence. Particular attention is given to the characterization of normal forms, a question with direct bearing on the termination properties of computer programs.

The main technical result extends a series of consistency/inconsistency results about the lambda-calculus : For any normal form, an expression without a normal form exists for which their equality can be consistently postulated as an extra axiom for the lambda-calculus. Extensions of theorems of Böhm and Morris are also proved, and a concept of approximate reduction developed.

A representation of lambda-expressions as directed graphs is considered in the second half of the thesis and applied as the basis for reduction algorithms. 'Correctness' is established by deriving necessary and sufficient conditions for the valid application of several general operations on graphs. In conjunction with the 'normal-order' rule of evaluation, efficiency of reduction is achieved without sacrificing the property of terminating whenever a normal form exists. Relation of the technique to function evaluation strategies and parameter passing mechanisms is briefly discussed.

ACKNOWLEDGEMENTS

To Professor Christopher Strachey, whose guidance and enthusiasm have been a constant source of encouragement ;

To Professor Dana Scott, whose insight made this thesis possible and whose research sets a standard we can only approximate(!) ;

To my colleagues, whose comments and suggestions have, sometimes unwittingly, triggered ideas in the author's mind.

I am grateful also to the Science Research Council for their financial support during the course of this research.

CONTENTS

CHAPTER 4 : A Graph Evaluation Technique for the λ-calculus

TABLES

INTRODUCTION

This dissertation examines two aspects of the λ-calculus of Church [1] – its lattice-theoretic models recently discovered by Scott[2], and evaluation strategies based on its conversion rules.

As our title suggests, the emphasis throughout is on the linguistic view of the λ-calculus. Its models will be analysed for their semantic content, its conversion rules for their import on the implementation of 'real' programming languages.

From this point of view, our labour is a contribution to two areas in the theory of programming languages, semantics and pragmatics. A third area of study, that of syntax, is relatively uninformative for the λ-calculus. We perceive these three areas in the same sense as Morris [3], and define them as follows:

1. *Syntax* concerns the synthesis, recognition, and analysis of the legal forms, or sentences, of a language.
2. *Semantics* deals with the relation between the legal forms of a language and the abstract objects they denote.
3. *Pragmatics* treats the relation between a language and its users, human or mechanical.

It would be futile, if not impossible, to attempt to say

exactly where the boundaries between these three areas lie, for there is necessarily some overlap and interaction between them. What matters is that *most* aspects of the study of programming languages can be classified as belonging to one of them.

The syntax/semantics distinction applied to programming languages has been appreciated and studied for some time. Too often however, we find that syntax is understood as above and semantics is then taken as being everything else. It is part of our thesis that this division is incomplete, in that it neglects the more subtle distinction between semantics and pragmatics. There is a difference between the formal description of a language, for which its syntax and pragmatics suffice, and an understanding of the language, for which semantics are required. Thus, specifications of the 'workings', or 'effect', of a language we classify as pragmatics, the role of semantics being elucidation of the 'meanings' of linguistic concepts. Logically, semantics precede pragmatics; the requirement is that the pragmatics of a language be consistent with its semantics, not vice versa.

In this respect it is fair to say that the λ-calculus has been deficient as a language; prior to Scott's models, it had a syntax and a pragmatics, but no semantics. The models studied here are semantic in nature. Evaluation strategies, however, quite definitely come under the heading of pragmatics. Since our contribution to the latter is mainly one of addition to earlier techniques, further introduction is postponed to

the beginning of Chapter 4. It is the former and its place in the development of a theory of computation which is the more significant and most needs preparatory discussion. It would be all too easy for us to present Scott's models as a "pure" study of the λ-calculus, but we feel there is more in them than that. This introduction is therefore part historical, part philosophical, and part indicative of the problems to be treated.

Perspective

As conceived here, the subject matter of computation embraces the whole spectrum from the notion of an 'algorithm' (formal and informal) to the artifices employed in everyday computing. A complete *theory of computation* is thus seen to comprise (at least) three distinct parts:

1. A theory of *computability*, which includes treatment of *what* constitutes an algorithm and of *what* one can hope to compute, without being too concerned about *how* to compute.
2. A theory of programming languages, i.e. of the usage and meaning of symbolism for the expression and communication of algorithms.
3. A theory of *computing* encompassing such topics as compiler design and the organization of 'machines' for executing the actions specified by 'programs'.

Of these three desiderata, the first has already been answered by the work of logicians such as Gödel, Turing and Church, while Chomsky grammars represent common

ground for the specification of the *syntax* of programming languages (though people will probably argue *ad infinitum* about which syntax "looks best"). Much is also understood about the 'operational' aspects of programming and over the years a considerable *empirical* knowledge of computer operating systems has been accumulated.

Of current approaches to the *semantics* of programming languages however, none has as yet attained the level of general acceptance. Most are highly operational in outlook, making use of all kinds of standard programming techniques - stacks, control trees, statement counters, etc.. The work of the PL/I Vienna group [4,5] is the most comprehensive achievement to date along these lines, but the description is very involved to say the least. How much this is due to the complexities of PL/I and how much the result of the method used is not clear, but somehow one feels it doesn't provide the whole story. For although the description defines a precise and unambiguous standard for the would-be compiler-writer, it does not seem well-suited to the development of a *theory* in which one can prove the 'correctness' of an implementation or the equivalence of programs. Both these goals were set long ago by McCarthy [6]. My impression is that the 'PL/I Machine' is itself an implementation of PL/I which, when a suitable theory has evolved, one would hope to prove correct.

For the purpose of proving things about programs, nearly all difficulties experienced with operational descriptions of semantics stem from the fact that they specify too much rather than too little, making choices

in areas where one wants to remains uncommitted. An obvious example is that of *procedures*. Operationally one has to devise mechanisms whereby a procedure can gain access to its parameters and return any results it may produce to the right place for the calling program. But the manner in which this is achieved is more or less irrelevant to an understanding of the meaning of a procedure as a mapping from its parameters to its results, with the "state-of-the-store" included in both (at least implicitly) in order to deal properly with side-effects. One looks for a method of semantic description which reflects this conception.

A second approach is that of *explication*, the foremost example of which is Landin's description of ALGOL 60 in terms of the λ-calculus [7]. The general idea is that one attempts to analyse the features of one language, the *explicandum*, in terms of those of another, the *explicatum*. Usually the latter is previously 'defined' and, therefore, presumably understood. The merits of explication lie mainly in its influence on language design, in the isolation of those features which are different from those which are mere variants of the same thing. Weaknesses or ambiguities in a language are exposed, and controversy over rival interpretations may be resolved. Explication does not seek to define the language in question however, only to explain it. The analogies drawn are just analogies, as sometimes the concept being explicated acquires more properties than one would expect. For example, one can use Church's representation of the integers to simulate arithmetic within the

λ-calculus, but this representation does not define the integers as one might then think it reasonable to apply an integer as a function to all sorts of objects.

This is not to disparage either the operational or the explicative approach to semantics, but only to say that the solutions obtained are provisional ones awaiting a more conceptual treatment.

Mathematical semantics

To be at all satisfactory, it seems certain that a formal treatment of programming language semantics must be *mathematical* in nature, and it must be applicable to all (or nearly all) languages with only minor modifications. Moreover, in striving for mathematical precision the high *intuitive* flavour of the programming concepts involved should not be sacrificed. After all, it is ordinary mortals who will judge the acceptability of a formal semantics, hence it is their predilections which must be considered first. The desire to *automate* the writing of compilers and the proofs of program properties has been suggested by some as possible benefits to be gained from the formalization of programming languages (as indeed they are), but to over-emphasize either of these *applications* in the first instance only confuses the problems of semantics with those of pragmatics. Of course, definitions of programming languages are subject to the constraint that they be *capable* of realization, but it is not necessary that an 'evaluating mechanism' be presented as part of a *semantic* description. To connect the theory with reality, it suffices that a simulation of the abstract entities

within the configurations of real machines be *possible*. This simulation need not be very efficient, for that is not the concern of semantics. The dominating criterion for 'acceptability' ought to be the ease with which a proposed theory allows natural expression of one's ideas.

In short, a theory of programming languages is required which is

1. mathematically based,
2. of general application,
3. consistent with one's intuition,
4. readily comprehended by the human intellect,
5. capable of practical realization.

The first half of this dissertation is devoted to a detailed investigation of one candidate [8] for such a theory. The approach is based on lattice theory, in particular on a theory of continuous functions over complete lattices due to Scott. The motivation for using complete lattices and continuous functions is fully discussed in two of his papers [9,10], to which we refer the reader for a general orientation (see also the present §1.2). In my opinion, this approach to semantics is the most promising to date and comes closest to satisfying the criteria listed above.

■ Here we endeavour to produce sound mathematical reasons for this claim. The method of our analysis is to take one result of Scott's research - his λ-calculus models - study these mathematically, and relate what we find to what we would expect and hope for "computationally".

Two views of the λ-calculus

The first, and logically correct, view of the λ-calculus is as a *formal system*, with axioms and rules just like any other formal system. For the λ-calculus, the axioms are ones of equality between terms and the rules are those of α-, β-, and (sometimes) η-conversion. From this point of view, one naturally speaks of *interpretations* and *models* of the λ-calculus.

At the same time, we shall think of the λ-calculus as an example of a programming language in its own right. Obviously it does not look much like one at first sight, but one should not be beguiled by its syntactic appearance; it can be given a more decorative and suggestive syntax, e.g. the so-called 'applicative subset' of Landin's ISWIM[11]. Here it is natural to talk of the *value* of a λ-expression and of the *semantics* of the λ-calculus.

The latter requires a little explanation however, since the λ-calculus may have many models. The point is that we do not want just any model, but we wish to find one which is somehow "the right one" or a "standard" one from a computational point of view. We might then refer to the interpretation of a λ-expression, X, in such a model as *the* meaning of X, or at least as the *standard* meaning of X. Thus, the questions we seek to resolve are; Is there a standard computational model? If so, does Scott's theory provide it?

Semantics of the λ-calculus

There is a strong temptation to say that the semantics of the λ-calculus reside in its conversion rules. The universe of λ-expressions is thus partitioned into a

collection of equivalence classes such that each belongs to one and only one class, and two λ-expressions belong to the same class iff they are interconvertible. One could then designate these equivalence classes as the 'abstract objects' which λ-expressions denote, the Church-Rosser Theorem being an assurance that one is consistent in doing so (i.e. each λ-expression then has a unique 'value').

This interpretation "fails" on several counts. First, it is not the one we want. It means, for instance, that

$$(\lambda x.xx)(\lambda x.xx)$$

and

$$(\lambda x.xxx)(\lambda x.xxx)$$

would not be equivalent, since they are not interconvertible, whereas from a computational viewpoint one would like them to be interpreted as the same thing.

More seriously, the interpretation does not reflect the natural relationship between the λ-calculus and the concept of a *function*. Since the λ-notation arose out of the need for a systematic notation for functions - to distinguish between a function itself and (an expression denoting) the result of a function for a undetermined value of its argument - it follows that the meaning of a λ-expression ought to be some kind of function.

The difficulty is that a λ-expression is more general than the functions encountered in conventional set theory. A λ-expression denotes a purely formal *rule*, or *operation*, for computing the result of a function application, but it differs from an ordinary 'typed' function in that it is not restricted to being applied only to elements of a fixed, pre-specified domain; its domain is regarded as being

determined *after* the rule is given by all things to which the rule is applicable, and this might include elements of quite arbitrary nature. In particular, a λ-expression can be meaningfully applied to *any* λ-expression, including *itself*. The universe of λ-expressions can thus be simply described as consisting of all terms fashioned out of variables by means of application and abstraction. Technically, this is a consequence of the requirement, that, as a formal system, the λ-calculus be *combinatorially complete* - any function definable intuitively by means of a variable can be represented formally as a λ-expression, c.f. Curry and Feys [12].

The aesthetic simplicity of this morphological requirement is deceptive however. The possibility of *self-application* which it affords has involved considerable effort on the part of logicians to establish the formal consistency of the λ-calculus and related systems, as even a cursory glimpse at the literature will reveal.

From a semantic point of view, the problem has been to find a "reasonable" interpretation at all. The domain, D, of an interpretation of the λ-calculus must be related to its own function space in a rather special way - it must include (at least up to isomorphism) part of its function space. It cannot contain *all* functions from D into D, since there are always "more" of these than there are elements of D. Whatever else this might mean, it implies that a λ-expression is not an *arbitrary* function.

The novel feature of Scott's models is his discovery that, by restricting attention to domains structured as

complete lattices, and to only the continuous functions on such domains, the necessary "self-referential" domains could be constructed.

The first of these *reflexive domains* he constructed was a domain(+) *isomorphic* to its own *continuous* function space. This construction will be outlined in §1.3 and the properties of the resulting λ-calculus model studied in Chapter 2.

Reiterating our earlier point, when we wish to be formally correct we shall talk of *interpretations* and *models*. Informally, we shall talk of *values* and *semantics*, moved by the desire to analyse these models for their computational relevance.

Motivation

The increasing influence of the λ-calculus on the development of programming languages in the last decade gives relevance to our analysis. Many characteristic features found in real languages strongly resemble those of the λ-calculus. The closeness of this resemblance has been studied by numerous authors, e.g. see [3,7], as a means of gaining insight into fundamental programming concepts, and the results of their research are reflected in the design of several languages, notably CPL [13], ISWIM [11] and PAL [14]. Concurrent with these studies, Strachey [15], Böhm [16] and others have utilised the

(+). Strictly, a family of domains.

λ-calculus itself in early attempts at the semantics of programming languages.

The present investigation is somewhat different in outlook. We do not use the λ-calculus to model any features of programming languages, nor is it our formalism for semantic description. Rather we treat the λ-calculus as a 'prototype' programming language.

The λ-calculus is particularly suitable as a testing ground for an apppoach to semantics. With a real programming language, even one that is 'well-defined', arguments for and against rival specifications of its semantics must be always somewhat subjective, since these are intended to define the language in question. Because of its precisely formulated mathematical properties, the λ-calculus transcends subjective debate. By defining its semantics in the same way as for any other programming language, there is an opportunity (unique?) to scrutinize our theories. At the same time, we regard a convincing descriptions ofiits semantics as a necessary capability of any proposed theory of programming languages.

CHAPTER 1

Prerequisites

This chapter introduces relevant material which will be used later. Since most is thoroughly documented in the literature, we restrict ourselves to a factual account of as much as we shall need. For a fuller treatment the reader is referred to Church [1] or Curry and Feys [12] for the λ-calculus, and to Scott [2,9] for the lattice-theory concepts. Our choice of terminology has also been influenced by Landin [17], Morris [3] and Strachey.

Each chapter is divided into several sections. The numbering of lemmas, theorems, etc., begins afresh in each section. Cross-references will be given without the qualifying noun, e.g. 1.2.3 to refer to Theorem 1.2.3.

§1.1 The λ-calculus

§1.1.1 Well-formed expressions

Expressions are formed as sequences of symbols drawn from an alphabet which has four *improper symbols*:

$$\lambda \ , \ . \ , \ (\ , \ )$$

and a denumerable class, denoted by Id, of symbols

$$a, b, c, \cdots, z, a', b', \cdots$$

called *variables*, or *identifiers*. Certain strings over this alphabet will be distinguished as being *well-formed expressions* (*wfes*), the class of all wfes being denoted by Exp.

The Greek letters ξ and ε_0 and decorated versions thereof (i.e. with primes and subscripts), will be used as *meta-variables* ranging over Id and Exp, respectively. Strictly, these meta-variables *denote* members of the respective classes, though most of the time we can skip the distinction. Identically decorated ones are to be understood as (denoting) the same identifier or wfe; differently decorated ones might or might not (in general not) be the same.

The following three-clause definition of *wfes* is combined with the definition of *free* and *bound* and some terminology to refer to parts of wfes:

1. *Identifiers*: an identifier ξ is a wfe; the occurrence of ξ in this wfe is free.
2. *Combinations*: if ε_0 and ε_1 are wfes, then $(\varepsilon_0)(\varepsilon_1)$ is a wfe, the parts ε_0 and ε_1 being called its *rator* and *rand*, respectively; an occurrence of an identifier in ε_0 (or ε_1) is free or bound in this combination according as it is free or bound in ε_0 (or ε_1).
3. *Abstractions*: if ξ is an identifier and ε is a wfe, then $(\lambda\xi.\varepsilon)$ is a wfe, the parts ξ and ε being called its *bv* and *body*, respectively; an occurrence of an identifier, other than ξ, in $(\lambda\xi.\varepsilon)$ is free or bound according as it is free or bound in ε; all occurrences of ξ in $(\lambda\xi.\varepsilon)$ are bound.

Notational conventions

Parentheses may be omitted from wfes consistent with the conventions:

1. Identifiers occurring as the rator or rand of a combination need not be parenthesized.
2. Association to the left:

   $\varepsilon_1\varepsilon_2\varepsilon_3\cdots\varepsilon_n$ stands for $(\cdots((\varepsilon_1)(\varepsilon_2))(\varepsilon_3)\cdots)(\varepsilon_n)$
3. Binding of the dot : the scope of the dot '.' in an abstraction extends as far to the right as possible, i.e. to the first unmatched closing parenthesis, or to the end of the expression, whichever occurs first.

Also, consecutive abstractions may be collapsed as in:

$\lambda\xi_1\xi_2\cdots\xi_n.\varepsilon$ stands for $\lambda\xi_1.\lambda\xi_2.\cdots.\lambda\xi_n.\varepsilon$

BNF-specification

Wfes can be specified in BNF-notation, with the above conventions for inserting missing parentheses, as

<wfe> ::= <id> | <wfe>(<wfe>) | (λ<id>.<wfe>)

Using the meta-variables ξ and ε, we abbreviate this as

ε ::= ξ | ε₀(ε₁) | (λξ.ε)

to be read, à la Landin, as:

A *wfe* is either an *identifier*, (denoted by) ξ,

or a *combination* with *rator*, ε_0, and *rand*, ε_1,

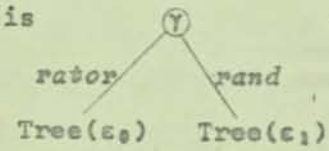
or an *abstraction* with *bv*, ξ, and *body*, ε.

The names *rator*, *rand*, *bv*, *body* are called *selectors*; they may be applied to wfes of the appropriate format to yield its components.

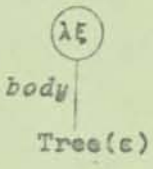
Tree representation of wfes

Related to these selectors, a wfe can be depicted as a *tree*, each of the three kinds of wfes being represented by a *node* in the tree with its selectors "pointing" at the node representing the appropriate component. Thus:

1. The tree of ξ is 

2. The tree of $\varepsilon_0(\varepsilon_1)$ is 

(γ is used to represent combinations)

3. The tree of $\lambda\xi.\varepsilon$ is 

Hereafter, the selectors on the branches will be omitted, with the convention that the left-branch of a combination is its rator,

N.B. In3., ξ has been included inside the node as we intend a sub-tree to be a sub-expression and do not wish to admit the occurrence of an identifier immediately following a 'λ' as a sub-expression.

Paths and sub-expressions

A *path* is a list of $n \geq 0$ selectors, other than *bv*. The path of length 0 will be denoted by *null*.

If $p = (s_1, s_2, \cdots, s_n)$ is a path of length > 0, then the first selector, s_1, and the list $(s_2, s_3, \cdots, s_n)$ are called the *head* and *tail* of p. If $p' = (s_1', \cdots, s_m')$ is a second path, then $p*p'$ denotes the path

$$(s_1, s_2, \cdots, s_n, s_1', \cdots, s_m') .$$

If p and q are two paths, then p is a *stem* of q if there is a path p' such that $p*p' = q$.

A path p is *applicable* to a wfe ε if $s_1 = head(p)$ is applicable to ε and if $tail(p)$ is then applicable to $s_1(\varepsilon)$. Then, define inductively

$$p(\varepsilon) = (p = null \rightarrow \varepsilon,\ tail(p)(head(p)(\varepsilon)))$$

A *sub-expression* of ε is a pair $\langle p,\varepsilon'\rangle$ consisting of a path p and a wfe ε' such that $p(\varepsilon) = \varepsilon'$. The sub-expression is said to be proper if p is not the null path. Two sub-expressions $\langle p_0,\varepsilon_0\rangle$, $\langle p_1,\varepsilon_1\rangle$ of the same wfe are *disjoint* if neither of p_0 and p_1 is a stem of the other.

If $S=\langle p,\varepsilon\rangle$ and $S'=\langle p',\varepsilon'\rangle$ are sub-expressions of ε_0 and ε_1, respectively, they are said to be *homologous* sub-expressions if $p=p'$, i.e. if S and S' occur in the same position in ε_0 and ε_1, respectively.

Contexts

A *context*, C[], is an expression with one sub-expression missing. Pictured as a tree, a context has one branch "unattached", so that one sub-tree is missing. As a symbol string, a suitable BNF-specification, with <c> for <context>, is

<c> ::= [] | <c>(<wfe>) | <wfe>(<c>) | (λ<id>.<c>)

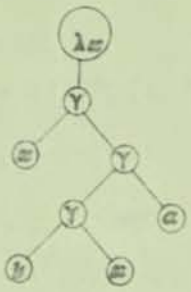
If C[] is a context and ε is a wfe, then C[ε] denotes the result of filling the "hole" in C[] with ε (i.e. replacing the empty square brackets in the linear representation, or the missing branch in the tree). Thus, C[ε] can be read: a wfe containing an occurrence of ε.

An example of a context is (in both forms)

C[] = λx.x([]a)

. Then

$$C[y,x] = \lambda z.z(y,x,a)$$



The rank of a wfe

Many proofs of theorems about the λ-calculus proceed by what is termed an induction on the *structural complexity* of wfes. The *rank* of a wfe is one measure of complexity, defined by:

1. An identifier is of rank 0.
2. If ε_0, ε_1 are of rank m, n, respectively, then $\varepsilon_0(\varepsilon_1)$ is of rank $m+n+1$.
3. If ε is of rank n, then $\lambda\xi.\varepsilon$ is of rank $n+1$.

Notice that, in terms of the tree representation, the rank of a wfe is equal to the sum of the number of γ- and λ-nodes in the tree.

§1.1.2 Substitution and conversion

To prevent the capture of bound variables, the definition of the substitution operation requires some care. The one we give will be such that the result, denoted by

$$\varepsilon' \equiv [\varepsilon_0/\xi]\varepsilon ,$$

of substituting ε_0 for ξ in ε is defined for *all* $\varepsilon_0, \xi, \varepsilon$.

Case 1. ε is an identifier ξ'

(a) If $\xi' = \xi$, then $\varepsilon' = \varepsilon_0$.

(b) If $\xi' \neq \xi$, then $\varepsilon' = \xi'$.

Case 2. ε is a combination $\varepsilon_1(\varepsilon_2)$

Then $\varepsilon' \equiv \varepsilon_1'(\varepsilon_2')$, where $\varepsilon_i' \equiv [\varepsilon_0/\xi]\varepsilon_i$ for $i=1,2$.

Case 3. ε is an abstraction $\lambda\xi'.\varepsilon_1$

(a) If $\xi'=\xi$, or if ξ not free in ε_1, then $\varepsilon'\equiv\lambda\xi'.\varepsilon_1$

(b) Otherwise, $\varepsilon' \equiv \lambda\xi''.[\varepsilon_0/\xi][\xi''/\xi']\varepsilon_1$

where ξ'' is the identifier determined as follows:

(i) If ξ' not free in ε_0, then $\xi''\equiv\xi'$.

(ii) If ξ' free in ε_0, then ξ'' is the first (for definiteness) identifier, other than ξ or ξ', in the alphabetical list of identifiers such that ξ'' does not occur free in ε_0 or ε_1.

α-conversion

If y does not occur free in M, a wfe of the form $\varepsilon \equiv C[\lambda x.M]$, where $C[]$ is any context, may be converted to $\varepsilon' \equiv C[\lambda y.M']$, where $M' \equiv [y/x]M$. This transformation is called an *α-conversion* of ε into ε', denoted by

$$\varepsilon \rightarrow_\alpha \varepsilon'$$

β-conversion

A wfe of the form $R \equiv (\lambda x.M)A$ is called a *β-redex*, and $R' \equiv [A/x]M$ is called the *contractum* of R. The part M of R is called the *base* of the redex. For any context $C[]$, the replacement of R in $\varepsilon \equiv C[R]$ by R' to yield $\varepsilon' \equiv C[R']$ is called a *β-contraction*; in the other direction, the replacement of R' by R is a *β-expansion*. We write

$$\varepsilon \xrightarrow{R}_\beta \varepsilon'$$

to be read: ε is transformed into ε' by contraction of the β-redex R.

η-conversion

A wfe of the form $P \equiv (\lambda x.Mx)$ is called an *η-redex* if x does not occur free in M, and M is called the *contractum* of P. The definitions of η-reduction, η-expansion and $\xrightarrow{P}_\eta$ follow by analogy with β.

Notation. We shall write

1. $\to_\beta$ or $\to_\eta$ when it is of no interest which redex has been contracted.
2. $\to$ without a subscript α, β or η when it is clear which is implied.
3. red and cnv for the transitive closure and equivalence relation, respectively, generated by $\to$.
4. prefixes α, β or η on red and cnv when restricted to a particular kind of conversion rule.

Normal forms

A wfe is said to be in *normal form* if it does not contain a redex as a sub-expression. A wfe is said to *have* a normal form if there is a wfe in normal form to which it is convertible. Again β- and/or η- may be prefixed to these terms.

Not all wfes can be converted to ones in normal form. Two we shall treat later are

$$(\lambda x.xx)(\lambda x.xx)$$

and the so-called paradoxical combinator

$$Y_\lambda \equiv \lambda f.(\lambda y.f(yy))(\lambda y.f(yy))$$

(We have used a subscript λ on Y to distinguish it from a lattice-theoretic operator to be introduced later.)

The class, N, of wfes in β-normal form can be defined inductively as follows:

1. $X_1, \cdots, X_m \in N \Rightarrow sX_1 \cdots X_m \in N$ ($m \geq 0$, s variable)

2. $X \in N \Rightarrow (\lambda x.X) \in N$ (x variable)

whence wfes in normal form have the structure

$$\lambda x_1 x_2 \cdots x_n . s X_1 X_2 \cdots X_m \quad , \; n, m \geq 0,$$

where each X_k is in normal form.

If, in 2., the restriction is added that X is not of the form $X'x$ with x not free in X', the wfe is in β-η-normal form.

§1.1.3 Descendants and ancestors

These notions, due to Morris, are a generalization of residuals as defined by Church and Curry. When a wfe is being reduced, they provide a means of associating parts of the wfes in a reduction sequence with parts of the initial wfe in the sequence, and vice versa. To help shorten the definition below, by *part* we shall mean well-formed sub-expression.

Suppose $\varepsilon \to \varepsilon'$. A function, *father*, from parts of ε' to parts of ε will be defined such that every part of ε' has a unique father in ε. The relation, *son*, is the inverse of father; not all parts of ε will have sons in ε', and some will have more than one son.

Let S' be a sub-expression of ε'. We consider the three kinds of conversion in turn:

1. $\varepsilon \to_\alpha \varepsilon'$. Then the father of S' is the homologous part of ε.

2. $\mathfrak{C} \to_\beta \mathfrak{C}'$ Let $R \equiv (\lambda x.M)A$ be the β-redex contracted.

2a. If S' is not part of the contractum of R, its father is the homologous part of $\mathfrak{C}$.

2b. If S' is part of an occurrence of the rand, A, which was substituted for x in M, its father is the homologous part of the occurrence of A in R.

2c. If S' is any other part of $[A/x]M$, its father is the homologous part of the occurrence of M in R.

N.B. Neither the redex being contracted nor any of the free occurrences of x in M have any sons. Parts of the rand, A, have k sons, where k is the number of free occurrences of x in M.

3. $\mathfrak{C} \to_\eta \mathfrak{C}'$. Let $P \equiv (\lambda x.Mx)$ be the η-redex contracted.

3a. If S'' is part of the contractum, M, its father is the homologous part of the occurrence of M in P.

3b. If S' is not part of M, its father is the homologous part of $\mathfrak{C}$.

N.B. Neither P nor Mx have any sons in $\mathfrak{C}'$.

The relations *descendant* and *ancestor* are the natural extensions of father and son by transitivity. Note that any descendant of a redex is itself a redex. After Church and Curry, we shall sometimes refer to the descendants of a redex as *residuals*.

Let

(1) $$\mathfrak{C} \equiv \mathfrak{C}_0 \to \mathfrak{C}_1 \to \mathfrak{C}_2 \to \cdots \to \mathfrak{C}_n \equiv \mathfrak{C}'$$

be a reduction of $\mathfrak{C}$ to $\mathfrak{C}'$. Then, $\mathfrak{C}_i$ is called the ith. *stage* in the reduction, and the contraction leading from $\mathfrak{C}_{i-1}$ to $\mathfrak{C}_i$ is called the ith. *step*.

Suppose $\mathcal{A}$ is a set of redexes in a wfe ε. Then the reduction (1) is said to be a *reduction relative to* $\mathcal{A}$ if the redex contracted in each step is a descendant of a redex in $\mathcal{A}$. Such a reduction is a *complete* reduction relative to $\mathcal{A}$ if the last stage contains no descendants of redexes in $\mathcal{A}$.

Lemma 1.1.1 (Curry and Feys)(The lemma of parallel moves)

If $\mathcal{A}$ is a set of β-redexes in ε, there is a complete reduction of ε relative to $\mathcal{A}$, and all complete reductions of ε relative to $\mathcal{A}$ end in the same wfe ε'. Further (Morris), parts of ε' have the same ancestors in ε whichever reduction is followed.

N.B. This lemma may fail if η-redexes are included in $\mathcal{A}$.

The following is central to the study of the λ-calculus:

Theorem 1.1.2 (The Church-Rosser Theorem)

If ε_0 cnv ε_1, there exists a wfe ε' such that

ε_0 red ε' and ε_1 red ε'.

Corollary 1.1.3

If ε_0 cnv ε_1 and ε_1 is in normal form, then ε_0 red ε_1 ; if also ε_0 is in normal form, then ε_0 and ε_1 are identical up to α-conversion.

Corollary 1.1.4

If two different reductions of ε both reduce it to wfes N_0, N_1 in normal form, then N_0 α-cnv N_1 ; hence, if a wfe has a normal form, this is unique up to α-conversion.

These results establish that the λ-calculus is *consistent*, in the sense that not all pairs of wfes are interconvertible (since distinct normal forms are not).

§1.1.4 The standardization theorem

It follows from 1.1.3 that one can always find a normal form for any wfe which has one by trying all possible reductions of it. However, there is a more direct method, namely that one need only try one reduction sequence, as we now outline.

For a β-redex $R = (\lambda x.M)A$, the occurrence of λx here is called the *head* of R. If R and S are two β-redexes in the same wfe, ϵ, then R is said to be *senior* to S if the head of R lies to the left of the head of S in the linear representation of ϵ. For two distinct β-redexes in the same wfe, one is always senior to the other, since a β-redex is uniquely determined by its head. For any wfe, ϵ, not in β-normal form, there is a unique redex R in ϵ senior to all other (if any) redexes in ϵ; R is then called the *leftmost* β-redex in ϵ.

When R is senior to S, then R will have a unique son if S is contracted, whereas if R is contracted, then S either has a unique son lying entirely to the right of the contractum of R, or all sons of S are internal to the contractum of R.

Now let

$$\epsilon = \epsilon_0 \xrightarrow{R_1} \epsilon_1 \xrightarrow{R_2} \epsilon_2 \xrightarrow{R_3} \cdots \xrightarrow{R_n} \epsilon_n = \epsilon' \tag{2}$$

be a β-reduction of ϵ to ϵ'. Then (2) is called a *standard reduction* iff R_{i+1} is not a son of a redex in ϵ_i senior to R_i, for $i = 1, 2, \cdots, n-1$. Alternatively, this can be stated: (2) is a standard reduction iff the head of R_{i+1} in ϵ_i lies in or to the right of the contractum of R_i, for $i = 1, \cdots, n-1$.

Theorem 1.1.5 (Curry, Feys)(The standardization theorem)

If ε β-red ε', then there is a standard β-reduction from ε to ε'.

(2) will be called a *normal reduction* (or a *normal order reduction*) iff each R_{i+1} is the left-most β-redex in ε_i; and we shall call (2) *the* normal reduction of ε if the last stage ε' is in normal form.

Corollary 1.1.6

If ε is any wfe having a β-normal form, then the normal reduction of ε will always find its normal form.

§1.1.5 Axiomatic formulation

Since we shall be considering interpretations and models of the λ-calculus, it is expedient to have a more precise statement of the axioms of the λ-calculus as a formal system.

Such a formulation, Λ, is given in Table 1, to which we add the following explanatory notes.

First, we have listed the primitive symbols of Λ - that is, the individual variables and the function and predicate letters - in the customary manner of logic (c.f. Mendelson [18]). As is standard practice there, the superscripts indicate the degree of each function and predicate letter; the subscripts are used simply for indexing purposes. The terms (formulae) are then defined inductively - for each n-ary function (predicate) letter, that letter followed by a list of n terms in parentheses is a term (formula). Abbreviations have then been introduced to bring the notation back into line with that earlier.

Table 1

Axiomatic formulation, A, of the λ-calculus

Primitive symbols

Individual variables: ξ_i , for $i \geq 1$

Function letters: f_1^2, f_i^1 , for $i \geq 1$

Predicate letter: A_1^2

Terms

Identifiers: ξ_i , for $i \geq 1$

Combinations: $f_1^2(\varepsilon_0, \varepsilon_1)$, for terms $\varepsilon_0, \varepsilon_1$

Abstractions: $f_i^1(\varepsilon)$, for $i \geq 1$

Formulae: $A_1^2(\varepsilon_0, \varepsilon_1)$, for terms $\varepsilon_0, \varepsilon_1$

Abbreviations: $\varepsilon_0(\varepsilon_1)$ stands for $f_1^2(\varepsilon_0, \varepsilon_1)$

$\lambda\xi_i.\varepsilon$ stands for $f_i^1(\varepsilon)$

$\varepsilon_0 = \varepsilon_1$ stands for $A_1^2(\varepsilon_0, \varepsilon_1)$

Axioms and rules

1. $=$ is an *equivalence relation*

(ρ) $\varepsilon = \varepsilon$ (reflexivity)

(σ) $\varepsilon_0 = \varepsilon_1 \Rightarrow \varepsilon_1 = \varepsilon_0$ (symmetry)

(τ) $\varepsilon_0 = \varepsilon_1, \varepsilon_1 = \varepsilon_2 \Rightarrow \varepsilon_0 = \varepsilon_2$ (transitivity)

2. $=$ is *substitutive*

(μ) $\varepsilon_0 = \varepsilon_1 \Rightarrow \varepsilon(\varepsilon_0) = \varepsilon(\varepsilon_1)$ (right monotony)

(ν) $\varepsilon_0 = \varepsilon_1 \Rightarrow \varepsilon_0(\varepsilon) = \varepsilon_1(\varepsilon)$ (left monotony)

(θ) $\varepsilon_0 = \varepsilon_1 \Rightarrow \lambda\xi.\varepsilon_0 = \lambda\xi.\varepsilon_1$

3. *conversion rules*

(α) $\lambda\xi.\varepsilon = \lambda\xi'.[\xi'/\xi]\varepsilon$, when ξ' not free in ε

(β) $(\lambda\xi.\varepsilon)\varepsilon_0 = [\varepsilon_0/\xi]\varepsilon$

(η) $\lambda\xi.\varepsilon(\xi) = \varepsilon$, when ξ not free in ε

4. *derived axiom*

(Subst) $\varepsilon_0 = \varepsilon_1 \Rightarrow C[\varepsilon_0] = C[\varepsilon_1]$, for contexts $C[\,]$

$(\mu), (\nu), (\theta) \Rightarrow$ *(Subst)*

Notice that a different unary function letter, f_i^1, had to be used for abstraction with respect to each variable, ξ_i. If a binary function letter, say f_2^2, had been used for abstraction, then $f_2^2(\varepsilon_0,\varepsilon_1)$ would have been a term, for all ε_0. The terms would not then have corresponded to the wfes defined earlier.

The Greek letters (ρ), (σ), etc., for the axioms and rules are the same as those used by Curry and Feys, except our (θ) is their (ξ); this has been changed to avoid a clash with our use of ξ as a meta-variable for identifiers. There is one 'derived axiom', *(Subst)*, for substitutivity. This axiom implies, and is implied by, (μ), (ν) and (θ) together. We have given it here because of the ease with which it may be stated using the notion of contexts.

Informally, it can allbe summarized by:

1. The terms (wfes) are all those formed from variables by application and abstraction.
2. $=$ is a substitutive equivalence relation which holds for all pairs of interconvertible wfes.

(Strictly speaking, the formulae defined in Table 1 are the *atomic* formulae, since they do not involve quantifiers or propositional connectives. The study of the λ-calculus and related systems as affected by quantifiers is another branch of the subject known as *illative* combinatory logic. We consider here only that part which Curry and Feys call *pure* combinatory logic - the study of combinators, and wfes, in their formal relation to one another with reference only to a relation of equality.)

§1.2 Concepts from lattice theory

For the conventional notions of lattice theory, the reader is referred to any standard text on the subject, e.g. [19]. All our lattices will be complete lattices, for which several formulations are possible. The one most convenient here is: a *complete lattice* is a set with a partial order relation under which every subset has a least upper bound.

Following Scott, for the notions relevant to complete lattices, we use the symbols:

$\sqsubseteq$ for the *partial ordering*,

$\bot$, $\top$ for the *least* and *greatest* elements,

$\bigsqcup$ for the *l.u.b.*(or *sup*) of arbitrary subsets,

$\bigsqcap$ for the *g.l.b.*(or *inf*) of arbitrary subsets,

$\sqcup$, $\sqcap$ for the *join* and *meet* of pairs of elements.

The elements $\bot$ and $\top$ are read as "bottom" and "top", respectively, by analogy with the usual presentation of the partial order diagrams of lattices.

For the λ-calculus models we shall study, it is not too important what the intended interpretations of these notions are. In line with our desire to relate these models to the study of computation though, it seems worthwhile presenting *some* of the supporting ideas.

The motivation derives from the kinds of *data types* and *functions* which arise as computational entities. One view of a data type is as a set, namely the set of all objects of that type. For some primitive types, e.g. the integers or the truth values, this view is adequate enough, but for other types it ignores certain natural relationships which exist between objects of that type. The "omission" is most striking in the case of functions; more especially,

for the *partial functions* generally computed by programs and algorithms. For these, some are *extensions* of others, in the sense of being *more defined* than others. For example, suppose f_i denotes the restriction of, say, the factorial function to the integers $\leq i$:

$$f_i(n) = \begin{cases} n! & \text{, if } 0 \leq n \leq i \\ \textit{undefined} & \text{, if } n > i \end{cases}$$

Then, for $j \leq i$, f_i is an extension of f_j; whenever $f_j(n)$ is defined, so is $f_i(n)$, and they then yield the same answer. Putting this the other way round, it is natural to say that the function f_j is an *approximation* to f_i, and indeed they both approximate the factorial function defined on all non-negative integers.

Whenever approximations are being treated, one wonders about *limits* of sets of approximations. Thus, in the above example, we have a sequence

$$f_1, f_2, \cdots, f_i, \cdots$$

of functions all of which approximate the factorial function, and, as i tends to infinity, it seems reasonable to say that this sequence 'converges' to the factorial function.

Having seen how the notions of approximations and limits arise, we can now sketch how they are brought within the realm of our mathematical treatment.

To make the partial functions more amenable to study, the first step is to adjoin the 'fictitious' element $\perp$ to each type, so that the partial functions become total, under the convention that their value is $\perp$ wherever they are undefined. In the case of the function-types, say from D to D', the element $\perp$ is to be thought of as the function

which is *everywhere undefined*:

$$i(x) = \perp' \quad \text{, for all } x \in D,$$

where the prime has been used to distinguish the bottom element of D'. One word of caution here; if f is a function from D to D', it is not necessary that $f(\perp)=\perp'$. There is no reason to exclude *non-strict* functions, which may have a 'value' even though their argument is undefined.

Next, we bring in the notion of approximation. Intuitively, one understands its properties as being precisely those of a partial ordering, i.e. reflexive, transitive and anti-symmetric. The relation $\sqsubseteq$ for each data type is conceived in this sense of approximation. When it holds between two elements x, y, written $x \sqsubseteq y$, we say that x is *weaker than* y, or x is *less than or equally defined as* y, or, simply, x *approximates* y.

The notion of limit we want can now be seen to be the *least upper bound* with respect to the partial orderings. For the example of the factorial function, we can write

$$\text{factorial} = \bigsqcup_{n=0}^{\infty} f_n \, .$$

For pairs of elements x, y, their l.u.b. is their join $x \sqcup y$.

As formulated so far, not every subset of a data type will have a l.u.b.. However, it is rather inconvenient to have to prefix all ones statements with such phrases as "provided the appropriate element exists". A more elegant solution is to make the assumption that every subset has a l.u.b.. One can imagine, if one likes, that sufficient extra elements are adjoined to each data type (and the partial ordering suitably extended) so that this holds.

Thus we now have data types which are complete lattices under their partial orderings. One need not feel

disconcerted about the extra elements which have been adjoined. There is no loss of generality in taking a data type to be a complete lattice. For what we have done is to *embed* all the usual elements in a larger space with a richer structure, to facilitate study of their natural inter-relationships. These larger spaces therefore include all the elements one would expect, plus a few more to simplify the mathematical treatment.

Of these 'extra' elements, the one worth mentioning is $\top$, the l.u.b. of the whole lattice. It is to be thought of as an "over-determined" element. An example will make this clearer. The *lattice* of (non-negative) integers is formed by adjoining $\bot$ and $\top$ to the set $\{0, 1, 2, \ldots, n, \ldots\}$ of integers, and taking the 'trivial' partial ordering:

$$\bot \sqsubseteq x \sqsubseteq \top \qquad , x = 0, 1, \ldots$$

(plus reflexity, of course). Then the join of any two proper elements (ordinary integers) is $\top$, e.g. $2 \sqcup 3 = \top$, the reason being that there is no integer which 2 and 3 can be said to approximate. Thus $\top$ represents a kind of *inconsistency*; it is what one would get if the l.u.b. of contradictory approximations were taken. It should be noted that this is the case only for entirely contradictory elements. Thus, for the two functions

$$f(x) = \text{if } x=0 \text{ then } 0 \text{ else } x$$

$$g(x) = \text{if } x=0 \text{ then } 1 \text{ else } x$$

their join, in the appropriate lattice, will be the function

$$h(x) = \text{if } x=0 \text{ then } \top \text{ else } x,$$

which is not $\top$, i.e. is not the function whose value is $\top$ for all x.

Of course, we have only sketched the ideas for the 'lower' type spaces, but it can all be generalized to the 'higher-types'. I hope I have done justice to the ideas and indicated the sense in which the various lattice concepts are understood here. For further elaboration on this, and on the notion of continuity, which will be introduced shortly, the reader should consult the papers of Scott. As he points out, the use of partial functions to model the λ-calculus is not surprising, in view of the concern about whether wfes have normal forms or not. Some (but not all) wfes without normal forms will turn out to have 'value' $\perp$, the totally undefined function.

We now resume our account of assorted technical material needed for the later chapters. For the remainder of this section, D, D', etc., all denote complete lattices.

A function $f:D \to D'$ is *monotonic* if, for all $x,y \in D$:

$$x \sqsubseteq y \;\Rightarrow\; f(x) \sqsubseteq' f(y) .$$

A subset $X \subseteq D$ is *directed* if every finite subset $(x_1, x_2, \cdots, x_n) \subseteq X$ has an upper bound in X:

(3) $\quad x_1 \sqcup x_2 \sqcup \cdots \sqcup x_n \sqsubseteq y$, for some $y \in X$.

Note that (a) directed sets are always non-empty,

(b) finite sets are directed iff they contain their l.u.b.,

(c) to establish that a set is directed, it suffices to show that (3) holds for 2-element subsets of X.

Let $f:D \to D'$ and $X \subseteq D$, and consider the equation

(4) $$f(\bigsqcup X) = \bigsqcup' \{f(x) : x \in X\}$$

Then, f is said to be

(a) *continuous* iff (4) holds for all *directed* $X \subseteq D$,

(b) *finitely additive* iff (4) holds for *finite* $X \subseteq D$,

(c) *completely additive* iff (4) holds for *all* $X \subseteq D$.

Note that continuous functions are always monotonic. The three properties (a)-(c) connect together by

Theorem 1.2.1 (Scott)

A function is completely additive iff it is continuous and finitely additive.

The cartesian product $D \times D'$ of two complete lattices is structured as a complete lattice $[D \times D']$ by the component-wise partial ordering:

$$(x,x') \sqsubseteq (y,y') \text{ iff } x \sqsubseteq y \text{ and } x' \sqsubseteq' y'$$

Theorem 1.2.2 (Scott)

A function $f:[D \times D'] \to D''$ is continuous in both its variables together iff it is continuous in each separately.

When such a function f is continuous, then, for any *fixed* $x \in D$, the function $g:D' \to D''$ defined by

$$g(y) = f(x,y) \quad , \text{ for all } y \in D'$$

is a continuous function of y. As an alternative statement of this definition of g, the λ-*notation* may be used:

$$g = \lambda y:D'.f(x,y) \quad .$$

These results generalise to the product of any finite number of complete lattices in the obvious way.

Theorem 1.2.3 (Scott)

The set of all *continuous* functions from D to D' form a complete lattice, denoted by $[D \to D']$, under the partial ordering defined for $f, g \in [D \to D']$ by

$$f \sqsubseteq g \quad \Leftrightarrow \quad f(x) \sqsubseteq' g(x) \text{ for all } x \in D,$$

with l.u.b.s of subsets $F \subseteq [D \to D']$ given by

$$(\bigsqcup F)(x) = \bigsqcup' \{f(x) : f \in F\} \quad , \text{ for all } x \in D.$$

As a function of two variables, $f \in [D \to D']$ and $x \in D$, the operation $f(x)$ of application is continuous in x and completely additive in f.

Corollary 1.2.4 (Scott)

If $f : [D \times D'] \to D''$ is continuous, then the function $h : D \to [D' \to D'']$ defined by

$$h(x) = \lambda y{:}D'.f(x,y) \quad , \text{ for all } x \in D,$$

is also continuous.

Again using the λ-notation, this h may be written

$$h = \lambda x{:}D.\lambda y{:}D'.f(x,y) \quad .$$

These uses of λ-*notation* should not be confused with that of the ordinary λ-calculus. The notation is part of our *meta-language*, and is used as a means of directly "naming" an element of the appropriate function lattice. Thus, if x is a variable which may range over D, and if $F[x]$ is an expression which may involve x (more precisely, an applicative combination) denoting an element of the domain D', then, as x varies over D, with other parts of $F[x]$ remaining fixed, $F[x]$ defines a (continuous) function from D to D' *denoted* by $\lambda x{:}D.F[x]$.

We refer to this as *functional abstraction* with respect to x ranging over D.

From 1.2.3 and the anti-symmetry of $\sqsubseteq$, there follows immediately

Corollary 1.2.5

The property of *extensionality* holds in function lattices: for $f, g \in [D \to D']$,

$$f(x) = g(x) \text{ for all } x \in D \Rightarrow f = g$$

The following theorem of Tarski and its consequences will figure prominently in our study.

Theorem 1.2.6 (The fixed-point theorem)

If f is any *monotonic* function from a complete lattice D to itself, then the element

$$p = \bigsqcap \{x \in D : f(x) \sqsubseteq x\}$$

is the minimal fixed-point of f:

(a)(fixedness) $\quad f(p) = p$

(b)(minimality) $\quad f(x) \sqsubseteq x \Rightarrow p \sqsubseteq x$, for $x \in D$.

When f here is continuous, this result can be sharpened to

Theorem 1.2.7 (Scott)

If $f: D \to D$ is continuous, then $p = \bigsqcup_{n=0}^{\infty} f^n(\perp)$.

Further, the function $Y: [D \to D] \to D$ defined by

$$Y(f) = \bigsqcup_{n=0}^{\infty} f^n(\perp) \quad , \text{ for } f \in [D \to D]$$

is then a *continuous* function of f, and is called the minimal fixed-point operator (for D). It has the properties

(a)(fixedness) $\quad f(Y(f)) = f$

(b)(minimality) $\quad f(x) \sqsubseteq x \Rightarrow Y(f) \sqsubseteq x$, for $x \in D$.

A pair of continuous functions

$$\phi : D \to D' \quad , \quad \psi : D' \to D \ ,$$

form a *retraction* of D' onto D if

$$\psi(\phi(x)) = x \qquad \text{, for } x \in D$$

$$\phi(\psi(x')) \sqsubseteq' x' \qquad \text{, for } x' \in D'$$

ϕ id then a 1-1, completely additive mapping of D into D'. When ψ is also additive, then (ϕ,ψ) is said to be an *additive retraction*. (Since ψ is continuous, there is no ambiguity in the use of additive here, rather than completely additive, because of 1.2.1.)

Theorem 4.2.8 (Scott)

If (ϕ,ψ) is a retraction of D' onto D, then

$$\phi(x) \sqsubseteq' x' \Rightarrow \phi(x) \sqsubseteq' \phi(\psi(x')) \quad \text{, for } x \in D,\ x' \in D'.$$

Hence, $\phi(\psi(x'))$ is the *largest* (in the sense $\sqsubseteq'$) element in the image $\phi(D) \subseteq D'$ which approximates x'.

A third method ($\times$ and $\to$ were the first two) of composing lattices D and D' to form a larger one is that of a *lattice sum*, denoted by $[D + D']$. This is formed as an (almost) disjoint union of D and D'; the "almost" is because the greatest and least elements of D and D' are to be identified in the sum. Thus, the elements of the sum consist of $\perp$, $\top$, and all proper elements of D and D'. (If D and D' have any proper elements in common, it is assumed they have been indexed differently before the sum is formed.) For x,y in the sum lattice, $x \sqsubseteq y$ holds iff it held between x and y before the sum was formed (this requires $x \neq y$ to be in the same 'half' of the sum).

Associated with lattice sums are two retractions back onto the two halves of the sum. We give the one onto D, the other is similar. Define

$$\phi : D \to [D + D']$$

$$\psi : [D + D'] \to D$$

by

$$\phi(x) = x \quad , \text{ for } x \in D$$

$$\psi(x) = \begin{cases} x & , \text{ if } x \in D \\ \perp & , \text{ otherwise} \end{cases} \quad , \text{ for } x \in [D + D']$$

For these mappings to be continuous it is required that τ be *isolated* in D and D' ; τ is isolated in D if there does not exist a *directed* $X \subseteq D$ such that $\tau \notin X$ but $\bigsqcup X = \tau$.

The mapping ψ above is called the *projection* of $[D + D']$ onto D, denoted by

$$\psi(x) = (x|D) \quad , \text{ for } x \in [D + D']$$

The mapping ϕ above is called the *injection* of D into $[D + D']$, denoted by

$$\phi(x) = (x \text{ in } [D + D']) \quad , \text{ for } x \in D .$$

We now have all the tools we need for the construction in the next section. Lattice sums will not be used until later in Chapter 2.

§1.3 The inverse-limit construction

This section is a précis of [2,§2]. Space does not allow what would in any case be a repetition of Scott's proofs, but the details of the construction will be needed.

The construction starts with a fixed, but arbitrary, complete lattice D, from which certain 'higher-type' spaces are defined by induction:

$$D_0 = D$$
$$D_{n+1} = [D_n \to D_n] \quad , \text{ for } n = 0,1,2,\ldots,$$

where the lattice structure of D_{n+1} is that given by 1.2.3. The elements of each D_n are called the functions of *type level n*. So that the construction is non-trivial, it is assumed that D has more than one element.

The key which enables the right kind of "self-referential" domains to be constructed lies in making the hierarchy of functions lattices *cumulative*, in a manner reminiscent of (but not exactly the same as) the standard development of the cumulative theory of types. Each D_n is *embedded* in the next higher space, D_{n+1}, by means of identifications and mappings between them.

For $n=0$, we simply define mappings

$$\phi_0 : D_0 \to D_1 \quad , \quad \psi_0 : D_1 \to D_0$$

by

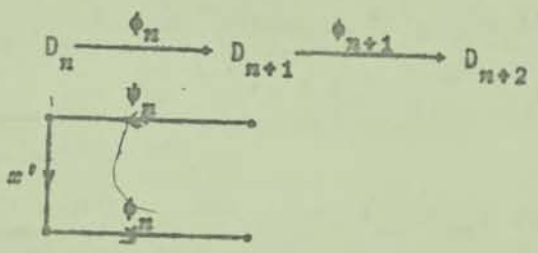
$$\phi_0(x) = \lambda y{:}D_0.x \quad , \text{ for } x \in D_0$$
$$\psi_0(x') = x'(\perp) \quad \text{for } x' \in D_1 .$$

Thus, ϕ_0 maps x into the function whose value is constantly x, and ψ_0 maps x' into the element $x'(\perp)$ which approximates all values in the range of x'.

For the higher-type spaces, a little more care is required. For $n \geq 0$, mappings

$$\phi_{n+1} : D_{n+1} \to D_{n+2} \ , \quad \psi_{n+1} : D_{n+2} \to D_{n+1}$$

are defined inductively by an "adjustment-of-types" suggested by the diagram, where $x' \in D_{n+1}$:



Thus, we define, for $x' \in D_{n+1}$,

$$\phi_{n+1}(x') = \phi_n \circ x' \circ \psi_n$$

$$= \lambda y' : D_{n+1} . \phi_n(x'(\psi_n(y'))) \ .$$

In words, first retract y' back onto D_n using ψ_n so that x' can then be applied, and embed the answer in D_{n+1} using ϕ_n. Similarly, for $x'' \in D_{n+1}$, define

$$\psi_{n+1}(x'') = \psi_n \circ x'' \circ \phi_n$$

$$= \lambda y : D_n . \psi_n(x''(\phi_n(y)))$$

Theorem 1.3.1 (Scott)

For all $n \geq 0$, the pair (ϕ_n, ψ_n) form an additive retraction of D_{n+1} onto D_n, and, for $x \in D_n$, $x' \in DD_{n+1}$,

$$\phi_n(x'(x)) = \phi_{n+1}(x')(\phi_n(x)) \ .$$

Further, by 1.2.8, $\phi_n(\psi_n(x'))$ is the *best* approximation in $\phi_n(D_n) \subseteq D_{n+1}$ to x'.

Now define mappings, for $n,m<\omega$,

$$\phi_{nm}: D_n \to D_m$$

which are compositions of the above ones to provide the embeddings of each D_n in each larger D_m by

$$\phi_{nm} = \lambda x{:}D_n.x \quad , \text{ if } n=m$$

$$\phi_{nm} = \phi_{(n+1)m} \circ \phi_n \quad , \text{ if } n<m$$

$$\phi_{nm} = \psi_m \circ \phi_{n(m+1)} \quad , \text{ if } n>m$$

Theorem 1.3.2 (Scott)

For $0\leq n\leq m<\omega$, the pair (ϕ_{nm},ϕ_{mn}) form an additive retraction of D_m onto D_n, and for $x \in D_n$, $x' \in D_{n+1}$,

$$\phi_{nm}(x'(x)) = \phi_{(n+1)(m+1)}(x')(\phi_{nm}(x)) \quad .$$

The limit space D_ω is now formed as a subset of

$$D_0 \times D_1 \times \cdots \times D_n \times \cdots$$

The *set* D_ω is defined by

$$D_\omega = \{\langle x_n\rangle_{n=0}^{\omega} : x_n \in D_n \text{ and } \psi_n(x_{n+1})=x_n \text{ , for all } n\}$$

Hereafter, we shall abbreviate sequences $\langle x_n\rangle_{n=0}^{\omega}$ by $\langle x_n\rangle$. Notice that not all sequences are taken, but only those for which $\psi_n(x_{n+1}) = x_n$ holds. It will then be the case that x_n is the *best* approximation in D_n to $x = \langle x_n\rangle \in D_n$.

Theorem 1.3.3 (Scott)

D_ω is a complete lattice under the partial ordering and l.u.b. defined "component-wise"

$$\langle x_n\rangle \sqsubseteq \langle y_n\rangle \quad \Leftrightarrow \quad x_n \sqsubseteq y_n \text{ for all } n\geq 0,$$

$$\bigsqcup X = \langle \bigsqcup\{x_n : x \in X\}\rangle \quad , \text{ for } X \subseteq D_\omega .$$

Associated with D_∞ are mappings

$$\phi_{n\infty}: D_n \to D_\infty \;, \quad \phi_{\infty n}: D_\infty \to D_n$$

defined by

$$\phi_{n\infty} = \lambda x{:}D_n.\langle \phi_{nm}(x)\rangle_{m=0}^{\infty}$$

$$\phi_{\infty n} = \lambda x{:}D_\infty.x_n$$

For $0 \le n < \infty$, $(\phi_{n\infty}, \phi_{\infty n})$ form an additive retraction of D_∞ onto D_n.

Notice that 1.3.2 shows that application, as a function of two arguments on the finite-type spaces, is 'preserved' in passing from $D_{n+1} \times D_n$ to $D_{m+1} \times D_m$, for $n \le m < \infty$. To simplify the notation, we can therfore justifiably adopt the following

Identification convention

Each element $x \in D_n$ is *identified* with the element $\phi_{nm}(x) \in D_m$, for all $0 \le n \le m \le \infty$, whence

(5) $$D_n \subseteq D_m \subseteq D_\infty \;,$$

and any ϕ_{nm} with $n \le m$ (this includes the special case $\phi_n = \phi_{n(n+1)}$) may be omitted as we wish.

For $x \in D_\infty$, subscripts will be used to denote the projection of x into each $D_n \subseteq D_\infty$:

(6) $$x_n = \phi_{n\infty}(\phi_{\infty n}(x)) = \phi_{nm}(x) \;, \quad 0 \le n < \infty$$

Theorem 1.3.4 For $x \in D_\infty$ and $n, m \ge 0$,

(a) $$x_n = \phi_{nn}(x_{n+1})$$

(b) $$x_n \sqsubseteq x_m \sqsubseteq x \;, \text{ for } n \le m$$

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

(d) $$(x_n)_m = x_k \;, \text{ where } k = \min(n, m)$$

(e) $$x \in D_n \iff x = x_n$$

(f) $$\perp_n = \perp \;, \quad \top_n = \top$$

To define *application* in D_ω, let $x = \langle x_n \rangle$ and $y = \langle y_n \rangle$ be any two elements of D_ω. For $n<\omega$, since $D_{n+1} = [D_n \to D_n]$, the application $x_{n+1}(y_n)$ is already well-formed as an operation on the finite-type spaces, and yields a value in D_n, which, by the convention above, is also in D_ω. If we form this application for each $n \geq 0$, a set of elements of D_ω is obtained. Application in D_ω can thus be defined as the l.u.b. of this set:

$$(7) \qquad x(y) = \bigsqcup_{n=0}^{\infty} x_{n+1}(y_n)$$

In reading this equation, one should be aware of where the ϕ_{nm}'s have been omitted. Thus, x and y have been first projected into D_{n+1} and D_n, respectively, the application performed in the known sense on $D_{n+1} \times D_n$, the result injected back into D_ω, and the l.u.b. of all results is then taken in D_ω.

Theorem 1.3.5 (Scott) As a function of two variables $x, y \in D_\omega$, the application operation (7) on $D_\omega \times D_\omega$ is completely additive in x and continuous in y.

Application and subscripting, in the special sense (6), are related by

Theorem 1.3.6 (Scott) For $x, y \in D_\omega$ and $n \geq 0$,

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

Corollary 1.3.7 (Scott) For $x, y \in D_\omega$

(a) $\qquad x_0(y) = \phi_0(x_0)(y) = x_0$

(b) $\qquad x(\perp)_0 = x_1(\perp) = \psi_0(x_1) = x_0 = x_0(\perp)$;

Proof. (a) From the identification $x_0 = \phi_0(x_0)$ and def. of ϕ_0.

(b) From 1.3.4(f) and 1.3.6 with $n=0$ and $y=\perp$; def. of ψ_0; 1.3.4(a) ; and part (a).

Theorem 1.3.8 (Scott) For $x, y \in D_\omega$,

(a) $x \sqsubseteq y \iff x(z) \sqsubseteq y(z)$ for all $z \in D_\omega$

(b) $x = y \iff x(z) = y(z)$ for all $z \in D_\omega$

Part (b) states that *extensionality* holds in D_ω.

Theorem 1.3.5 shows that every element of D_ω determines a continuous function from D_ω to itself. Conversely, every continuous function over D_ω can be represented by an element of D_ω, as follows:

Theorem 1.3.9 (Scott)

For any *continuous* function $f: D_\omega \to D_\omega$, let

$$\overline{f} = \bigsqcup_{n=0}^{\infty} (\lambda y{:}D_n . f(y)_n)$$

Then, $x(y) = f(y)$, for all $y \in D_\omega$.

From this last theorem and 1.3.5, we have

$$D_\omega \cong [D_\omega \to D_\omega]$$

under the correspondence

$$x \in D_\omega \longleftrightarrow \lambda y{:}D_\omega . x(y)$$

This isomorphism is to be added to the identification convention given earlier.

Note that (5) implies $\bigcup_{n=0}^{\infty} D_n \subseteq D_\omega$

In fact, it can be proved that D_ω is the completion, in the sense of $\sqsubseteq$ in D_ω, of this union.

A useful result for treating sequences of elements of D_ω is

Theorem 1.3.10 (Scott)

If each $u^{(n)} \in D_\omega$, and if $x = \bigsqcup_{n=0}^{\infty} u^{(n)}$, then

$x_n = u^{(n)}_n$ for all $n < \omega$ $\iff$ $(u^{(n+1)})_n = u^{(n)}_n$ for all $n < \omega$.

If, further, each $u^{(n)} \in D_n$, then

$$x_n = u^{(n)} \text{ for all } n<\omega \iff \psi_n(u^{(n+1)}) = u^{(n)} \text{ for all } n<\omega.$$

N.B. The superscript notation had to be used here since x_n now has the special meaning given in (6).

This theorem proves useful when specific sequences $\langle x_n \rangle_{n=0}^{\omega}$ for elements of D_ω are required. For it does *not* in general follow from the definition (7) of application in D_ω that

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

E.g., for

$$\Delta = \lambda x{:}D_\omega.\ x(x)$$

$$I = \lambda x{:}D_\omega.\ x$$

it is easy to show, using 1.3.10, that, for $n \geq 1$,

$$\Delta(I)_n = I_n$$

whereas $$\Delta_{n+1}(I_n) = I_{n-1}$$

For ease of reference, the construction is summarised in Table 2.

For further discussion, in particular on the relationship between the inverse limits obtained for different choices of D_0, see [2]. Much the same technique can be used to construct other kinds of self-referential domains, e.g. the lattice of flow diagrams [10].

Table 2

Construction of D_∞

D_0 = fixed complete lattice

$D_{n+1} = [D_n \to D_n]$, $n = 0,1,2,\cdots$

Retractions: $\phi_n : D_n \to D_{n+1}$, $\psi_n : D_{n+1} \to D_n$

$$\phi_0(x) = \lambda y{:}D_0.x$$
$$\psi_0(x') = x'(\perp)$$

$$\phi_{n+1}(x') = \phi_n \circ x' \circ \psi_n$$
$$\psi_{n+1}(x'') = \psi_n \circ x'' \circ \phi_n$$

The inverse limit

$$D_\infty = \{\langle x_n\rangle_{n=0}^{\infty} : \psi_n(x_{n+1}) = x_n \in D_n \text{ for all } n \geq 0\}$$

Identification:

$$x = \phi_{nm}(x) \quad \text{, for } x \in D_n,\ 0 \leq n \leq m \leq \infty.$$

whence $D_n \subseteq D_m \subseteq D_\infty$, for $0 \leq n \leq m \leq \infty$

Subscripting:

$$x_n = \phi_{n\infty}(\phi_{\infty n}(x)) = \phi_{\infty n}(x) \quad \text{, for } 0 \leq n < \infty$$

$$x_{n+1}(y) = x_{n+1}(y_n) = x(y_n)_n \quad \text{, } 0 \leq n < \infty$$

$$x(\perp)_0 = x_1(\perp) = \psi_0(x_1) = x_0.$$

D_∞-application

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

Extensionality

$$x = y \iff x(z) = y(z) \text{ for all } z \in D_\infty$$

Isomorphism

$$D_\infty \cong [D_\infty \to D_\infty]$$

$$x \in D_\infty \iff \lambda y{:}D_\infty.x(y)$$

CHAPTER 2

Lattice-theoretic Models of the λ-calculus

In this chapter and the next, we focus our attention on the semantic aspects for the axiomatic formulation, A, of the λ-calculus. A model of A with the complete lattice D_∞ as its domain will be described, then we shall investigate how the structure of this model fits with and reflects our intuitive ideas about the λ-calculus.

§2.1 A model with domain D_∞

A wfe ε can be said to be representable by an element $\bar{\varepsilon}$ of D_∞ (i.e. by a sequence $\langle x_n \rangle_{n=0}^{\infty}$ with $x_n \in D_n$, etc.) if, roughly speaking, the behaviour of $\bar{\varepsilon}$ under the special sense of application in D_∞ "parallels" that of ε under conv. For example, the combinator

$$K = \lambda x.\lambda y.x$$

is representable by $\bar{K}$ in D_∞ if

$$\bar{K}(x)(y) = x \quad \text{, for all } x,y \in D_\infty,$$

where application on the LHS is to be read as application in D . For the combinators K and

$$S = \lambda x.\lambda y.\lambda z.xz(yz) \quad ,$$

suitable sequences are given by

$$K_{n+2} = \lambda x{:}D_{n+1}.\lambda y{:}D_n.\psi_n(x) \quad \text{, for } n \geq 0,$$
$$K_1 = \psi_1(K_2) = \lambda x{:}D_0.x$$
$$K_0 = \psi_0(K_1) = \perp$$

$$S_{n+3} = \lambda x{:}D_{n+2}.\lambda y{:}D_{n+1}.\lambda z{:}D_n.x(\phi_n(z))(yz) \quad , n \geq 0$$

$$S_2 = \psi_2(S_3) = \lambda x{:}D_1.\lambda y{:}D_0.x(\Delta)$$

$$S_1 = \psi_1(S_2) = \lambda x{:}D_0.x$$

$$S_0 = \psi_0(S_1) = \Delta$$

All the standard combinators are similarly representable in D_ω. Our interest in S and K derives from the fact that they form a *primitive set of combinators* - i.e. any wfe can be expressed as a combination of S, K and its free variables. Hence all wfes are representable in D_ω.

Unfortunately, this method alone is inadequate. Apart from the inconvenience of first having to re-write wfes in terms of S and K (or some other primitive set of combinators) before interpreting them, there are more fundamental objections to be overcome. The 'expressibility' of wfes as S-K-combinations relies on properties of conversion whose validity has not yet been established. Furthermore, the combinatorial rules

$$K(x)(y) = x \quad , \text{ for all } x,y$$

$$S(x)(y)(z) = xz(yz) \quad , \text{ for all } x,y,z$$

for S and K alone yield a 'weaker' system than that based on cnv. Sets of combinatorial rules fully equivalent to cnv can be given, but these are always many in number and difficult to work with.(+)

This is not to say that the above method is vacuous, for it does confirm the *feasibility* of a λ-calculus model

---

(+) In [1], Church lists no less than 38 rules for a full combinatory equivalent of conversion, using the combinators $I=(\lambda a.a)$ and $J=(\lambda a.\lambda b.\lambda c.\lambda d.ab(adc))$ as a primitive set (for the λI-calculus).

with domain D_ω. And, once the required properties of cnv have been validated, wfes *could* be interpreted in this way, albeit somewhat indirect.

To establish a model with domain D_ω, a more direct approach along the conventional lines of model theory is called for. Thus, an interpretation of the system A must be given and its axioms verified.

This interpretation will be formed by the following assignment of relations and operations (of the appropriate degree) over D_ω to the primitive symbols listed in Table 1:

(I1) Variables are interpreted as ranging over D_ω.

(I2) f_1^2 is interpreted as the binary operation of functional application on $D_\omega \times D_\omega$.

(I3) For $i=1,2,\cdots$, f_i^1 is interpreted as the unary operation of functional abstraction with respect to ξ_i ranging over D_ω.

(I4) A_1^2 is interpreted as the binary relation of lattice equality in D_ω.

Continuing in this way, the interpretation of the terms and formulae can be derived by the obvious structural induction and we could then verify that the axioms of A are all 'true' under this interpretation.

Such a rigid adherence to formality is unnecessary however. We choose to follow a more informal presentation which is no less precise, but is more immediately transparent. We adopt this treatment safe in the knowledge

that, if we so desired, the presentation can be made formally exact along the lines sketched above. In any case, the clauses and equations we shall write are not unlike the usual definitions of such concepts from logic as 'satisfiability' and 'truth' (c.f. [18,p.50-51]).

§2.2 Semantics of the λ-calculus

The approach we follow treats the λ-calculus as a simple 'programming' language - a language of 'pure', type-free functions. From this standpoint, the current section is a prototype for definitions of the *semantics* of more conventional programming languages.

The first part of the description of any formal language is its *syntax* - the delineation of the well-formed expressions (or sentences) of the language. In §1.1.1, the syntax of the λ-calculus was given in the BNF-like form:

$$\varepsilon ::= \xi \mid \varepsilon_0(\varepsilon_1) \mid (\lambda\xi.\varepsilon)$$

where ξ and ε are meta-variables ranging over the classes Id and Exp of all identifiers and all wfes, respectively. In writing wfes below, ambiguities about missing parentheses are to be resolved by the notational conventions in §1.1.1.

Syntax can always be viewed in several ways, so it is well to be clear at the outset of the view we take here. In a *concrete* syntax, expressions consist merely of strings of symbols with no underlying structure; in an *abstract* syntax, expressions are formed as structured objects built

up from, and decomposable into, their constituent sub-parts. There is an operation known as *parsing* which links these two syntaxes, providing a structural analysis, called a *parse tree*, of those symbol strings which represent well-formed expressions.

For our semantic purposes, we need only know that suitable parsers for the λ-calculus can be defined; the vagaries of any particular parser need not concern us. Accordingly, though we shall enclose *text* (i.e. symbol strings) within a special pair of brackets, ⟦ and ⟧, such text will be regarded as being in abstract, parsed form. Thus, in writing $⟦\varepsilon_0(\varepsilon_1)⟧$, we designate a structured object which is an instance of a *combination*, whose *rator* and *rand*, respectively, are denoted by ε_0 and ε_1. Similarly, $⟦\lambda\xi.\varepsilon⟧$ designates an *abstraction*, whose *bv* and *body*, respectively, are denoted by ξ and ε.

Basic to this presentation of semantics is the manner in which variables are treated. Variables in formal languages correspond to pronouns in verbal languages - they *stand for* objects and can be used in different contexts to denote different objects. Just as sentences in natural languages are meaningless until pronouns are connected with their *antecedents*, so with variables their *denotations* must be known before any meaning can be attributed to terms involving variables. For the variables (identifiers) of λ-calculus, their denotations may range over the domain D_ω. In abstract terms, this association of 'values' with identifiers is nothing more than a mapping from identifiers to elements

of D_∞, denoted by

$$\rho : Id \to D$$

where, for the sake of simplicity, D will now be written for D_∞, so that

$$D \cong [D \to D]$$

We let Env stand for the class of all such mappings, ρ, or *environments*, as they will be called

Given an environment, the meaning of a wfe will now be defined in terms of that of its immediate sub-parts. In this way, each ε,ρ-pair is associated with an element of D, referred to as the *value* of ε in an environment ρ. This is just another mapping, $\mathcal{V}$, defined on the cartesian product Exp X Env into D. In fact, it proves convenient to "curry" this function, so that $\mathcal{V}$ takes a wfe as its argument and produces a function from Env to D as its result

$$\mathcal{V} : Exp \to [Env \to D]$$

Corresponding to each of the three cases in the syntax of wfes, there is one *semantic clause* in the structural definition of the function $\mathcal{V}$:

(S1). Identifiers

The meaning of an identifier, ξ, in an environment, ρ, is ascertained by "looking up" its value in ρ. In view of the functional nature of ρ, this is achieved by application of ρ to ξ.

$$\mathcal{V}[\![\xi]\!](\rho) = \rho[\![\xi]\!]$$

(S2). Combinations

The isomorphism between B and [D → D] derived from the well-definedness of the operation $\bigsqcup_{n=0}^{\infty} a_{n+1}(y_n)$ of application for all pairs a,y in D. For a combination, $c_0(c_1)$, the value of its rator can be construed as function over D and so be meaningfully applied to the value of its rand. Hence, we define

$$v[\![c_0(c_1)]\!](\rho) = v[\![c_0]\!](\rho) \,\dot{+}\, (v[\![c_1]\!](\rho)) \ ,$$

where the application indicated by $\dot{+}$ is in D.

(S3). Abstractions

An abstraction, $\lambda\xi.t$, in an environment, ρ, is interpreted as a function from D to D (itself in D by isomorphism). When applied to an argument β from D, the result of this function is the value of its body, c, in an environment, ρ', identical to ρ in all respects, except that the bound variable, ξ, is now associated with the argument β. We shall use the notation $\rho[\beta/\xi]$ for this *extended environment*, ρ', defined by

$$\rho[\beta/\xi] = \lambda\xi'{:}Id.((\xi' \theta \xi) \to \beta,\ \rho[\xi'])$$

As β varies over D, $v[\![c]\!](\rho[\beta/\xi])$ determines the function as which $(\lambda\xi.c)$ is to be interpreted:

$$v[\![\lambda\xi.c]\!](\rho) = \lambda\beta{:}D.v[\![c]\!](\rho[\beta/\xi])$$

These semantics are summarised in Table 3.

Remember that the 'λ'-s on the RHS of the last two equations above are not the same as the 'λ'-s occurring in wfes but are used merely as an expository device, as part of our meta-language.

Table 8

D_∞-semantics of the λ-calculus

Syntax

$$\varepsilon ::= \xi \mid \varepsilon_0(\varepsilon_1) \mid (\lambda\xi.\varepsilon)$$

Notation

Id	Class of all identifiers
Exp	Class of all wfes
$D \cong [D \to D]$	Domain of values of wfes
$\beta : D$	Elements of the domain D

Semantic functions

$\rho : Id \to D$	The environment-mapping
Env	Class of all environments

$$\rho[\beta/\xi] = \lambda\xi' : Id.((\xi' = \xi) \to \beta,\ \rho[\xi'])$$

$$\nu : Exp \to [Env \to D]$$

(S1) $\quad \nu[\![\xi]\!](\rho) = \rho[\xi]$

(S2) $\quad \nu[\![\varepsilon_0(\varepsilon_1)]\!](\rho) = \nu[\![\varepsilon_0]\!](\rho)(\nu[\![\varepsilon_1]\!](\rho))$

(S3) $\quad \nu[\![\lambda\xi.\varepsilon]\!](\rho) = \lambda\beta : D.\ \nu[\![\varepsilon]\!](\rho[\beta/\xi])$

§2.3 Semantic equivalence

Using the definitions of the last section, a semantic counterpart of the notion of convertibility between wfes will now be defined, and the truth of the axioms in Table 1 established by a series of lemmas.

Definition. Two wfes $\varepsilon_0, \varepsilon_1$ are said to be *(semantically) equivalent*, written $\varepsilon_0 \equiv \varepsilon_1$, if they have the same value in D for *all* assignments of values to their free variables:

$$\varepsilon_0 \equiv \varepsilon_1 \iff \nu[\![\varepsilon_0]\!](\rho) = \nu[\![\varepsilon_1]\!](\rho) \quad \text{for all } \rho .$$

It should be noted that this relation is genuinely *semantic*, rather than *syntactic*, in nature; in fact, it is *not* a recursively enumerable relation.

Lemma 2.3.1 $\equiv$ is an equivalence relation.

Proof. Immediate from the corresponding property of lattice equality in D.

Lemma 2.3.2

(a) $\varepsilon_0 \equiv \varepsilon_1 \Rightarrow (\lambda\xi.\varepsilon_0) \equiv (\lambda\xi.\varepsilon_1)$, for $\xi \in Id$

(b) $\varepsilon_0 \equiv \varepsilon_0',\ \varepsilon_1 \equiv \varepsilon_1' \Rightarrow \varepsilon_0(\varepsilon_1) \equiv \varepsilon_0'(\varepsilon_1')$

(N.B. Axioms (μ) and (ν) are special cases of (b).)

Proof. Let ρ be any environment.

(a) Suppose $\varepsilon_0 \equiv \varepsilon_1$. Then,

$\nu[\![\lambda\xi.\varepsilon_0]\!](\rho) = \lambda\beta{:}D.\ \nu[\![\varepsilon_0]\!](\rho[\beta/\xi])$, by semantic clause (S3)

$= \lambda\beta{:}D.\ \nu[\![\varepsilon_1]\!](\rho[\beta/\xi])$, from $\varepsilon_0 \equiv \varepsilon_1$ and equality of lattice functions

$= \nu[\![\lambda\xi.\varepsilon_1]\!](\rho)$, by (S3)

(b) Suppose $\varepsilon_i \equiv \varepsilon_i'$, $i=0,1$. Then,

$$\begin{aligned}\nu[\varepsilon_0(\varepsilon_1)](\rho) &= \nu[\varepsilon_0](\rho)(\nu[\varepsilon_1](\rho)) && \text{, By (S2)}\\ &= \nu[\varepsilon_0'](\rho)(\nu[\varepsilon_1'](\rho)) && \text{, since } \varepsilon_i \equiv \varepsilon_i'\\ &= \nu[\varepsilon_0'(\varepsilon_1')](\rho) && \text{, by (S2)}\end{aligned}$$

■

Theorem 2.3.3 $\equiv$ is substitutive:

$\varepsilon_0 \equiv \varepsilon_1 \Rightarrow C[\varepsilon_0] \equiv C[\varepsilon_1]$ for all contexts $C[\,]$

Hence, replacement of any sub-part of a wfe by a $\equiv$-equivalent wfe doesn't change the meaning of the whole expression.

Proof. Immediate from 2.3.2, since (μ), (ν), (θ) together imply (*Subst*).

To prove the validity of conversion in D, we first state some properties of environments.

Lemma 2.3.4

For any $\rho \in Env$, and all $\beta,\beta' \in D$, $\xi,\xi' \in Id$,

(a) $\xi = \xi' \Rightarrow \rho[\beta/\xi][\beta'/\xi'] = \rho[\beta'/\xi']$

(b) $\xi \neq \xi' \Rightarrow \rho[\beta/\xi][\beta'/\xi'] = \rho[\beta'/\xi'][\beta/\xi]$

Proof. Immediate from the definition of extended environments.

Lemma 2.3.5

If ξ does not occur free in ε, then, for all $\rho \in Env$, $\beta \in D$,

$$\nu[\varepsilon](\rho[\beta/\xi]) = \nu[\varepsilon](\rho) \quad .$$

Proof. This lemma and 2.3.7 below are proved by very similar structural inductions on ε. Since the details for the latter are somewhat less trivial, we choose to prove 2.3.7 rather than the present, almost self-evident lemma.

Corollary 2.3.6

If ξ does not occur free in ε, then, for all $\rho \in Env$, $\beta,\beta' \in D$,

$$\nu[\varepsilon](\rho[\beta/\xi][\beta'/\xi']) = \nu[\varepsilon](\rho[\beta'/\xi']) .$$

Proof. Immediate from the two previous lemmas.

Lemma 2.3.7 (The substitution lemma)

For all $\varepsilon,\varepsilon_0 \in Exp$, $\xi \in Id$, and $\rho \in Env$,

(1) $$\nu[[\varepsilon_0/\xi]\varepsilon](\rho) = \nu[\varepsilon](\rho[\nu[\varepsilon_0](\rho)/\xi]) .$$

Proof. By induction on the *rank* of ε. This proof should be read in conjunction with the definition of substitution in §1.1.2. Denote

$\varepsilon' = [\varepsilon_0/\xi]\varepsilon$

$\rho' = \rho[\nu[\varepsilon_0](\rho)/\xi]$

$L = \nu[\varepsilon'](\rho)$ = LHS of (1)

$R = \nu[\varepsilon](\rho')$ = RHS of (1)

Case1. ε is an identifier ξ'

Then, $$R = \nu[\xi'](\rho') = \rho'[\xi'] = \begin{cases} \nu[\varepsilon_0](\rho) & \text{, if } \xi' = \xi \\ \rho[\xi'] & \text{, if } \xi' \neq \xi \end{cases}$$

If $\xi' = \xi$, then $\varepsilon' = \varepsilon_0$, so $L = \nu[\varepsilon_0](\rho) = R$.

If $\xi' \neq \xi$, then $\varepsilon' = \xi'$, so $L = \nu[\xi'](\rho) = \rho[\xi'] = R$.

Case 2. ε is a combination $\varepsilon_1(\varepsilon_2)$

Then, $\varepsilon' = \varepsilon_1'(\varepsilon_2')$, where $\varepsilon_i' = [\varepsilon_0/\xi]\varepsilon_i$ for $i=1,2$. The induction hypothesis implies that (1) holds for ε_1 or ε_2 in place of ε. Then,

$L = \nu[\varepsilon_1'(\varepsilon_2')](\rho) = \nu[\varepsilon_1'](\rho)(\nu[\varepsilon_2'](\rho))$, By (S2)

$= \nu[\varepsilon_1](\rho')(\nu[\varepsilon_2](\rho'))$, ind.hyp.

$= \nu[\varepsilon_1(\varepsilon_2)](\rho') = R$, by (S2).

Case 3. ε is an abstraction $\lambda\xi'.\varepsilon_1$

(a) If $\xi'=\xi$, or if ξ not free in ε_1, then $\varepsilon' = \lambda\xi'.\varepsilon_1$, so

$$L = \nu[\lambda\xi'.\varepsilon_1](\rho) = \lambda\beta':D.\nu[\varepsilon_1](\rho[\beta'/\xi'])$$

$$R = \nu[\lambda\xi'.\varepsilon_1](\rho') = \lambda\beta':D.\nu[\varepsilon_1](\rho'[\beta'/\xi'])$$

From the definition of ρ', the result now follows by 2.3.4(a) if $\xi'=\xi$, and by 2.3.6 if $\xi'\neq\xi$ and ξ not free in ε_1.

(b) Otherwise, i.e. if $\xi'\neq\xi$ and ξ free in ε_1, let ξ'' be the identifier determined in Case 3(b) of the definition of substitution. Whether (i) or (ii) there is the case, we have $\xi''\neq\xi$ and ξ'' not free in ε_0. If (i) holds, then $\xi''=\xi'$; if (ii) holds, then $\xi''\neq\xi'$ and ξ'' not free in ε_1. Let

$$\varepsilon_2 = [\xi''/\xi']\varepsilon_1 \quad , \quad \varepsilon_2' = [\varepsilon_0/\xi]\varepsilon_2$$

Then,

$$\varepsilon' = \lambda\xi''.\varepsilon_2' \quad ,$$

and since

$$rank(\varepsilon_2) = rank(\varepsilon_1) < rank(\varepsilon) \quad ,$$

the induction hypothesis implies that (1) holds (for *any* ε_0,ξ,ρ) with ε_1 or ε_2 in place of ε. Then,

$L = \nu[\lambda\xi''.\varepsilon_2'](\rho) = \lambda\beta:D.\nu[\varepsilon_2'](\rho[\beta/\xi''])$, by (S2)

$= \lambda\beta:D.\nu[\varepsilon_2](\rho'')$, by ind.hyp.(1) for ε_2 with $\rho[\beta/\xi'']$ for ρ

where $\rho'' = \rho[\beta/\xi'']['\nu[\varepsilon_0](\rho[\beta/\xi''])/\xi]$

$= \rho[\beta/\xi'']['\nu[\varepsilon_0](\rho)/\xi]$, by 2.3.5, since ξ'' not free in ε_0

$= \rho['\nu[\varepsilon_0](\rho)/\xi][\beta/\xi'']$, by 2.3.4(b), since $\xi''\neq\xi$

$= \rho'[\beta/\xi'']$, by def. of ρ'

$= \lambda\beta:D.\nu[[\xi''/\xi']\varepsilon_1](\rho'')$, by def. of ε_2

$L = \lambda\beta{:}D.v[e_1](\rho''[v[\xi''](\rho'')/\xi'])$, by ind.hyp.(1) for e_1 with ξ'', ξ', ρ'' for e_0, ξ, ρ

$= \lambda\beta{:}D.v[e_1](\rho''[\beta/\xi'])$, by (S1) and def. of ρ''

$= \lambda\beta{:}D.v[e_1](\rho'[\beta/\xi''][\beta/\xi'])$, inserting def. of ρ''

$= \lambda\beta{:}D.v[e_1](\rho'[\beta/\xi'])$, { by 2.3.4(a) if case (i) for ξ'' / by 2.3.6 if case (ii) for ξ''

$= v[\lambda\xi'.e_1](\rho')$, by (S3)

$= R$

■

Lemma 2.3.8

(a) α-conversion is valid: if ξ' not free in e, then

$$\lambda\xi.e = \lambda\xi'.[\xi'/\xi]e$$

(b) β-redexes are equivalent to their contractums:

$$(\lambda\xi.e)e_0 = [e_0/\xi]e$$

(c) η-conversion is valid: if ξ not free in e, then

$$\lambda\xi.e(\xi) = e$$

Proof. Let ρ be any environment.

(a) $v[\lambda\xi'.[\xi'/\xi]e](\rho) = \lambda\beta{:}D.v[[\xi'/\xi]e](\rho[\beta/\xi'])$, by (S3)

$= \lambda\beta{:}D.v[e](\rho[\beta/\xi'][v[\rho[\beta/\xi']]])$, by 2.3.7

$= \lambda\beta{:}D.v[e](\rho[\beta/\xi'][\beta/\xi])$, by (S1)

$= \lambda\beta{:}D.v[e](\rho[\beta/\xi])$, by 2.3.6, since ξ' not free in e

$= v[\lambda\xi.e](\rho)$, by (S3)

(b) $v[(\lambda\xi.e)e_0](\rho) = (\lambda\beta{:}D.v[e](\rho[\beta/\xi]))(v[e_0](\rho))$, by (S2, S3)

$= v[e](\rho[v[e_0](\rho)/\xi])$, by application in D

$= v[[e_0/\xi]e](\rho)$, by 2.3.7

(c) $\nu[\lambda\xi.e(\xi)](\rho) = \lambda\beta{:}D.\nu[e(\xi)](\rho[\beta/\xi])$, By (S3)

$= \lambda\beta{:}D.\nu[e](\rho[\beta/\xi])(\nu[\xi](\rho[\beta/\xi]))$, by (S2)

$= \lambda\beta{:}D.\nu[e](\rho[\beta/\xi])(\beta)$, By (S1)

$= \lambda\beta{:}D.\nu[e](\rho)(\beta)$, by 2.3.5, since ξ not free in e

$= \nu[e](\rho)$, by extensionality in D

■

Theorem 2.3.9

$$e_0 \text{ cnv } e_1 \Rightarrow e_0 = e_1$$

Proof. Follows from 2.3.3 and 2.3.8 by definition of cnv and transitivity of =.

Summarising §§2.2,2.3, we have

Theorem 2.3.10

The semantics given in Table 3 and the relation = defined above constitute a model for the λ-calculus system based on α-β-η-conversion.

§2.4 Properties of D_ω-model

It is rarely the case that the models of a formal system are 'isomorphic' to the system itself; in general, the models have a much richer structure and many stronger properties hold in them.

The D_ω-model of the λ-calculus is no exception to this 'rule'. To begin with, D_β is much more than a *set* of objects, it is a *complete lattice*, with the interesting

partial order relation $\sqsubseteq$. Just as lattice equality determined the relation $=$, so $\sqsubseteq$ induces a partial order relation on Exp, which we might as well denote by the same symbol $\sqsubseteq$.

Definition. $e_0 \sqsubseteq e_1 \Leftrightarrow v[e_0](\rho) \sqsubseteq v[e_1](\rho)$ *for all* ρ.

Theorem 2.4.1

The relation $\sqsubseteq$ is substitutive:

$e_0 \sqsubseteq e_1 \Rightarrow C[e_0] \sqsubseteq C[e_1]$ for all contexts C[].

Proof. Analogous to 2.3.3.

Suppose e is a wfe without free variables (a *combinator*). Using (S1)-(S3) to fully expand $v[e](\rho)$, we then obtain an expression which is independent of the environment ρ, e.g.

$$v[\lambda x.\lambda y.yx](\rho) = \lambda\beta_1{:}D.\lambda\beta_2{:}D.v[yx](\rho') \quad \text{, by (S3) twice}$$

where $\rho' = \rho[\beta_1/x][\beta_2/y]$

$$= \lambda\beta_1{:}D.\lambda\beta_2{:}D.v[y](\rho')(v[x](\rho')) \quad \text{, by (S2)}$$

$$= \lambda\beta_1{:}D.\lambda\beta_2{:}D.\beta_2(\beta_1) \quad \text{, by (S1) twice and def. of } \rho'$$

Moreover, apart from the indication of types on its bound variables, this expanded expression is (up to α-conversion) just the same as e itself. For simplicity, we shall therefore allow combinators themselves to denote their values in D. For example, $(\lambda x.x)$ and $(\lambda x.\lambda y.x)$ stand for the *functions* I and K, respectively, from D to D, defined by

$$I(x) = x \quad \text{, for all } x \in D$$

$$K(x)(y) = x \quad \text{, for all } x,y \in D .$$

Similarly, if e contains free variables, the full expansion of $\nu[\![e]\!](\rho)$ depends only on the values of these free variables in the environment ρ. Hence, e can be regarded as a *parametric form* for its value under suitable assignments to its free variables.

We might say that this convention is an *embedding* of the λ-calculus in D. In applying this convention, by virtue of 2.3.9, ordinary λ-conversion may be used to manipulate (expressions denoting) elements of D.

Many of the results derived so far have asserted the equality of elements in D or the $=_\beta$-equivalence of various wfes. Since models are not of much interest if they are inconsistent, it is pertinent to ask whether non-equivalent wfes exist.

Lemma 2.4.2

The combinators $K=(\lambda x.\lambda y.x)$ and $K(I)=(\lambda x.\lambda y.y)$ are *incomparable*:

$$K \not\sqsubseteq K(I) \qquad K(I) \not\sqsubseteq K$$

Proof. Suppose $K \sqsubseteq K(I)$. By monotonicity of functions over D, we would then have

$$K\,d\,e\,\rho(y) \sqsubseteq K(I)\,d\,e\,\rho(y)$$

whence $$x \sqsubseteq y$$

Since this holds for *all* $x,y \in D$, not only would all elements of D be comparable (under $\sqsubseteq$), they would all be equal. Hence, the supposition that $K \sqsubseteq K(I)$ contradicts the facts that D has an infinite number of

elements and contains incomparable elements(+).

The proof that $K(I) \not\subseteq K$ is similar.

Thus, $K \neq K(I)$ and the D_∞-model is *non-trivial*. Searching deeper, we ask how close a match there is between the *semantic* notion of equivalence introduced via the model and the purely formal, *syntactic* relation of convertibility between wfes. In particular, under what conditions does the converse of 2.3.9 hold ?

The following theorem enables us to resolve this question for wfes reducible to normal form.

Theorem 2.4.3 (Böhm [20]) (++)

For two distinct wfes M_0, M_1 in β-η-normal form, we can find variables $z_1, z_2, \cdots, z_t$ $(t \geq 0)$, combinators $U_1, U_2, \cdots, U_t$, and wfes $H_1, H_2, \cdots, H_o$ $(o \geq 0)$ such that

$$M_i' H_1 H_2 \cdots H_o \text{ cnv } v_i \quad , \quad i = 0,1 \ ,$$

where $M_i' = [U_1, \cdots, U_t / z_1, \cdots, z_t] M_i \quad , \ i = 0,1 \ ,$

and the variables v_0, v_1 occur free in H_k, for some $1 \leq k \leq o$.

A notationally simpler statement of this theorem is

Theorem 2.4.4

For any two wfes e_0, e_1 reducible to distinct β-η-normal forms, we can find a context C[] such that

$$C[e_i] \text{ cnv } v_i \quad , \quad i = 0,1 \ .$$

(+) Provided the initial D_0 in the construction of $D_\infty (= D)$ has more than one element.

(++) An account of the proof of this theorem is given in the Appendix, where we also prove a generalization of Böhm's theorem to be stated in Chapter 3.

Proof. From the last theorem, take

$$C[\,] = (\lambda x_1 x_2 \cdots x_t.[\,])U_1 \cdots U_t H_1 \cdots H_o \quad .$$

We can now derive the following partial converse to 2.3.9.

Theorem 2.4.5

If $\varepsilon_0, \varepsilon_1$ are *both* reducible to β-η-normal form, then

$$\varepsilon_0 \textit{ not-cnv } \varepsilon_1 \Rightarrow (\varepsilon_0 \not\sqsubseteq \varepsilon_1 \textit{ and } \varepsilon_1 \not\sqsubseteq \varepsilon_0) \Rightarrow \varepsilon_0 \neq \varepsilon_1 \ .$$

Proof. Suppose ε_0 *not-cnv* ε_1, and (using 2.4.4) let $C[\,]$ be a context such that $C[\varepsilon_i]\text{cnv}v_i$, for $i = 0,1$. Also, let ρ be an environment such that $\rho[v_0]=K$ and $\rho[v_1]=K(I)$, so that

$$\nu[C[\varepsilon_i]](\rho) = \nu[v_i](\rho) = \begin{cases} K & \text{, for } i=0 \\ K(I) & \text{, for } i=1 \end{cases}$$

Then, we would have

$$\begin{aligned} \varepsilon_0 \sqsubseteq \varepsilon_1 &\Rightarrow C[\varepsilon_0] \sqsubseteq C[\varepsilon_1] && \text{, by 2.4.1} \\ &\Rightarrow \nu[C[\varepsilon_0]](\rho) \sqsubseteq \nu[C[\varepsilon_1]](\rho) && \text{, by def. of } \sqsubseteq \\ &\Rightarrow K \sqsubseteq K(I) \end{aligned}$$

Hence, $\varepsilon_0 \sqsubseteq \varepsilon_1$ would contradict 2.4.2. Similarly, $\varepsilon_1 \sqsubseteq \varepsilon_0$ also leads to a contradiction. ∎

The substance of Böhm's theorem is that wfes having distinct β-η-normal forms are "really" distinct. If one were to postulate, as an extra axiom for the λ-calculus, the equality of two such wfes, the resulting system would be *inconsistent*, i.e. all wfes could then be proved equal.

In the next section we show that this result does not extend to wfes without normal forms.

§2.5 Fixed-point operators

From here on, extensive use will be made of the identification conventions introduced earlier, viz.

(a) Combinators themselves may be used to denote their values in D_∞, e.g. $Y_\lambda = \nu[Y_\lambda](\rho)$ for all ρ, and can be manipulated by ordinary λ-conversion.

(b) For $0 \le n \le m \le \infty$, each $x \in D_n$ is identified with $\phi_{nm}(x) \in D_m$, and hence $D_n \subseteq D_m \subseteq D_\infty$.

(c) For $x \in D_\infty$ and $n \ge 0$, x_n denotes $\phi_{n\infty}(\phi_{\infty n}(x))$.

In manipulating subscripts, 1.3.4, 1.3.6 and 1.3.7 will be used many times.

We consider the λ-calculus *paradoxical combinator*

$$Y_\lambda = \lambda f.(\lambda y.f(yy))(\lambda y.f(yy))$$

so called because of its *fixed-point property*:

$$Y_\lambda(f) \text{ cnv } f(Y_\lambda(f))$$

The theorem which follows was first proved by David Park.

Theorem 2.5.1

In D_∞, Y_λ is equal to the lattice-theoretic operator

$$Y = \lambda f{:}D_\infty.\ \bigsqcup_{n=0}^{\infty} f^n(\perp) \quad .$$

Proof. Since Y_λ is a fixed-point operator and Y obtains the *minimal* fixed-point of continuous functions (by 1.2.7) $Y \sqsubseteq Y_\lambda$ must hold.

For the converse relation, by virtue of extensionality in D_∞, it suffices to show that

$$Y_\lambda(f) \sqsubseteq \bigsqcup_{n=0}^{\infty} f^n(\perp) \quad , \ f \in D_\infty \ .$$

Let $X = (\lambda y.f(yy))$

so that $Y_\lambda(f) = X(X)$

We first prove four lemmas about the projections of X into the "finite-type" spaces.

Lemma 2.5.2

$$X_{n+1}(X_n) = f_{n+1}(X_n(X_{n-1})) \quad \text{, for } n \geq 1 .$$

Proof. $X_{n+1}(X_n) = X(X_n)_n$, By 1.3.6

$= f(X_n(X_n))_n$, by def. of X

$= f_{n+1}(X_n(X_n)_n)$, by 1.3.6

and the result follows since

$$X_n(X_n)_n = (X_n)_{n+1}(X) = X_n(X) = X_n(X_{n-1})$$

by 1.3.6, 1.3.4(d) and 1.3.6. ■

Lemma 2.5.3 $X_0 = f_0(\perp)$.

Proof. $X_0 = X(\perp)_0$, by 1.3.7(b)

$= f(\perp(\perp))_0$, by def. of X

$= f(\perp)_0$, since $\perp(\perp)=\perp$

$= f_0(\perp)$, by 1.3.7(b) ■

Lemma 2.5.4 $X_1(X_0) = f_1(f_0(\perp))$.

Proof. $X_1(X_0) = X(X_0)_0$, by 1.3.6

$= f(X_0(X_0))_0$, by def. of X

$= f(X_0)_0$, by 1.3.7(a) with $y=X_0$

$= f_1(X_0)$, by 1.3.6 with $n=0$

$= f_1(f_0(\perp))$, by 2.5.3 ■

Lemma 2.5.5

$$X_{n+1}(X_n) = f_{n+1}(f_n(\cdots(f_1(f_0(\perp)))\cdots)) \quad \text{, for } n \geq 0.$$

Proof. Apply 2.5.2 n times and use 2.5.4.

Now, since $f_n \sqsubseteq f$ for all $n \geq 0$, we have

$$X_{n+1}(X_n) \sqsubseteq f^{n+2}(\perp)$$

Hence,
$$Y_\lambda(f) = X(X) \stackrel{i}{=} \bigsqcup_{n=0}^{\infty} X_{n+1}(X_n) \quad , \text{ By def. of application}$$
$$\sqsubseteq \bigsqcup_{n=0}^{\infty} f^{n+2}(\perp) = Y(f) \quad , \text{ since } \perp \sqsubseteq f(\perp) \sqsubseteq f^2(\perp)$$

This completes proof of 2.5.1.

Starting with Y_λ, we can generate an infinite sequence of fixed-point operators, using the combinator

$$G = \lambda y.\lambda f.f(yf)$$

Define $\quad Y_0 = Y_\lambda \quad , \quad Y_{i+1} = Y_i G \quad (i \geq 0)$

Translating results proved by Morris [3, p.71] and Böhm [16, p.195] into the present notation, we have

Lemma 2.5.6 For all $i \geq 0$,

(a) $\quad Y_i \text{ cnv } GY_i$

(b) $\quad Y_i F \text{ cnv } F(Y_i F)$, for all wfes F

(c) $\quad Y_0 \text{ not-cnv } Y_1$.

(Further to (c), it is believed that

$$Y_i \text{ not-cnv } Y_j \quad , \text{ for all } i \neq j.)$$

There is an interesting relationship between the combinator G and fixed-point operators in D_∞, viz. that all fixed-point operators are fixed-points of G, from which we can deduce that each Y_i in the above sequence is equal to the minimal fixed-point operator Y.

Lemma 2.5.7

(a) For all $n \geq 0$, $\quad G^n(\perp)(f) = f^n(\perp)$.

(b) For any fixed-point operator H, $\quad G(H) = H$.

(c) $\quad G(Y) = Y = Y(G)$.

Proof.

(a) By induction on n.

For $n=0$, $G^0(\Lambda) = \Lambda$, so $G^0(\Lambda)(f) = \Lambda = f^0(\Lambda)$.

For $n \geq 0$,
$$\begin{aligned} G^{n+1}(\Lambda)(f) &= G(G^n(\Lambda))(f) \\ &= f(G^n(\Lambda)(f)) && \text{, by def. of } G \\ &= f(f^n(\Lambda)) && \text{, by ind.hyp.} \\ &= f^{n+1}(\Lambda) \end{aligned}$$

(b) If H is a fixed-point operator, then, for $f \in D_\omega$,
$$H(f) = f(H(f)) = G(H)(f) .$$

(c) Since Y is a fixed-point operator, $G(Y) = Y$ follows from part (b). For the other half of (c), for $f \in D_\omega$,
$$\begin{aligned} Y(G)(f) &= \left(\bigsqcup_{n=0}^{\infty} G^n(\Lambda)\right)(f) \\ &= \bigsqcup_{n=0}^{\infty} G^n(\Lambda)(f) && \text{, since application is additive in its first argument} \\ &= \bigsqcup_{n=0}^{\infty} f^n(\Lambda) && \text{, by part(a)} \\ &= Y(f) . \end{aligned}$$
■

Theorem 2.5.8

For $i=0,1,2,\cdots$, $Y_i = Y$ in D_ω.

Proof. By induction on i. The case $i=0$ is just 2.5.1. For $i \geq 0$, if $Y_i = Y$, then
$$Y_{i+1} = Y_i(G) = Y(G) = Y \quad \text{, by 2.5.7(c)}$$
■

Thus, though Y_0 and Y_1 are not interconvertible, they are equivalent in D_ω. Therefore, the equality of Y_0 and Y_1 can be *consistently* adjoined to the system based on α-β-η-conversion.

§2.6 Normal versus non-normal forms

Intuitively, it does not seem unreasonable that the fixed-point operators of §2.5, none of which have a normal form, should all be equivalent.

Intuitive arguments about the λ-calculus can be misleading, however. One might expect that wfes having a normal form are somehow "different" from those with no normal form. For some time, it was thought that such a result could be established in the D_∞-model, but this turns out not to be the case.

Let(+) $F = \lambda f.\lambda x.\lambda y.x(fy)$

By β-η-reduction, the identity $I = \lambda x.x$ is one fixed-point of F:

$$F(I) \to_\beta \lambda x.\lambda y.x(Iy) \to_\beta \lambda x.\lambda y.xy \to_\eta \lambda x.x = I$$

We show that I is in fact the *minimal* fixed-point, $Y(F)$, of F in D_∞, whence the normal form I is equivalent to $Y_\lambda(F)$, which has no normal form (since it is a wfe of the λI-calculus and contains a sub-expression, Y_λ, with no normal form).

Theorem 2.6.1

In D_∞, I is equivalent to $Y_\lambda(F)$.

Proof. Denote $J = Y_\lambda(F) = Y(F)$.

Then, $J = F(J) = \lambda x.\lambda y.x(J(y))$.

Since I is one fixed-point of F, we have $J \sqsubseteq I$.

(+) In terms of standard combinators, $F = CB$, where

$C = \lambda x.\lambda y.\lambda z.xzy$, $bB = \lambda x.\lambda y.\lambda z.x(yz)$

For the converse relation, we first prove

Lemma 2.6.2 For $n \geq 0$,

$$J(x_n) \sqsupseteq x_n \quad \text{, for all } x \in D_\infty .$$

Proof. By induction on n. Let $y \in D_\infty$. For $n=0$,

$$J(x_0)(y) = x_0(J(y)) \quad \text{, by def. of } J$$
$$= x_0 = x_0(y) \quad \text{, by 1.3.7(a) twice}$$

Since this holds for all y, $J(x_0) = x_0$, by 1.3.8(b). For $n \geq 0$,

$$J(x_{n+1})(y) = x_{n+1}(J(y)) \quad \text{, By def. of } J$$
$$\sqsupseteq x_{n+1}(J(y_n)) \quad \text{, since } y \sqsupseteq y_n$$
$$\sqsupseteq x_{n+1}(y_n) \quad \text{, by ind.hyp. for } y$$
$$= x_{n+1}(y) \quad \text{, by 1.3.6}$$

Hence, $J(x_{n+1}) \sqsupseteq x_{n+1}$, by 1.3.8(a) this time. ∎

Resuming proof of 2.6.1, we have, for all $x \in D_\infty$,

$$J(x) = J\left(\bigsqcup_{n=0}^{\infty} x_n\right) \quad \text{, by 1.3.4(c)}$$
$$= \bigsqcup_{n=0}^{\infty} J(x_n) \quad \text{, by continuity}$$
$$\sqsupseteq \bigsqcup_{n=0}^{\infty} x_n \quad \text{, by 2.6.2}$$
$$= x \quad \text{, by 1.3.4(c)}$$

Hence, $J \sqsupseteq I$, by 1.3.8(a), and the proof of 2.6.1 is complete.

The repercussions of this example are considerable. First, let us make the

Definition. An equivalence relation $\triangleleft$ between wfes will be called *normal* if $\varepsilon_0 \sim \varepsilon_1$ implies either both ε_0 and ε_1 have normal forms or both do not. A model of the

λ-calculus will be called *normal* if the interpretation of equality in the model is normal.

The equivalence of I and $I_\lambda(F)$ shows that the D_∞-model of the λ-calculus is a *non-normal* model. Furthermore, for any normal form, the existence of an equivalent non-normal form can now be deduced.

Theorem 2.6.2

Given *any* normal form N, one can find a wfe X with no normal form such that the system resulting from the addition of $N = X$ as an extra axiom for the λ-calculus is not inconsistent.

Proof. Choose an occurrence (free or bound) of a variable x in N which is not the rator of a combination(+). Let X be the wfe obtained by replacing this occurrence of x by the equivalent wfe $I(x) = I_\lambda(F)(x)$. Then, $N = X$ by 2.3.3 ; and the replaced occurrence of x not being the rator of a combination is sufficient to guarantee that X has no normal form.

The consistency of $N = X$ as an extra axiom now follows from the consistency of the D_∞-model.

In contrast with the situation for distinct normal forms, we have

---

(+) Such an occurrence must exist, e.g. choose the *right-most* occurrence of a variable in N.

Theorem 2.6.4

Let M be any wfe having a normal form. If X is the non-normal form equivalent to M determined in the proof of 2.6.3, there is no context C[] such that

$$C[M] \text{ cnv } K \quad , \quad C[X] \text{ cnv } K(I) \quad .$$

Hence (c.f. [16, p.185]), there is no context such that

$$C[c] \text{ cnv } \begin{cases} K & \text{, if } c \text{ cnv } M \\ K(I) & \text{, otherwise} \end{cases}$$

Proof. The existence of such contexts would contradict the incomparability of K and $K(I)$ in D_β.

Theorem 2.6.3 is interesting because it shows that the normal and non-normal forms are not *essentially* different. However, this does not mean that they will be indistinguishable in *all* models ; whether or not a non-normal form 'equal' to a normal form can be found will depend on the model we are using. In logical terms all we can say is that the λ-calculus is not categorical in this respect ; some models will be normal, some not.

To satisfy the requirements which motivated the present study, it is obvious from the close analogy between programs which terminate and wfes with a normal form which models are to be preferred. The termination of programs is as much a part of their equivalence as, say, the fact that they produce the same output on a teletype, line-printer, or any other device, and it would be nice if all these aspects of equivalence could be incorporated within a single concept. In other words, we would like a notion of equivalence between programs which is as strong

as possible, including termination properties within its compass. A proof of the equivalence of two programs would then imply that either both programs terminate, or neither does. On these grounds, non-normal models of the λ-calculus, such as D_∞, are not as helpful as they might be.

By making a different choice for the initial retraction (ϕ_0,ψ_0), Park [21] discovered an even more anomalous possibility. In his models, the combinator Y_λ is not equal to the minimal fixed-point operator Y.

But such 'imperfections' do not detract from the inverse-limit construction. The fact that somewhat contrived variations can produce 'pathological' situations in no way diminishes the power of the method. On the contrary, the construction is quite general and will be used in the next section to produce a model which averts most of the difficulties experienced with D_∞.

In rejecting one model in favour of another, we do so for no other reason than the teleological one that each model can only be judged in terms of its usefulness for its intended application, which here means its fitness to serve as a base domain for a theory of computation. For this purpose, we seek domains which have just those properties we want, no more and no less.

§2.7 A model with atoms

The reservations expressed in §2.6 all stemmed from the equivalence of I and $I_\Delta(I)$. Re-examination of the proof of 2.6.1 reveals that this result depends on the principle of extensionality, which holds in D_ω precisely because *all* its elements are functions (up to isomorphism).

But herein lies the explanation of the inadequacy of D_ω. For, besides functions, a plausible *theory* of computation must also deal with more primitive quantities - integers, truth values, etc. - for which the consequences of extensionality seem foreign to our intuition; e.g. the integer 3 becomes equal to the function $(\lambda x.3(x))$.

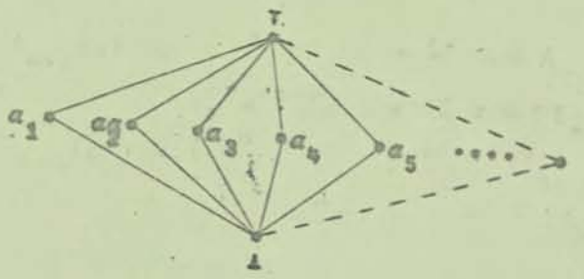
In the normal course of events, these primitives, or *atoms* as we shall call them, cannot be meaningfully applied as functions. In the present treatment the element Δ has been introduced to handle just such 'undefined' situations, so we set

$$a(x) = \Delta \quad \text{, for all } x\text{, for atoms } a \text{ .}$$

Thus, under application, the atoms behave as the totally undefined function. Other than this, atoms are regarded as indivisible objects, about which all we know is that they are pairwise distinct. Hence, the natural way to embed a *set* $\{a_1, a_2, a_3, \ldots\}$ of atoms in a (complete) *lattice* is to append the elements Δ and τ and impose the trivial partial ordering:

$$\Delta \sqsubseteq a_i \sqsubseteq \tau \quad \text{, for all atoms } a_i\text{.}$$

Diagrammatically:



Such a lattice will be referred to as a *domain of atoms*, denoted by A.

One *could* now use A as the initial domain, D_0, in the construction of D_ω. But from what we have seen above, the resulting 'identification' of an atom $a \in A = D_0$ with the functions $\phi_{0m}(a)$, for $0<m\leq\omega$, does not seem desirable. However, a simple alteration in the construction achieves the effect we want, producing a domain $E_\omega \cong A + [E_\omega \to E_\omega]$.

We begin with a domain A of atoms as the initial domain E_0. To form each higher-type domain E_{n+1}, we take the *lattice sum* of the atoms in A and the continuous functions over the previous domain E_n:

$$E_0 = A$$

$$E_{n+1} = A + [E_n \to E_n] \quad , \quad n \geq 0 \ .$$

This time, each E_n is embedded in E_{n+1} by mapping the atom- and function-parts of E_n into the corresponding parts of E_{n+1}, appropriate retractions (ϕ_n, ψ_n) for which are:

$$\phi_0(x) = x \quad , \text{ for } x \in A$$

$$\psi_0(x') = \begin{cases} x' & , \text{ if } x' \in A \\ \bot & , \text{ if } x' \in [E_0 \to E_0] \end{cases} \quad (*)$$

(*) We cannot take $\psi_0(x')=x'(\bot)$ here, since, for $x' \in [E_0 \to E_0]$, $\psi_0(x')=\bot$ is the only definition making (ϕ_0,ψ_0) a retraction. But for the higher-type functions, the same "adjustment-of-types" technique as before can be used.

$$\phi_{n+1}(x) = \begin{cases} x & \text{, if } x \in A \\ \phi_n \circ x \circ \psi_n & \text{, if } x \in [E_n \to E_n] \end{cases}$$

$$\psi_{n+1}(x') = \begin{cases} x' & \text{, if } x' \in A \\ \psi_n \circ x' \circ \phi_n & \text{, if } x' \in [E_{n+1} \to E_{n+1}] \end{cases}$$

Now form

$$E_\omega = (\langle x_n \rangle_{n=0}^{\omega} : \psi_n(x_{n+1}) = x_n \in E_n \text{ for all } n \geq 0)$$

and structure E_ω as a complete lattice by "component-wise" operations and relations, i.e.

$$x \sqsubseteq y \iff x_n \sqsubseteq_n y_n \text{ for all } n \geq 0, \quad \text{etc.}$$

Mappings

$$\phi_{nm} : E_n \to E_m \quad , n, m \geq 0$$

$$\phi_{n\omega} : E_n \to E_\omega \quad , n \geq 0$$

$$\phi_{\omega m} : E_\omega \to E_m \quad , m \geq 0 \ ,$$

with the same properties as before, are introduced to provide the embeddings

$$E_n \overset{\sqsubset}{=} E_m \subseteq E_\omega \quad , 0 \leq n \leq m \leq \omega \ ,$$

whence we can adopt the same identification and notational conventions as for D_ω.

Observe that, for any $x = \langle x_n \rangle_{n=0}^{\omega}$ in E_ω,

either (a) $x_n = x \in A$ for all $n \geq 0$,

or (b) $x_{n+1} \in [E_n \to E_n]$ for $n \geq 0$, *and* $x_0 = \perp$

In case (b), x determines a (continuous) function over E_ω as before:

$$(2) \qquad x = \lambda y : E_\omega . \bigsqcup_{n=0}^{\omega} \phi_{n+1}(y_n) \quad , \text{ for } x \in [E_\omega \to E_\omega].$$

Hence, up to isomorphism, every element of E_ω is either an atom or a continuous function over E_ω:

$$E_\omega \subseteq A + [E_\omega \to E_\omega] \ .$$

In the other direction, clearly $A \subseteq E_\infty$, by the correspondence $a = \langle a_n \rangle_{n=0}^{\infty}$, where all $a_n = a \in A$. And, as for 1.3.9, *every* continuous function $f: E_\infty \to E_\infty$ can be represented by an element of E_∞:

$$f = \bigsqcup_{n=0}^{\infty} (\lambda y{:}D_n.f(y)_n) \quad .$$

Thus, we have established

Theorem 2.7.1 E_∞ is a complete lattice, and

$(3)_\beta$ $$E_\infty \cong A + [E_\infty \to E_\infty]$$

Since we agreed that proper atoms (i.e. other than τ) are to have the same applicative behaviour as the totally undefined function, application can be extended to an operation

$$Apply: E_\infty \to [E_\infty \to E_\infty]$$

defined on the *whole* of E_∞. For $x,y \in E_\infty$, we define

$$Apply(x)(y) = \begin{cases} \tau & , \text{ if } x = \tau \\ \bot & , \text{ if } x \in A,\ x \neq \tau \ (\dagger) \\ \bigsqcup_{n=0}^{\infty} x_{n+1}(y_n) & , \text{ otherwise} \end{cases}$$

Using the projection mappings associated with lattice sums, this can be re-written

(4) $$Apply(x)(y) = \bigsqcup_{n=0}^{\infty} (x_{n+1}|[E_n \to E_n])(y)$$
$$= (x|[E_\infty \to E_\infty])(y)$$

In words, to apply any $x \in E_\infty$ as a function, first project x into the function-part of E_∞.

($\dagger$) Since the separate "tops" are identified in forming a lattice sum, def.1.1.2.

Similarly, for abstraction, suppose $F[y]$ is an applicative combination involving the variable y. As y varies over E_∞ (with other parts of $F[y]$ remaining fixed), $F[y]$ determines a continuous function $\lambda y{:}E_\infty.F[y]$, which is rendered as an element of E_∞ by use of the injection mapping associated with lattice sums. Thus, we write

$$(\lambda y{:}E_\infty.F[y]) \ in \ E_\infty$$

Further, to exhibit any function $f \in [E_\infty \to E_\infty]$ as an element of E_∞, first inject f into the lattice sum $A + [E_\infty \to E_\infty]$, and then use the isomorphism (3).

Most of the time we can omit this injection mapping without danger of confusion, regarding it as another convention of identification. The projection mappings, however, cannot be omitted, since an element is not always the same as its projection onto one half of the sum.

Lemma 2.7.2 For $x \in E_\infty$,

$$x \in [E_\infty \to E_\infty] \iff x = \lambda s{:}E_\infty.x(s)$$

Hence, a *restricted* version of extensionality holds in E_∞, viz. that

$$x(s) = y(s) \text{ for all } s \in E_\infty \Rightarrow x = y ,$$

provided x and y are *both* in $[E_\infty \to E_\infty]$. An alternative formulation valid for all $x, y \in E_\infty$ is

$$Apply(x)(s) = Apply(y)(s) \text{ for all } s \in E_\infty$$
$$\Rightarrow (x|[E_\infty \to E_\infty]) = (y|[E_\infty \to E_\infty])$$

Proof. For the first part, ($\Leftarrow$) is obvious. In the other direction ($\Rightarrow$), the result is simply a restatement of equation (2), using the isomorphism (3). The second part is immediate from the first and the definition (4) of $Apply$. ■

Thus, although full extensionality fails (because all atoms behave alike under application), *functions* are still conceived as being equal if they are equal in extension. The alternative formulation in 2.7.2 means that from the knowledge that two elements of E_∞ always yield the same result when applied as functions, all one can conclude is that their interpretations as functions (i.e. their projections into the function-part of E_∞) are the same.

An interpretation of λ-calculus in E_∞ can now be developed along the lines of §§2.1-2.3. Taking care to insert the projection (and injection) mappings where necessary, the definitions and theorems stated there pass over to this interpretation virtually unchanged.

The central part of this interpretation is paraphrased in Table 4, where, to avoid any confusion, we have used a different letter, $\mathcal{E}$, for the main semantic function and the symbol $=^*$ for equivalence under these semantics.

To save us the trouble of re-writing all our earlier results, we shall use a superfixed "*" to designate the translated versions of D_∞-results which hold for E_∞ (=E now). For example, Lemma 2.3.5* states: if ξ does not occur free in ε, then, for all $\rho \in Env$, $\beta \in E$,

$$\mathcal{E}[\![\varepsilon]\!](\rho[\beta/\xi]) = \mathcal{E}[\![\varepsilon]\!](\rho) .$$

The earlier proofs can be taken over almost verbatim for these *-versions. The only extra consideration is that, for proofs given by arguing for all $x \in D$, there are now three cases: (i) $x = \tau$, (ii) $x \in A$, $x \neq \tau$, (iii) $x \in [E \to E]$.

Table 6.4

E_∞-semantics of the λ-calculus

Domains

A	Domain of atoms
$E \cong A + [E \to E]$	Domain of values of wfes

The function *Apply*

$$Apply : E \to [E \to E]$$

$$Apply(x)(y) = (x|[E \to E])(y)$$

Semantic functions

$\rho : Id \to E$	The environment mapping
Env	Class of all environments

$$\rho[\beta/\xi] = \lambda\xi' : Id.((\xi' = \xi) \to \beta,\ \rho[\xi'])$$

$$\mathcal{E} : Exp \to [Env \to E]$$

(S1*) $\quad \mathcal{E}[\![\xi]\!](\rho) = \rho[\xi]$

(S2*) $\quad \mathcal{E}[\![e_0(e_1)]\!](\rho) = Apply(\mathcal{E}[\![e_0]\!](\rho))(\mathcal{E}[\![e_1]\!](\rho))$

$$= (\mathcal{E}[\![e_0]\!](\rho)|[E \to E])(\mathcal{E}[\![e_1]\!](\rho))$$

(S3*) $\quad \mathcal{E}[\![\lambda\xi.e]\!](\rho) = (\lambda\beta : E.\mathcal{E}[\![e]\!](\rho[\beta/\xi]))$ *in* E

Semantic equivalence

$$e_0 \equiv^* e_1 \iff \mathcal{E}[\![e_0]\!](\rho) = \mathcal{E}[\![e_1]\!](\rho) \text{ for all } \rho.$$

In this way, we have

Theorem 2.7.3

The semantics given in Table 4 constitute a model for the λ-calculus system based on α-β- conversion.

On the other hand, η-conversion is *not* valid in this model, because full extensionality does not now hold:

Example 2.7.4

Let $\varepsilon_0 = \lambda y.xy$, $\varepsilon_1 = x$,

so that ε_0 η-cnv ε_1. But $\varepsilon_0 \not=^* \varepsilon_1$, since if ρ is an environment such that $\rho[x] = a \in A$, and $a \not= \tau$, then

$$\mathcal{E}[\lambda y.xy](\rho) = \lambda\beta{:}E.Apply(a)(\beta) = \perp ,$$

whereas $$\mathcal{E}[x](\rho) = \rho[x] = a .$$

Recall that η-conversion would assert that

$\lambda\xi.\varepsilon(\xi) = \varepsilon$, provided ξ not free in ε.

Corresponding to the restricted extensionality property in E, a restricted version of η-conversion *is* valid, namely, when ε is an abstraction:

Lemma 2.7.5

Provided ξ does not occur free in $(\lambda\xi'.\varepsilon')$,

$$\lambda\xi.(\lambda\xi'.\varepsilon')\xi =^* \lambda\xi'.\varepsilon'$$

Proof. This equivalence holds since it can be deduced by α-β-conversion alone:

$$\begin{aligned} \lambda\xi.(\lambda\xi'.\varepsilon')\xi &\to_\beta \lambda\xi.[\xi/\xi']\varepsilon' \\ &\to_\alpha \lambda\xi'.[\xi'/\xi][\xi/\xi']\varepsilon') \\ &= \lambda\xi'.\varepsilon' \end{aligned}$$

where the last step is trivial if $\xi'=\xi$, and if $\xi'\neq\xi$, it follows since then ξ not free in $(\lambda\xi'.\varepsilon')$ implies ξ not free in ε'.

■

This lemma confirms the first result in 2.7.2 (with $s = \mathcal{E}[\lambda\xi'.\varepsilon'](\rho)$), since the interpretation (S3*) of an abstraction is in the function-part of E.

The structural properties of this model are very similar to those of the earlier model. Thus, the relation $\sqsubseteq$ in E determines a substitutive partial order relation, denoted by $\sqsubseteq^*$, between wfes:

$$\varepsilon_0 \sqsubseteq^* \varepsilon_1 \Leftrightarrow \mathcal{E}[\varepsilon_0](\rho) \sqsubseteq \mathcal{E}[\varepsilon_1](\rho) \text{ for all } \rho.$$

Lemma 2.7.6 (2.4.1*)

$$\varepsilon_0 \sqsubseteq^* \varepsilon_1 \Rightarrow C[\varepsilon_0] \sqsubseteq^* C[\varepsilon_1] \text{ for all contexts } C[].$$

Again, K and $K(I)$ are incomparable (under $\sqsubseteq^*$), and Böhm's theorem implies

Theorem 2.7.7 (2.4.5*)

If ε_0 and ε_1 have distinct β-η-normal forms, then $\varepsilon_0 \not\sqsubseteq^* \varepsilon_1$ and $\varepsilon_1 \not\sqsubseteq^* \varepsilon_0$.

For the sequence

$$Y_0 (= Y_\lambda),\ Y_1,\ Y_2, \cdots,\ Y_i, \cdots$$

of combinators given in §2.5, first extend Y to an operator Y^* applicable to the whole of E:

$$Y^* = \lambda f{:}E.\bigsqcup_{n=0}^{\infty} (f|[E \to E])^n(\bot)$$

so that

$$Y^*(f) = \begin{cases} \top & \text{, if } f = \top \\ \bot & \text{, if } f \in A,\ f \neq \top \\ \bigsqcup_{n=0}^{\infty} f^n(\bot) & \text{, if } f \in [E \to E] \end{cases}$$

Again, Y^* is the *minimal* fixed-point operator (for E), and by the methods of §2.5, we have

Theorem 2.7.8 (2.5.8*)

For $i=0,1,2,\ldots$, $Y_i = Y^\infty$ in E_∞.

When we come to the comparison of normal and non-normal forms, this model is more promising than the previous one, since the equivalence of I and $Y_\lambda(F)$ does not hold:

Theorem 2.7.9 $\quad I \neq^* Y_\lambda(F)$

Proof. Otherwise, we would have $I(x) =^* Y_\lambda(F)(x)$, and, by α-β-conversion and the definition of F,

$$x =^* \lambda y.x(Y_\lambda F y) \ .$$

But if ρ is an environment such that $\rho[x] = a \in A$, and $a \neq \tau$, $a \neq \perp$, then

$$\mathcal{E}[x](\rho) = a \ ,$$

whereas $\mathcal{E}[\lambda y.x(Y_\lambda F y)](\rho) = \lambda\beta{:}E.Apply(a)(Y^*(F)(\beta)) = \perp$. □

Thus, I and $Y_\lambda(F)$ can be distinguished by atoms in E_∞. The proof as given in 2.6.2 fails this time since only restricted versions of 1.3.8 hold in E_∞, and the projections are slightly different (because of the presence of the atoms).

Encouraged by this result, we shall further investigate the normal/non-normal form question in Chapter 3. The results we shall prove there make it seem highly likely that the E_∞-model is a normal model.

Such a gain in 'normality' we regard as far outweighing the loss of η-conversion. In any case, the whole question of the unrestricted usage of η-conversion has been a cause of some controversy over the years, as

as reflected by the general concern in the literature with *intensional/extensional* systems. The properties of E_ω can be said to represent a compromise between the two extremes of full extensionality and full intensionality, a compromise motivated by the quantities encountered in the formalisation of programming languages. Thus, the principle of extensionality still holds for functions, which is just as we want it to be. But since we have admitted the possibility that wfes need not *always* be interpreted as functions, unrestricted η-conversion is denied.

In fact, using $\sqsubseteq^*$, something can be proved about η-conversion in E_ω, viz. that an η-redex is $\sqsubseteq^*$ than its contractum:

Theorem 2.7.10

Provided ξ does not occur free ε,

$$\lambda\xi.\varepsilon(\xi) \sqsubseteq^* \varepsilon$$

Proof. For any environment ρ,

$$\begin{aligned}
\mathcal{E}[\![\lambda\xi.\varepsilon(\xi)]\!](\rho) &= \lambda\beta{:}E.Apply(\mathcal{E}[\![\varepsilon]\!](\rho[\beta/\xi]))(\beta) \\
&= \lambda\beta{:}E.Apply(\mathcal{E}[\![\varepsilon]\!](\rho))(\beta) && \text{, since } \xi \text{ not free in } \varepsilon \\
&= (\mathcal{E}[\![\varepsilon]\!](\rho)|[E \to E]) && \text{, using 2.7.2} \\
&\sqsubseteq \mathcal{E}[\![\varepsilon]\!](\rho) && \text{, from properties of projections,}
\end{aligned}$$

with equality holding in the case that $\mathcal{E}[\![\varepsilon]\!](\rho) \in [E \to E]$, i.e. agreeing with what we said earlier that extensionality rule holds for functions. ■ ends

Corollary 2.7.11 $\quad \varepsilon_0$ η-red $\varepsilon_1 \Rightarrow \varepsilon_0 \sqsubseteq^* \varepsilon_1$

Proof. Immediate from transitivity and substitutivity of $\sqsubseteq^*$.

In particular, 2.7.11 holds for c_0, c_1 having distinct β-normal forms; thus, the incomparability, under $\sqsubseteq^*$, of β-η-normal forms does not extend to distinct β-normal forms.

Moreover, the non-equivalence of β-η-normal forms also does not extend to distinct β-normal forms. For, although 2.7.4 shows that, in general, we only have $c_0 \sqsubseteq^* c_1$ in 2.7.11, distinct β-normal forms can be found for which the stronger relationship $c_0 =^* c_1$ holds:

Example 2.7.12

Consider $c_0 = x(\lambda y.xy)$, $c_1 = xx$

Let ρ be any environment and abbreviate $\rho[x]$ by σ $(\sigma \in E)$. Then,

$$\mathcal{E}[\![c_0]\!](\rho) = Apply(\sigma)(\lambda\beta{:}E.Apply(\sigma)(\beta))$$

$$= \begin{cases} \top & , \text{ if } \sigma = \top \\ \bot & , \text{ if } \sigma \in A, \ \sigma \neq \top \\ \sigma(\sigma) & , \text{ if } \sigma \in [E \to E] \quad \text{(using 2.7.2)} \end{cases}$$

and $$\mathcal{E}[\![c_1]\!](\rho) = Apply(\sigma)(\sigma) = \begin{cases} \top & , \text{ if } \sigma = \top \\ \bot & , \text{ if } \sigma \in A, \ \sigma \neq \top \\ \sigma(\sigma) & , \text{ if } \sigma \in [E \to E] \end{cases}$$

Hence, $c_0 =^* c_1$. ∎

Thus, some distinct β-normal forms are $=^*$-equivalent, some not. It seems difficult to formulate any general rule as to when η-conversion is valid, however, since it depends on considerations *external* to an η-redex, e.g. compare 2.7.4, where the replacement of $(\lambda y.xy)$ by x is not valid, with 2.7.12, where it is valid.

§2.8 Summary of Chapter 2.

The inverse-limit construction has been applied twice to produce complete lattices, D_∞ and E_∞, in which the λ-calculus can be interpreted.

Such interpretations have been specified in Tables 3,4. The validity of conversion has been established by defining, in terms of lattice equality, semantic counterparts of the relation of convertibility between wfes.

Thus, D_∞ and E_∞ serve as domains of non-trivial models for the λ-calculus. In both models, distinct β-η-normal forms are incomparable under the respective partial orderings, and an infinite sequence of combinators are all equal to the appropriate lattice-theoretic minimal fixed-point operator.

The D_∞-model is a non-normal model of the full system based on α-β-η-conversion; the E_∞-model is an (apparently) normal model of the system based on α-β-conversion alone.

It remains an open question whether, by the inverse-limit method, one can construct a normal model in which η-conversion is valid, though for the more immediate goals in a theory of computation a normal model of α-β-conversion alone is to be preferred.

Chapter 3

On the Characterization of Normal Forms

In Chapter 2, the existence of a consistent, non-normal model of the λ-calculus has been established, so that the equality of a normal form and a wfe with no normal form can be adjoined to the λ-calculus without rendering the system inconsistent.

Nevertheless, consistency is not everything, and the λ-calculus undoubtedly also has models in which the normal forms *are* distinguishable from the non-normal forms. Moreover, the feeling persists that a model can be found in which there is a meaningful characterization along the lines "A wfe has a normal form if and only if its interpretation in the model ...".

Before such a characterization can be given, however, it is helpful to know that a model *is* normal. Here, we shall investigate this aspect for the E_∞-model of the λ-calculus. With normality established, one can then search for counterparts of the normal forms.

It is worth noting that any property characterizing the normal forms *must* be undecidable, since it is well-known that there is no general decision procedure for the question of whether a wfe has a normal form or not. Thus, 'distinguishable' does not mean 'decidable'; rather, in saying that a model distinguishes between the normal and non-normal forms, we mean that if N and X are any two wfes,

M having a normal form but X not, then $M \neq X$, where $=$ denotes the interpretation of equality in the model.

Throughout this chapter, conversion and reduction will be understood as α-β-conversion and β-reduction, respectively, unless otherwise stated; sometimes we shall use the prefix 'β-' to emphasize the point. Many of the definitions and results can be extended to include η-conversion, but we need not do so here.

§3.1 Extension of Böhm's theorem

In order to derive an extension of 2.4.4 to the case of a normal form and a wfe with no normal form, we first need to generalize the notion of wfes being in normal form.

Definitions

A β-redex $(\lambda x.M)A$ is said to be the *head redex* of a wfe X just when X is of the form

(1) $$\lambda t_1 t_2 \cdots t_n.(\lambda x.M)AX_1X_2\cdots X_m \quad , \quad n,m \geq 0,$$

The reduction of X which proceeds by contracting the head redex at each stage is called the *head reduction* of X.

A wfe is in *head normal form* if it does not contain a head redex, and hence is of the form

(2) $$\lambda t_1 t_2 \cdots t_n.xX_1X_2\cdots X_m \quad , \quad n,m \geq 0, \ x \text{ variable},$$

where $X_1, \cdots, X_m$ can be *arbitrary* wfes. The form (2) will be called the (unique) *first-level structure* determined by n,m,x, and will be denoted by (n,m,x).

Let $\mathcal{H}$ denote the set of all wfes in head normal form. Introduce an equivalence relation $\approx$ on $\mathcal{H}$, where $H_0 \approx H_1$ holds iff $H_0, H_1 \in \mathcal{H}$ have the same first-level structure:

$$H_0 \approx H_1 \iff n_0 = n_1,\ m_0 = m_1,\ s_0 = s_1,$$

where (n_i, m_i, s_i) is the first-level structure of H_i, $i = 0,1$.

Lemma 3.1.1

For any $H \in \mathcal{H}$, if H' is a wfe to which H is reducible, then $H' \in \mathcal{H}$ and $H \approx H'$.

Proof. Immediate from the observation that, when H is of the form (2)', any reduction of H must be internal to the $X_1, \ldots, X_m$.

Definition

A wfe X is said to *have* a head normal form if X is convertible to some $H \in \mathcal{H}$; H is then said to be a head normal form *of* X.

Lemma 3.1.2

If X has a head normal form H, then

(a) there is an $H' \in \mathcal{H}$ such that X red H' and $H' \approx H$,

(b) if H_0 and H_1 are any two head normal forms of X, then $H_0 \approx H_1$.

Thus, the head normal form of X is unique, up to $\approx$.

Proof.

(a) Since X cnv H, by the Church-Rosser Theorem there exists a wfe H' such that

$$X \text{ red } H' \quad , \quad H \text{ red } H' \quad ;$$

whence, by 3.1.1, $H' \in \mathcal{H}$ and $H' \approx H$.

(b) Suppose X cnv $H_i \in \mathbb{H}$, $i=0,1$, so that H_0 cnv H_1. Again by the Church-Rosser Theorem, there is an $H \in \mathbb{X}$ such that H_0 red H , H_1 red H ;

by 3.1.1, $H_1 \approx H \approx H_0$.

When it exists, the head redex of any wfe X is also the left-most β-redex of X. Head reduction thus comprises the initial stages of the normal reduction of X, namely until X is reduced to a wfe in head normal form. Just as normal reduction always finds a normal form when one exists, we now show that head reduction has the same property in relation to head normal forms.

Theorem 3.1.3 A wfe has a head normal form iff its head reduction terminates

Proof. Clearly, if head reduction terminates, it does so in head normal form.

Conversely, suppose X has a head normal form, so that, by the preceding lemma, X red H for some $H \in \mathbb{H}$. Then, by the standardization theorem, there is a standard reduction

(3) $$X=A_0 \xrightarrow{R_1} A_1 \xrightarrow{R_2} A_2 \longrightarrow \cdots \xrightarrow{R_n} A_n=H$$

where R_i denotes the β-redex contracted in the ith. step.

Let k be the *least* integer for which A_k is in head normal form. Then, A_{k-1} is not in head normal form but A_k is, so the redex R_k may be the head redex of A_{k-1}. Let j be the *least* integer such that $R_{j+1}, R_{j+2}, \cdots, R_k$ are all head redexes of $A_j, A_{j+1}, \cdots, A_{k-1}$, respectively ; i.e. j is the least $j \leq k$ for which the part of (3) from A_j to A_k is a head reduction.

We claim that $j=0$, so that the part of reduction (3) from A_0 to A_k is the required head reduction of X.

For if $j=0$, then, by choice of j, the redex R_{j+1} is the head redex of A_j, but R_j is not the head redex, R say, of A_{j-1}. Hence, R_j must lie in or to the right of R, and R_{j+1} is the unique residual of R in A_j, which contradicts (3) being a standard reduction. The supposition $j=0$ is therefore impossible.

We now list some elementary properties of wfes with no head normal form (h.n.f. for short).

Theorem 3.1.4:

Let X be any wfe with no head normal form.

(a) For any variable x, $(\lambda x.X)$ has no h.n.f.

(b) If X is an abstraction $(\lambda x.X')$, then X' has no h.n.f.

(c) For any wfe U and variable x, $[U/x]X$ has no h.n.f.

(d) For any wfe U, $X(U)$ has no h.n.f.

(e) For any wfes $U_1,\cdots,U_m$ and variables $x_1,\cdots,x_n$, the wfe $(\lambda x_1 x_2\cdots x_n.XU_1U_2\cdots U_m)$ has no h.n.f.

Proof. Let $X=A_0\to A_1\to\cdots\to A_i\to\cdots$ be the (non-terminating) head reduction of X.

(a), (b) Trivial.

(c) Define $A_i' = [U/x]A_i$, $i\geq 0$.

Then, $A_0'\to A_1'\to\cdots\to A_i'\to\cdots$

is the non-terminating head reduction of $[U/x]X$.

(d) Let m be the least integer (≥ 0, possibly infinite) for which A_m is an abstraction. Then, for $i\geq m$, each A_i is an abstraction, say $A_i = \lambda y.B_i$. Now define

$$A_i' = \begin{cases} A_i(U) & \text{, for } i \leq m. \\ [U/y]B_{i-1} & \text{, for } i > m. \end{cases}$$

Then, $A_0' \to A_1' \to \cdots \to A_i' \to \cdots$

is the non-terminating head reduction of $X(U)$.

(e) Follows from several applications of (d) and (a).

We can now state the promised extension of Böhm's Theorem. The proof can be best understood after one is familiar with the techniques employed in the proof of 2.4.4. The statement of our extension, however, is not dependent on these ideas. In order not to disrupt the flow of this chapter, we therefore leave the proof to the Appendix.

Theorem 3.1.5

For any two wfes M, X, with M having a β-normal form but X not, there exists a context $C[\,]$ such that

$$C[M] \text{ cnv } v_0 \ ,$$

and *either* (i) $C[X] \text{ cnv } v_1$,

or (ii) $C[X] \text{ cnv } \lambda t.v_0 X'$,

where t is a variable other than v_0,

and X' is some wfe,

or (iii) $C[X]$ has no head normal form,

where v_0, v_1 are free variables of $C[\,]$.

With this theorem we can partially resolve the problem of distinguishing the normal and non-normal forms in E_∞.

For if (i) above is the case, then $C[M] \not=^* C[X]$, since the distinct variables v_0, v_1 cannot be equivalent. If (ii) is the case, letting ρ be an environment such that $\rho[v_0] = a \in A$, with $a \neq \tau$, we have

$$\mathcal{E}[C[M]](\rho) = \rho[v_0] = a$$

while $$\mathcal{E}[C[X]](\rho) = \mathcal{E}[\lambda t.v_0 X' \Psi](\rho)$$

$$= \lambda\beta{:}E_\infty.Appl_\gamma(a)(\mathcal{E}[X'](\rho[\beta/t]))$$

$$= \bot.$$

Hence, $C[M] \not=^* C[X]$; therefore $M \not=^* X$ in both cases (i) and (ii)(*), and the problem has been reduced to that of whether a variable can be equivalent (in E_∞) to a wfe with no head normal form.

More generally, though, we ask "Can a head normal form be equivalent to a wfe with no head normal form ?" One would think not, but this requires proof.

For this purpose, we extend the λ-calculus to include a special constant Ω, conceived of as the 'unspecified' or 'undefined' *expression* which 'approximates' all proper wfes, in the sense of $\sqsubseteq^*$ defined in §2.7:

$$\Omega \sqsubseteq^* e \qquad \text{, for all wfes } e.$$

Ω is to be interpreted as the element ⊥ in E_∞, so we now add

$$(S4^*) \qquad \mathcal{E}[\Omega](\rho) = \bot \qquad ;$$

to the semantic clauses (S1*)-(S3*) in Table 3.

Ω has been introduced into our treatment primarily for reasons of convenience. The symbol ⊥ might have been used for Ω, but we have preferred a different symbol to make it clear in the ensuing sections just when we are

---

(*) Notice that the proof for case (ii) required use of the atoms in E_∞. The result can fail for this case in D_∞.

dealing with wfes themselves (involving Ω) and when with their values (involving $\perp$) in E_∞. Indeed, we wish to emphasise that Ω is *not*, and is not intended as, an alternative notation for $\perp$; Ω is a *syntactic* entity - a special wfe - quite distinct from the *semantic* usage of $\perp$ as the 'undefined' *value*. Note, however, that Ω adds nothing essentially new to the λ-calculus, as wfes with the required properties can be readily found:

Example 3.1.6

Let $\Delta = (\lambda y.yy)$. Then, $\Delta\Delta$ has no head normal form, and $\Delta\Delta \beta\text{-conv } Y_\lambda(\lambda x.x)$.

Hence, for any ρ,

$$\mathcal{E}[\![\Delta\Delta]\!](\rho) = Y^*(I) \quad \text{, with } Y^* \text{ as in } \S 2.7$$

$$= \bigsqcup_{n=0}^{\infty} (I[\![E \to E]\!])^n(\perp)$$

$$= \bigsqcup_{n=0}^{\infty} I^n(\perp) = \perp \ ,$$

so that $\Delta\Delta =^* \Omega$. ∎

Lemma 3.1.7

(a) $\Omega(e) =^* \Omega$, for all wfes e

(b) $\lambda\xi.\Omega =^* \Omega$, for all variables ξ

Proof. Immediate from (S4*) and the properties

$$\perp = K(\perp) = \perp(e) \quad \text{, for all } e \text{ in } E_\infty.$$

This lemma establishes the validity of two "conversion rules" for Ω :

(Ω_1) A combination with Ω as its rator may be replaced by Ω in any context.

(Ω_2) An abstraction with Ω as its body may be replaced by Ω in any context.

We shall call the extended system with these rules the λ-Ω-*calculus*.

Theorem 3.1.8

If X has a head normal form, then $X \not=^* \Omega$.

Proof. Let $H = \lambda t_1 t_2 \cdots t_n . s X_1 X_2 \cdots X_m$ be the head normal form of X. To show that $H \not=^* \Omega$, it suffices to show that there is some environment in which $s X_1 \cdots X_m$ has value other than $\perp$.

Let ρ be an environment such that $\rho[s] = K^m(e)$, where $e \neq \perp$ is an element of E_∞. Then,

$$\mathcal{E}[s X_1 \cdots X_m](\rho) = K^m(e)(\mathcal{E}[X_1](\rho)) \cdots (\mathcal{E}[X_m](\rho))$$
$$= e \neq \perp \qquad \square$$

This theorem implies that any wfe equivalent to Ω cannot have a head normal form, but does the converse hold ? We state this formally as

Question 1. Does it hold that *all* wfes having no head normal form are $=^* \Omega$?

If so, the desired normality of E_∞ then follows, together with the characterization :

> A wfe e is reducible to head normal form iff there is an environment ρ such that
> $$\mathcal{E}[e](\rho) \neq \perp .$$

In the rest of this chapter, we attempt to resolve Question 1 (in the affirmative) by two independent approaches, also of considerable interest and significance in themselves.

Added in print:

The author has recently received copy of a thesis by Barendregt(4) which may contain the answer to Question 1. He defines

> A term (wfe), M, is *solvable* if there exist terms $N_1,\cdots,N_k$ such that $MN_1\cdots N_k$ has a normal form.

and has proved (his 3.2.16 and 3.2.20½)

1). The equality of all unsolvable terms can be consistently adjoined to the λ-calculus.

2). A closed term, M, is solvable iff the addition of $M = I_\lambda(K)$ as an extra axiom renders the λ-calculus inconsistent.

Unsplvable terms and wfes with no head normal form are co-extensive. Question 1 would appear to be resolved if the result in 2) remains true when $I_\lambda(K)$ is replaced by *any* unsolvable closed term. A deeper intake of Barendregt's thesis may reveal, or suggest, proof of the latter.

---

(4) Barendregt, H.P., Some extensional term models for combinatory logics and λ-calculi, Utrecht, 1971.

§3.2 Approximate reduction

The concept of *approximation* has been central to our theme. All our partial orderings are understood in this sense, and our domains have been structured as complete lattices so that the notion of *limits* of sets of approximations can be studied.

The constant Ω is an object which 'approximates' any wfe ε : $\Omega \sqsubseteq^* \varepsilon$,

whence, $C[\Omega] \sqsubseteq^* C[\varepsilon]$, for all contexts C[].

Thus, *an* approximation to a wfe results if Ω is substituted for any sub-expression.

For a notion of *limit* which is at all useful, however, we shall want to form our approximations in a special way. The limiting process we shall describe is based on β-reduction and seems the one most natural to us.

Definition. A β-redex is said to be an *outermost* redex of a wfe when it is not a sub-expression of any other β-redex. (Clearly, the outermost redexes of any wfe form a set of *disjoint* sub-expressions.)

In outline, the idea behind the limiting process we have in mind runs as follows. Suppose we wish to 'evaluate' a wfe ε, and we start reducing ε by some means or other. If ε has a normal form and the reduction actually reaches this normal form, everything is fine. If not, at some stage, despairing of ever finding a normal form, suppose we decide to abandon the attempt. Then we need not simply conclude from this that ε does not appear

to have a normal form and discard the results of the reduction. For if we have reduced ε to a wfe ε_1, we might feel that the appareht non-termination of the reduction is caused by some 'undefinedness' in the behaviour of the redexes in ε_1, similar to the irreducibility of $\Delta\Delta$ (c.f. 3.1.6). Thus, a reasonable 'guess' as to the value of ε is the expression, A_1, obtained by replacing the outermost β-redexes of ε_1 by the constant Ω.

Remembering this first approximation, A_1, now embark on a fresh reduction of ε. Again suppose we do not find a normal form and decide to stop the reduction, when ε has been reduced to a wfe ε_2, say. Replacing redexes in ε_2 by Ω as before, we obtain a second approximation, A_2, to ε.

Continuing in this way, as we examine more and more reductions we obtain more and more approximations, the cumulative sum of which yield a gradually 'better' approximation to ε.

Passing to the limit, now consider the totality of *all* wfes to which ε is reducible, i.e. the whole *reducibility class* $\{\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_i, \cdots\}$ of ε (+). For each ε_i in this class, let A_i denote the result of replacing all outermost β-redexes of ε_i by Ω. The obvious question to ask is whether the sum knowledge of all these A_i converges to the value of ε itself.

To investigate this question, we first introduce some further terminology.

(+) Algorithms for enumerating this class can easily be given.

Definition. A wfe, A, of the λ-Ω-calculus is said to be in *approximate normal form* (or, sometimes, in *β-Ω-normal form*) if it contains no β-redex and the constant Ω does not occur either as the rator of a combination or as the body of an abstraction, so that neither of the 'rules' for Ω :

$$(\Omega_1) \qquad \Omega(\varepsilon) \rightarrow \Omega$$
$$(\Omega_2) \qquad \lambda\xi.\Omega \rightarrow \Omega$$

can be applied to A. If, in addition, Ω does not occur at all in A, then A is said to be in *proper* normal form.

A wfe A is said to be *Ω-reducible* to B if A can be transformed into B by application of rules (Ω_1), (Ω_2) alone.

Lemma 3.2.1

If A is in β-normal form, then A is Ω-reducible to an equivalent wfe in approximate normal form.

Proof. Follows from 3.1.7 and the observations that application of the Ω-rules always decreases the length of a wfe and cannot introduce any β-redexes.

Definition. For any ε, the wfe in β-normal form obtained by replacing all outermost β-redexes in ε by Ω will be called the (unique) *direct approximant* of ε.

Example 3.2.2

Consider $\varepsilon_0 = \lambda x.x(R_1 y)(yR_2)(\lambda x.R_3)$,

where R_1, R_2, R_3 are β-redexes. The direct approximant of ε_0 is $A_0' = \lambda x.x(\Omega y)(y\Omega)(\lambda x.\Omega)$,

which is Ω-reducible to the approximate normal form

$$A_0 = \lambda x.x(\Omega)(y\Omega)(\Omega) \quad .$$

Lemma 3.2.3

Let A' be the direct approximant of any wfe $\mathfrak{C}$.

Then (a) $A' \sqsubseteq^* \mathfrak{C}$,

and (b) A' is Ω-reducible to an equivalent wfe in approximate normal form.

Proof. (a) follows from $\Omega \sqsubseteq^* \mathfrak{C}_0$ by transitivity and substitutivity of $\sqsubseteq^*$. (b) follows from 3.2.1

Definition. A wfe $\mathfrak{C}$ is said to be *approximately reducible* to A if (a) A is in approximate normal form,

and (b) $\mathfrak{C}$ is β-reducible to a wfe whose direct approximant is Ω-reducible to A.

A is then called a *reduced approximant* of $\mathfrak{C}$.

Lemma 3.2.4

If A is any reduced approximant of $\mathfrak{C}$, then $A \sqsubseteq^* \mathfrak{C}$.

Proof. Let $\mathfrak{C}'$ be the wfe to which $\mathfrak{C}$ is β-reducible whose direct approximant, A', is Ω-reducible to A. Then,

$$A =^* A' \sqsubseteq^* \mathfrak{C}' =^* \mathfrak{C} \quad ,$$

by 3.2.1, 3.2.3(a), and the consistency of β-reduction. ■

Example 3.2.2 (continued)

Suppose the contractum of the redex R_3 in $\mathfrak{C}_0$ is $(\lambda u.suR_4)$, where R_4 is another β-redex. Then, $\mathfrak{C}_0$ β-red $\mathfrak{C}_1$, where

$$\mathfrak{C}_1 = \lambda x.x(R_1 y)(yR_2)(\lambda x.\lambda u.suR_4) \quad .$$

The direct approximant of $\mathfrak{C}_1$ is

$$A_1' = \lambda x.x(\Omega y)(y\Omega)(\lambda x.\lambda u.su\Omega) \quad ,$$

which is Ω-reducible to the approximate normal form

$$A_1 = \lambda x.x(\Omega)(y\Omega)(\lambda x.\lambda u.su\Omega) \quad .$$

Then, A_1 is a reduced approximant of $\mathfrak{C}_0$ and

$$A_0 \sqsubseteq^* A_1 \sqsubseteq^* \mathfrak{C}_1 =^* \mathfrak{C}_0 \qquad ■$$

We shall show that the relationship in this example between β-reduction and direct approximants holds for all wfes.

Theorem 3.2.5

(4) $\quad e_0 \ \beta\text{-red}\ e_1 \ \Rightarrow \ A_0 \sqsubseteq^{\circ} A_1$,

where A_0, A_1 are the direct approximants of e_0, e_1.

Proof. Since $\sqsubseteq^{\circ}$ is transitive, it is sufficient to prove the theorem for the case when e_0 is reduced to e_1 by contraction of a single β-redex R.

Let $R_1, R_2, \cdots, R_n$ be the set of outermost redexes in e_0.

If the redex R is not one of these outermost redexes, then R must be internal to one of them and disjoint from the rest. e_0 and e_1 are then identical, except at some sub-expression internal to one of their outermost redexes, hence A_0 and A_1 will be identical.

Now suppose R is one of the outermost redexes in e_0, say $R=R_1$, for definiteness. Let X be the contractum of R, and let $C[\,]$ be the context of R and X in e_0 and e_1, respectively, i.e. the context such that

$$e_0 = C[R] \quad , \quad e_1 = C[X] \quad .$$

The redex R is disjoint from all the other outermost redexes in e_0, so, for $i=2,3,\cdots,n$, each R_i has a unique residual in e_1 identical to R_i itself. Moreover, $R_2, R_3, \cdots, R_n$ occur as sub-parts of the context $C[\,]$, so if $C'[\,]$ denotes the result of replacing these redexes in $C[\,]$ by Ω, then $A_0 = C'[\Omega]$. For A_1, a deeper analysis is required, depending on the structure of the contractum X of R.

Case 1. X is a variable or a combination.

Then, X cannot be contained in an outermost redex, Q, of ϵ_1. For if it were, then Q would be the residual of a redex in ϵ_0 containing the redex R, which would contradict the fact that R is an outermost redex of ϵ_0.

Thus, in this case, the outermost redexes of ϵ_1 consist of $R_2, R_3, \cdots, R_n$ and the outermost redexes of X. If X' denotes the direct approximant of X, we then have $A_1 = C'[X']$, whence

$$A_0 = C'[\Omega] \sqsubseteq^* C'[X'] = A_1 \quad .$$

Case 2. X is an abstraction.

If X does not occur as the rator of a combination in ϵ_1, then $A_1 = C'[X']$ follows as in Case 1.

Now suppose X is the rator of a combination $X(U)$ in ϵ_1. Then, $X(U)$ is the unique descendant of a combination $R(U)$ in ϵ_0. The redex $X(U)$ must be an outermost redex of ϵ_1, otherwise R would not have been an outermost redex of ϵ_0. But this time some of the R_i, say $R_2, R_3, \cdots, R_m$ $(m \leq n)$ without loss of generality, may be sub-parts of the rand, U, of $X(U)$, so they are not outermost redexes of ϵ_1. Thus, the outermost redexes of ϵ_1 consist of $R_{m+1}, \cdots, R_n$ and $X(U)$.

Let $C_0[\,]$ be the context such that

$$\epsilon_0 = C_0[R(U)] \quad , \quad \epsilon_1 = C_0[X(U)] \quad .$$

Let U' be the direct approximant of U (i.e. the result of replacing the redexes $R_2, R_3, \cdots, R_m$ in U by Ω). Let $C_0'[\,]$ be the context obtained by replacing the redexes $R_{m+1}, \cdots, R_n$ in $C_0[\,]$ by Ω. Then,

$$A_0 = C_0'[\Omega(U')] =^* C_0'[\Omega] = A_1 \quad . \quad \blacksquare$$

This theorem shows that the direct approximant of a wfe is $\sqsubseteq^*$ than any of its reduced approximants. Thus, the direct approximants of successive stages in a reduction get no "worse" as the reduction continues. In other words, the process of taking direct approximants is *monotonic* with respect to β-reduction.

However, we are concerned not just with monotonicity, but also with *continuity* and *limits*, for which purpose we would like to know whether the successive direct approximants 'converge' to ϵ in the limit.

In fact, for any *particular* reduction this may not be the case. For instance, the normal order reduction algorithm can be inadequate in this sense:

Example 3.2.6

Consider the wfe $\epsilon = x(\Delta\Delta)(R)$,

where x is a variable not otherwise occurring, and R is a β-redex which is not equivalent to Ω (e.g. any redex which has a normal form). Then, $\Delta\Delta$ is the left-most β-redex in ϵ, so, since $\Delta\Delta$ is reducible only to itself, the successive stages in normal order reduction of ϵ are all identical to ϵ. Hence, the successive direct approximants in this reduction are all equal to $A = x(\Omega)(\Omega)$.

Certainly, $A \sqsubseteq^* \epsilon$, but $A \not\equiv^* \epsilon$. ∎

It is easy to see why normal order reduction "failed" for this example; it continues indefinitely with the non-terminating reduction of $\Delta\Delta$, so never gets round to looking at any redexes further to the right.

If we consider *all* possible reductions of wfes, however, the prospects are more hopeful. Let $\mathfrak{B}(\mathfrak{e})$ denote the set of *all* reduced approximants of $\mathfrak{e}$:

(5) $\quad \mathfrak{B}(\mathfrak{e}) = \{A: A \text{ in approx.n.f.}, \text{ and } \mathfrak{e} \text{ approx.red. } A\}$

Without loss of generality we can suppose that $\Omega \in \mathfrak{B}(\mathfrak{e})$; if not, it does no harm to append Ω to $\mathfrak{B}(\mathfrak{e})$.

An important property of these sets is

Theorem 3.2.7

For any wfe $\mathfrak{e}$, the set $\mathfrak{B}(\mathfrak{e})$ is *directed*.

Proof. Suppose $A_0, A_1 \in \mathfrak{B}(\mathfrak{e})$.

For $i=0,1$, let $\mathfrak{e}_i$, with direct approximant A_i', be a wfe such that

$$\mathfrak{e} \ \beta\text{-red}\ \mathfrak{e}_i \quad , \quad A_i' \ \Omega\text{-red}\ A_i \quad .$$

Then, $\mathfrak{e}_0$ cnv $\mathfrak{e}_1$, so, by the Church-Rosser Theorem, there exists a wfe $\mathfrak{e}'$ such that $\mathfrak{e}_i$ red $\mathfrak{e}'$. Hence,

$$\mathfrak{e} \text{ red } \mathfrak{e}_i \text{ red } \mathfrak{e}' \quad , \ i=0,1.$$

Now let A be the approximate normal form to which the direct approximant, A', of $\mathfrak{e}'$ is Ω-reducible. Then, A is a reduced approximant, of $\mathfrak{e}$, and, by 3.2.1 and 3.2.5,

$$A_i \ \mathrm{m}^* \ A_i' \ \sqsubseteq^* \ A' \ \mathrm{m}^* \ A \quad , \ i=0,1,$$

whence

$$A_0 \sqcup^* A_1 \ \sqsubseteq^* \ A \quad ,$$

i.e., in E_∞,

$$\mathfrak{A}[A_0](\rho) \sqcup \mathfrak{A}[A_1](\rho) \sqsubseteq \mathfrak{A}[A](\rho) \quad \text{for all } \rho. \qquad \blacksquare$$

Returning to the question of whether the reduced approximants of $\mathfrak{e}$ themselves tend to $\mathfrak{e}$ in the limit, this can now be stated more succinctly as

Question 2. For all wfes ε, does it hold that

(6) $\quad \mathcal{A}[\![\varepsilon]\!](\rho) = \bigsqcup\{\mathcal{A}[\![A]\!](\rho) : A \in \mathcal{B}(\varepsilon)\}$, for all ρ,

the sup being taken in the E_ω-lattice ?

We feel confident that the answer to this question must be "Yes". To support this claim, we outline a possible method for proof of (6), by structural induction on ε :

Case 1. ε is an identifier, ξ

Then, ξ is the only reduced approximant of ε, so (6) certainly holds in the case.

Case 2. ε is an abstraction, $\lambda\xi.\varepsilon'$

If A' ($\not\equiv \Omega$) is a reduced approximant of ε', then $\lambda\xi.A'$ is a reduced approximant of ε ; conversely, every non-Ω reduced approximant of $\lambda\xi.\varepsilon'$ must be of the form $\lambda\xi.A'$, with A' a reduced approximant of ε'. Hence (apart from Ω),

(5') $\quad \mathcal{B}(\varepsilon) = \{\lambda\xi.A' : A' \in \mathcal{B}(\varepsilon')\ ,\ A' \not\equiv \Omega\}$.

The induction hypothesis would imply that (6) holds for ε', i.e.

(6') $\quad \mathcal{A}[\![\varepsilon']\!](\rho) = \bigsqcup\{\mathcal{A}[\![A']\!](\rho) : A' \in \mathcal{B}(\varepsilon')\}$, for all ρ.

If Ω is the only reduced approximant of ε', then

$$\mathcal{B}(\varepsilon') = \{\Omega\} = \mathcal{B}(\varepsilon)$$

and $\quad \varepsilon = \lambda\xi.\varepsilon' \ \equiv^* \ \lambda\xi.\Omega \ \equiv^* \ \Omega$,

in which case (6) holds for ε.

If ε' has reduced approximants other than Ω, then

$$\mathcal{A}[\![\varepsilon]\!](\rho) = \mathcal{A}[\![\lambda\xi.\varepsilon']\!](\rho)$$

$$= \lambda\beta{:}E.\mathcal{A}[\![\varepsilon']\!](\rho[\beta/\xi]) \qquad \text{, by (S3*)}$$

$= \lambda\beta{:}E.\ \bigsqcup\{\mathcal{E}[\![A']\!](\rho[\beta/\xi]) : A' \in \mathcal{B}(\varepsilon')\}$, by (6')

$= \bigsqcup\{\lambda\beta{:}E.\mathcal{E}[\![A']\!](\rho[\beta/\xi]) : A' \in \mathcal{B}(\varepsilon')\}$, by def. of $\bigsqcup$ for functions over E

$= \bigsqcup\{\mathcal{E}[\![\lambda\xi.A']\!](\rho) : A' \in \mathcal{B}(\varepsilon')\}$, by (S3*)

$= \bigsqcup\{\mathcal{E}[\![A]\!](\rho) : A \in \mathcal{B}(\varepsilon)\}$, by (6')

Case 3. ε is a combination, $\varepsilon_1(\varepsilon_2)$

The induction hypothesis would imply that (6) holds for ε_1 and ε_2. Then, omitting the projections for the time being, we have

$\mathcal{E}[\![\varepsilon]\!](\rho) = \mathcal{E}[\![\varepsilon_1]\!](\rho)(\mathcal{E}[\![\varepsilon_2]\!](\rho))$, by (S2*)

$= (\bigsqcup\{\mathcal{E}[\![A_1]\!](\rho) : A_1 \in \mathcal{B}(\varepsilon_1)\})(\bigsqcup\{\mathcal{E}[\![A_2]\!](\rho) : A_2 \in \mathcal{B}(\varepsilon_2)\})$, by ind. hyp.

$= \bigsqcup\{\mathcal{E}[\![A_1]\!](\rho)(\mathcal{E}[\![A_2]\!](\rho)) : A_1 \in \mathcal{B}(\varepsilon_1)\ ,\ A_2 \in \mathcal{B}(\varepsilon_2)\}$,

by additivity and continuity of application in its first and second arguments, respectively.

The difficulty in this case lies in relating the reduced approximants of ε_1 and ε_2 to those of ε. Thus, to infer the truth of (6) for ε, what now seems to be needed is a result to the effect that:

> For each pair (A_1, A_2) with $A_1 \in \mathcal{B}(\varepsilon_1)$ and $A_2 \in \mathcal{B}(\varepsilon_2)$, there exists an $A \in \mathcal{B}(\varepsilon)$ such that $A_1(A_2) \sqsubseteq^{\#} A$.

Although proof of this eludes us at the moment, somehow the 'directedness' of $\mathcal{B}(\varepsilon)$ ought to do the trick. At least, it is an encouraging indication for the solution.

In a sense, equation (6) represents a kind of completeness property for β-reduction which is the best we can hope for. As we generate the reducibility class of $\mathfrak{C}$, so we obtain more and more of its reduced approximants. The principle embodied in (6), if true, is that if one is prepared to generate enough of the reducibility class, then one can approximate $\mathfrak{C}$ as close as one likes - intuitively, we can 'compute' the *value* of $\mathfrak{C}$ (in E_∞). Then, we could even say that the approximate normal forms constitute an (effectively given) *basis* for the λ-calculus in these models. It seems very satisfying that these ideas can be expressed as such a neat equation (6).

We can see immediately that a positive solution to (6) would resolve Question 1 in §3.1. For if a wfe is not in head normal form, its direct approximant is Ω-reducible to Ω ; and if a wfe has no head normal form, no wfe to which it is reducible is in head normal form, so Ω is the only reduced approximant of wfes with no head normal form.

Notice also that the RHS of (6) is always $\sqsubseteq^*$ than the LHS, since every reduced approximant of $\mathfrak{C}$ is $\sqsubseteq^*$ $\mathfrak{C}$; and if a wfe has a (proper) normal form, equality must hold in (6), since then the normal form is one of the reduced approximants. Further, (6) holds for all wfes with no normal form we have investigated to date. Notable in this respect is the paradoxical combinator, Y_λ, given in §2.5 :

Example 3.2.8

All wfes to which Y_λ is β- reducible are of the form

$$\varepsilon_n = \lambda f.f^n(XX) \quad , n \geq 0,$$

where $$X = \lambda y.f(yy) \quad .$$

Thus, the reduced approximants of Y_λ are

$$A_n = \lambda f.f^n(\Omega) \quad , n \geq 0.$$

Then,

$$\mathcal{E}[A_n](\rho) = \lambda\beta{:}E.(\beta|[E \to E])^n(\perp) \quad ,$$

so $$\bigsqcup_{A \in B(Y_\lambda)} \mathcal{E}[A](\rho) = \bigsqcup_{n \geq 0}^{\infty} \lambda\beta{:}E.(\beta|[E \to E])^n(\perp)$$

$$= Y^* = \mathcal{E}[Y_\lambda](\rho)$$

■

§3.3 Extension of a theorem of Morris

In relation to 3.2.8, Morris proved in his thesis [8] what amounts to the result that, for any context C[] such that $C[Y_\lambda(f)]$ has a β-normal form, there is an integer n (dependent on the context C[]) such that $C[f^n(\Omega)]$ has the same normal form. From this he was able to deduce a minimal fixed-point property for Y_λ which we shall refer to in §3.4. Here, we shall derive a generalization of Morris's theorem which holds for arbitrary wfes.

Theorem 3.3.1

Let ε be any wfe. For any context C[], if $C[\varepsilon]$ has a β-normal form, we can find a reduced approximant, A, of ε such that $C[A]$ is β-reducible to the same normal form.

For the proof we first need to introduce some concepts discussed by Morris.

Definition. Suppose $\mathcal{R}$ is a set of disjoint sub-expressions of A. Then A is said to *match B except at $\mathcal{R}$* if A can be transformed into B by replacement of sub-expressions in $\mathcal{R}$.

Lemma 3.3.2 (Morris)

Suppose (a) A matches B except at $\mathcal{R}$, and

(b) $A \xrightarrow{R}_{\beta} B$, where

(i) R is not a sub-part of a member of $\mathcal{R}$,
(ii) the rator of R is not a member of $\mathcal{R}$.

Then there is a β-redex, Q, in B such that A' matches B' except at $\mathcal{R}'$, where $B \xrightarrow{Q} B'$, and $\mathcal{R}'$ is the set of outermost sons of members of $\mathcal{R}$.

To save us having to introduce essentially the same notation in the theorems below, denote a β-reduction of A to A' by

$$(7) \qquad A=A_0 \xrightarrow{R_1} A_1 \xrightarrow{R_2} A_2 \longrightarrow \cdots \xrightarrow{R_n} A_n=A'$$

where R_i is the β-redex contracted in the ith. step of (7) from A_{i-1} to A_i, for $i=1,2,\cdots,n$.

Corollary 3.3.3

Suppose (a) A matches B except at $\mathcal{R}$,

(b) the reduction (7) from A to A' is such that

(i) each R_i is not a sub-part of a descendant of a member of $\mathcal{R}$,
(ii) the rator of each R_i is not a descendant of a member of $\mathcal{R}$.

Then, there is a β-reduction of length n from B to a wfe, B', such that A' matches B' except at $\mathcal{R}'$, where

$\mathcal{A}'$ is the set of outermost descendants of members of $\mathcal{A}$.

Further, if we here take B as the result of substituting Ω for members of $\mathcal{A}$ in A, then B' is the result of substituting Ω for members of $\mathcal{A}'$ in A'.

Proof. For the first part, apply 3.3.2 n times. For the second part, if we think of Ω as a variable for the moment, then all occurrences of Ω will be free; any descendant of an occurrence of Ω must then be Ω itself. ■

With this corollary we shall be able to substitute arbitrary wfes for sub-expressions which are "inessential" to a reduction.

We shall also need some preliminary results about the *relative* length of various reductions, so that we can give proofs by induction on the length of a reduction.

Consider the following situation. Suppose A has a β-normal form N, and A is converted to B by contraction of a single β-redex, S, in A. Then, of course, B is reducible to N. If we consider the *normal reductions* of A and B to N, it should be obvious that the one for B is *no longer than* that for A (we shall prove this in the sequel), but here we want to investigate cases in which it is actually *shorter*.

Lemma 3.3.4

Suppose (a) the reduction (7) is a normal β-reduction of A to A' (i.e. each R_i is the left-most β-redex in A_{i-1}),

(b) $A \xrightarrow{S}_{\beta} B$,

(c) B' is the result of a complete reduction relative to the residuals of S in $A'(= A_n)$.

Then, there is a normal β-reduction from B to B' of length $m \leq n$, with equality only if none of the redexes $R_1, R_2, \cdots, R_n$ in (7) is a residual of S.

Comment.

This lemma is more or less known previously; what is new here is the relative lengths of the reductions.

The lemma is a partial converse to one derived by Curry and Feys [12, p.140] in their proof of the standardization theorem. We shall prove our result by more or less the same technique employed there.

The converse is only partial because it is vital for our lemma that each R_i be the *left-most* β-redex. If (7) is only a standard (rather than a normal) reduction, there is certainly a standard reduction from B to B', but it is not necessarily true that $m \leq n$.

Proof of 3.3.4.

For $i = 1, 2, \cdots, n$, let Σ_i denote the set of residuals of S in A_i.

We shall determine a reduction

$$(8) \qquad B = B_0 \xrightarrow{Q_1} B_1 \xrightarrow{Q_2} B_2 \longrightarrow \cdots \xrightarrow{Q_m} B_m = B'$$

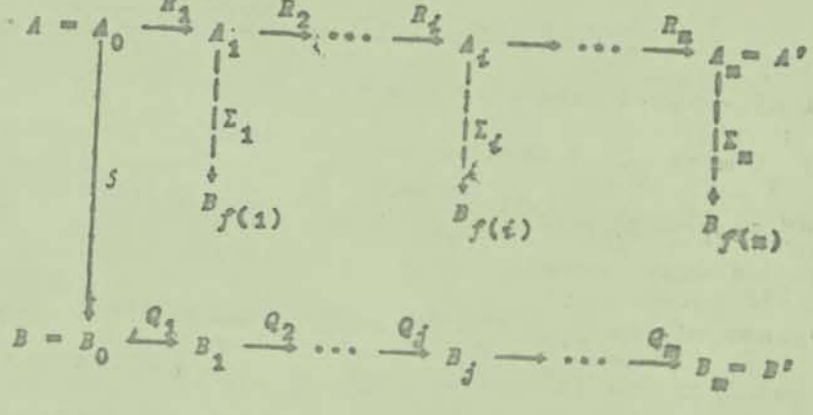
and an auxiliary function $f(i)$, defined for $0 \leq i \leq m$, satisfying the following conditions:

(i) The reduction (8) is normal, i.e. each Q_j is the left-most β-redex in B_{j-1}.

(ii) $f(0) = 0$, and either $f(i+1) = f(i)$ or $f(i+1) = f(i)+1$.

(iii) If $j = f(i)$, then a complete reduction relative to Σ_i converts A_i to B_j.

We can depict the relationship between these quantities as :



where the dotted arrows labelled with Σ_i's indicate complete reductions relative to Σ_i. What we need to show is that the diagram "closes up" in the bottom-right corner.

The proof of (i),(ii),(iii) will be by induction on an index q (over the range $0 \leq q \leq n$) of the A_q in §7).

For $q=0$, we need only define $B_0 = B$ and $f(0)=0$ for (i),(ii),(iii) to be satisfied.

Now assume that (i),(ii),(iii) are satisfied for some $0 \leq q < n$, and let $p=f(q)$; i.e. our induction hypothesis is that the reduction (8) is normal up to B_p and (ii),(iii) hold for all $i \leq q$. Consider two cases according as R_{q+1} - the left-most β-redex in A_q, by supposition (a) - is or is not a residual of S in A_q.

Case 1. R_{q+1} is a residual of S in A_q.

Then, defining $f(q+1)=f(q)=p$, conditions (i) and (ii) trivially hold. For condition (iii), notice that

$$A_q \xrightarrow{R_{q+1}} A_{q+1} \xrightarrow{\Sigma_{q+1}} X$$

is one complete reduction of A_q relative to Σ_q; hence $X = B_p$, by the lemma of parallel moves and induction hypothesis (iii) with $i=q$.

Case 2. R_{q+1} is not a residual of S in A_q.

Then, since R_{q+1} is the *left-most* β-redex in A_q, after a complete reduction relative to Σ_q from A_q to B_p, it must have a unique(+) residual, say P_{q+1}, which is the left-most β-redex in B_p. Hence, take P_{q+1} for Q_{p+1}, $f(q+1)=p+1=f(q)+1$, and let B_{p+1} be the result of contracting Q_{p+1} in B_p. Then, (i) and (ii) are immediate from these choices. For (iii), we have two complete reductions

$$A_q \xrightarrow{R_{q+1}} A_{q+1} \xrightarrow{\Sigma_{q+1}} X$$

$$A_q \xrightarrow{\Sigma_q} B_p \xrightarrow{Q_{p+1}} B_{p+1}$$

of A_q relative to the set $\Sigma_q \cup \{R_{q+1}\}$ of redexes in A_q, so $X = B_{p+1}$ by the lemma of parallel moves.

This completes the induction. Now *define* $n = f(n)$. From (iii) and supposition (c), we have $B_n = B'$. Using (ii), we see that $n = f(n) \leq n$, and, from the manner in which f was defined, equality holds only if Case 2 applies for all q. ∎

(+) It is here that the supposition that (7) is a *normal*, rather than just a *standard*, reduction is required.

Corollary 3.3.5

Suppose (a) A has a normal form N,

(b) $A \xrightarrow{R}_{\beta 1} B$,

(c) n, m are the lengths of the normal β-reductions of A, B, respectively, to N.

Then, $m \leq n$, with equality only if no redex contracted in the normal reduction of A to N is a residual of S.

Proof. Taking $A' = N$, let (7) be the normal reduction of A to N and apply 3.3.4. Then, $B' = N$, since A' (= N in normal form) can contain no residuals of S. ■

Theorem 3.3.6

Let ε be any wfe and C[] be any context such that C[ε] has a β-normal form N. Then we can find a wfe, ε', to which ε is β-reducible such that C[ε'] can be reduced to N without contraction of any residual of a β-redex in ε'.

Further, if X is the direct approximant of ε', then C[X] is also β-reducible to N.

Proof. Taking A = C[ε] and $A' = N$, let (7) be the normal β-reduction of C[ε] to N. If none of the redexes $R_1, R_2, \ldots, R_n$ in (7) is a residual of a redex in ε, there is nothing to prove. Otherwise, let S be a redex in ε such that some R_k is a residual of S (e.g. choose S such that k is least). Now apply 3.3.5 with B = C[ε^*], where ε^* is the result of contracting S in ε; thus, the normal reduction of C[ε^*] to N is of length less than n. With the obvious induction on the length of normal reduction, the first half of the theorem now follows.

For the second half, let R be the set of outermost β-redexes in ε', so that ε' matches its direct approximant, X,

except at $\mathcal{A}$. Applying 3.3.3, we then have a β-reduction of $C[X]$ to a wfe N' which matches N except at the outermost descendants of $\mathcal{A}$. But the members of $\mathcal{A}$ are all β-redexes, and the descendants of any β-redex are themselves β-redexes. Since N is in normal form, there can be no descendants of members of $\mathcal{A}$ in N; hence, $N' = N$.

□

Finally, it must be shown that the conclusion of 3.3.6 remains true if the direct approximant, X, is replaced by its approximate normal form. For this it is enough to show that applications of the rules for Ω cannot affect any reduction to proper normal form.

Theorem 3.3.7

Let N denote an arbitrary proper normal form. Then,

(a) $C[\Omega(\varepsilon)]$ β-red N $\Rightarrow$ $C[\Omega]$ β-red N

(b) $C[\lambda\xi.\Omega]$ β-red N $\Rightarrow$ $C[\Omega]$ β-red N

(c) $C[\Omega]$ β-red N $\Rightarrow$ $C[\varepsilon]$ β-red N , for all ε.

Proof.

(a) Taking $A = C[\Omega(\varepsilon)]$ and $A' = N$, without loss of generality we can suppose that (7) is the normal β-reduction of $C[\Omega(\varepsilon)]$ to N without contraction of a residual of any redex in ε (if not, first apply 3.3.6). Any descendant of $\Omega(\varepsilon)$ will then be of the form $\Omega(\varepsilon')$. Applying 3.3.3 with $\mathcal{A} = \{\Omega(\varepsilon)\}$ and $B = C[\Omega]$, we now have a β- reduction of $C[\Omega]$ to N', where N' matches N except at descendants of $\Omega(\varepsilon)$; but there can be no such descendants in N for then N would not be in proper normal form. Hence, $N' = N$.

(b) Suppose (7) is the normal β-reduction of $C[\lambda\xi.\Omega]$ to N. Any descendant of $\lambda\xi.\Omega$ is the same wfe, $\lambda\xi.\Omega$. Hence, (b) follows from 3.3.3 as in (a), since $\lambda\xi.\Omega$ cannot be the rator of a redex, R_k, in the reduction (7). For if it were, the contractum of R_k would be Ω; since (7) is a normal reduction, the redexes $R_{k+1}, \ldots, R_n$ must then all lie to the right of this occurrence of Ω, so N would not be in proper normal form as supposed.

(c) Similarly.

□

This completes proof of 3.3.1.

Corollary 3.3.8

Suppose X has no head normal form, but $C[\ ]$ is a context such that $C[X]$ has a (proper) β-normal form N. Then, $C[\mathfrak{e}]$ β-red N, for all wfes $\mathfrak{e}$.

Proof. Since X has no head normal form, Ω is its only reduced approximant; hence, $C[\Omega]$ β-red N, by 3.3.1, and the result follows by 3.3.7(c).

The following converse of 3.3.1 holds:

Theorem 3.3.9

Let A be a reduced approximant of $\mathfrak{e}$, and $C[\ ]$ be a context such that $C[A]$ has a (proper) β-normal form, N. Then, $C[\mathfrak{e}]$ β-red N.

Proof. Let $\mathfrak{e}'$ be a wfe to which $\mathfrak{e}$ is β-reducible such that the direct approximant of $\mathfrak{e}'$ is Ω-reducible to A. Then, A matches $\mathfrak{e}'$ except at the set, $\mathcal{A}$, of occurrences of Ω in A. Applying 3.3.3, there exists N' such that $C[\mathfrak{e}']$ β-red N' and N matches N' except at descendants of $\mathcal{A}$. But since

B is in proper normal form, and all descendants of Ω are equal to Ω, there can be no descendants of Ω in B; hence, $B = B'$ and $C[c'] \beta$-red B. The result follows since $c \beta$-red c'.

Corollary 3.3.10

Suppose X has no head normal form, and let $C[\ ]$ be any context. Then, $C[X]$ has a β-normal form, B, iff $C[\Omega] \beta$-red B.

This property, of course, *characterizes* wfes with no head normal form. For if X does have a head normal form, we can find contexts such that $C[X]$ has a normal form but $C[\Omega]$ does not.

Except for a few minor details, much the same reasoning may be used to extend 3.3.1 and 3.3.9 to contexts such that $C[c]$ has a head normal form. But since we have neither the patience nor the inspiration to carry it out here, we content ourselves with the statement of

Theorem 3.3.11

If $C[c]$ has a head normal form B, we can find a reduced approximant, A, of c such that $C[A] \beta$-red $B' \simeq B$.

§3.4 Extensional equivalence

We said earlier that full extensionality does not hold in E_ω. There we were really discussing the principle of *functional extensionality* - whether to regard wfes as equivalent just when their interpretations as functions, viz. their projections into $[E_\omega \to E_\omega]$, have the same extension. As we saw in §2.7, this kind of equivalence is inadequate as soon as one allows wfes to be interpreted as something other than functions, i.e. as atoms.

But there seems no reason to deny a wider principle of extensionality under which wfes are conceived as being equivalent if they agree in the results of all operations which may be applied *to them*. Application of wfes *as* functions is one such operation, so this stronger equivalence will imply functional equivalence, but not vice versa. Other operations which spring readily to mind - abstraction, and application of functions *to* wfes - can be combined into one here by studying the properties of wfes when used in larger contexts.

Nothing useful results, however, if we consider *arbitrary* contexts. But if we restrict our attention to contexts yielding normal forms, we arrive at a concept of *extensional equivalence* for the λ-calculus, whereby wfes will be defined as being extensionally equivalent if there is no difference in the normal forms they produce when used in larger contexts. Distinct normal forms must of course be interpreted as non-equivalent (or even, incomparable) objects, otherwise the interpretation would be inconsistent.

Definition (Morris)

A wfe ε_0 is said to be *extended by* ε_1, written $\varepsilon_0 \lhd \varepsilon_1$, and ε_1 is said to *extend* ε_0, if whenever $C[\varepsilon_0]$ has a normal form, then $C[\varepsilon_1]$ has the same normal form. Further, ε_0 and ε_1 are said to be *extensionally equivalent*, written $\varepsilon_0 = \varepsilon_1$, if each is extended by the other:

$$\varepsilon_0 = \varepsilon_1 \Leftrightarrow \varepsilon_0 \lhd \varepsilon_1 \text{ and } \varepsilon_1 \lhd \varepsilon_0 .$$

We shall define the *context-extension* of a wfe, ε, as the set of all pairs $\langle C[], N\rangle$, consisting of a context $C[]$ and a wfe, N, *in normal form*, such that $C[\varepsilon]$ is reducible to N. Thus, wfes are extensionally equivalent iff they have the same context-extension; and ε_0 is extended by ε_1 iff the context-extension of ε_0 is a subset of that of ε_1.

Some elementary properties of these relations are summarized as

Theorem 3.4.1

(a) $\lhd$ is reflexive, transitive and substitutive:

$$\varepsilon_0 \lhd \varepsilon_1 \Rightarrow C[\varepsilon_0] \lhd C[\varepsilon_1] \text{ , for all contexts } C[]$$

(b) $=$ is a substitutive equivalence relation and is a normal relation on wfes:

$$\varepsilon_0 = \varepsilon_1 \Rightarrow \text{either } \varepsilon_0, \varepsilon_1 \text{ both have a normal form or both do not}$$

(c) Interconvertible wfes are extensionally equivalent:

$$\varepsilon_0 \text{ cnv } \varepsilon_1 \Rightarrow \varepsilon_0 = \varepsilon_1 \quad \text{(but not conversely)}$$

(d) Normal forms are maximal under $\lhd$:

ε_0 has a normal form and $\varepsilon_0 \lhd \varepsilon_1 \Rightarrow \varepsilon_0 \text{ cnv } \varepsilon_1$

(e) Wfes with no head normal form are minimal under $\lhd$:

X has no head normal form $\Rightarrow X \lhd \varepsilon$ for all wfes ε

(f) Wfes with no head normal form are extensionally equivalent.

Proof. (a)-(d) were proved by Morris; (e) is a restatement of our 3.3.8; and (f) is obvious from (e).

Theorem 3.4.2 (Morris)

Suppose ~ is a non-trivial (i.e. non-universal) equivalence relation between wfes such that

(a) ~ is substitutive,

(b) ~ is an extension of cnv: e_0 cnv $e_1 \Rightarrow e_0 \sim e_1$

(c) ~ is normal.

Then, $$e_0 \sim e_1 \Rightarrow e_0 = e_1$$

whence = is the *maximal* normal interpretation of equality between wfes.

This latter property makes the relation = particularly interesting. For, conditions (a) and (b) are satisfied by the interpretation of equality in any λ-calculus model, and (c) also holds if the model is normal. If the model is non-normal, there are wfes N and X, respectively with and without normal forms, such that $N \sim X$, but $N \neq X$. Hence

Theorem 3.4.3

A model of the λ-calculus is a normal model *iff* the interpretation of equality in the model is a sub-relation of extensional equivalence, =.

Normality of E_∞ therefore follows if one can show *either* that semantic equivalence implies extensional equivalence:

(9) $$e_0 \mathrel{\mathbb{E}^*} e_1 \Rightarrow e_0 = e_1 \quad ,$$

or the stronger property

(10) $$e_0 \sqsubseteq^* e_1 \Rightarrow e_0 \lessdot e_1 \quad .$$

But, unfortunately, proofs of (9) and (10) seem just as elusive as normality itself.

More to the point, and what motivates our present interest in $=_\eta$, is the view expressed by extensional equivalence that one is *ultimately* concerned only with normal forms, which fits quite nicely with the ideas encountered in §§3.2,3.3. Indeed, if an equivalence between wfes is to mean anything in a computational sense, then one would like wfes to be considered equivalent if they lead to identical results (normal forms) when used in any 'completeable' computation, i.e. if they have the same context-extension. Thus we ask

Question 3. Does extensional equivalence imply semantic equivalence in E_∞:

(11) $\qquad \varepsilon_0 =_\eta \varepsilon_1 \;\Rightarrow\; \varepsilon_0 =^* \varepsilon_1$?

Roughly speaking, is the value of a wfe in E_∞ uniquely determined by knowledge of its context-extension ?

Further, does the converse of (10) hold:

(12) $\qquad \varepsilon_0 \preccurlyeq \varepsilon_1 \;\Rightarrow\; \varepsilon_0 \sqsubseteq^* \varepsilon_1$?

Clearly (11) is implied by (12), and if (11) holds, Question 1 of §3.1 can be resolved. Since $\Delta\Delta =^* \Omega$ and $\Delta\Delta$ has no head normal form, from (11) and 3.4.1(f) it would follow that all wfes with no head normal form are $=^* \Omega$, whence E_∞ would be a normal model. By 3.4.3, (9) would also hold, so that the relation of semantic equivalence in E_∞ is then precisely that of extensional equivalence between wfes. In other words, proof of (11) implies that E_∞ is the *maximal* normal model of the λ-calculus.

Probably (ii) would also resolve Question 2 of §3.2, in view of the intimate connection (c.f. 3.3.1 and 3.3.9) between the properties of reduced approximants and the notion of context-extension. To illustrate this connection we again consider the combinator Y_λ.

Theorem 3.4.4

If X is *any* fixed-point of a wfe F, i.e. if $F(X)$ cnv X, then (a) $Y_\lambda(F) \lhd X$,

(b) $Y_\lambda(F) \sqsubseteq^{\mathcal{H}} X$.

Proof. Part (a) was proved previously by Morris, but we give here a simpler proof made possible by our results about reduced approximants.

(a) Suppose $C[\,]$ is a context such that $C[Y_\lambda(F)]$ has a normal form N. By 3.3.1, there is a reduced approximant, A, of Y_λ such that $C[A(F)]$ red N. From 3.2.8, every reduced approximant of Y_λ is of the form $\lambda f.f^n(\Omega)$; hence, there is some $n \geq 0$ such that $C[(\lambda f.f^n(\Omega))F]$ red N, so $C[F^n(\Omega)]$ red N.

Applying 3.3.7(c), we have $C[F^n(X)]$ red N. Since $F(X)$ cnv X by supposition, we have $F^n(X)$ cnv X, whence $C[X]$ has normal form N.

Since this holds for all contexts $C[\,]$, it follows that $Y_\lambda(F) \lhd X$.

(b) For any environment ρ,

$$\mathcal{E}[Y_\lambda(F)]\uparrow(\rho) = Y^\circ(\mathcal{E}[F](\rho))$$

and $$\mathcal{E}[X](\rho) = \mathcal{E}[F(X)](\rho) \quad \text{, since } X \text{ cnv } F(X)$$

$$= (\mathcal{E}[F](\rho)|[E \to E])(\mathcal{E}[X](\rho)) .$$

Since Y^* is the *minimal* fixed-point operator, we therefore have $\mathcal{E}[Y_\lambda(F)](\rho) \sqsubseteq \mathcal{E}[X](\rho)$ □

Thus, Y_λ produces fixed-points which are minimal with respect to both $<$ and $\sqsubseteq^*$. While this does not actually prove either (10) or (12), it is strongly suggestive.

If these two relations, $<$ and $\sqsubseteq^*$, can be proved the same, it would follow that normal forms are *maximal* elements under $\sqsubseteq^*$. Whether this property then characterizes the normal forms in the E_∞-model we leave to the future.

§3.5 Summary and conclusions

The discursive, somewhat speculative nature of this chapter calls for a review of what progress has been achieved.

The challenge uppermost in our minds throughout this chapter has been establishing the normality of the E_∞-model of the λ-calculus. With the presence of the atoms, the problem has been reduced (§3.1) to that of distinguishing between wfes with and without head normal forms, about which we conjectured (Question 1) that

(13) $\quad \varepsilon$ has no head normal form $\iff \varepsilon =^* \Omega$.

In quest for the solution to Question 1, two closely related, but independent concepts have been investigated, namely approximate reduction (§§3.2,3.3) and extensional equivalence (§3.4). The former led us

to enquire (Question 2, equation(6)) whether wfes can be regarded as the 'limit' of their reduced approximants, the truth of which represents a completeness property for reduction (see comments towards the end of §3.2). In [22.p.35] Scott states:

> "One has to take the reduction rules (for the λ-calculus) 'on faith' and has no 'standards' to which they must match."

We suggest that (6) is a reasonable standard to expect 'computationally'.

Similarly, extensional equivalence is a natural concept to anyone concerned with programming. The idea conveyed becomes even more acceptable when one realises that the correspondence (between the λ-calculus and programming languages) is not so much between wfes and programs, but between wfes and *procedures*. Expressed in terms of procedures, extensional equivalence means:

> Two procedures are extensionally equivalent if each may be used in place of the other in *any* terminating program without affecting the outcome of the program(♦).

The second attractive feature of extensional equivalence is its maximality property (3.4.2). Hence we asked (Question 3) whether semantic equivalence, $=^s$, is implied by extensional equivalence. If so, we have seen that E_∞ then provides the *maximal normal model* ...

(♦) There are complications about *side-effects*, but the notion can easily be amended to incorporate them. However, this might not be necessary; for if two procedures yield the same results but have different side-effects, presumably programs can be written making use of these differences.

of the system based on α-β-conversion.

Theoretically, this would be a very significant result. There would be a compelling mathematical reason to justify our claim that there *is* something special, something 'canonical', about E_ω - that $=^\omega$ lies not just somewhere between the two extremes (for normal models) of convertibility and extensional equivalence, $=$, but is precisely the same as $=$, which is as we would like it to be.

At the other end of the spectrum, as it were, the author has his doubts that a 'model' can be found in which the interpretation of equality turns out to be *exactly* convertibility between wfes. This assertion depends on what one considers a "model", so let me explain. The domains D_ω and E_ω were constructed quite *independently* of the λ-calculus, in order to provide semantic analogues for the 'type-free' functions encountered there. The notion of equality in these two domains is also semantic in conception, being akin to such ideas as 'truth' in, say, first-order logic. This is in contradistinction with the situation for many first-order theories, whose models are often essentially 'proof-theoretic' in nature. Useful though they are, there is little *semantic* content in them. Rather, the author feels they constitute more of an alternative formulation, or at best a 'self-model', of the theory itself.(*)

Our λ-calculus models are not of this kind; indeed,

(*) We cite, e.g., the model constructed in [18, p.65-68] to show that every consistent first-order theory has a denumerable model, whence the completeness of any first-order predicate calculus, i.e. sentences provable iff logically valid ('true' under *all* interpretations).

the various equivalences in them are proposed as candidates for the completion of the analogy depicted below:

	Syntactic		Semantic
Logic:	Provability	———	Truth
	↕		↕
λ-calculus:	Convertibility	———	?

Thus the author is thinking of 'model' in a genuine semantic sense. The interpretation should be well-motivated, in much the same way that the standard interpretation of formal number theory captures one's intuitive feel for the natural numbers. It is the known incompleteness of number theory, and of any other 'sufficiently strong', consistent, recursively axiomatizable theory(+), which prompts the author to doubt the existence of a λ-calculus 'model' with equality 'isomorphic' to convertibility. We would expect that similar incompleteness results hold with the "standard computational model" of the λ-calculus for which we are searching (c.f. page 8). In other words, we contend that the λ-calculus is 'sufficiently strong', for it is certainly consistent (Church-Rosser) and recursively axiomatizable. In any case, would we really "want" a model in which, for instance, the fixed-point combinators Y_0, Y_1 given in §2.5 are not equivalent ? I think not.

The ease with which many ideas about computation have been expressed in our models exemplifies the utility of the present lattice-theoretic approach and lends support

---

(+) Our terminology here is that of Mendelson [18, Ch. 3] ...

to our thesis that the right kind of domains on which to base a *mathematical* theory of computation are now emerging.

Extrapolating from our studies of the λ-calculus, had the non-equivalence (essential for consistency) of wfes with distinct normal forms implies the (hardly surprising!) corollary that terminating programs which yield 'different' results cannot be considered equivalent. Normality of reduce equivalence between programs is desired so that programs that which terminate are distinguished from those which do not. Maximality (of $=^*$) would be a pleasing, but not entirely expected bonus.

It is almost superfluous to add that all results we know of are consistent with our conjectures. One cannot hope that they will hold in all models but we feel that those construction of the domain E_∞ is somehow sufficiently 'well-behaved' that equations (6), (11) and (13) are true. The exact parallel between the reduced approximants of Y_λ and the terms in the *sup*-expression for Y^* (see 3.2.8) is one indication of what we mean by 'well-behaved'.

Also notable in this connection are the properties of reduced approximants of *any* wfe (see 3.3.1, 3.3.9, 3.3.11). They *prove* that if one is interested in wfes only in those they may be used in larger expressions to yield normal (or head normal) forms, then knowledge of their reduced approximants is sufficient.

This seems (to us) to be a natural extension to the λK-calculus of the view inherent in the λI-calculus (+) programs

(+) The system as in §1.1 but with the added restriction that an abstraction, $(\lambda\xi.\varepsilon)$, is not accepted as a wfe unless its bound variable ξ occurs free at least once in its body.

Commenting on Church's reasons for preferring the λI-calculus, Curry and Feys state [12,p.106] that he regarded a wfe as 'significant' only if all its sub-parts are, and that he had in mind an interpretation in which only the normal forms are 'significant'. For the λK-calculus, as well as the normal forms, it is reasonable to maintain that non-normal forms are 'significant' in terms of their *eventual* usage to produce normal forms. From this point of view, 3.3.10 suggests that wfes with no head normal form are the ones we should consider 'meaningless', or 'least-defined', in the λK-calculus.

A second reason Church gave was that he could do anything he wanted without the λK-calculus. This may or may not be so, but our contention is that in practice it is not convenient to limit attention to *strict* functions, i.e. those which are undefined if their argument is undefined. Any respectable programming language will involve non-strict constructs, so that a program might terminate even though parts of it would not if they were to be executed. The conditional expressions of ALGOL 60 [23] provide the most obvious example; an ALGOL expression of the form

if ... then ... else ...

can have a value when one of its 'arms' is undefined. To explicate the meaning of such constructs using strict functions alone (if this be possible at all) would be artificial.

Lattice theory is particularly applicable to the study of another important non-strict construct in programming languages, that of recursion, in view of the latter's connection with operations of taking fixed-points of functions.

Indeed, it is intended to *define* the semantics of recursion via the appropriate, lattice-theoretic, *minimal* fixed-point operator, Y. The benefits of this approach derive from the repertoire of induction 'rules' thus made available for proving theorems about programs. The most general form of such arguments is the *fixpoint induction* of Park [29], which includes earlier versions of the technique as direct applications, e.g. McCarthy's principle of 'recursion induction' [6] and an induction rule of Scott [22].

A special case of recursion is that of *iteration*. Using the minimality of fixed-points produced by Y, Strachey has proved [25] a theorem about loops which is valid under quite general assumptions. Roughly, the theorem states that a while-command

while <*expression*> do <*command*>

is equivalent (in the absence of labels and jumps) to a generalised assignment command, the 'evaluation' of whose right-hand side involves only the application of a *pure* function, i.e. one which is not only without side-effects but is *independent* of the 'store'. What is more the theorem tells us precisely what this function is, in terms of Y and the meaning of the constituents of the while-command. The theorem is, more or less, what one would expect, but it is reassuring to know that one can *prove* a result which previously was a matter of instinctive belief.

Hopefully this brief sketch conveys something of the power of the method for treating programming concepts. The importance of Y is undeniable, and is one reason for our

interest in the various λ-calculus fixed-point combinators. To us, these combinators, none of which have a normal form, are 'significant', which is we have studied models of the λK-calculus rather than the λI-calculus.

The construction of the domains D_∞ and E_∞ (and others) is thus seen as a two-ended activity. On the one hand, analysis of many fundamental concepts in the field of computation is facilitated. At the same time, our knowledge of the λ-calculus itself is enhanced and extended by the structure of its models. Some new theorems about the λ-calculus have been deduced (and the proof of some older ones simplified) *from the properties of lattices alone.*

To logicians the result of greatest interest is that the equality of a normal form and a wfe with no normal form can be consistently adjoined to the λ-calculus (Theorem 2.6.1); as far as we know this has not been proved before. Notice that the result holds for both the λK- and λI-calculus, since the wfes, I and $Y_\lambda(F)$, in the example of §2.6 are both wfes of the λI-calculus.

The 'impossibility' result in 2.6.4 bears a striking resemblance to problems in the theory of computability. We speculate that the undecidability of other problems can be similarly derived using the *non-normal* D_∞-models, as follows:

1. Explicate the quantities concerned in the λ-calculus, say in much the same way that arithmetic can be 'simulated' in the λ-calculus (see e.g. [1]).

2. Thus formulate the problem as a proposition about wfes and let P and Q be the classes of wfes for which the proposition holds and does not hold, respectively.

3. Show that the decidability of the proposition is equivalent to the existence of a context C[] such that

$$C[\varepsilon] \text{ red } \begin{cases} K & , \text{ if } \varepsilon \in P \\ K(I) & , \text{ if } \varepsilon \in Q \end{cases}$$

4. Show there is a normal form $N \in P$ and a non-normal form $X \in Q$ which are D_∞-equivalent, whence the existence of a context as in 3. would contradict the incomparability of K and $K(I)$.

Perhaps a word is in order here as to why we have chosen to present the results throughout our dissertation in terms of contexts, C[], rather than 'functions'. The reason is partly a technical one based on the fact that we have wanted to consider *all* wfes, not just those without free variables, the *closed* wfes as they are sometimes called. For wfes with free variables the results can be expressed more concisely in terms of contexts (compare the two alternative statements, 2.4.3 and 2.4.4, of Böhm's theorem). In any case, proof of theorems for closed wfes often requires proof of a more general theorem for wfes with free variables. Similarly, even though whole programs must be closed - all variables must be declared, either explicitly or by default conventions - parts of programs, e.g. procedures, need not be closed.

One respect in which the λ-calculus suffers greatly is in the 'opacity' of its traditional presentation. Proof of all but its most trivial properties often involves an exhaustive case analysis which can be a daunting prospect to the would-be student; he can admire the meticulousness of these analyses but the insight gained is not very much. For the would-be expositor the diligence required is considerable; it is all too easy for him to make mistakes when confronted with the task of committing his proofs to paper, as witnessed by the number of erroneous statements published about the λ-calculus. In the apposite words of Curry and Feys [12,preface]:

> "Since some of our fellow sinners are among the most careful and competent logicians on the contemporary scene, we regard this as evidence that the subject is refractory. Thus fullness of exposition is necessary for accuracy; and excessive condensation would be false economy here, even more than it is ordinarily."

But the complexities of such exposition are no longer *inevitable*. If the right kind of properties can be established in its models, one can look forward to a simpler, more conceptual treatment of the λ-calculus. In this connection, it is interesting to note that our results in §2.6 were deduced solely by *model-theoretic* arguments. The only constraint on the use of models to prove theorems about the λ-calculus is that one must be careful to avoid circularity of argument - it would not count for much to use properties of a model to prove theorems which have been

used to establish these properties of the model.

Admittedly some of our proofs themselves have been rather lengthy, but the effort is worthwhile for the more sophisticated methods of reasoning which can now be employed. In computing parlance, we can anticipate a 'bootstrapping' to a 'higher-level' presentation of studies of the λ-calculus.

CHAPTER 4

A Graph Evaluation Technique for the λ-calculus

§4.1 Introductory

In recent years a fertile area of research effort has centered on the design of mechanical implementations of the λ-calculus. The simplicity of the conversion rules renders the λ-calculus a suitable language for this activity, enabling ready comparison of alternative evaluation *strategies* uncluttered by details of transient significance. At the same time, the decisions faced have enough in common with those posed by 'procedure-oriented' programming languages that the subject is neither academic nor unrewarding.

Two kinds of implementation can be distinguished:

1. Substitution mechanisms operate directly on wfes, iteratively searching for and contracting redexes by *literal substitution* of 'arguments' for bound variables. (As a variant of this category, the literal substitutions are sometimes *simulated*; see, e.g., Wegner [26,p.197] or McGowan [17,p.16] on this point.)
2. Interpreters for the λ-calculus do not modify the initial wfe during its evaluation. Once read as input, and suitably 'pre-processed' into some

internal representation, the initial expression forms the *control-part*, or *program*, of a machine whose 'run-time' operation utilizes various information structures — symbol (name) tables, return links, stacks, code pointers, etc. — to maintain a record of the partial results of evaluation and the currently reached pointsof execution. Machines in this category have been termed *fixed-program*, or *reentrant*, *machines* by Wegner, loc.cit..

For programming languages, implementations of the second kind are the more relevant of the two. To define them and prove their correctness, however, an indirect approach via mechanisms of the first kind may well be advisable, since these bear a closer relation to the reductions of wfes involved and can yield a clearer perception of what one is trying to implement. Indeed, McGowan's Fixed Program Machine [27,Ch.4] evolved in just this way from Wegner's Basic Lambda Calculus Machine [26, §3.8.2].

In abstract terms, the conversion rules comprise merely a set of available operations for transforming expressions (symbol strings, even). The general task in implementation, therefore, is the discovery of *algorithms*, which organize the application of these rules for some useful purpose as efficiently as possible.

For the λ-calculus, ideally one would like an algorithm which takes any wfe as input, replying either "The normal form of the given wfe is ...", or "The given

wfe has no normal form". This has long since been proved impossible. Letting S denote the set of wfes having a normal form, membership of S is undecidable as a property of arbitrary wfes; in the terminology of recursive function theory, S is not a recursive set.

It is recursively enumerable however. That is, there is a recursive function on the natural numbers whose range (under a suitable Gödel numbering) is the whole set S. It is thus possible to find an algorithm which lists the members of S, testing each in turn to see if it is the wfe given as input; if a wfe has a normal form, sooner or later it must occur in the enumeration, but if a wfe has no normal form the algorithm will continue indefinitely. Such an algorithm is said to *partially decide* whether a wfe has a normal form - it terminates provided a normal form exists.

Only in some special cases, such as $\Delta\Delta$ and the paradoxical combinator Y_λ, can one show that a normal form does not exist. Even then, the argument must of necessity be specific to the example, for there is no *systematic* way of even partially deciding that a wfe has no normal form; otherwise, by 'interleaving' the two partial decision algorithms one could answer the question of the existence of a normal form for any wfe.

Needless to say, iteratively generating and testing more and more members of S is not a practical proposition, nor as the evocative descriptions sometimes applied reveal - the "British Museum" algorithm, the "Chinese Army" approach, etc.. Further the method only asserts that a normal form exists. It does not tell us what this normal form is or how to find

it; for that we shall have to look elsewhere. But it shows that our ambitions must be limited to an algorithm which succeeds in finding a normal form whenever one exists, and fails to terminate otherwise.

Fortunately, we do not have to look very far for such algorithms. The key is provided by one of the consequences of the Church-Rosser Theorem - that the reducibility class of a wfe contains its normal form if this exists. Since this class *is* recursively enumerable, we can test each member in turn to see if it is *in* normal form - a question which is decidable, more or less by direct observation.

This algorithm amounts to trying all possible reductions, a technique still containing a degree of 'arbitrariness' not usually associated with evaluation mechanisms. Its implementation need not be seriously considered (though it might pose some interesting taxonomic and storage administration problems), for it is immediately superseded by a more 'deterministic' algorithm. From the standardization theorem it follows that only the *normal reduction* sequence need be generated - if a normal form exists, contraction of the left-most β-redex at each stage will eventually reach it.

One cannot of course tell *in advance* whether the method will ever find a normal form, but as far as the theory is concerned, normal reduction achieves all that can be asked; it *guarantees* finding the normal form of all wfes having one.

Implementations of normal reduction, however, leave something to be desired. Their storage requirements tend to grow alarmingly and the number of operations performed is often quite large, even for apparently 'simple' wfes. Indeed, for the λI-calculus it has been proved [12,p.142] that normal reduction is the *least* efficient method, in that the length (i.e. the number of β-contractions) of normal reduction of a λI-wfe is an upper bound on the length of any of its reductions. For the λK-calculus the situation is not much better; in nearly all cases where a normal form exists there is a shorter reduction than normal reduction, though the latter is the only general method for which one can be sure of finding a normal form. Some amelioration of the excessive storage demands is possible using the simulated substitution techniques referred to earlier, but the *length* of a reduction is not decreased by these means.

Faced with these inefficiencies some have settled for less 'complete', but more manageable, algorithms which are successful in the majority of cases. Nearly all these methods are based on the so-called *applicative order* of reduction, where the argument, A, of a β-redex, $(\lambda x.M)A$, is operated on *before* it is substituted for free occurrences of x in M. This may fail to terminate for some wfes where a normal form does exist and could be found by normal reduction, e.g. when the argument has no normal form, say $A = \Delta\Delta$, but the variable x does not occur free in M with M having a normal form. It should be noted, however, that, contrary to some statements the author has seen, applicative reduction of wfes *cannot* lead to erroneous results; failure

to terminate does not mean that the reduction is incorrect. 1.1.4 assures us that if any reduction terminates it must do so in the same normal form (up to α-conversion) as would normal reduction. In other words, no reduction can give a "wrong" answer.

Evaluation mechanisms based on applicative reduction are by now part of the stock in trade of language implementors. Evaluation of arguments before substitution corresponds to an ALGOL call-by-value, where the actual parameters of a procedure call are evaluated before the procedure-body is entered. The *eval*-function of LISP [28] illustrates the general technique. Landin's SECD-machine [17] gives a specific run-time organization for its implementation.

Here a different approach will be pursued. We maintain the desire for an algorithm which always finds a normal form when one exists, but we seek a method which is "better" than normal reduction.

The stimulus for the present study derives from some observations about β-reduction when performed on the tree representation (see §1.1.1) of wfes, and about normal reduction on these in particular.

First, consider a β-redex, $(\lambda x.M)A$, the tree representation of which is

where T_M, T_A are the trees reppesenting M, A, respectively.

Then the tree representation of the contractum of this redex can be obtained from the tree representing the body, M, by

replacing free occurrences of the variable x by *copies* of the tree representing the rand, A. For example, the redex

$$(\lambda x.x(ax))((\lambda z.zb)c)$$

and its contractum $((\lambda z.zb)c)(a((\lambda z.zb)c))$ are depicted in Figs.1(a) and 1(b).

The obvious question to ask is whether one needs several copies of the tree representing the rand. Does it suffice simply to replace free occurrences of x in T_M by *pointers* to the one copy of T_A occurring as the rand of the redex being contracted ? For the example above, the contractum would then be represented by the more compact structure in Fig.1(c).

We shall show that such 'assignment-of-pointers' to the one copy of the rand can, with one precaution left till later, be used as the basis for more efficient reduction algorithms.

The structure being handled is of course no longer a tree but is a (directed) *graph*, hence the name *graph reduction* will be applied to the process we shall describe.

The advantages to be gained from using graphs rather than trees are almost self-evident. As illustrated by our first example, a graph provides a more compact representation of a wfe than does a tree. Each sub-graph (i.e. a node together with all branches and nodes "below" it) represents possibly more than one sub-expression in the linear expansion of the whole graph. In particular, a sub-graph which is a redex can represent a set of redexes in the linear expansion; by contracting the one redex in the graph we can thus effect contraction of several (identical) redexes <u>simultaneously</u>.

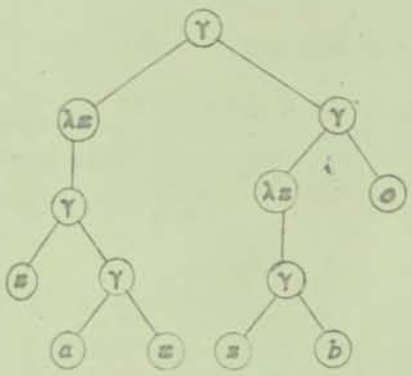


Figure 1(a)

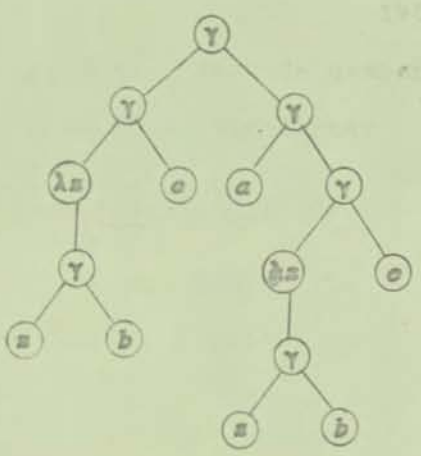


Figure 2(b)

Figure 1(c)

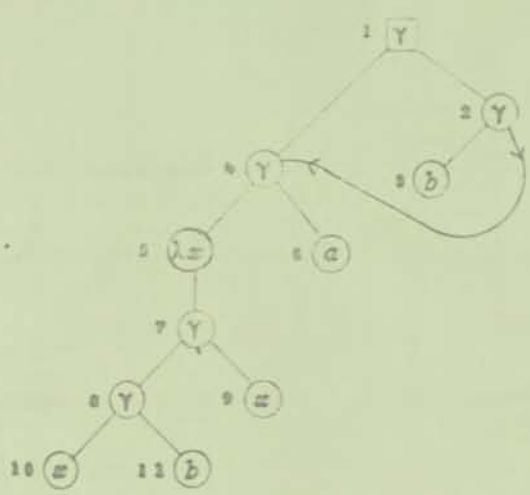


Figure 2(a)

Figure 2(b)

For example, in the graph in Fig.2(a), the node numbered 4 represents the two underlined redexes in the linear expansion

$$X = (\underline{(\lambda x.xbx)a})(b(\underline{(\lambda x.xbx)a}))$$

Contracting this one redex in the graph (using the 'pointer-assignment' technique above), Fig.2(b) is obtained, representing the wfe

$$X' = (aba)(b(aba))$$

whereas ordinary reduction of X to X' would have required two contractions.

The deficiencies of normal-order reduction are most striking when the rand of the redex being contracted is itself a redex. Consider the redex $Q = (\lambda x.M)R$, with R also a redex. Then, in general, the wfes in the normal reduction sequence beginning with Q will contain several descendants (residuals in this case) of the redex R. *Very roughly*, the pointer-assignment technique keeps these descendants of R "linked together", so that if one, say R_1, ever occurs as the left-most redex, its contraction also effects contraction of the other descendants of R. If a second descendant, R_2, were to occur later as a left-most redex, a (partly) reduced form of it would be available in the graph; using straightforward normal reduction on the linear expansion, however, there would be some duplication of effort since contraction of R_2 repeats that of R_1. Such repetition can be avoided by reducing the rand, R, before the left-most redex, Q, is contracted (as would happen in applicative-order reduction), but, as we know, this can

fail if it should turn out that all descendants of *R* are "cancelled out" in the normal reduction of *Q*.(+) At the time that *Q* is about to be contracted, it cannot be decided whether such "cancellation" will happen in the subsequent reduction. Using pointers we can achieve the next best thing. It is, if you like, as if the reduction of the rand is stored away as a sub-problem; if the answer is ever needed later, *then* we can retrieve the sub-problem, solve it (or at least attempt to solve it), *overwrite* the problem with its solution (so that later references do not have to re-solve the same problem), and return the answer to the required place.(++)

Of course, the loosely-formed argument of the last few paragraphs neither defines nor proves anything, but it was not meant to. It helps to motivate what might otherwise appear a somewhat contrived operation.

A general class of *lambda-graphs* and their linear expansion to yield the represented wfe will be described in §4.2. The major operation on lambda-graphs is that of *contraction of redex-nodes*, where a node will be called a *redex-node* if it represents a (β-)redex in the linear expansion. *Necessary* conditions for these operations to be well-defined will be treated as they arise, eventually arriving at definitions of *admissible* and *R-admissible* graphs. Contraction of a redex-node, *R*, will be 'correct' when the graph is *R*-admissible; the result of this operation

---

(+) By being part of the rand, *Z*, of a redex $(\lambda y.X)Z$ for which the variable *y* does not occur free in the base, *X*.

(++) Programs do this kind of thing all the time, though one does not always think of them in these terms!

will then be an admissible graph. It will be shown that admissible graphs, if not already *R*-admissible, can always be transformed into *R*-admissible ones, so that the conditions are *sufficient* for sequences of contractions to be possible.

§4.3 gives the formal statement and proof of these results. A specific algorithm based on 'normal-order' evaluation is treated in §4.4 ; it is shown to terminate whenever a normal form exists, and always do so in no more (usually, less) contractions than would normal-order reduction operating on wfes as symbol strings. Implementation of this technique, its relation to function evaluation strategies, and some directions for future research are discussed in §§4.5, 4.6, 4.7.

§4.2 Graph reduction

The definitions of nodes, graphs, etc., which follow are not the most general possible, but will be adequate for our purposes. They are a 'hybrid' of those given in texts on graph-theory and on compound data structures.

A *universe*, U, of objects called *nodes* will be defined to consist of all nodes of three distinct types, where the definition of each type determines the *names* of

1. a *predicate* which holds only for nodes of that type,
2. a finite number of functions, called *selectors*, applicable to nodes of that type,
3. a finite number of *data components* which nodes of that type may have.

When defined, the result of applying a selector to a node

will be another node (not necessarily of the same type). The data components of a node will be objects other than nodes; their nature, but not their values, is also fixed by definition of each type.

The three types, denoted by γ, λ and τ (for τ-node read terminal node), of nodes will be used to represent combinations, abstractions and identifiers, respectively. Their predicates, selectors and data components are given by the following table (where a dash means none):

Type	Predicate	Applicable selectors	Data components	
			Name	Nature
γ	$Is\gamma$	*rator*, *rand*	–	–
λ	$Is\lambda$	*body*	*bv*	identifier
τ	$Is\tau$	–	*var*	identifier

A (well-formed) *lambda-graph* is defined as a quintuple (X, s_1, s_2, s_3, Z), where

(G1) X is a finite set of nodes in U,

(G2) s_1, s_2, s_3 are functions from X to X such that, for $N \in X$,

(a) $s_1(N), s_2(N)$ are defined iff $Is\gamma(N)$,

(b) $s_3(N)$ is defined iff $Is\lambda(N)$,

(G3) Z is a node in X called the *root* of the graph.

The functions s_1, s_2, s_3 are designated as the *rator*, *rand*, *body* selectors, respectively.

In drawing these graphs on paper:

1. Each node in X is represented by a circle enclosing its type and data components (if any).

2. A node, N_0, is connected to a node, N_1, by a directed line, called a *pointer*, labelled with a selector name, s, iff $s(N_0) = N_1$.

3. The root is distinguished by using a square in place of a circle in 1..

Conventions: (a) The type, τ, of terminal nodes will be omitted.

(b) Selector names may be omitted as for trees.

(c) Arrows may be omitted on "downwards" pointers.

(d) Numbers may be written at the side of nodes in order to refer to them in examples.

Several lambda-graphs are shown in Figure 3. A lambda-graph is a tree as defined earlier if there is only one pointer to each node (except the root).

The definitions of paths, stems, the result of applying a path to a node, etc., are analogous to those in §1.1.1. A node N_1 is said to be *accessible* from a node N_0 if there is a path p *from* N_0 *to* N_1, i.e. such that $p(N_0)=N_1$. Such a path is said to *pass through* a node N, and N is said to *lie on* the path p, if $q(N_0)=N$ for some stem q of p.

To shorten the treatment below, we shall simply write *graph* for *lambda-graph*, and *sub-node of* for *node accessible from*. Each node is included as a sub-node of itself, by the null path. A *non-null* path from a node N to itself will be called an *N-cycle*.

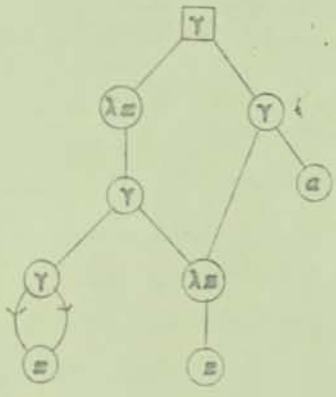


Figure 3(a)

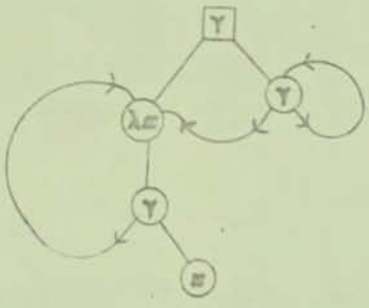


Figure 3(b)

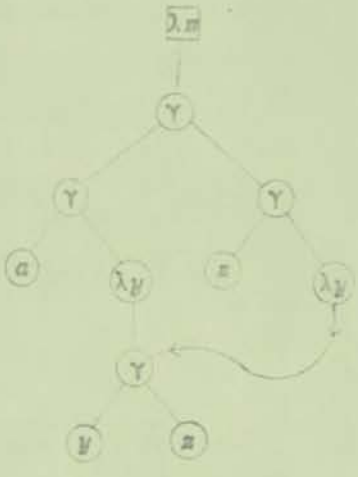


Figure 3(c)

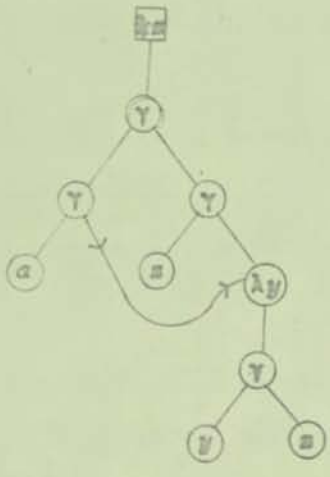


Figure 3(d)

Figure 3

Linear expansion of graphs

Let G be a graph $\langle X, s_1, s_2, s_3, Z \rangle$. For $N \in X$, consider the wfe

$$\varepsilon = Flatten(N, \emptyset)$$

defined recursively as follows:

1. If $Is\tau(N)$, then $\varepsilon = var(N)$.
2. If $Is\gamma(N)$, then $\varepsilon = \varepsilon_1(\varepsilon_2)$,
   where $\varepsilon_i = Flatten(N_i(N), G)$, $i=1,2$.
3. If $Is\lambda(N)$, then $\varepsilon = \lambda\xi.\varepsilon_3$,
   where $\xi = \#v\$N)$ and $\varepsilon_3 = Flatten(s_3(\xi N), G)$.

Then,

$$Wfe(G) = Flatten(Z, G)$$

is said to be the *wfe represented by* G.

This wfe will be finite in length when G is *acyclic*. Since infinite wfes will not be considered here, the first condition for a graph to be *admissible* is that it be *acyclic*.

For the graphs in Figure 3, (a) represents

$$(\lambda x.cx(\lambda z.z))((\lambda z.z)c)$$

(b) is cyclic, and (c),(d) both represent

$$\lambda x.a(\lambda y.yx)(s(\lambda y.yx))$$

As we saw in §4.1, each node of an acyclic graph represents a *set* of sub-expressions in the linear expansion. Node N represents a sub-expression $\langle p, \varepsilon \rangle$ of $Wfe(G)$ if p is a path from Z to N and $\varepsilon = Flatten(N, G)$. The set of all such sub-expressions will be called the *peers* of N in G:

$$Peers(N, G) = \{\langle p, \varepsilon \rangle : \varepsilon = Flatten(N, G),\ p(Z) = N\}.$$

Lemma 4.2.1 If G is acyclic, the peers of each node of G are disjoint as sub-expressions of $Wfe(G)$.

Proof. Let p, q be two distinct paths from Z to N. If p were a stem of q, then the path p' such that $p \cdot p' = q$ would be an N-cycle.

Contraction of redex-nodes

Let R be a redex-node of G. We consider the following sequence of operations:

(C1) Let $P=rator(R)$, $v=bv(P)$, $M=body(P)$, $A=rand(R)$.

(C2) Let F be the set of sub-nodes of P in G.

(C3) Let $V = \{F \in F : Is\tau(F), var(F)=v\}$.

(C4) Adjust all pointers to R so that they point to M.

(C5) Adjust all pointers to nodes in V so that they point to A.

(C6) If R was the root of G, mark M as the new root, except when $M \in V$, in which case mark A as the new root(+).

The operation (C1)-(C6) can be said to be *correct* if the resulting graph represents a wfe to which Wfs(G) is reducible.

Example 4.2.2

Applying (C1)-(C6) to the graph of Fig.2(a) in §4.1, for $R=4$ we have

$$P=5 \ , \ v=x \ , \ M=7 \ , \ A=6$$

$$P=\overset{5}{(}7,8,9,10,11) \ , \ V=\{9,10\}$$

The resulting graph is clearly that of Fig.2(b), and the operation is correct for this example.

Example 4.2.3

Consider the graph in Fig.4(a), with linear expansion

$$e_0 = \underline{(\lambda x.xf)(gf)}(\underline{(\lambda x.xf)(gf)}x(\lambda x.xx))$$

Where the underlined sub-expressions are the peers of node 2.

(+) The exception treats the special case $M = \textcircled{v}$.

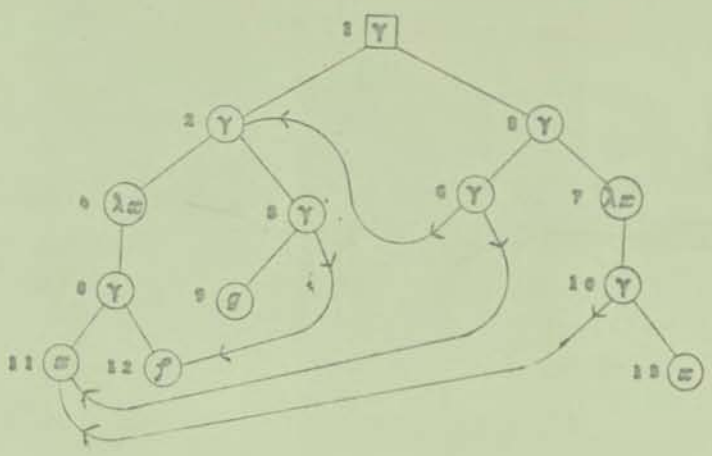


Figure 4(a)

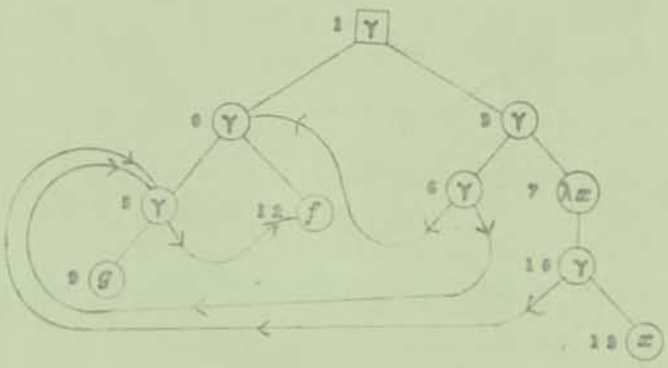


Figure 4(b)

Taking $R \neq 2$ in (C1)-(C6), we have

$$F=4 \ , \ v=x \ , \ N=8 \ , \ A=5$$

$$F = \{4,8,11,12\} \ , \ V = \{11\}$$

The resulting graph in Fig.4(b) has linear expansion

$$c_1 = (gff)((gff)(gf)(\lambda x.(gf)x))$$

Thus, the two redexes in c_0 represented by node 2 have been correctly contracted, but the operation has caused undesired replacements of two other occurrences of x in c_0 by (gf).

This example fails because of the incompatible uses of node 11 to represent occurrences of x in c_0. The peers

of node 11 are the underlined occurrences of x in

$$(\lambda x.\underset{\uparrow}{x}f)(gf)((\lambda x.\underset{\uparrow}{x}f)(gf)\underset{\uparrow}{x}(\lambda x.\underset{\uparrow}{x}x))$$
$$\qquad\qquad\quad a \qquad\qquad\quad a \quad b \quad c$$

where a,b,c distinguish three different kinds of occurrences of x in ε_8. The a-occurrences are internal to the base of the redexes being contracted; in ε_{11} they have been correctly replaced by (gf). The b-occurrence is free in ε_8, and the c-occurrence is bound elsewhere in ε_8 (i.e. not by the rators of the redexes being contracted); these occurrences should not be replaced.

The second condition for a graph to be admissible is designed to prevent such multi-purpose usage of terminal nodes. It will ensure that the peers of terminal nodes are either all free in the linear expansion, or they are all "similarly bound", in the sense that the *smallest* sub-expressions of ε in which they occur bound are themselves the peers of a *single* λ-node.

The binders-relation for graphs

Let N be a node of a graph G with root S, and let

$$\varepsilon_N = Flatten(N,G)$$

Let X, L, T be the set of nodes, λ-nodes, r-nodes, respectively, of G. Define

$NodesOn(p,N,G)$ = the set of nodes on the path p from N in G
$= \{N' \in X : q(N)=N'$ for some stem q of $p\}$

$BvsOn(p,N,G)$ = the set of *bvs* of λ-nodes in $NodesOn(p,N,G)$
$= \{x : x=bv(L),$ some $L \in L \cap NodesOn(p,N,G)\}$

FreeOccs(x,N,G) = the set of nodes which represent a free occurrence of x in e_N

$= \{T \in \mathsf{T} : x = var(T),$ and $x \notin BvsOn(p,N,G)$ for some p s.t. $p(N)=T\}$

FreeNodes(N,G) = the set of sub-nodes of N which represent a free occurrence of a variable in e_N

$= \{T \in \mathsf{T} : T \in FreeOccs(var(T),N,G)\}$

FreeVars(N,G) = the set of variables occurring free in e_N

$= \{x : x = var(T),$ some $T \in FreeNodes(N,G)\}$

Suppose $T \in \mathsf{T}$, and let $x = var(T)$. Then, a node $L \in \mathsf{L}$ is said to be a *binder* of T in G if $x = bv(L)$ and $T \in FreeOccs(x, body(L), G)$. In case T also represents a free occurrence of x in the linear expansion of the whole graph G, the special mark '$*$' will be used ; $*$ is said to be a *binder* of T in G if $T \in FreeOccs(x,Z,G)$.

Now define

Binders$(T,G) = \{B \in \mathsf{L} \cup \{*\} : B$ is a binder of T in $G\}$

When these sets are singletons for *all* terminal nodes, the function from T into $\mathsf{L} \cup \{*\}$ mapping each terminal node into its unique binder will be called the *binder-function* of G.

Admissible graphs

A graph G is said to be an *admissible graph* of a wfe e if

(A1) G is acyclic and $e = Wfe(G)$.

(A2) Each terminal node of G has a unique binder.

The tree representation of a wfe is always an admissible graph. Of the graphs in Figs.1-4, those in Figs.1,2,3(a),3(d) are admissible, those in Figs.3(b),3(c),4 are not.

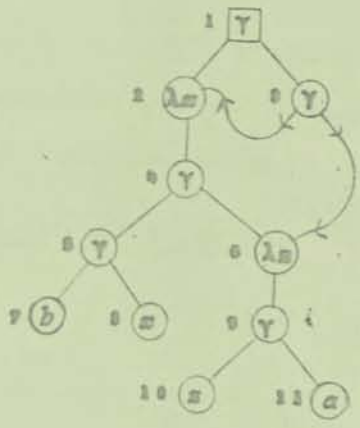


Figure 5(a)

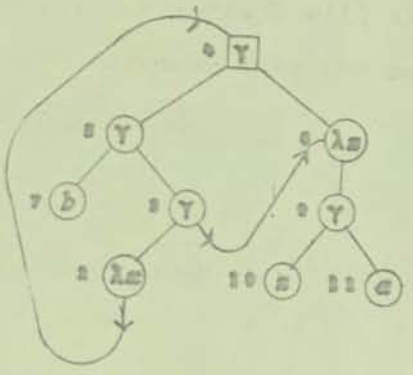


Figure 5(b)

Example 4.2.4

Consider Fig.5(a), with linear expansion

$$e = (\lambda x.bx(\lambda z.zc))((\lambda x.bx(\lambda z.zc))(\lambda z.zc))$$

Taking R=1 in (C1)-(C6), we have

$$F=2 \ , \ v=x \ , \ M=4 \ , \ A=3$$

$$F = \{2,4,5,6,7,8,9,10,11\} \ , \ V = \{8\}$$

The resulting graph is shown in Fig.5(b), the root having been adjusted by (C6). Although Fig.5(a) is an admissible graph of e, the graph in Fig.5(b) is inadmissible, since it is cyclic, and does not represent a wfe to which e is reducible.

This example has failed because there are *two* pointers to node 2 in Fig.5(a), only one of which is the rator-pointer of the redex being contracted. Adjusting the pointer to node 8 to point to node 3 has thus caused an occurrence of x in e external to the base of R to be changed.

R-admissible graphs will prevent this irregularity by ensuring that there is only *one* pointer to the rator-node

of R, namely the one from R itself. The graph will then be such that the rator-node of R represents only the rators of the peers of R in the linear expansion.

R-admissible graphs

A graph $G = (X, s_1, s_2, s_3, Z)$ containing a redex-node R is said to be an *R-admissible graph* of a wfe e if it is an admissible graph of e and

(AS) The rator-node of R is accessible only via the rator-pointer of R, i.e.
$s_i(N) = s_1(R) \Rightarrow N = R$, $i=1$.

There are several ways of turning an admissible graph into an R-admissible one. A trivial method is to transform the graph into the equivalent tree, but that would destroy the purpose of the exercise. Suppose, however, that we make one copy (as a graph, not as a tree) of the rator-subgraph of R and adjust the rator-pointer of R to point to this copy. With notation as in (C1)-(C6) the nodes in F are to be copied, their pointers set as in the subgraph being copied, and the rator-pointer of R set to point to the copy of node F.

Example 4.2.4 (continued)

Using a prime to indicate the copy of a node, the result of the above operation on Fig.5(a) is shown in Fig.5(c). Applied to this graph, the steps in (C1)-(C6) then adjust the pointer from node 5' to node 6' to point to node 3, and node 4' is marked as the new root, Fig.5(d). The operation is now correct, yielding an admissible graph of

$$b((\lambda x.bx(\lambda x.xa))(\lambda x.xa))(\lambda x.xa) \quad .$$

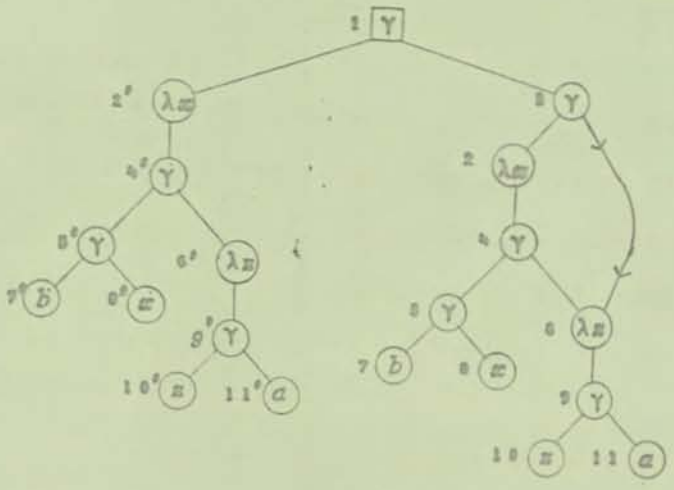


Figure 5(c)

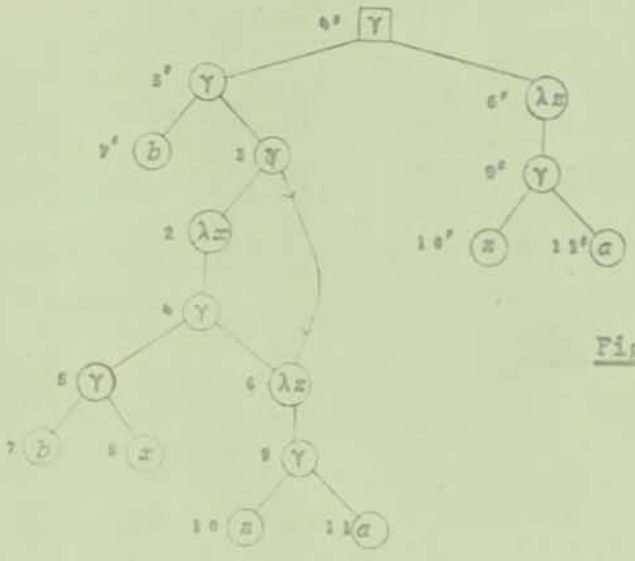


Figure 5(d)

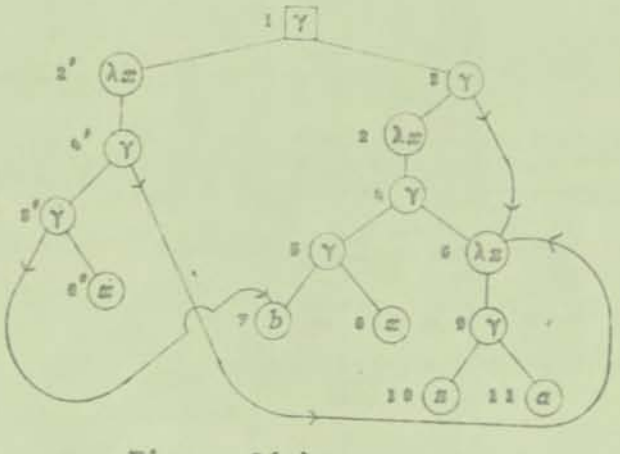


Figure 5(e)

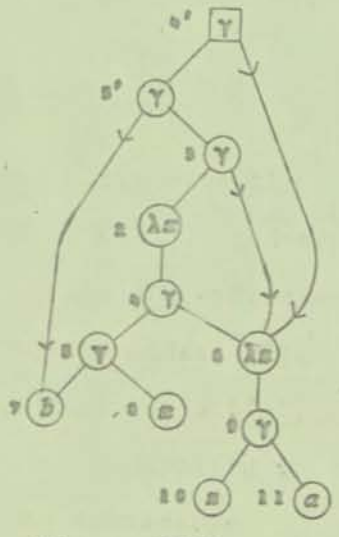


Figure 5(f)

Not all the nodes in F need be copied however. Obviously, the less that has to be copied the better. In general, if N is a node in F and $\varepsilon_N, \varepsilon_P$ are the wfes represented by nodes N, F, respectively, there will be no need to copy node N', or any of its sub-nodes, if no variable occurring free in ε_N is bound in ε_P. When this is the case, then ε_N could be "taken out" of ε_P by a β-expansion.

Definition. A proper sub-expression $\langle p, \varepsilon' \rangle$ of a wfe ε is said to be *directly abstractable* from ε if no free occurrence of a variable in $\langle p, \varepsilon' \rangle$ is bound in ε.

A sub-expression $\langle q, \varepsilon'' \rangle$ of a wfe ε is said to be *abstractable* from ε if it is part of a sub-expression $\langle p, \varepsilon' \rangle$ of ε (i.e. there is some non-null stem p of q such that $p(\varepsilon)=\varepsilon'$) which is directly abstractable from ε.

Example 4.2.5

The underlined parts of

$$\varepsilon = \lambda x.\underline{abx}(\lambda y.y(\underline{bcy})(\underline{(\lambda z.azz)}y))$$

are directly abstractable from ε. Any part which does not extend outside the underlines is abstractable from ε.

Example 4.2.4(cont.)

In Fig.5(a), nodes 6,7,11 represent parts directly abstractable from the wfe, ε_P, represented by node 2; nodes 6,7,9,10,11 represent parts abstractable from ε_P. Performing the copy-operation without copying these latter nodes, Fig. 5(e) is an R-admissible graph of the same wfe as Figs.5(a), 5(c). Fig.5(f) is then the result of (C1)-(C6) applied to redex-node $R=1$ in Fig.5(e); Fig.5(f) is an admissible graph of the same wfe as Fig.5(d).

§4.3 Proof of correctness

With the ideas of the last section in mind, the copy- and contraction-operations will now be precisely formulated, and it will be proved that they achieve the desired effect.

For the proofs which follow, G is an *admissible* graph

(1) $$G = (X, s_1, s_2, s_3, Z)$$

b is the binder-function of G, c denotes $Wfe(G)$, and c_N denotes the wfe represented by node N :

(2) $$c_N = Flatten(N, G)$$

A path from the root to a node N will be called simply a *path to* N. 'Directly abstractable' and 'abstractable' will often be abbreviated as 'd.a.' and 'abs', respectively. To save having to qualify allour equations, selectors and paths will be applied to nodes whether they are of the right type or not, on the understanding that both sides of equations are undefined if the selector or path is not applicable.

The definition of *abstractable* in §4.2 was given with reference to wfes as linear symbol strings. This will first be connected with admissible graphs by showing that *either all* peers of a node in an admissible graph are abstractable from its linear expansion, *or none* is (4.3.5).

Lemma 4.3.1

If p is a path to N, then $\langle p, c_N \rangle$ is d.a. from c iff $BvsOn(p, Z, G) \cap FreeVars(N, G)$ is empty.

Proof. This lemma is just a restatement of the definition of d.a. sub-expressions in terms of the functions given earlier.

Lemma 4.3.2

If p is a path to N, then

(3) $\langle p, \varepsilon_N \rangle$ d.a. from ε $\Leftrightarrow$ $b(T)=*$ for all $T \in FreeNodes(N,G)$

Hence, the RHS of (3) being independent of the path to N, if one peer of N is d.a. from ε, then all peers of N are d.a. from ε.

Proof.

($\Rightarrow$) Suppose $\langle p, \varepsilon_N \rangle$ d.a. from ε. Then,

$T \in FreeNodes(N,G) \Rightarrow var(T) \in FreeVars(N,G)$

$\Rightarrow var(T) \notin BvsOn(p,\Sigma,G)$, by 4.3.1

and

$T \in FreeNodes(N,G) \Rightarrow T \in FreeOccs(var(T),N,G)$

$\Rightarrow var(T) \notin Bvs(q,N,G)$,

for some path q from N to T

Taken together, these yield

$T \in FreeNodes(N,G) \Rightarrow var(T) \notin BvsOn(p*q,\Sigma,G)$

$\Rightarrow T \in FreeOccs(var(T),\Sigma,G)$

$\Rightarrow *$ is a binder of T in G

$\Rightarrow b(T)=*$, since G is admissible.

($\Leftarrow$) Suppose $b(T)=*$ for all $T \in FreeNodes(N,G)$. Then,

$z \in FreeVars(N,G) \Rightarrow z=var(T)$ for some $T \in FreeNodes(N,G)$

$\Rightarrow z \notin BvsOn(p,\Sigma,G)$, since $*$ is the only binder of T

Hence, $\langle p, \varepsilon_N \rangle$ is d.a. from ε, by 4.3.1. □

Lemma 4.3.3 If $T \in FreeNodes(N,G)$ and $b(T)=*$, then every path to N passes through $b(T)$.

Proof. Let p be any path to N, and let $L=b(T)$, $z=var(T)$. Since $T \in FreeNodes(N,G)$, there is some path q from N to T such that $L \notin NodesOn(q,N,G)$. Then, $p*q$ is a path to T, and $z \in BvsOn(p,\Sigma,G)$, otherwise $*$ would be a second binder

of T. Let p' be the longest stem of p such that $x=bv(L')$, where $L'=p'(Z)$. Then L' must be L, otherwise L' would be a second binder of T. ∎

Corollary 4.3.4

(a) If $b(T)=*$, every path to T passes through $b(T)$.

(b) If $b(T)=e$ and p is any path to T, then

$$var(T) \notin BvsOn(p,Z,G)$$

Proof. (a) By 4.3.3, since $T \in FreeNodes(N,G)$.

(b) Otherwise, T has a binder other than $*$.

Lemma 4.3.5

If one peer of N is abs. from c, then all peers of N are abs. from c.

Proof. Let p be a path to N such that $\langle p,c_N\rangle$ is abs. from c, and let p_0 be the *longest* stem of p such that $\langle p_0,c_D\rangle$ is d.a. from c, where $D=p_0(Z)$. Let p_1 be the path such that $p = p_0 \cdot p_1$.

Let q be any path to N other than p. For $\langle q,c_N\rangle$ to be abs. from c, it suffices to show that q passes through D. Let q_0 be the *shortest* stem of q such that $q_0(Z)$ meets the path p_1 from D to N (exists since $q(Z)$ and $p_1(D)$ meet at N), and let $A=q_0(Z)$. Let p_2 be the stem of p_1 such that $p_2(D)=A$.

If $A=D$, the result is proved.

If $A \neq D$, then $\langle p_0 \cdot p_2, c_A\rangle$ is not d.a. from c (by choice of p_0). Hence, by 4.3.2, $\langle q_0,c_A\rangle$ is not d.a. from c, and there is a $T \in FreeNodes(A,G)$ such that $b=b(T)=e$. Since $\langle p_0,c_D\rangle$ is d.a. from c, $T \notin FreeNodes(D,G)$, by 4.3.2,

whence $L \in NodesOn(p_2, D, G)$; and $L \in NodesOn(q_0, E, G)$, by 4.3.3. Then, $L \notin A$ would contradict the choice of q_0, and $L = A$ would contradict $T \in FreeNodes(A, G)$. $A \notin D$ is therefore impossible. □

Subgraphs

For a node F of G, the *F-subgraph* of G is defined as

(4) $$(F, e_1', e_2', e_3', F)$$,

where F is the set of sub-nodes of F in G, and e_i' is the restriction of e_i to F ($i=1,2,3$).

Define b_F as the restiction of b to $T \in F$:

(5) $$b_F(T) = \begin{cases} b(T) & \text{, if } b(T) \in F \\ * & \text{, otherwise} \end{cases}$$

Clearly, any F-subgraph of an admissible graph is also admissible, and b_F is its binder function.

Lemma 4.3.6

If p and q are any two paths from F to H in the admissible graph (1), then

(a) $\langle p, e_H \rangle$ d.a. from e_F $\Longleftrightarrow$ $\langle q, e_H \rangle$ d.a. from e_F
$\Longleftrightarrow$ $b_F(T) = *$ for all $T \in FreeNodes(H, G)$

(b) $\langle p, e_H \rangle$ abs. from e_F $\Longleftrightarrow$ $\langle q, e_H \rangle$ abs. from e_F

Proof. Immediate by applying 4.3.2 and 4.3.5 to the subgraph.

With this last lemma, for admissible graphs it is now unambiguous to say that a *node*, H, is abs.(or d.a.) from a *node*, F, without reference to a path from F to H. Similarly, for terminal nodes, we can say that a *node*, T , is free in a *node*, F, meaning that all peers of T are free occurrences of $var(T)$ in e_F.

Lemma 4.3.7

If T is a terminal sub-node of F, then

T abs. from $F \iff b_p(T)=e$ or $b_p(T)=b(T)$ abs. from F.

Proof. If $b_p(T)\neq e$, then $b_p(T)=b(T)$, by (5).

($\Leftarrow$) Trivial, since T is a sub-node of its binder when $b_p(T)\neq e$, and if $b_p(T)=e$ then T is free in F.

($\Rightarrow$) Suppose T abs. from F and $L=b_p(T)\neq e$. Let p and q be paths from F to L and from L to T, respectively. Since T is abs. from F, there is a stem p' of $p\cdot q$ such that $p'(F)=N$ is d.a. from F. Then p' must be a stem p (else, $T \in FreeNodes(N,G)$ and $b_p(T)\neq e$, so N d.a. from F would contradict 4.3.6(a)). Hence, L is abs. from F, since it is a sub-node of N. ■

Lemma 4.3.8

Let F be any node in an admissible graph (1). Then, any sub-node N of F accessible other than via F is abstractable from F.

Proof. If $N=F$, the lemma is vacuous.

Let p be a (non-null) path from F to N, and let q be a path to N not via F. Let p',q' be the *shortest* stems of p,q such that $p'(F) = q'(Z) = A$, say. It suffices to show that A is d.a. from F. Suppose not. Then, by 4.3.6(a), there is a $Z \in FreeNodes(A,G)$ such that $L=b_p(T)\neq e$; hence, $L \in NodesOn(p',F,G)$, by 4.3.4(a) applied to the F-subgraph. But, by (5), $L=b_p(T)=b(T)$, hence also $L \in NodesOn(q',Z,G)$, by 4.3.4(a). There is now a similar contradiction to that in 4.3.5, whence A must be d.a. from F. ■

The-copy-operation

Let G be the *admissible* graph (1) with binder-function b. Suppose R is a redex-node of G. Let $P = s_1(R)$ and

F = the set of sub-nodes of P

$C = \{P\} \cup \{N \in F : N \text{ not abs. from } P\}$.

Form a set C' of copies of the nodes in C (i.e. nodes in C' are identical in type and data components to those in C, but are indexed differently). Let ϕ be the function from C onto C' mapping each node in C to its copy in C', and let $\psi = \phi^{-1}$. Define functions

$$f : X \to (X-C) \cup C'$$

$$g : (X \cup C') \to X$$

by

$$f(N) = \begin{cases} \phi(N) & , \text{ if } N \in C \\ N & , \text{ if } N \in (X-C) \end{cases}$$

$$g(N) = \begin{cases} \psi(N) & , \text{ if } N \in C' \\ N & , \text{ if } N \in X \end{cases}$$

Define

(6) $\quad G' = Copy(R,G) = (X \cup C', s_1', s_2', s_3', Z)$

where, for $i=1,2,3$,

$$s_i'(N) = \begin{cases} f(s_i(g(N))) & , \text{ if } N \in C' \\ s_i(N) & , \text{ if } N \in X,\ N \neq R \end{cases}$$

and

$$s_1'(R) = f(P) = \phi(P)$$

$$s_i'(R) = s_i(R) \quad , i=2,3$$

Theorem 4.3.9

If R is any redex-node of an admissible graph G of a wfe e, then $Copy(R,G)$ is an R-admissible graph of e.

We shall prove this result by a series of lemmas. Since two graphs are being treated, a path will be given a graph as a second argument to distinguish in which graph the path is being taken.

Lemma 4.3.10 For all $N \in X \cup C'$,

(a) $N \in X \Rightarrow g(f(N)) = N$

(b) $g(s_i'(N)) = s_i(g(N))$, for $i=1,2,3$

(c) $p(g(N),G) = g(p(N,G'))$, for all paths p.

Proof. (a) Immediate from the definitions of f and g.

(b) If $N \in C'$, then $g(s_i'(N)) = g(f(s_i(g(N))))$
$= s_i(g(N))$

If $N \in X$, $N \neq R$ or if $N=R$ for $i=2,3$, then

$g(s_i'(N)) = g(s_i(N)) = s_i(N)$, since $s_i(N) \in X$
$= s_i(g(N))$, since $N \in X$

If $N=R$ and $i=1$, then

$g(s_1'(R)) = g(f(F)) = F = s_1(R) = s_1(g(R))$.

(c) Straightforward from (b) by induction on the length of the path p. ∎

Corollary 4.3.11 G' is acyclic.

Proof. Otherwise, suppose p is an N-cycle in G', i.e. $p(N,G') = N$. Then,

$g(N) = g(p(N,G')) = p(g(N),G)$, by 4.3.10(c),

in contradiction of G being acyclic. ∎

Lemma 4.3.12 For all $N \in X \cup C'$,

(7) $Flatten(N,G') = Flatten(g(N),G)$.

Hence, G and G' represent the same wfe (since $g(S)=S$ and S is the root of both G and G').

Proof. By induction on the *size* of N in G', where the *size* of N is defined as the number of nodes in the N-subgraph of G' (i.e. the number of sub-nodes of N in G').

Let $\varepsilon', \varepsilon$ denote the LHS, RHS of (7), respectively.

If $Is\tau(N)$, then

$$\varepsilon = var(g(N)) = \begin{cases} var(N) = \varepsilon' & , \text{ if } N \in X \\ var(\psi(N)) = \varepsilon' & , \text{ if } N \in C' \end{cases}$$

If $Is\gamma(N)$, let $\quad \varepsilon_i' = Flatten(e_i'(N), G') \quad , i=1,2$

$$\varepsilon_i = Flatten(e_i(g(N)), G) \quad , i=1,2$$

Since G' is acyclic by 4.3.11, the size of $e_i'(N)$ in G' is < the size of N in G' ; thus,

$$\varepsilon_i' = Flatten(g(e_i'(N)), G) \quad , \text{ by ind.hyp.}$$

$$= Flatten(e_i(g(N)), G) = \varepsilon_i \quad , \text{ by 4.3.10(b)}$$

Hence, from the definition of *Flatten*,

$$\varepsilon' = \varepsilon_1'(\varepsilon_2') = \varepsilon_1(\varepsilon_2) = \varepsilon \quad .$$

If $Is\lambda(N)$, the induction is similar. ∎

Lemma 4.3.13 G' satisfies condition (A2).

Proof. By 4.3.7 and definition of C, for any terminal node T in G,

$$T \in C \iff b(T) \neq * \text{ and } b(T) \in C \quad .$$

Thus, terminal sub-nodes of F are copied iff their (unique) binder in G is copied. For terminal nodes T' in $X \cup C'$, define

$$b'(T') = \begin{cases} f(b(g(T'))) & , \text{ if } T' \in C' \\ b(T') & , \text{ if } T' \in X \end{cases}$$

From 4.3.10(c) and the fact that γ and ψ preserve the type and data components of nodes, $b'(T')$ must be a binder of T' in G', and there can be no other binder (either * or a λ-node) of T' in G'. ∎

Lemma 4.3.14 G' satisfies condition (A3) for redex R.

Proof. Since $s_1'(R)=\phi(F)$, we have to show that

$$H\neq R \text{ or } i=2,3 \Rightarrow s_i'(H) \neq \phi(H) .$$

The result follows easily from the definitions of f, g and s_i' in all cases except when $H \in C^0$ and $s_i(g(H)) \in C$. For this case,

$$s_i'(H) = f(s_i(g(H))) = \phi(s_i(\phi(H))).$$

If $\phi(F) = \phi(s_i(\phi(H)))$, then $F=s_i(\phi(H)) \in C$, in contradiction of G being acyclic. ■

This completes the proof of 4.3.9.

The contraction operation

Let R be a redex-node of $G = (X, s_1, s_2, s_3, Z)$ and suppose G is now *R-admissible*, with binder-function b. Let $\varepsilon = Vfe(G)$.

First, we need to be a little more careful about bound variables than in the informal description (C1)-(C6) in §4.2. Specifically, we can suppose that *bvs* of λ-nodes are pairwise distinct and all different from the free variables of ε. For if not, with the existence of the bᵢnder-function for graphs, the bound variables can be changed as follows:

(B1) Let $L_1, L_2, \cdots, L_n$ be the λ-nodes of G, and let $v_1, v_2, \cdots, v_n$ be distinct variables not occurring in the graph.

(B2) For $j=1,2,\cdots,n$, change $bv(L_j)$ to v_j.

(B3) For each terminal node with $b(F)\neq\Omega$, change its *var*-component to the *bv* of its binder.

(These changes can, in fact, be performed not just for

R-admissible graphs but for any admissible graph.) We shall show later that this assumption about bound variables involves no loss of generality; by suitably 'pre-processing' the initial graph, the conditions can always be satisfied in a sequence of contractions on graphs.

Now let $F = s_1(R)$, $A = s_2(R)$,

$M = s_1(F)$, $v = bv(F)$,

let F,A,M be the set of sub-nodes of F,A,M, respectively, and let

$$V = \{T \in M : Is\tau(T) \text{ and } b(T) = F\} .$$

The relation between these can be depicted as

Let

$$C = \begin{cases} A & \text{, if } M \in V \text{ (i.e. if } M \Leftrightarrow (v)) \\ M & \text{, if } M \notin V \end{cases}$$

Now define

(8) $$G' = Contract(R,G) = (X', s_1', s_2', s_3', Z')$$

where $$X' = X - V - \{R,F\}$$

$$Z' = \begin{cases} Z & \text{, if } R \neq Z \\ C & \text{, if } R = Z \end{cases}$$

and, for $N \in X'$,

$$s_i'(N) = \begin{cases} C & \text{, if } s_i(N) = R \\ A & \text{, if } s_i(N) \in V \\ s_i(N) & \text{, otherwise} \end{cases}$$

Theorem 4.3.15

If R is a redex-node in G (after changes (B1)-(B3) to its bound variables, if necessary), and if G is an R-admissible graph of a wfe ε, then *Contract*(R,G) defined in (8) is an admissible graph of the wfe ε' which is the result of a complete reduction relative to the peers of R in ε.

Again this theorem will be proved by a series of lemmas. That G is R-admissible can be stated as

(9) $$s_i(N) = F \iff N=R \ , \ i=1 \ .$$

In particular, $F \notin A$, since $A=s_2(R)$. The condition that G be R-admissible will be used to guarantee that the nodes in V represent *only* free occurrences of v in the bases of the peers of R in ε, so that adjustment of pointers to them does not inadvertently change any other parts of ε.

Lemma 4.3.16 Nodes in V are accessible only via the rator-pointer from R to F.

Proof. $T \in V \Rightarrow b(T)=F \notin \bullet \Rightarrow T$ accessible only via F, by 4.3.4(a). The lemma now follows from (9).

Lemma 4.3.17

$A \cap V$ is empty, and hence

$$N \in A \Rightarrow s_i'(N) = s_i(N) \ .$$

Proof. Suppose q were a path from A to a node $T \in V$, and let p be any path to R. Then, $p \cdot rand \cdot q$ is a path to T, hence passes through $b(T)=F$, by 4.3.4(a). But this is impossible since $F \notin A$ and

(i) p cannot pass through F, otherwise *p-rator* would contain a non-null path from F to itself,

(iii q cannot pass through F, otherwise q also passes through R (by 4.3.16), and *rand-q* then contains a non-null path from R to itself.

Thus, paths from A to nodes in V cannot exist.

For the second part, if $N \in A$, then $\sigma_i(N) \in A$, whence $\sigma_i(N) \notin V$, by the first part ; and $\sigma_i(N) \neq N$, else G would be cyclic. Hence $\sigma_i'(N) = \sigma_i(N)$, by definition of σ_i'. ∎

Now denote

$$\varepsilon_N = Flatten(N,G) \quad , \text{ for } N \in X$$
$$\varepsilon_N' = Flatten(N,G') \quad , \text{ for } N \in X' .$$

Lemma 4.3.18 $\varepsilon_A = \varepsilon_A'$.

Proof. From 4.3.17,

$$N \in A \Rightarrow \varepsilon_N = \varepsilon_N'$$

follows by induction on the size of N in G, whence $\varepsilon_A = \varepsilon_A'$, since $A \in A$. ∎

This lemma shows that, in G', node A still represents the rand of the redex being contracted. To show that variables are not captured in the adjustment of pointers, we need to show that no variable free in ε_A is bound in ε_P. When, after (B1)-(B3), the *bvs* of λ-nodes of G' are pairwise distinct and differ from the free variables in ε, it suffices to show that the binder (in G) of free nodes of A is not in M.

Lemma 4.3.19

If T is a terminal node in A and $b_A(T)=*$, then either $b(T)=*$ or $b(T) \notin M$.

Proof. Suppose $L=b(T)\neq *$, and let p be a path to R. By 4.3.3, the path p *rand* to A passes through L, and, since $L \notin A$, L must lie on the path p to R. If L were now a sub-node of F, there would be a path in G from L to R to F to N to L. ∎

We can now show that, in G', node C represents the contractum of c_R. First, define a function

$$h : M \to (M \cup \{A\}) - V$$

by $$h(N) = (N \in V \to A, N)$$

Then, by definition of s_i',

(10) $$s_i'(N) = h(s_i(N)) \quad , \text{ for } N \in M, \ i=1,2,3.$$

Lemma 4.3.20 For nodes $N \in M$,

$$[c_A/v]c_N = c'_{h(N)} \ .$$

Proof. By induction on the size of N in G.

1. If $I \in \tau(N)$,

   (a) If $N \in V$, then $h(N)=A$, $c_N = var(N) = v$, and
   $$[c_A/v]c_N = c_A = c_A' \ .$$

   (b) If $N \notin V$, then $h(N)=N$, $\xi \equiv var(N) \neq v$, and
   $$[c_A/v]c_N = \xi \Rightarrow c_N' \ .$$

2. If $I \in \lambda(N)$, let $\xi = bv(N)$. Then, $\xi \neq v$, since N is a proper sub-node of F, and ξ is not free in c_A, by 4.3.19. Thus,

$$\varepsilon_N = Flatten(N,G) = \lambda\xi.Flatten(s_\xi(N),G) \quad ,$$

so $[\varepsilon_A/v]\varepsilon_N = \lambda\xi.[\varepsilon_A/v](Flatten(s_\xi(N),G))$

$= \lambda\xi.Flatten(h(s_\xi(N)),G')$, by ind.hyp.

$= \lambda\xi.Flatten(s_\xi'(N),G')$, by (10)

$= Flatten(N,G')$

$= \varepsilon_N' = \varepsilon'_{h(N)}$, since $N \leq V$.

ξ. If $\xi\gamma(N)$, the proof is similar. ∎

Corollary 4.3.21 $[\varepsilon_A/v]\varepsilon_M = \varepsilon_C'$

Proof. From 4.3.20, since $h(M)=C$.

Lemma 4.3.22 The wfe, ε', represented by G' is the result of contracting all peers of R in ε.

Proof. If $R=Z$, then C is the root of G' and the lemma is just 4.3.21.

If $R\neq Z$, then Z is the root of G and G'. Let P be the set of all paths from Z to R in G. Consider any path q from Z in G which does not have a stem in P and is not the stem of any path in P (i.e. q does not pass through R and cannot be extended to pass through R).

Let N be a node on q. Then, $s_\xi(N) \neq R$ and $s_\xi(N) \notin V$, by choice of q and 4.3.16, so $s_\xi'(N) = s_\xi(N)$, by definition of s_ξ'. Since this holds for all nodes N on all such paths q from the common root, Z, of G and G', ε therefore matches ε' except at peers of R.

For any path $p \in P$, we have $p(Z,G)=R$ and $p(Z,G')=C$. The sub-expression $\langle p,\varepsilon_R\rangle$ of ε has thus been replaced in ε' by $\langle p,\varepsilon_C'\rangle$, which is the contractum of ε_R, by 4.3.21. ∎

That G' is acyclic is immediate since e' is a wfe of finite length. It was not necessary to *first* prove that G' was acyclic this time. The inductions in 4.3.19 and 4.3.20 are on the size of nodes in G ; these are well-founded since, G being acyclic, sub-nodes of a node in G have smaller size than the node itself. For the copy-operations, the induction in 4.3.12 was on the size of nodes in *Copy*(R,G), hence for this induction to be well-founded, *Copy*(R,G) had first to be proved acyclic.

Finally, to show that G' is an *admissible* graph of e', it remains to prove

Lemma 4.3.23 G' satisfies condition (A2).

Proof. Let T and L be the sets of τ- and λ-nodes, respectively, of G. Then, T' = T-V and L' = L-{P} are the corresponding sets for G'. Define a function

$$b' : T' \to L' \cup \{*\}$$

as the restriction of the binder-function, b, for G to T' :

$$b'(T') = b(T') \quad \text{, for all } T' \in T' \text{ .}$$

To show that b' is the binder-function for G', there is no need to argue about paths of G' ; that the wfe, e', represented by G' *is* the result of a reduction from e (by 4.3.22), and the properties of *bvs* of nodes in L after (B1)-(B3) need only be used.

If $b(T') = *$, then no node in L, and therefore no node in L', has *var*(T') as its *bv*-component. Hence all peers of T' in e' are free occurrences of *var*(T'), so e is their unique binder in G'.

If $L=b(T')\neq s$, then all peers of T' in ε are parts of peers of L in ε. Peers of T' in ε' must then be parts of peers of L in ε', since bound occurrences cannot become free in a reduction and L is the only λ-node with $var(T')$ as its *bv*-component. Hence, L is the unique binder of T' in G'.

This completes the proof of 4.3.15. □

We now return to the question of the bound variable changes in (B1)-(B3) and show that the assumption that these changes have always been made is justified. We argue that although the above *proofs* use the *names* of variables, the two operations themselves can be performed without reference to the *specific* names of variables.

The contraction-operation is already so formulated; the set, V, of nodes representing occurrences of the variable being substituted for is defined in terms of the binder-function for (admissible) graphs alone.

For the copy-operation, in order to determine the set, C, of nodes to be copied (the set of nodes not abs. from F), it suffices to determine those sub-nodes which are *directly* abstractable from F (since, by definition of abstractable and our results for admissible graphs, a node is abs. from F iff it is a sub-node of one which is d.a. from F). By 4.3.6(a), a node, N, is d.a. from F *iff*

$$T \in FreeNodes(N,G) \rightarrow b_F(T) = s \quad .$$

But a (terminal)node, T, is a free node of N iff $b_N(T)=s$, so that N is d.a. from F iff

(11) $$b_N(T) = N \rightarrow b_F(T) = s \quad .$$

Thus, again, only the binder-function is required.

Both operations also uniquely determine the binder-function for the transformed graphs (c.f. 4.3.13, 4.3.23). For the *internal* workings of these operations in a sequence of transformations on graphs, therefore, the var- and bv-components of nodes are redundant. Only when the graph is to be flattened to a linear symbol string need bound variables be inserted, and then any assignment in accordance with (B1)-(B3) suffices.

In an implementation of this method of reducing wfes, obtaining the binder-function for the initial graph will be one of the tasks of a pre-processor (see §4.5). By these means, the need for α-conversions *during* the course of a reduction can be averted.

§4.4 Normal-order graph reduction

The last two sections have presented a general technique for contracting redexes in graphs which represent wfes. For a specific *algorithm* one now has to give a 'rule' which determines the redex-node on which to operate in each step. We shall treat the *normal-order* rule that the left-most β-redex is contracted in each step. This node will be called the *left-most redex-node*.

Let ε be any wfe. For the initial graph, let G_0 be an admissible graph of ε. Any admissible graph of ε suffices, e.g. its tree may be used, though there are more

"economical" ones. For $i \geq 0$, define inductively

$$(12)\begin{cases} R_i = \text{the left-most redex-node in } G_i \\ G_i' = \begin{cases} G_i & \text{, if } G_i \text{ is } R_i\text{-admissible} \\ Copy(R_i, G_i) & \text{, otherwise} \end{cases} \\ G_{i+1} = Contract(R_i, G_i') \end{cases}$$

Example 4.4.1

Figure 6 presents the normal-order graph reduction of

$$e = (\lambda f.f(f(fa)))((\lambda x.\lambda y.yx)b)$$

to its normal form $e' = abbb$

starting from its tree representation, Fig.6(a). In each graph the left-most redex-node is distinguished by underlining the number labelling it. Where G_i' differs from G_i, a ' has been used on the figure letter, i.e. Figs.6(c'), 6(d'). In these two applications of the copy-operation, node 4 is not copied, since it is d.a. from node 5.

The initial wfe, e, has been reduced to e' by 5 contractions in these graphs, whereas normal reduction operating on string representations would have required 7 contractions. The critical step which achieves this shortening of the reduction is that from Fig.6(b) to 6(c), where the contraction of redex-node 2 in 6(b) represents three contractions in its linear expansion. □

Since, by the results of §4.3, normal-order graph reduction (NGR, for short) correctly represents one reduction of a wfe, whenever this terminates then normal β-reduction (NBR, for short) must also terminate. For the converse, we shall prove something stronger (4.4.3 below),

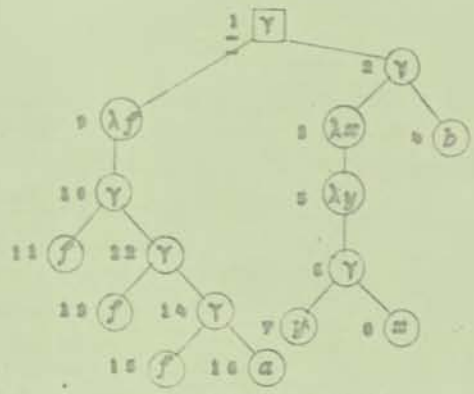


Figure 6(a)

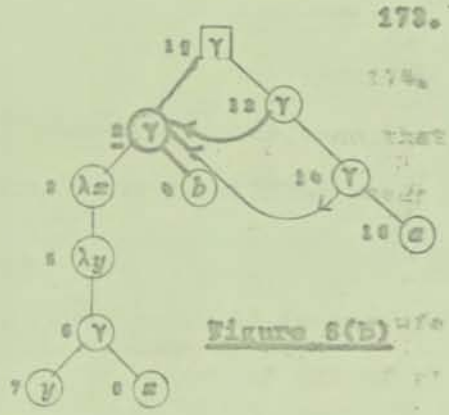


Figure 6(b)

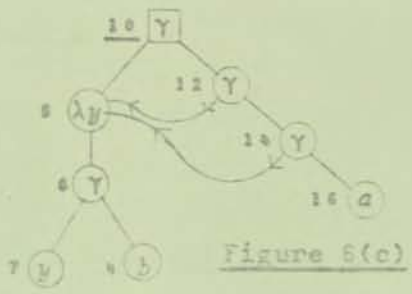


Figure 6(c)

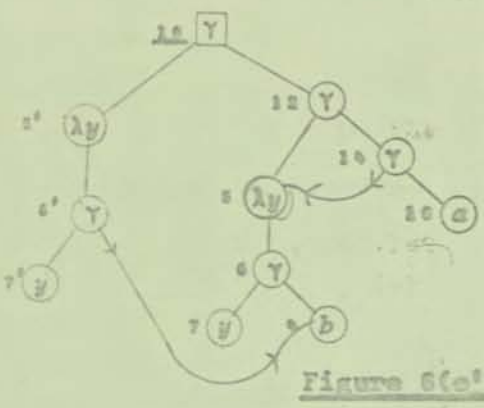


Figure 6(c')

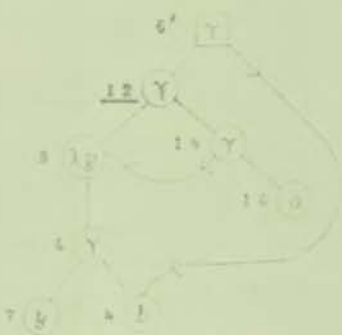


Figure 6(d)

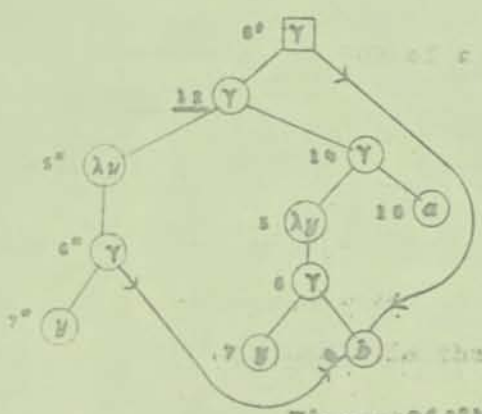
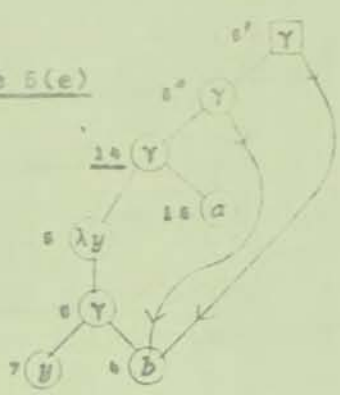


Figure 6(d')

Figure 6(e)



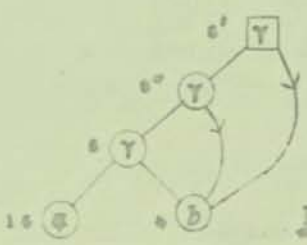


Figure 6(f)

namely that the length of NGR is never greater than that of NBR. The following preliminary lemma will be used:

Lemma 4.4.2

Suppose $\mathcal{E}$ has a normal form N, and $\mathcal{E}'$ is any wfe to which $\mathcal{E}$ is reducible. Then, the length of NBR of $\mathcal{E}'$ is $\leq$ that for $\mathcal{E}$.

Proof. Apply 3.3.5 to each step in a reduction of $\mathcal{E}$ to $\mathcal{E}'$.

Theorem 4.4.3

Suppose $\mathcal{E}$ has a normal form N, and let n be the length of NBR of $\mathcal{E}$ to N. For $i \geq 0$, let $\mathcal{E}_i$ be the wfe represented by the ith. stage, G_i, in NGR of $\mathcal{E}$, and let m_i be the length of NBR of $\mathcal{E}_i$ to the normal form N.(+) Then,

either (a) $\mathcal{E}_i$ is in normal form,

or (b) $0 < m_i \leq n-i$.

Hence, alternative (a) must hold for some $i \leq n$, so NGR of $\mathcal{E}$ terminates after at most n graph-contractions.

Proof. By induction on i.

For $i=0$, we have $m_0 = n$, by definition of G_0.

Suppose the result holds for $i =$ some $j \geq 0$, and $\mathcal{E}_j$ is not in normal form. The induction hypothesis is thus

(13) $\qquad 0 < m_j \leq n-j$.

Let $\qquad \mathcal{A}_j = (R_j^1, R_j^2, \cdots, R_j^k)$, $k \geq 1$.

be the peers of the left-most redex-node, R_j, in G_j (k depends on j, of course). For definiteness, suppose R_j^1 is the member of $\mathcal{A}_j$ which is the left-most β-redex in $\mathcal{E}_j$.

(+) The existence and finite length of the normal β-reduction of $\mathcal{E}_i$ to N follow since $\mathcal{E}_i$ is a wfe to which $\mathcal{E}$ is reducible.

From (12) and 4.3.9, G_j' is an R_j-admissible graph of ε_j ; hence, by 4.3.15, G_{j+1} is an admissible graph of ε_{j+1}, and ε_{j+1} is the result of a complete reduction relative to $\mathcal{R}_j$.

Let ε_j^1 be the result of contracting R_j^1 in ε_j. By choice of R_j^1, ε_j^1 is the first stage in NBR of ε_j; hence, the length, m_j^1, of NBR of ε_j^1 to H is

(14) $$m_j^1 = m_j - 1 \quad .$$

From the lemma of parallel moves, ε_{j+1} is the result of a complete reduction relative to the residuals of $\mathcal{R}_j$ in ε_j^1 (for the present case, these residuals are just $\mathcal{R}_j - \{R_j^1\}$, since the redexes in $\mathcal{R}_j$ are disjoint). Then,

$$m_{j+1} \leq m_j^1 \quad , \text{ by 4.4.2}$$

$$\leq n-(j+1) \quad , \text{ by (13) and (14)}$$

whence either (a) or (b) holds for $i = j+1$. ∎

The same proof can be used to show that any method of reducing wfes for which each "step" includes contraction of the left-most redex will terminate whenever a normal form exists.

It is worth emphasizing, however, that graph reduction does not contract the members of $\mathcal{R}_j$ one by one. The intermediate stages in a complete $\mathcal{R}_j$-reduction are by-passed, graph reduction achieving *simultaneous* contraction of the redexes in $\mathcal{R}_j$. The number of such complete relative reductions is therefore a fair measure of the length of NGR of a wfe.

Theorem 4.4.3 provides a *qualitative* comparison of the 'efficiency' of NGR and NBR. Quantitative results

are not so easy to discover, however, other than the

Theorem 4.4.4

For any two integers m,n with $2 \leq m \leq n$, one can find a wfe, ε, such that the lengths of NGR and NBR of ε are m and n, respectively.

Proof. It suffices to consider 2cases: $m=n$, and $m=2$. The result for general m,n with $2 \leq m \leq n$ follows by suitably intermixing examples for these 2 cases.

For the case $m=n$, take, e.g., $\varepsilon = I^n(I)$.

For the case $m=2$, take, e.g.,

$$\varepsilon = (\lambda y.xyy \cdots y)((\lambda y.yy)x)$$

$n-1$ times

□

It would be pleasing if one could show also that NGR is never longer than the applicative-order reduction (AR, for short) of a wfe (when the latter terminates). Empirical observations confirm the author's suspicion that this is true, but the result is not so easy to establish, since the intermediate wfes in AR are not so closely related to those in NGR.

§4.5 Features of implementation

To program an implementation of the present method of reducing wfes is a straightforward task. There being little programming originality required, a full description of one, while not a wasted exercise, would be out of place here. However, one or two "tricks" which the author found

helpful for the sake of efficiency when he implemented the technique do merit a short discussion.

Nodes can be represented by *vectors* containing (representations of) their type and data components. Each vector has an address, namely the location of the beginning of the vector, whence application of a selector function is represented by an element of the vector which is the address of the resulting vector.

This vector-representation constitutes the minimum information necessary for the copy- and contraction-operations on graphs. In practice, extra elements will be included in the vectors to simplify some of the decisions encountered. (The specific order in which the vector elements represent these components is of no importance, as long as this is done in a systematic fashion.)

To decide at each stage whether the graph is R-admissible, where R is the redex to be contracted next, it must be determined whether there is only one pointer to the rator of R. One solution is to include a count of the number of references to each node, so that one enquires whether the count for the rator of R is one. (Note that, although the question is a Boolean one – is there one or more than one reference – to decide this it is necessary to maintain throughout the reduction a record of the actual number of references to nodes.)

As given by (C1)-(C6) in §4.3, one part, (C4), of the contraction-operation involves adjusting pointers to a redex-node, R, to point to the node, M, representing the

base of R. Node R thus becomes "detached" from the rest of the graph and could be returned to free storage. Instead, a fourth type of node, an *I-node*, this detached node can be more usefully employed. An I-node introduces a degree of 'indirectness' into the pointers, its sole purpose being to point to another node; references to I-nodes are to be interpreted as references to the node to which the I-node points. Overwriting R with an I-node pointing to N, operation (C4) can now be effected without having to search the graph for all references to R.

I-nodes can also be used to simplify operation (C5). To show this, we indicate how the binder-function may be incorporated into the data structures being manipulated.

First, construct the *tree* of the initial wfe e, and let b be its binder-function. For each terminal node, replace all references to it by pointers to its binder. (If $b(T)=\Lambda$, it has no binder; in this case, choose one representative, A say, from among the terminal nodes which represent free occurrences of $var(T)$ in e and replace all references to such nodes by pointers to A.) The cyclic nature of the resulting structure is resolved by marking the binder-pointers in some distinguished way, for these are to be interpreted differently from other pointers; viz., when a graph is being 'flattened' (after *bv*-assignments as in (B1),(B2)), or a subgraph is being copied, binder-pointers are regarded as pointers to terminal nodes with *var*-component the *bv* of the λ-node to which they point.

This *pre-processing* phase can, of course, be achieved in other ways. For instance, it is not necessary to first

construct the tree of the initial wfe; the required structure can be formed as the initial wfe is being input. In effect, binder-pointers are *internal names* for bound variables; the idea is one manifestation of a technique found in several compilers for programming languages, known as *canonicalization* of variables.

For the present application, the binder-pointers have the advantage that no longer need one search for those nodes which represent occurrences of the variable being substituted for — these occurrences are now represented by those binder-pointers which lead to the rator, F, of the redex, R, being contracted. Since F would become detached by the contraction operation (provided the graph is R-admissible), part (C5) can now be achieved by overwriting F with an I-node pointing to the rand of R.

In theory, the distinguishing "marks" on the binder-pointers to F should now be removed. In practice, again this is unnecessary; for we now extend the interpretation of pointers marked in this way such that they are not treated as binder-pointers if they point to an I-node. It can be seen that the handling of these pointers is such that marked pointers cannot lead to I-nodes when they are to be genuinely treated as binder-pointers.

One can, if one likes, regard there as being two 'modes' when expanding graphs — *ordinary* mode as given above, and *special* mode. Tracing a binder-pointer switches from ordinary mode to special mode, in which I-nodes revert to ordinary, λ-nodes are treated as occurrences of bound variables, and other nodes are errors.

Summarising these usages of I-nodes, contraction of a redex-node, *R*, with rator-node, *F*, can be stated as:

(CI1). Overwrite *R* with I-node pointing to the body of *F*.

(CI2). Overwrite *F* with an I-node pointing to the rand of *R*.

In these terms, the contraction of redexes in graphs has become almost a trivial operation. The entire "cost" of the technique resides in the overheads necessary to organize the transformations (CI1),(CI2) and to ensure the validity of their application. That is, the redex-node to be contracted at each stage must be determined, and if the graph is not *R*-admissible, the copy-operation must first be applied.

Any standard technique for finding redexes in wfes (as symbol strings) can be adapted for locating redex-nodes in graphs. Thus, although normal-order reduction is defined as an iterative process – successively contract the left-most redex – the efficiency of its implementation is improved by being programmed as a pair of mutually recursive operations, *RTNF* and *RTLF*. The former provides the general control of *reduction to normal form*, the latter controls *reduction to lambda-form* . *RTLF* is an amended version of *RTNF* used for the rators of combinations, attempting to reduce them to abstractions ; if successful, the combination is then a β-redex, the process continuing by operating on the contractum of this redex.

For a wfe *e*, *RTNF* is defined by the usual 3-way switch on the structure of *e* :

Case 1. e is an identifier, ξ

Then, ξ is the normal form of e.

Case 2. e is an abstraction, $(\lambda\xi.e_1)$

Apply *RTNF* to e_1. If this terminates with a normal form e_1', then $(\lambda\xi.e_1')$ is the result for e.

Case 3. e is a combination, $e_1(e_2)$

Apply *RTLF* to the rator, e_1. If this terminates with e_1', test whether e_1' is an abstraction :

(a) If so, then

(a_1) Contract the β-redex $e_1'(e_2)$ to yield e'.

(a_2) Apply *RTNF* to e'; if this terminates with e'', then e'' is the result for e.

(b) If not, apply *RTNF* to the rand, e_2; if this terminates with e_2', then $e_1'(e_2')$ is the result for e.

To define *RTLF* : case 1 is as above; in case 2, return e itself as the result; case 3 is as above, except use *RTLF* in place of *RTNF* in (a_2).

An implementation of normal-order *graph* reduction using this recursive control-strategy is straightforward, the three cases corresponding to the three node types (other than I-nodes). The only variation is in case 3(a_1), which is achieved for graphs by (CI1) and (CI2) (possibly preceded by an application of the copy-operation).

By these means, an iterative process searching from the root for the left-most redex at each stage is avoided. In effect, the recursive nature of control utilises the

knowledge of where each (left-most) redex was located to guide its search for the next redex.

The copy-operation can also be programmed as a recursive operation, starting from the λ-node, F, representing the rator of the redex, with two precautions:

1. The structure must be copied as a graph, not as a tree.
2. Nodes abstractable from F are not copied.

To treat precaution 1., copied nodes must be "tagged" in some way to prevent their being re-copied. Either a different tag (e.g. a unique integer) should be used for the copy-operation at different stages, or, if the same one is used each time, it must be removed at the end of the operation. For the duration of each copy-operation, some indication of the correspondence between nodes and their copies must be kept, in order to deal correctly with any further references to nodes already copied.

To treat precaution 2., we saw earlier that only a test for directly abstractable sub-expressions is needed. As yet, however, there does not seem to be an easy way to implement this ; though decidable, the direct test of the implication (11) at the end of §4.3 might require considerable effort. Clearly, this is one area awaiting innovative suggestions. One possibility is that there will be a simple decision procedure using connectivity matrices, or some other graph-theoretic concept, though the fluctuating size of the present graphs might present some administrative difficulties with such matrices.

§4.6 Relation to function evaluation strategies

Current programming languages allow the parameters of a procedure to be passed in three ways — by *value* and by *name*, e.g. as in ALGOL 60 [23], and by *reference*, e.g. as in CPL [13].

For the λ-calculus, call-by-reference is superfluous, since the λ-calculus is a language without assignment. Call-by-value corresponds to applicative-order reduction, as mentioned in §4.1. Call-by-name similarly corresponds to normal-order reduction ; indeed, the ALGOL report specifies the effect of call-by-name as one of literal substitution, though most implementations achieve this effect via calls of a *subroutine* generated from the actual parameter (Randell and Russell [29]).

As with the corresponding methods of reducing wfes, call-by-name is often expensive in time and storage, while call-by-value may involve inessential, non-terminating computations. Graph reduction illustrates an intermediate approach which overcomes these defects. As a variant of call-by-name, evaluation of an expression (argument) represented in the graph has the desirable side-effect that *all* references to it are replaced by their value, thus avoiding repeated evaluations of the same expression. Effectively, the call-by-value mechanism is *set up* at call-time, but its application is delayed until execution reaches the *first* point at which the value is actually required.

The latter view prompts the author to refer to this method of passing parameters as *call-by-need*. Techniques

implementing call-by-name and call-by-value are readily adaptable for the evaluation of parameters called by need, as follows:

1. Associate the formal parameter with a new location ℓ, e.g. the one which would be allocated for call-by-value.
2. Let R be the subroutine which would be generated for call-by-name.
3. Assign to ℓ a subroutine, S, which, *if* and when called, in turn calls the subroutine R ; if and when R terminates, then
   (i) Let v denote the result given by R.
   (ii) *Overwrite* ℓ with a constant subroutine, S', which delivers the value v but does nothing else.
   (iii) Return v as the result of S.
4. Treat occurrences of the formal in the procedure body as calls for the subroutine currently accessed via location ℓ.

It is as a consequence of action 3(ii) that second and later (in the *dynamic* sense) references to ℓ yield immediate access to the value v, without re-invoking the subroutine S.

How useful this method of passing parameters will prove remains to be seen. With 'real' programming languages, judgement is clouded by the possibility of side-effects which is sometimes the whole purpose in calling parameters by name. Some convincing examples

where call-by-need is desirable might make its inclusion *in the language* worthwhile, however, as an aid for the programmer; for although call-by-need *could* be programmed around in many languages, it is much more satisfactory if the details are left to the compiler and/or run-time system.

As an vehicle for an investigation of call-by-need, the design of *interpreters* for the λ-calculus operating according to this strategy is one area for further research. The discovery of a 'fixed-program' machine for graph reduction will, we feel, alleviate its one remaining unsavoury feature — the copy-operation — by manipulating suitable run-time structures.

At the level of substitutionsmechanisms for the λ-calculus, the *basic lambda-calculus machine* (BLCM) of Wegner [26,§2.8.2] also used pointers, but only as a device to simulate literal substitution of arguments for variables. When a redex $(\lambda x.M)A$ occurs at the 'top' of the BLCM workstack,

(a) the part λx is removed from the stack, and

(b) free occurrences of x in M are replaced by a 'pointer' to the beginning of A.

When a pointer occurs at the top of the workstack, it is replaced by a copy of the expression, E say, to which it points. Graph reduction suggests it would be advantageous, and would not affect the question of termination, to operate on the expression E *before* it is copied. Clearly, the whole mechanism then becomes recursive, but the technical modifications should not pose any problems.

A class of finite directed graphs for representing wfes has been considered, and general conditions isolated under which operations on graphs perform reductions of wfes.

As the basis for reduction algorithms, graphs provide a more compact representation than trees or symbol strings. A subgraph represents a *set* of sub-expressions, in consequence of which a single contraction-operation on a graph accomplishes simultaneous contraction of a set of redexes in its linear expansion.

Naming conflicts have been prevented by "canonicalizing" the usage of bound variables. Replacing bound variables by pointers to (the node representing) their binding λ, graph reduction operates internally without reference to the names of variables and, hence, without the need for α-conversion.

This treatment of bound variables can equally be applied to the linear presentation of wfes, though it seems much more appropriate in the case of graphs. The idea is not new, and is but one solution to what is largely a technical problem. In effect, the degree of redundancy present in the bound-variable-notation has been removed. For, in writing $(\lambda x.M)$, choice of the specific name, x, is unimportant ; what one wishes to convey is that certain sub-expressions of M — those which happen to be denoted as the free occurrences of x in M — are tied to the binding λ in $(\lambda x.M)$ in a certain way. Pointers provide a "variable-free" notation for the

same concept. Indeed, the use of pointers resolves naming conflicts by doing aways with names !

As an application of the general technique, the normal-order version, NGR, of graph reduction combines the desirable features of two conventional methods. NGR succeeds in finding a normal form whenever one exists (like normal-order reduction, NBR, on symbol strings), and does so without the duplication of effort entailed in NBR (like applicative-order reduction, AR).

We note, however, that NGR does not always achieve the shortest reduction possible. A simple example is the wfe

$$X = (\lambda x.xxx)(\lambda y.y(\underline{(\lambda z.z)b}))$$

for which NGR (like NBR in this case) takes 4 contractions to reach its normal form

$$N = ab(\lambda y.yb)$$

whereas there is a reduction of X to N in 3 steps (by contracting the underlined reddex first). Thus, are there other orders of reduction which further improve the performance of graph reduction ? Is there even a *minimal* reduction algorithm, i.e. one for which there is never a shorter reduction to normal form ?

In the meantime, a statistical comparison of NGR and NBR would be a useful exercise. The present 4.4.4 shows that all psirs of relative lengths (with NGR the shorter) are *possible*, but just how much more efficient is NGR on the average ? The main difficulty here is that the criteria for selecting a fair sample are not at all clear.

Judged in the terms of the introduction to this chapter, graph reduction extends the range of simulated substitution mechanisms. But the technique has enough novelty to merit a search for equivalent interpreters/ fixed-program-machines.

APPENDIX

Böhm's Theorem and its extension

This appendix presents an account of 2.4.3 from which 3.1.5 will be deduced by a simple amendment. The treatment broadly follows that of Böhm, though a few ideas have been borrowed from an alternative proof by Curry to be published in his Combinatory Logic, Volume 2. Particular attention will be given to one or two parts which the author feels were not fully covered by Böhm's proof.

To simplify the considerable notational difficulties as much as possible, a few abbreviations have been introduced. Phrases like "for $j=1\cdots n$" have been omitted when it is clear what the range of the index is. Capital letters stand for wfes throughout, small t,u,v,w,x,y,z for variables. The notation $M[y_1,y_2,\cdots]$ indicates that $y_1,y_2,\cdots$ may occur free in M; $M[G_1,G_2,\cdots]$ then denotes the result of substituting the G_k for all free occurrences of the y_k in M.

In this appendix, we revert to conventional usage of '=' for cnv and '≡' for syntactic identity.

The proof utilizes many times the property that K^aK^b selects the (the1)-th. argument out of $(a+b+1)$ arguments:

(1) $\quad K^aK^bX_1X_2\cdots X_aYZ_1Z_2\cdots Z_b = Y$, for all $a,b\geq 0$,

where $K^0\equiv I$. Two properties of rank, defined in §1.1.1, are

used:- substituting one variable for another leaves the rank unchanged, and *proper* sub-expressions have smaller rank than the whole wfe.

Let N denote the set of all wfes in β-η-normal form. Each $M \in N$ is of the form (c.f. §1.1.2):

(2) $$M \equiv \lambda x_1 x_2 \cdots x_n . s X_1 X_2 \cdots X_m \quad , n,m \geq 0,$$

where each $X_k \in N$, and n,m,s are called the *order*, *grade*, and *principal variable* (p.v., for short), respectively of M.

Proof of 2.4.3

Let $M_0, M_1 \in N$ with $M_0 \not\equiv M_1$. The letter i will be used throughout to index the range 0,1. In the form of (2), denote

(3) $$M_i \equiv \lambda x_1 \cdots x_{n_i} . Z_i \quad , \text{ where } Z_i \equiv s_i X_{i1} X_{i2} \cdots X_{im_i} .$$

Let $k = \mathrm{Max}(n_0, n_1)$. Given any $j \geq k$, the pair (N_0, N_1) defined by

(4) $$N_i \equiv M_i x_1 x_2 \cdots x_j \equiv Z_i x_{n_i+1} \cdots x_j$$

is uniquely determined. Clearly, each $N_i \in N$ and(*)

(5) $$M_0 = M_1 \iff N_0 = N_1 .$$

Using (3), write (4) more uniformly as

(6) $$N_i \equiv s_i X_{i1} \cdots X_{ij} \cdots X_{ig_i} \quad ,$$

where

(7) $$g_i = m_i + j - n_i$$

(8) $$X_{i(m_i+j)} \equiv x_{n_i+j} \quad , \text{ for } j = 1,2,\cdots,j-n_i .$$

(*)(Curry) That M_0, M_1 are in β-η-normal form is needed here. For if $M_0 \equiv y$ and $M_1 \equiv (\lambda x.yx)$, then $M_0 \equiv yx \equiv M_1$.

X_{ij} will be called the *j*th.*argument* in M_i.

Define the following *equivalence relation* on N:

(9) $$M_0 \sim M_1 \iff n_0=n_1 \cdot n_0-m_0=n_1-m_1 \quad .$$

Applying this definition to M_i in (8) and using (7),

(10) $$M_0 \sim M_1 \iff M_0 \sim M_1 \iff n_0=n_1 \cdot g_0=g_1 \quad .$$

Lemma A1. 2.4.3 holds when $M_0 \neq M_1$.

Proof. From (10), it suffices to consider two cases:

Case 1. $n_0 \neq n_1$.

Choose $f=\bar{n}$ and consider the wfes $M_i v_0 v_1[n_0,n_1]$. Since $n_0 \neq n_1$, different selectors can be substituted for them which will pick out v_0 and v_1, respectively, from among the arguments in $M_i v_0 v_1$. Thus, if

(11) $$U_0 = K^{g_0} X \quad , \quad U_1 = K^{1+g_1} Y \quad ,$$

then, from (8) and (1),

(12) $$M_i v_0 v_1[U_0,U_1] = v_i \quad .$$

For the M_i, there are four sub-cases according as each n_i is free or bound in M_i. We shall treat the "mixed" case ; the others are similar. Suppose n_0 is free in M_0, but n_1 is bound in M_1, i.e. $n_1=n_k$ for some k. Then, (12)' can be re-written as

(13) $$([U_0/n_0]M_i)n_1 \cdots n_{k-1}(U_1)n_{k+1} \cdots n_n v_0 v_1 = v_i \quad .$$

Case 2. $n_0 = n_1 \cdot g_0 \neq g_1$.

For definiteness, suppose $g_0 > g_1$. Let $p = g_0-g_1$, and choose $f = 1+\bar{n}+p$. Then, from (8), the last $(p+1)$

arguments in each H_i are the variables $x_{1+n}, \cdots, x_j$.

Consider the g_1-th. argument position. In H_1, we have simply $X_{0g_1} = x_j$. In H_0, since $g_1 = g_0 - p$, the g_1-th. position is p places to the left of $X_{0g_0} = x_j$; hence, $X_{0g_1} = x_{j-p} = x_{1+n}$. Thus, H_0 and H_1 differ in their g_1-th. argument. We therefore select this argument by substituting $U = K^{g_1-1}I$ for the p.v. x ($= x_0 = x_1$), and for x_{1+n} and x_j we substitute selectors to yield the variables v_0 and v_1. Specifically, consider $H_i[x_{1+n}, x_j, x]$. Then

(14) $$H_i[K^p v_0, v_1, U] = v_i \quad .$$

The K^p is needed here to cancel the p arguments following the g_1-th. in H_0.

For the H_i, the result follows analogously to the manner in which (13) followed from (12), there being two sub-cases this time, viz. x free or bound in H_i. ∎

We now have to treat the case $H_0 \neq H_1$, $H_0 \sim H_1$, for which the H_i in (6) have the same p.v. $x = x_0 = x_1$ and the same number $g = g_0 = g_1$ of arguments. First, two preliminary lemmas:

Lemma A2. For $j = 1, 2, \cdots, g$,

(15) $$rank(X_{ij}) \leq rank(H_i) \quad ,$$

with equality iff $m_i = 0 = n_i$, in which case $rank(H_i) = 0$.

Proof. For $j \leq m_i$, X_{ij} is a proper sub-expression of H_i, except when $m_i = 0$ (in which case there is nothing to prove). For $j > m_i$, the X_{ij} are variables (by (8)) so have rank 0. Finally, if $n_i = 0 = m_i$, then $H_i = x_i$, so $rank(H_i) = 0$. ∎

Lemma A3.

If $H_0 \not\sim H_1$ but $H_0 \sim H_1$, then, setting $f=\bar{\pi}$ in (6),

(16) $$X_{0j} \not\sim X_{1j}$$

for some j in the range $i \leq j \leq g$.

Proof. Otherwise $H_0 = H_1$, in contradiction of (5).

From these two lemmas, when $H_0 \not\sim H_1$ but $H_0 \sim H_1$, we can find corresponding X_{ij} such that (15) and (16) hold. In fact, for at least one of the X_{ij}, inequality must hold in (15); otherwise, each $H_i = s_i$ and then $H_0 \not\sim H_1$ would imply $H_0 \not\sim H_1$. Thus the *sum* of the ranks of the X_{ij} is less than the *sum* of the ranks of the H_i.

For this pair, either $X_{0j} \not\sim X_{1j}$ or $X_{0j} \sim X_{1j}$. In the latter case, the whole procedure can be repeated for the X_{ij}. In this way we shall construct (Lemma A4) a *chain* of pairs of wfes, starting with (H_0, H_1), such that the members of each pair are —equivalent, except the last pair, where the observation about the sum of the ranks will be invoked to infer the finiteness of the length of the chain.

The use of this chain to prove 2.4.3 then divides into two cases, according as the p.v.s of the chain are pairwise distinct or not. If so, there is freedom to make independent substitutions for them, which will enable us (Lemma A5) to "unwind" the chain, eventually producing a pair for which $\not\sim$ holds. If not, we show (Lemma A6) how it is still possible to make substitutions in such a way that the first chain is transformed into a second chain in which the p.v.s are pairwise distinct, and distinct from the p.v.s of the first chain.

The chain construction

We abstract on the notation so far by introducing three operators M, J, X, the first defined on all pairs (H_0, H_1), the other two defined on all pairs for which $H_0 \neq H_1$, $H_0 \sim H_1$.

(17) $M(H_0, H_1)$ = the pair of the form (6) obtained from (H_0, H_1) with the choice $f = \overline{1}$.

(18) $J(H_0, H_1)$ = the least j determined by Lemma A3.

(19) $X(H_0, H_1)$ = the jth. argument pair in $M(H_0, H_1)$, for $j = J(H_0, H_1)$.

We shall now abbreviate the two arguments H_0, H_1 of these three operations using the index i, e.g. $J(H_i)$ for $J(H_0, H_1)$. Also, we shall write, e.g., $H_i = M(H_i)$ for $(H_0, H_1) = M(H_0, H_1)$.

Notice that M is *idempotent* in the sense

(20) $$M(M(H_i)) = M(H_i) \quad .$$

Thus, replacing (H_0, H_1) by $M(H_i)$ in any of (17)-(19) yields the same result.

Lemma A4.:

For any pair (H_0, H_1) for which $H_0 \neq H_1$, there exists an integer σ, $0 \leq \sigma < \omega$, and a chain

$$(H_0^{(k)}, H_1^{(k)}) \quad , \; k = 0, 1, \cdots, \sigma,$$

of pairs of wfes, all of the form (6), such that

(21) $$H_0^{(k)} = H_1^{(k)} \quad , \; k = 0, 1, \cdots, \sigma - 1 \; ,$$

$$H_0^{(\sigma)} \neq H_1^{(\sigma)} \quad .$$

Proof. To construct the $M_i^{(k)}$, the subsidiary $N_i^{(k)}$ will also be defined. First set

$$N_i^{(0)} = M_i \quad , \quad M_i^{(0)} = M(M_i). \tag{22}$$

For $k \geq 0$, if $M_0^{(k)} \not\sim M_1^{(k)}$, set $s = k$; otherwise set

$$N_i^{(k+1)} = X(N_i^{(k)}) \quad , \quad M_i^{(k+1)} = M(N_i^{(k)}) \quad . \tag{23}$$

By straightforward induction on k,

$$N_0^{(k)} \sim N_1^{(k)} \iff M_0^{(k)} \sim M_1^{(k)} \tag{24}$$

That s is finite is immediate from Lemma A2 and the definitions of X and M, by considering the sum of the ranks of successive pairs in these chains. (21) follows from the inductive definitions of the $M_i^{(k)}$. □

Some caution is required before we can say precisely whether the p.v.s of the chain are pairwise distinct or not. This concerns the bound variables in each $N_i^{(k)}$. In the form of (8), for $k = 0, 1, \cdots, s$, denote

$$N_i^{(k)} = \lambda x_1 x_2 \cdots x_{n_i^{(k)}} \cdot S_i^{(k)} \quad , \tag{25}$$

and let

$$n_k = \text{the larger of the } n_i^{(k)} \quad , \tag{26}$$

$$r_k = n_0 + n_1 + \cdots + n_k = r_{k-1} + n_k \quad . \tag{27}$$

To keep the notation systematic, it is necessary to change the bound variables of $N_i^{(k)}$ so that they differ from those of other pairs in the chain. For this purpose, let

$$y_1, y_2, \cdots, y_p \quad , \text{ where } p = r_s \quad , \tag{28}$$

be a list of pairwise distinct variables not occurring anywhere in the chain (possible since the chain is finite).

Now, for $q=1,2,\dots,n_i^{(k)}$, perform α-conversions

$$y_{q+r_{k-1}} / s_q$$

in (25). The (leading) bound variables of all the $H_i^{(k)}$ will then be pairwise distinct.

In fact, we can suppose that all these α-conversions have been performed in the initial pair (H_0,H_1). For, from the manner in which the chain was constructed, c.f. (8) and Lemma A3, it can be seen that each $H_i^{(k)}$ is either a variable standing alone or is a part of the initial H_i.

After these changes have been made, the relationship between the $E_i^{(k)}$ and the $H_i^{(k)}$ can be expressed as

(29) $$E_i^{(k)} y_{1+r_{k-1}} \cdots y_{r_k} = H_i^{(k)} \quad .$$

For $k=0,1,\dots,e-1$, since $H_0^{(k)} = H_1^{(k)}$, we can unambiguously denote

(30) $$s_k = \text{the p.v. of } H_i^{(k)}$$

(31) $$\sigma_k = \text{the grade of } H_i^{(k)} \quad .$$

Then, each $H_i^{(k)}$ can be written in the form

(32) $$H_i^{(k)} = s_k x_{i1}^{(k)} \cdots x_{ij}^{(k)} \cdots x_{i\sigma_k}^{(k)} \quad ,$$

and

(33) *either* s_k is free in the initial H_0 and H_1,
or $s_k = y_t$, for some $t \le r_k$.

For $k=e$, the form of $H_i^{(k)}$ is

(34) $$H_i^{(e)} = s_i^{(e)} x_{i1}^{(e)} \cdots x_{ij}^{(e)} \cdots x_{iu_i^{(e)}}^{(e)} \quad ,$$

and

(35) *either* $s_i^{(e)}$ is free in H_i,
or $s_i^{(e)} = y_t$, for some $t \le r_e$.

Lemma A5. If the p.v.s

$$z_0, z_1, \cdots, z_{s-1}, z_0^{(s)}, z_1^{(s)}$$

of the chain constructed from (N_0, N_1) by Lemma A4 are pairwise distinct (except possibly $z_0^{(s)} = z_1^{(s)}$)(*), then 2.4.3 holds for (N_0, N_1).

Proof. The pairwise distinctness of the p.v.s is stated as

(36) $$q \neq l \Rightarrow z_q \neq z_l$$

(37) $$z_k \neq z_l^{(s)}$$

For $k = 0, 1, \cdots, s-1$, let j_k be the index of the argument position of $N_l^{(k+1)}$ in $N_l^{(k)}$!

(38) $$j_k = J(N_l^{(k)}) = J(N_l^{(k)})$$,

and let U_k be the combinator which will select this argument out of the $N_l^{(k)}$, discarding the rest of $N_l^{(k)}$:

(39) $$U_k = K^{j_k - 1} K S^{k - j_k}$$.

Consider all wffs now as they involve $z_0, \cdots, z_{s-1}$ as free variables, i.e. $N_l^{(k)}[z_0, \cdots, z_{s-1}]$. Let

(40) $$Q_l[z_0, \cdots, z_{s-1}] = N_l U_1 U_2 \cdots U_p$$

$$= N_l^{(0)} U_{1+r_0} \cdots U_p \quad , \text{ by (20)}$$

To establish the lemma, it suffices to show that

(41) $$Q_l[U_0, \cdots, U_{s-1}] = N_l^{(s)}[U_0, \cdots, U_{s-1}] = N_l \quad .$$

For then $R_0 \neq R_1$, since (by (37)) none of the substitutions U_k / z_k can alter either the p.v. $z_l^{(s)}$ or the grade $g_l^{(s)}$ of $N_l^{(s)}$ in (34). Lemma A1 can then be applied to (R_0, R_1) to

---

(*) Allowed since $N_0^{(s)} \neq N_1^{(s)}$ requires only one of $z_0^{(s)} \neq z_1^{(s)}$, $g_0^{(s)} \neq g_1^{(s)}$ to hold.

complete the present lemma.

To prove (41), for $k=0,1,\cdots,s$, let

$$L_i^{(k)} = Q_i[U_0,\cdots,U_{k-1},z_k,\cdots,z_{s-1}] \tag{42}$$

$$R_i^{(k)} = (M_i^{(k)}[U_0,\cdots,U_{k-1},z_k,\cdots,z_{s-1}])U_{1+r_k}\cdots U_p . \tag{43}$$

Equation (41) is just $L_i^{(s)} = R_i^{(s)}$. We shall show that $L_i^{(k)} = R_i^{(k)}$ for $k=0,1,\cdots,s$, by induction on k.

For $k=0$, this is just (40). As induction hypothesis, suppose the result is true for some $k<s$. Then we have

$$L_i^{(k+1)} = [U_k/z_k]L_i^{(k)} \quad \text{, by (42)}$$

$$= [U_k/z_k]R_i^{(k)} \quad \text{, by ind.hyp.}$$

$$= U_k Y_{i1}^{(k)}\cdots Y_{ij}^{(k)}\cdots Y_{ir_k}^{(k)} U_{1+r_k}\cdots U_p \quad \text{, by (43,32,36,33)}$$

where $Y_{ij}^{(k)} = [U_0,\cdots,U_k/z_0,\cdots,z_k]X_{ij}^{(k)}$
$= [U/z]X_{ij}^{(k)}$, for short

$$= Y_{ij_k}^{(k)} U_{1+r_k}\cdots U_p \quad \text{, by (39,1)}$$

$$= ([U/z]X_{ij_k}^{(k)})\, U_{1+r_k}\cdots U_p \quad \text{, by def.of } Y_{ij_k}^{(k)}$$

$$= ([U/z]M_i^{(k+1)})\, U_{1+r_k}\cdots U_p \quad \text{, by (38,28) and def.M}$$

$$= [U/z](M_i^{(k+1)} U_{1+r_k}\cdots U_p) \quad \text{, by (33)}$$

$$= [U/z](M_i^{(k+1)} U_{1+r_{k+1}}\cdots U_p) \quad \text{, by (29) with } k=k+1$$

$$= ([U/z]M_i^{(k+1)})\, U_{1+r_{k+1}}\cdots U_p \quad \text{, by (33)}$$

$$= R_i^{(k+1)}$$

■

It remains to consider the case in which the p.v.s of the chain are not pairwise distinct. When this occurs, the above proof may fail. For two different pairs, $M_i^{(k)}$ and $M_i^{(q)}$ with $k=q_0$, the selection of the next pairs from among their arguments may require different v_k and v_q. But if $s_k=s_q$ and $v_k \neq v_q$, this is impossible. Similarly, if $s_k=s_i^{(o)}$ for some k and i, the substitution v_k/s_k might conflict with the selectors (obtained from Lemma A1) to be substituted for the $s_i^{(o)}$ in the R_i given in (41).

The way round this difficulty involves use of the combinators

(44) $$C_k \equiv \lambda t_0 t_1 \cdots t_k . t_k t_0 t_1 \cdots t_{k-1} \quad .$$

Given $(k+1)$ arguments, C_k brings the last argument to the front. Thus, if one substitutes C_k/z_k in $M_i^{(k)}$ with large enough k, then all the arguments in $M_i^{(k)}$ will be "absorbed" inside the body of C_k, with at least one bound variable remaining at the front. By applying these to more variables, a new chain (of the same length) will be constructed in which the p.v.s are pairwise distinct.

Lemma A6

For the chain obtained from (M_0, M_1) by Lemma A4, there exist integers $p_0, p_1, \cdots, p_{s-1}, q_0, q_1$, such that, letting

(45) $$x \equiv z_0, z_1, \cdots, z_{s-1}, z_0^{(o)}, z_1^{(o)}$$

(46) $$C \equiv C_{p_0}, C_{p_1}, \cdots, C_{p_{s-1}}, C_{q_0}, C_{q_1}$$

(47) $$P_i^{(k)} \equiv N([C/x]M_i^{(k)}) \quad , \; k=0,1,\cdots,s,$$

then the $P_i^{(k)}$ form a chain in which the p.v.s are pairwise

distinct, and all differ from the p.v.s in the list z. Further,

(48) $$p_0^{(k)} \sim p_1^{(k)} \quad , \; k=0,1,\cdots,s-1 \; ,$$

(49) $$p_0^{(s)} \not\sim p_1^{(s)} \quad .$$

Proof. Since there may now be equals in the list z, there is some restriction on the choice of $p_0,p',\cdots,p_{s-1},q_0,q_1$; viz., if two variables in z are equal, then the corresponding integers in the list $p_0,\cdots,p_{s-1},q_0,q_1$ must be the same, otherwise there would be incompatibilities in the substitutions $C_{\frac{1}{2}}z$.

For $k=0,1,\cdots,s-1$, from (22):

(50) $$[C/z]N_t^{(k)} = C_{p_k} x_{i1}^{(k)} \cdots x_{ig_k}^{(k)} \quad , \text{ where } x_{ij}^{(k)} = [C/z]z_{ij}^{(k)}$$

$$= \lambda t_{g_k} \cdots t_{p_k} . t_{p_k} x_{i1}^{(k)} \cdots x_{ig_k}^{(k)} t_{g_k} \cdots t_{p_k-1}$$

provided $p_k \geq g_k$. For whatever integers will be later specified for the p_k, let e_k be the number of bound variables in (50):

(51) $$e_k = 1+p_k-g_k \quad , \; k=0,1,\cdots,s-1,$$

and let

(52) $$u_1^{(k)}, u_2^{(k)}, \cdots, u_{e_k}^{(k)} \quad , \; k=0,1,\cdots,s-1,$$

be variables not occurring in the earlier chain. Then

(53) $$p_t^{(k)} = ([C/z]N_t^{(k)})u_1^{(k)} \cdots u_{e_k}^{(k)} \quad , \text{ from (47)}$$

$$= u_{e_k}^{(k)} x_{i1}^{(k)} \cdots x_{ig_k}^{(k)} u_1^{(k)} \cdots u_{e_k-1}^{(k)} \quad , \text{ from (50)}$$

Hence, (48) holds provided $p_k \geq g_k$, and

(58) $$u_{\sigma_0}^{(0)}, u_{\sigma_1}^{(1)}, \cdots, u_{\sigma_{s-1}}^{(s-1)}$$

are the p.v.s of

$$p_t^{(0)}, p_t^{(1)}, \cdots, p_t^{(s-1)} .$$

Proof. For $k=s$, consider two cases (exhaustive since $u_0^{(s)} \neq u_1^{(s)}$):

Case 1. $g_0^{(s)} \neq g_1^{(s)}$.

Let $f_t = g_t^{(s)}$ to simplify the notation; then $f_0 \neq f_1$. (*) Similarly to (50), from (34) we have

(55) $$[C/z]N_t^{(s)} = \lambda \psi_{f_t} \cdots \psi_{q_t} \cdot \psi_{q_t} r_{t1}^{(s)} \cdots r_{tf_t}^{(s)} \psi_{f_t} \cdots \psi_{q_t-1} ,$$

provided $q_t \geq f_t$. Now let

(56) $$d_t = 1 + q_t - f_t ,$$

(57) $$d = \text{the larger of } d_0, d_1 ,$$

and let $w_2, \cdots, w_d$ be variables not occurring earlier. Then d is the larger of the orders of $[C/z]N_t^{(s)}$; hence, for $q_t \geq f_t$,

(60) $$p_t^{(s)} = ([C/z]N_t^{(s)}) w_1 w_2 \cdots w_d$$

$$= w_{d_t} r_{t1}^{(s)} \cdots r_{tf_t}^{(s)} w_1 \cdots w_{d_t-1} w_{d_t+1} \cdots w_d , \text{ by (56)}$$

If we now choose

(59) $$p_0 = p_1 = \cdots = p_{s-1} = q_0 = q_1 = h ,$$

where h = the largest of $g_0, \cdots, g_{s-1}, f_0, f_1$,

then $d_0 \neq d_1$, so (49) holds, and the p.v.s of the $p_t^{(k)}$-chain are those in the list (54) and w_{d_0}, w_{d_1}; hence they are pairwise distinct.

Case 2. $g_0^{(e)} = g_1^{(e)}$, $s_0^{(e)} \neq s_1^{(e)}$.

Let $g = g_0^{(e)} = g_1^{(e)}$. Then (55)-(58) all hold with g in place of f_i. We cannot choose integers as before since then we would have $d_0 = d_1$. The following choices suffice for this case:

(60) h = the largest of $g_0, \cdots, g_{e-1}, g$

(61) $q_0 = h$, $q_1 = h+1$

(62) $p_k = \begin{cases} h+1 & \text{, if } s_k = s_1^{(e)} \\ h & \text{, otherwise (includes case } s_k = s_0^{(e)}) \end{cases}$

Then, $d_0 = 1+h-g$ and $d_1 = 2+h-g$, so $d_0 \neq d_1$ and (49) again holds ; the p.v.s are pairwise distinct as before.

In both cases, the p.v.s of $P_i^{(k)}$ are clearly all different from those in the list z. The technique of Lemma A5 can now be applied to unwind this chain.

This completes proof of Lemma A6 and 2.4.3.

Proof of 3.1.5

Summarizing the proof of 2.4.3, we see that it is characterized by four parameters : the operators N, J, X which determine a step in the construction of the chain, and the relation $\sim$ which controls the construction. To prove 3.1.5 therefore, we show how these parameters can be extended to the case of a normal form and a non-normal form.

Let N and X be wfes with and without β-normal forms, respectively. Recalling the definition (9) of $\sim$, it can

be seen that it depends solely on the *first-level structure* (c.f. §3.1) of M_0 and M_1; that is, it does not depend on the X_{ij} in (3). Thus, $\sim$ can be extended to M and X, provided these first-level structures exist. For M_0 this must always be the case - take M_0 = the β-normal form(*) of M. For M_1 it will be the case iff X has a head normal form (h.n.f.); when this is so, take M_1 = the h.n.f. of M. We can suppose now that M_0, M_1 are expressed as in (3), the only exception to what held there being that the X_{ij} with $i=1$ are not necessarily in (or even have) a normal form. Define

$$(63) \qquad M \sim X \Leftrightarrow \begin{cases} false & \text{, if } X \text{ has no h.n.f} \\ M_0 \sim M_1 & \text{, if } X \text{ has a h.n.f., } M_1 \end{cases}$$

Now we define M', J', X' applicable to M, X when $M \sim X$. When this is the case, then $M_0 \sim M_1$, from (63). Form the N_i from these M_i as in (6); thus define

$$(64) \qquad M'(M,X) = M(M_0, M_1)$$

To define J' and X', notice that, since X has no normal form, its h.n.f., M_1, also cannot have a normal form. For $i=1$ in (6) therefore, there must be some value of j in the range $1 \leq j \leq q$ such that X_{1j} has no normal form (compare with Lemma A3). Hence we can define

$$(65) \qquad J'(M,X) = \text{the least } j \text{ such that } X_{1j} \text{ has no normal form}$$

$$(66) \qquad X'(M,X) = \text{the } j\text{th. argument of } M'(M,X), \text{ for } j = J'(M,X).$$

(*) Note we do not need to take β-η-normal forms here.

These three operators and the relation $\sim$ now determine a chain of pairs analogous to that in Lemma A4. However, there is an extra complication this time - it is possible that the construction will continue indefinitely. Indeed, it was when the author realised this possibility that he discovered the example of the pair $(I, I_{\beta}P)$ treated in §2.6 ; it is easy to verify that the construction does not terminate for this pair (Exercise).

The reason for this difficulty is that the sum-of-the-ranks argument may fail this time, simply because the successive ranks of the non-normal forms in the chain can be quite unrelated to each other. All is not lost however, as the first member of each pair in the chain $(\ )$ is in normal form. For $i=0$, Lemma A2 still applies, with strict inequality unless the normal form has rank 0. Thus, the rank of successive first members steadily decreases until it reaches 0. At this point we shall forcibly terminate the construction, by replacing the definition of $\sim$ in (63) by

$$(67) \qquad N \sim X \Leftrightarrow \begin{cases} \text{false} & \text{, if } X \text{ has no h.n.f.} \\ N_0 \sim N_1 & \text{, if } X \text{ has a h.n.f., } \operatorname{rank}(N_0)\neq 0. \\ \text{false} & \text{, if } X \text{ has a h.n.f., } \operatorname{rank}(N_0)=0. \end{cases}$$

The chain obtained with this definition of $\sim$ controlling the construction will always be finite in length. The proofs of Lemmas A5,A6 hold for these chains virtually unchanged (reference to 3.1.4 is needed at one or two points). To establish 3.1.5, all that remains to be proved is

Lemma A7. 3.1.5 holds when $N \neq X$.

Proof. Corresponding to the definition (67) of $=_\beta$ there are three cases to consider :

Case 1. X_β has no h.n.f.

Then, let N_0 be the β-normal form of N :

$$N_0 \equiv \lambda x_1 x_2 \cdots x_n . x X_1 X_2 \cdots X_m \quad , n,m \geq 0,$$

and let $U \equiv X^m v_0$. If x is free in N_0, then

(68) $([U/x]N_0)x_1 \cdots x_n \equiv X^m v_0 X_1 \cdots X_m \equiv v_0$,

and $([U/x]X)\, x_1 \cdots x_n$ has no h.n.f., by 3.1.4.

If x is bound in N_0, say $x \equiv x_k$, then

(69) $N_0 x_1 \cdots x_{k-1} U x_{k+1} \cdots x_n \equiv X^m v_0 X_1 \cdots X_m \equiv v_0$

and $X x_1 \cdots x_{k-1} U x_{k+1} \cdots x_n$ has no h.n.f., by 3.1.4.

N_0 can now be replaced by N in (68) and (69) since they are interconvertible.

Case 2. X has a h.n.f. and rank $(N_0) \neq 0$.

Since $N \neq X$ by supposition, we have $N_0 \neq N_1$. The result then follows analogously to Lemma A1.

Case 3. X has a h.n.f. and rank$(N_0) = 0$.

Since rank$(N_0)=0$, N_0 is a variable, say $N_0 \equiv z$. Let N_1 be the head normal form of X :

$$N_1 \equiv \lambda \phi_1 \phi_2 \cdots \phi_n . x X_1 X_2 \cdots X_m \quad , n,m \geq 0.$$

If $x \not\equiv z$ or $n \not\equiv m$ or x is bound in N_1, then $N_0 \not= N_1$, so proceed as in Lemma A1. Now suppose $x \equiv z$ and $n \equiv m$. Then, $n > 0$, otherwise x would be a normal form of X. Then,

$$([X^{n-1} v_0 / x]N_0)\phi_1 \cdots \phi_{n-1} \equiv X^{n-1} v_0 \phi_1 \cdots \phi_{n-1} \equiv v_0$$

$$([X^{n-1} v_0 / x]N_1)\phi_1 \cdots \phi_{n-1} \equiv \lambda \phi_n . X^{n-1} v_0 X_1 \cdots X_m \equiv \lambda \phi_n . x X_m$$

In all cases, the specification of a content C[] for 3.1.5 is now straightforward (c.f. proof of 2.4.4 from 2.4.3).

This completes proof of Lemma A7 and of 3.1.5.

Remarks

The proof of 3.1.5 is not effective in the sense of computability theory; it is undecidable whether X has a head normal form, and the function J' is not computable. Thus, 3.1.5 is only an existence theorem, whereas for 2.4.3 there is an effective method of finding appropriate combinators and wfes.

Apart from the extended result in 3.1.5, the main "new" features here are the *detailed* proofs of Lemmas A5,A6. Also, the first alternative in (62) corrects a slip in Böhm's original proof.

Curry's proof of 2.4.3 is considerably shorter but does not seem to generalize as readily for 3.1.5. The author spent some time trying to adapt Curry's method but found he needed one result he could not see how to prove:

(?) If X has no normal form and z is a variable occurring free in X, does there exist a sufficiently large integer, h, such that $[C_h/z]X$ has no normal form ?

If X has no head normal form, (?) is trivial (see 3.1.4); and if X has a normal form N, then $[C_h/z]X$ has a normal form at least for any h greater than rank(N).

Note that (?) was not needed in the present proof, since the construction of the chains involves substitutions

only of variables for variables, and the unwinding of the chains is independent of the normal/non-normal form question.

If (?) can be proved, the author can envisage a much shorter (because it is notationally simpler) "double-induction" proof of 3.1.5. On that optimistic note, we bring this somewhat protracted appendix to a close.

REFERENCES

[1] Church,A. (1941)

The Calculi of Lambda-Conversion, Annals of Mathematics Studies, No.6, Princeton University Press.

[2] Scott,D. (1970)

Lattice-theoretic models for the λ-calculus, Princeton, (unpublished).

[3] Morris,J.H. (1968)

Lambda-calculus models of programming languages, Ph.D. dissertation, M.I.T., Cambridge, Mass..

[4] Lucas,P. and Walk,K. (1969)

On the formal description of PL/I, in *Annual Review in Automatic Programming* 6,3.

[5] Walk,K., et al. (1968)

Abstract syntax and interpretation of PL/I, IBM Laboratory, Vienna, Tech.Report TR 25.082.

[6] McCarthy,J. (1963)

A basis for a mathematical theory of computation, in *Computer Programming and Formal Systems* (eds. Braffort,P. and Hirschberg,D.), Amsterdam : North-Holland.

[7] Landin,P.J. (1965)

A correspondence between ALGOL 60 and Church's lambda-notation, *Comm.A.C.M.* 8, 2 & 3, 89-101, 158-165.

[8] Scott,D. and Strachey,C. (1971)

Towards a mathematical semantics for computer languages, in *Proceedings of the Symposium on Computers and Automata*, Microwave Research Institute Symposium Series, 21, Polytechnic Institute of Brooklyn. (in press)

[9] Scott,D. (1970)

Outline of a mathematical theory of computation, Technical Monograph PRG-2, Programming Research Group, Oxford.

[10] Scott,D. (1970)

The lattice of flow diagrams, in *Symposium on Semantics of Algorithmic Languages* (ed.Engeler,E.), Springer-Verlag.

[11] Landin,P.J. (1966)

The next 700 programming languages, *Comm.A.C.M.* 9, 3, 157-164.

[12] Curry,H.B. and Feys,R. (1958)

Combinatory Logic 1, Amsterdam : North-Holland.

[13] Barron,D., et al. (1963)

The main features of CPL, *Computer J.* 6, 2, 134-143.

[14] Wozencraft,J.M. and Evans,A. (1969)

Notes on Programming Linguistics, M.I.T., Cambridge, Mass..

[15] Strachey,C. (1966)

Towards a formal semantics, in *Formal Language Description Languages for Computer Programming* (ed. Steel,T.B.), Amsterdam : North-Holland.

[16] Böhm,C. (1966)

The CUCH as a formal and description language, in *Formal Language Description Languages for Computer Programming* (ed,Steel,T.B.), Amsterdam: North-Holland.

[17] Landin,P.J. (1966)

A λ-calculus approach, in *Advances in Programming and Non-numerical Computation* (ed. Fox,L.), Oxford : Pergamon Press.

[18] Mendelson,E. (1964)

Introduction to Mathematical Logic, Princeton : Van Nostrand.

[19] Abbott,J.C. (1969)

Sets, Lattices, and Boolean Algebras, Boston : Allyn and Bacon.

[20] Böhm,C. (1968)

Alcune proprieta delle forme β-η-normali nel λ-K-calcolo, Consiglio Nazionale delle Ricerche, No.696, Rome. (in Italian)

[21] Park,D. (1970)

The Y-combinator in Scott's lambda-calculus models, Symposium on Theory of Programming, University of Warwick. (unpublished)

[22] Scott,D. (1969)

A type-theoretical alternative to CUCH,ISWIM, OWHY, Oxford. (unpublished)

[23] Naur,P., et al. (1963)

Revised report on the algorithmic language Algol 60, *Comm.A.C.M.* 6, 1, 1-23.

[24] Park,D. (1970)

Fixpoint induction and proofs of program properties, *Machine Intelligence 5*, 59-77, (eds. Michie,D. and Meltzer,B.),Edinburgh University Press.

[25] Strachey,C. (1970)

Loop theorem, Lecture notes, Programming Research Group, Oxford. (unpublished)

[26] Wegner,P. (1968)

Programming Languages, Information Structures, and Machine Organization, McGraw-Hill.

[27] McGowan,C.L. (1971)

Correctness results for lambda calculus interpreters, Ph.D. dissertation, Cornell.

[28] McCarthy,J. (1960)

Recursive functions of symbolic expressions and their computation by machine - Part I, *Comm.A.C.M.* 3, 4, 184-195.

[29] Randell,B. and Russell,L.J. (1964)

ALGOL 60 implementation, London : Academic Press.