# The $\lambda$-calculus

**Peter Hoffman**

This is quite a rich subject, but rather under-appreciated in many mathematical circles, apparently. It has 'ancient' connections to the foundations of both mathematics and also of computability, and more recent connections to functional programming languages and denotational semantics in computer science. This write-up is largely independent of the lengthier Computability for the Mathematical—[**CM**], within which it will also appear.

Do not misconstrue it as negative criticism, but I find some of the literature on the $\lambda$-calculus to be simultaneously quite stimulating and somewhat hard to read. Reasons for the latter might be:

(1) neuron overload caused by encyclopædiæ of closely related, but subtly different, definitions (needed for deeper knowledge of the subject perhaps, or maybe a result of the experts having difficulty agreeing on which concepts are central and which peripheral—or of me reading only out-of-date literature!);

(2) authors whose styles might be affected by a formalist philosophy of mathematics ('combinatory logicians'), while I am often trying to 'picture actual existing abstract objects', in my state of Platonist original sin; and

(3) writing for readers who are already thoroughly imbued (and so, familiar with the jargon and the various 'goes without saying's) either as professional universal algebraists/model theorists or as graduate students/professionals in theoretical computer science.

We'll continue to write for an audience assumed to be fairly talented upper year undergraduates specializing in mathematics. And so we expect a typical (perhaps unexamined) Platonist attitude, and comfortable knowledge of functions, equivalence relations, abstract symbols, etc., as well as, for example, using the '=' symbol between the names of objects only when wishing to assert that they are (names for) the same object.

The text [**HS**] is one of the definite exceptions to the 'hard for me to read' comment above. It is very clear everywhere. But it requires more familiarity with logic and model theory (at least the notation) than we need below. It uses model theoretic language in doing the material we cover, and tends to be more encyclopædic, though nothing like [**Ba**]. So the following 111 or so pages will hopefully serve a useful purpose, including re-expressing things such as combinatorial completeness and the 'equivalence' between combinators and $\lambda$-calculus in an alternative, more familiar language for some of us

beginners. Also it's nice not to need to be continually reminding readers that "=" is used for identicalness of objects, with the major exception of the objects most discussed (the terms in the $\lambda$-calculus), where a different symbol must be used, because "=" has been appropriated to denote every equivalence relation in sight! At risk of boring the experts who might stumble upon this paper, it will include all but the most straightforward of details, and plenty of the latter as well (and stick largely to the extensional case).

**Contents.**



## VII-1 The Formal System $\Lambda$ .

To get us off on the correct path, this section will introduce the $\lambda$-calculus strictly syntactically as a formal system. Motivation will be left for somewhat later than is customary.

**Definition of $\Lambda$** . This is defined inductively below to be a set of non-empty finite strings of symbols, these strings called *terms*. The allowable symbols are

$$\lambda \;||\; \bullet \;||\; ) \;||\; ( \;||\; x_0 \;||\; x_1 \;||\; x_2 \;||\; x_3 \cdots$$

All but the first four are called *variables*. The inductive definition gives $\Lambda$ as the smallest set of strings of symbols for which (i) and (ii) below hold:

(i) Each $x_i \in \Lambda$ (atomic terms).

(ii) If $A$ and $B$ are in $\Lambda$, and $x$ is a variable, then

$$(AB) \in \Lambda \qquad \text{and} \qquad (\lambda x \bullet A) \in \Lambda \; .$$

**Definition of free and bound occurrences of variables.**

Notice that the definition below is exactly parallel to the corresponding one in 1$^{\underline{\text{st}}}$order logic, **[LM], Section 5.1**. That is, '$\lambda x\bullet$' here behaves the same as the quantifiers '$\forall x$' and '$\exists x$' in logic.

(i) The occurrence of $x_i$ in the term $x_i$ is free.

(ii) The free occurrences of a variable $x$ in $(AB)$ are its free occurrences in $A$ plus its free occurrences in $B$.

(iii) There are no free occurrences of $x$ in $(\lambda x \bullet A)$ .

(iv) If $y$ is a variable different than $x$, then the free occurrences of $y$ in $(\lambda x \bullet A)$ are its free occurrences in $A$

(v) A *bound* occurrence is a non-free occurrence.

**Definition of $A^{[x\mapsto C]}$, the substitution of the term $C$ for the variable $x$ in the term $A$.**

In effect, the string $A^{[x\mapsto C]}$ is obtained from the string $A$ by replacing each free occurrence of $x$ in $A$ by the string $C$. To be able to work with this mathematically, it is best to have an analytic, inductive definition, as below. This has the incidental effect of making it manifest that, as long as $A$ and $C$ *are* terms, such a replacement produces a string which *is* a term.

(i) If $x = x_i$, then $x_i^{[x\mapsto C]} := C$ .

(ii) If $x \neq x_i$, then $x_i^{[x\mapsto C]} := x_i$ .

(iii) $(AB)^{[x\mapsto C]} := (A^{[x\mapsto C]}B^{[x\mapsto C]})$ .

(iv) $(\lambda x \bullet A)^{[x\mapsto C]} := (\lambda x \bullet A)$ .

(v) If $y \neq x$, then $(\lambda y \bullet A)^{[x\mapsto C]} := (\lambda y \bullet A^{[x\mapsto C]})$ .

But we really only bother with substitution when it is 'okay', as follows:

**Definition of "$A^{[x\mapsto C]}$ is okay".**

This is exactly parallel to the definition of substitutability in 1$^{\underline{\text{st}}}$order logic. You can give a perfectly tight inductive definition of this, consulting, if necessary, the proof in VI-3 that $SUB$ is recursive. But somewhat more informally, it's just that, when treated as an occurrence in $A^{[x\mapsto C]}$, no free occurrence of a variable in $C$ becomes a bound occurrence in any of the copies of $C$ substituted for $x$.

(In CS texts, it is often convenient to force every substitution to be okay. So they have a somewhat more complicated definition than (i) to (v) above,

from which 'okayness' [Or should it be 'okayity'?] follows automatically.)

**Bracket Removal Abbreviations.**

From now on, we usually do the following, to produce strings which are not in $\Lambda$, but which are taken to be names for strings which are in fact terms in the $\lambda$-calculus.

(i) Omit outside brackets on a non-atomic term.

(ii) $A_1A_2\cdots A_n \ := \ (A_1A_2\cdots A_{n-1})A_n$ (inductively), i.e.

$$ABC = (AB)C \ \ ; \ \ ABCD = ((AB)C)D \ \ ; \ \text{ etc.}$$

(iii) $\lambda x \bullet A_1A_2\cdots A_n \ := \ \lambda x \bullet (A_1A_2\cdots A_n)$ , so, for example,

$$\lambda x \bullet AB \ \neq \ (\lambda x \bullet A)B \ .$$

(iv) For any variables $y_i$,

$$\lambda y_1y_2\cdots y_n \bullet A \ := \ \lambda y_1 \bullet \lambda y_2 \bullet \cdots \lambda y_n \bullet A \ := \ \lambda y_1 \bullet (\lambda y_2 \bullet (\cdots (\lambda y_n \bullet A)\cdots))$$

The second equality in (iv) is the only possibility, so there's nothing to memorize there. The first equality isn't a bracket removal. Except for it (and so in the basic $\lambda$-calculus without abbreviations), it is evident that there is no need at all for the symbol '$\bullet$'. After all, except for some software engineers, who happily invent jargon very useful to them (but which confuses syntax with semantics), no logician sees the need to use $\forall x \bullet F$ in place of $\forall x F$. But the abbreviation given by the first equality in (iv) does force its use, since, for example,

$$\lambda xy \bullet x \ \neq \ \lambda x \bullet yx \ .$$

If we had dispensed with '$\bullet$' right from the beginning, the left-hand side in this display would be $\lambda x \lambda y x$, and would not be confused with its right-hand side, namely $\lambda xyx$ .

There is another statement needed about abbreviations, which 'goes without saying' in most of the sources I've read. But I'll say it:

(v) If $A$ is a term, and $S, T$ are (possibly empty) strings of symbols such that $SAT$ is a term, and if $R$ is an abbreviated string for $A$ as given by some of (ii) to (iv) just above, then $SRT := SAT$ ; that is, the abbreviations in (ii) to (iv) apply to *subterms*, not just to entire terms. But (i) doesn't of

course; one must restore the outside brackets on a term when it is used as a subterm of a larger term.

We should comment about the definition of the occurrences of subterms. The implicit definition above is perfectly good to begin with: a subterm occurrence is simply a connected substring inside a term, as long as the substring is also a term when considered on its own.

However, there is a much more useful inductive definition just below. Some very dry stuff, as in Appendix B of **[LM]**, shows that the two definitions agree. The inductive definition is:

(0) Every term is a subterm occurrence in itself.

(i) The atomic term $x_i$ has no proper subterm occurrences.

(ii) The proper subterm occurrences in $(AB)$ are all the subterm occurrences in $A$ together with all the subterm occurrences in $B$ .

(iii) The proper subterm occurrences in $(\lambda x \bullet A)$ are all the subterm occurrences in $A$ .

Finally we come to an *interesting* definition! But why this is so will remain a secret for a few more pages.

**Definition of the equivalence relation $\approx$ .**

This is defined to be the smallest equivalence relation on $\Lambda$ for which the following four conditions hold for all $A, B, A', B', x$ and $y$:

$(\alpha)$ : $\lambda x \bullet A \;\approx\; \lambda y \bullet A^{[x \to y]}$
if $A^{[x \to y]}$ is okay and $y$ has no free occurrence in $A$.

$(\beta)$ : $(\lambda x \bullet A)B \;\approx\; A^{[x \to B]}$ if $A^{[x \to B]}$ is okay.

$(\eta)$ : $\lambda x \bullet (Ax) \;\approx\; A$ if $A$ has no free occurrence of $x$ .

(Congruence Condition) :

$$A \approx A' \quad \text{and} \quad B \approx B' \quad \Longrightarrow \quad AB \;\approx\; A'B' \quad \text{and} \quad \lambda x \bullet A \;\approx\; \lambda x \bullet A' \; .$$

**Exercise.** Show that

$$\lambda x \bullet A \;\approx\; \lambda x \bullet A' \quad \Longrightarrow \quad A \approx A' \; .$$

**Remarks**. The names $(\alpha), (\beta)$ and $(\eta)$ have no significance as far as I know, but are traditional. The brackets on the left-hand side of $(\eta)$ are not

necessary. Most authors in this subject seem to use $A \equiv B$ for our $A = B$, and then use $A = B$ for our $A \approx B$ . I strongly prefer to reserve '=' to mean 'is the same string as', or more generally, 'is the same thing as', for things from Plato's world. Think about the philosophy, not the notation, but do get the notation clear.

And most importantly, this definition more specifically says that *two terms are related by '$\approx$' if and only if* there is a finite sequence of terms, starting with one and ending with the other, such that each step in the sequence (that is, successive terms) comes from $(\alpha), (\beta)$ or $(\eta)$ being applied to a subterm of some term, where those three 'rules' may be applied in either direction.

The last remark contains a principle, which again seems to go without saying in most expositions of the $\lambda$-calculus:

**Replacement Theorem VII-1.1.** *If $A, B$ are terms, and $S, T$ are (possibly empty) strings of symbols such that $SAT$ is a term, then*
*(i) $SBT$ is also a term;*
*(ii) if $A \approx B$ then $SAT \approx SBT$ .*

This is not completely trivial, but is very easy by induction on the term $SAT$. See Theorem 2.1/2.1$^*$ in **[LM]** for an analogue.

Here are a few elementary results.

**Proposition VII-1.2.** *For all terms $E$ and $E_i$ for $1 \leq i \leq n$ such that no variable occurs in two of $E, E_1, \cdots E_n$, we have*

$$(\lambda y_1 y_2 \cdots y_n \bullet E) E_1 E_2 \cdots E_n \;=\; E^{[y_1 \to E_1][y_2 \to E_2] \cdots [y_n \to E_n]} \; .$$

**Proof.** Proceed by induction on $n$, using $(\beta)$ liberally. (The condition on non-common variable occurrences is stronger than really needed.)

**Proposition VII-1.3.** *If $A$ and $B$ have no free occurrences of $x$, and if $Ax \approx Bx$, then $A \approx B$ .*

**Proof.** Using the 'rule' $(\eta)$ for the first and last steps, and the replacement theorem (or one of the congruence conditions) for the middle step,

$$A \;\approx\; \lambda x \bullet Ax \;\approx\; \lambda x \bullet Bx \;\approx\; B \; .$$

**Exercise.** Show that dropping the assumption $(\eta)$, but assuming instead the statement of **VII-1.3** , the resulting equivalence relation agrees with $\approx$ .

Now we want to consider the process of moving from a left-hand side for one of $(\alpha), (\beta)$ or $(\eta)$ to the corresponding right-hand side, but applied to a subterm, often proper. (However, for rule $(\alpha)$, the distinction between left and right is irrelevant.) Such a step is called a reduction.

If one such step gets from term $C$ to term $D$, say that $C$ reduces to $D$.

Now, for terms $A$ and $B$, say that $A \overline{\succeq} B$ if and only if there is a finite sequence, starting with $A$ and ending with $B$, such that each term in the sequence (except the last) reduces to the next term in the sequence.

Of course, $A \overline{\succeq} B$ implies that $A \approx B$ but the converse is false.

To show that not all terms are equivalent in the $\approx$–sense, one seems to need a rather non-trivial result, namely the following theorem.

**Theorem VII-1.4.**(Church-Rosser) *For all terms $A, B$ and $C$, if $A \overline{\succeq} B$ and $A \overline{\succeq} C$, then there is a term $D$ such that $B \overline{\succeq} D$ and $C \overline{\succeq} D$.*

We shall postpone the proof (maybe forever); readable ones are given in [**Kl**], and in [**HS**], Appendix 1.

Say that a term $B$ is normal or in normal form if and only if no sequence of reductions starting from $B$ has any step which is an application of $(\beta)$ or $(\eta)$ (possibly to a proper subterm). An equivalent statement is that $B$ contains no subterm of the form of the left-hand sides of $(\beta)$ or $(\eta)$, i.e.

$(\lambda x \bullet A)D$ ,

or

$\lambda x \bullet (Ax)$ if $A$ has no free occurrence of $x$.

Note that we don't require "$A^{[x \to D]}$ is okay" in the first one. Effectively, we're saying that neither reductions $(\beta)$ nor $(\eta)$ can ever be applied to the result of making changes of bound variables in $B$.

The claims just above and below do constitute little propositions, needed in a few places below, particularly establishing the following important corollary of the Church-Rosser theorem :

*For all $A$, if $A \overline{\succeq} B$ and $A \overline{\succeq} C$ where $B$ and $C$ are both normal, then $B$ and $C$ can be obtained from each other by applications of rule $(\alpha)$, that is, by a change of bound variables.*

So a given term has at most one normal form, up to renaming bound variables.

In particular, no two individual variables, regarded as terms, are related by $\approx$, so there are many distinct equivalence classes. As terms, variables are normal, and are not related by '$\approx$' to any other normal term, since there aren't any bound variables to rename ! But also, of course, you can

find infinitely many closed terms (ones without any free variables) which are normal, no two of which are related by '$\approx$'.

Note also that *if $B$ is normal and $B \overline{\succeq} C$, then $C \overline{\succeq} B$ .*
But a term $B$ with this property is not necessarily normal, as the example just below shows.

**VII-2 Examples and Calculations in $\Lambda$ .**

First, here is an example of a term which has no normal form:

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

Note that reduction $(\beta)$ applies to it, but it just reproduces itself. To prove rigorously that this has no normal form, one first argues that, in a sequence of single applications of the three types of reduction, starting with the term above, every term has the form

$$(\lambda y \bullet yy)(\lambda z \bullet zz) \ ,$$

for some variables $y$ and $z$. Then use the fact (not proved here) that, if $A$ has a normal form, then there is a finite sequence of reductions, starting from $A$ and ending with the normal form.

Though trying to avoid for the moment any motivational remarks which might take away from the mechanical formalistic attitude towards the elements of $\Lambda$ (not $\Lambda$ itself) which I am temporarily attempting to cultivate, it is impossible to resist remarking that the existence of terms with no normal form will later be seen to be an analogue of the existence of pairs, (algorithm, input), which do an infinite loop!

Another instructive example is, where $x$ and $y$ are different variables,

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

For this one, there is an infinite sequence of reductions, leading nowhere, so to speak. Just keep working inside the brackets ending at the far right over-and-over, as with the previous example. On the other hand, applying rule $(\beta)$ once to the leftmost $\lambda$, we just get $y$, the normal form. So not every sequence of reductions leads to the normal form, despite one existing. This example also illustrates the fact, refining the one mentioned above, that by always reducing with respect to the leftmost $\lambda$ for which a $(\beta)$ or $(\eta)$-reduction

exists (possibly after an ($\alpha$)-'reduction' is applied to change bound variables, and doing the ($\beta$)-reduction when there is a choice between ($\beta$)and ($\eta$) for the leftmost $\lambda$), we get an algorithm which produces the normal form, if one exists for the start-term. It turns out that having a normal form is undecidable, though it is semi-decidable as the 'leftmost algorithm' just above shows.

Now comes a long list of specific elements of $\Lambda$ . Actually they are mostly only well defined up to reductions using only ($\alpha$), that is, up to change of bound variables. It is important only that they be well defined as equivalence classes under $\approx$ . We give them as *closed* terms, that is, terms with no free variables, but leave somewhat vague which particular bound variables are to be used. Also we write them as normal forms, all except $\underline{Y}$ . For example,

**Definitions of $\underline{T}$ and $\underline{F}$ .**

$$\underline{T} \;:=\; \lambda xy \bullet x \;:=\; (\lambda x \bullet (\lambda y \bullet x)) \quad ; \quad \underline{F} \;:=\; \lambda xy \bullet y \;:=\; (\lambda x \bullet (\lambda y \bullet y)) \;,$$

where $x$ and $y$ are a pair of distinct variables. The second equality each time is just to remind you of the abbreviations.

Since terms obtained by altering $x$ and $y$ are evidently equivalent to those above, using ($\alpha$), the classes of $\underline{T}$ and $\underline{F}$ under $\approx$ are independent of choice of $x$ and $y$. Since all the propositions below concerning $\underline{T}$ and $\underline{F}$ are 'equations' using '$\approx$', not '=', the choices above for $x$ and $y$ are irrelevant.

We shall continue for this one section to work in an unmotivated way, just grinding away in the formal system, by which I mean roughly the processing of strings by making reductions, as defined a few paragraphs above. But the notation for some of the elements defined below is quite suggestive as to what is going on. As Penrose [**Pe**], p. XXXX has said, some of this, as we get towards the end of this section, is "magical"... and ... "astonishing"! See also the exercise in the next subsection.

**VII-2.1** *For all terms $B$ and $C$, we have*
(a) $\underline{T}BC \approx B$ ; and (b) $\underline{F}BC \approx C$ .

**Proofs.** For the latter, note that

$$\underline{F}B = (\lambda x \bullet (\lambda y \bullet y))B \approx (\lambda y \bullet y)^{[x \mapsto B]} = \lambda y \bullet y \;.$$

Thus

$$\underline{F}BC = (\underline{F}B)C \approx (\lambda y \bullet y)C \approx y^{[y \mapsto C]} = C \;.$$

For **VII-2.1**(a), choose some variable $z$ not occurring in $B$, and also different from $x$. Then

$$\underline{T}B = (\lambda x \bullet (\lambda y \bullet x))B \approx (\lambda x \bullet (\lambda z \bullet x))B \approx (\lambda z \bullet x)^{[x \mapsto B]} = \lambda z \bullet B \ .$$

Thus

$$\underline{T}BC = (\underline{T}B)C \approx (\lambda z \bullet B)C \approx B^{[z \mapsto C]} = B \ .$$

**Definition of $\underline{\neg}$ .** Now define

$$\underline{\neg} \ := \ \lambda z \bullet z \ \underline{F} \ \underline{T} \ = \ \lambda z \bullet ((z \ \underline{F}) \ \underline{T}) \ \neq \ (\lambda z \bullet z) \ \underline{F} \ \underline{T} \ ,$$

for some variable $z$. Once again, using rule $(\alpha)$, this is independent, up to $\approx$, of the choice of $z$.

**VII-2.2** *We have* $\underline{\neg} \ \underline{T} \approx \underline{F}$ *and* $\underline{\neg} \ \underline{F} \approx \underline{T}$ .

**Proof.** Using **VII-2.1**(a) for the last step,

$$\underline{\neg} \ \underline{T} = (\lambda z \bullet z \ \underline{F} \ \underline{T})\underline{T} \approx (z \ \underline{F} \ \underline{T})^{[z \mapsto \underline{T}]} = \underline{T} \ \underline{F} \ \underline{T} \approx \underline{F} \ .$$

Using **VII-2.1**(b) for the last step,

$$\underline{\neg} \ \underline{F} = (\lambda z \bullet z \ \underline{F} \ \underline{T})\underline{F} \approx (z \ \underline{F} \ \underline{T})^{[z \mapsto \underline{F}]} = \underline{F} \ \underline{F} \ \underline{T} \approx \underline{T} \ .$$

**Definitions of $\underline{\wedge}$ and $\underline{\vee}$ .** Let

$$\underline{\wedge} \ := \ \lambda xy \bullet (x \ y \ \underline{F}) \ \text{ and } \ \underline{\vee} \ := \ \lambda xy \bullet (x \ \underline{T} \ y) \ \ ,$$

for a pair of distinct variables $x$ and $y$.

**VII-2.3** *We have*

$$\underline{\wedge} \ \underline{T} \ \underline{T} \approx \underline{T} \ \ ; \ \ \underline{\wedge} \ \underline{T} \ \underline{F} \approx \underline{\wedge} \ \underline{F} \ \underline{T} \approx \underline{\wedge} \ \underline{F} \ \underline{F} \approx \underline{F} \ ,$$

*and*

$$\underline{\vee} \ \underline{T} \ \underline{T} \approx \underline{\vee} \ \underline{T} \ \underline{F} \approx \underline{\vee} \ \underline{F} \ \underline{T} \approx \underline{T} \ \ ; \ \ \underline{\vee} \ \underline{F} \ \underline{F} \approx \underline{F} \ .$$

**Proof.** This is a good exercise for the reader to start becoming a $\lambda$-phile.

**Definitions of $\underline{1st}$, $\underline{rst}$ and $[A,B]$ .**

$$\underline{1st} \;:=\; \lambda x \bullet x\; \underline{T} \;\;;\;\; \underline{rst} \;:=\; \lambda x \bullet x\; \underline{F} \;\;;\;\; [A,B] \;:=\; \lambda x \bullet xAB \;\;,$$

where $x$ is any variable for the first two definitions, but, for the latter definition, must not occur in $A$ or $B$, which are any terms.

Note that $[A,B]$ is very different than $(AB)$

**VII-2.4** *We have*

$$\underline{1st}[A,B] \approx A \quad \text{and} \quad \underline{rst}[A,B] \approx B \;.$$

**Proof.** Choosing $y$ not occurring in $A$ or $B$, and different than $x$,

$$\underline{1st}[A,B] = (\lambda x \bullet x\; \underline{T})(\lambda y \bullet yAB) \approx (x\; \underline{T})^{[x \mapsto (\lambda y \bullet yAB)]}$$

$$= (\lambda y \bullet yAB)\underline{T} \approx \underline{T}AB \approx A \;,$$

using **VII-2.1**(a). The other one is exactly parallel.

**Definitions of $\underline{ith}$ and $[A_1, A_2, \cdots, A_n]$ .**

Inductively, for $n \geq 3$, define

$$[A_1, A_2, \cdots, A_n] \;:=\; [A_1, [A_2, \cdots, A_n]] \;,$$

so $[A,B,C] = [A,[B,C]]$ and $[A,B,C,D] = [A,[B,[C,D]]]$, etc.

Thus, quoting **VII-2.4**,

$$\underline{1st}[A_1, A_2, \cdots, A_n] \approx A_1 \quad \text{and} \quad \underline{rst}[A_1, A_2, \cdots, A_n] \approx [A_2, A_3, \cdots, A_n] \;.$$

Here $[A]$ means $A$, when $n = 2$. We wish to have, for $1 \leq i < n$ ,

$$\underline{(i+1)th}[A_1, A_2, \cdots, A_n] \approx A_{i+1} \approx \underline{ith}[A_2, \cdots, A_n] \approx \underline{ith}(\underline{rst}[A_1, A_2, \cdots, A_n]) \;.$$

So it would be desirable to define $\underline{ith}$ inductively so that

$$\underline{(i+1)th}\; E \;\approx\; \underline{ith}\; (\underline{rst}\; E)$$

for all terms $E$. But we can't just drop those brackets and 'cancel' the $E$ for this !! However, define

$$\underline{(i+1)th} \;:=\; \underline{B}\; \underline{ith}\; \underline{rst} \;,$$

where

$$\underline{S} \;:=\; \lambda xyz \bullet (xz)(yz) \quad \text{and} \quad \underline{B} \;:=\; \underline{S}\,(\underline{T}\,\underline{S})\,\underline{T}\;.$$

(Of course $\underline{1th}$ is the same as $\underline{1st}$, and maybe we should have alternative names $\underline{2nd}$ and $\underline{3rd}$ for $\underline{2th}$ and $\underline{3th}$ !) This is all we need, by the second 'equation' below:

**VII-2.5** *We have, for all terms $A, B$ and $C$,*

$$\underline{S}ABC \approx (AC)(BC) \quad \text{and} \quad \underline{B}ABC \approx A(BC)\;.$$

**Proof.** The first equation is a mechanical exercise, using $(\beta)$ three times, after writing $\underline{S}$ using bound variables that don't occur in $A, B$ or $C$. Then the second one is a good practice in being careful with brackets, as follows:

$$\underline{B}ABC = \underline{S}\,(\underline{T}\,\underline{S})\,\underline{T}ABC = (\underline{S}\,(\underline{T}\,\underline{S})\,\underline{T}A)BC \approx (\underline{T}\,\underline{S})A(\underline{T}A)BC$$

$$= (\underline{T}\,\underline{S}\,A)(\underline{T}A)BC \approx \underline{S}(\underline{T}A)BC \approx (\underline{T}A)C(BC) = (\underline{T}AC)(BC) \approx A(BC)\;.$$

We have twice used both the first part and **VII-2.1**(a).

Don't try to guess what's behind the notation $\underline{S}$ and $\underline{B}$—they are just underlined versions of the traditional notations for these 'operators'. So I used them, despite continuing to use $B$ (not underlined) for one of the 'general' terms in stating results. For those who have already read something about combinatory algebra, later we shall also denote $\underline{T}$ alternatively as $\underline{K}$, so that $\underline{S}$ and $\underline{K}$ are the usual notation (underlined) for the usual two generators for the *combinators*. The proof above is a foretaste of *combinatory logic* from three subsections ahead. But we could have just set

$$\underline{B} \;:=\; \lambda xyz \bullet x(yz)\;,$$

for our purposes here, and proved that $\underline{B}ABC \approx A(BC)$ directly.

**Definitions of $\underline{I}$ and $A^n$ .** Define, for any term $A$,

$$A^0 \;:=\; \underline{I} \;:=\; \underline{S}\,\underline{T}\,\underline{T}\;,$$

and inductively

$$A^{n+1} \ := \ \underline{B}\ A\ A^n \ .$$

**VII-2.6** *For all terms $A$ and $B$, and all natural numbers $i$ and $j$, we have*

$$A^0B = \underline{I}\ B \approx B \quad ; \quad \underline{I} \approx \lambda x \bullet x \quad ; \text{ and } \quad A^i(A^jB) \approx A^{i+j}B \ .$$

Also $A^1 \approx A$.

**Proof.** This is a good exercise, using induction on $i$ for the 3rd displayed identity.

**Exercise.** (i) Sometimes $A^2 \not\approx AA$ . For example, try $A = \underline{T}$ or $\underline{F}$ .

(ii) Do the following sometimes give four distinct equivalence classes of terms :

$$A^3 \ ; \ AA^2 \ ; \ AAA = (AA)A \ ; \ A(AA) \approx A^2A \ ?$$

**Definitions of $\underline{s}$, $\underline{isz}$ and $\overline{n}$ .** Define, for natural numbers $n$,

$$\overline{n} \ := \lambda uv \bullet u^n v \quad ; \quad \underline{s} \ := \lambda xyz \bullet (xy)(yz) \quad ; \quad \underline{isz} \ := \lambda x \bullet x(\lambda y \bullet \underline{F})\underline{T} \ .$$

Here, $u$ and $v$ are variables, different from each other, as are $x, y$ and $z$.

**VII-2.7** *We have, for all natural numbers $n$,*

$$\underline{s}\ \overline{n} \approx \ \overline{n+1} \quad ; \quad \underline{isz}\ \overline{0} \approx \underline{T} \quad ; \quad \underline{isz}\ \overline{n} \approx \underline{F} \text{ if } n > 0 \ .$$

**Proof.** For any $n$, using four distinct variables,

$$\underline{isz}\ \overline{n} = (\lambda x \bullet x(\lambda y \bullet \underline{F})\underline{T})(\lambda uv \bullet u^n v) \approx (\lambda uv \bullet u^n v)(\lambda y \bullet \underline{F})\underline{T}$$

$$\approx (\lambda v \bullet (\lambda y \bullet \underline{F})^n v)\underline{T} \approx (\lambda y \bullet \underline{F})^n \underline{T} \qquad (*)$$

So when $n = 0$, we get

$$\underline{isz}\ \overline{0} \approx (\lambda y \bullet \underline{F})^0 \underline{T} \approx \underline{T} \ ,$$

since $A^0B \approx B$ quite generally.

Note that $(\lambda y \bullet \underline{F})B \approx \underline{F}$ for any $B$, merely because $\underline{F}$ is a closed term. So when $n > 0$, by $(*)$ and **VII-2.6** with $i = 1$ and $j = n - 1$,

$$\underline{isz}\ \overline{n} \approx (\lambda y \bullet \underline{F})((\lambda y \bullet \underline{F})^{n-1}\underline{T}) \approx \underline{F} \ .$$

The latter '$\approx$' is from the remark just before the display.

Finally, for any closed term $B$ and many others, the definition of $\underline{s}$ and rule $(\beta)$ immediately give

$$\underline{s}\ B \approx \lambda yz \bullet By(yz)$$

So

$$\underline{s}\ \overline{n} \approx \lambda yz \bullet (\lambda uv \bullet u^n v)y(yz) \approx \lambda yz \bullet (\lambda v \bullet y^n v)(yz)$$

$$\approx \lambda yz \bullet y^n(yz) \approx \lambda yz \bullet y^{n+1}z) \approx \ \overline{n+1}\ .$$

The penultimate step uses **VII-2.6** with $i = n$ and $j = 1$. and depends on knowing that $A^1 \approx A$.

**Exercise.**

(a) Define $\ \underline{+}\ :=\ \lambda uvxy \bullet (ux)(vxy)$ . Show that $\underline{+}\ \overline{k}\ \overline{\ell}\ \approx\ \overline{k+\ell}$ .

(m) Define $\ \underline{\times}\ :=\ \lambda uvy \bullet u(vy)$ . Show that $\underline{\times}\ \overline{k}\ \overline{\ell}\ \approx\ \overline{k\ell}$ .

**Definitions of $\underline{pp}$ and $\underline{p}$ .** For distinct variables $x, y, z, u$ and $w$, define

$$\underline{pp}\ := \lambda uw \bullet [\ \underline{F}\ ,\ \underline{1st}w(\underline{rst}w)(u(\underline{rst}w))\ ]\ ,$$

and

$$\underline{p}\ := \lambda xyz \bullet \underline{rst}(x(\underline{pp}\ y)[\ \underline{T}\ ,\ z\ ])\ .$$

**VII-2.8** (Kleene) *For all natural numbers $n > 0$, we have* $\underline{p}\ \overline{n}\ \approx\ \overline{n-1}$ .

**Proof.** First we show

$$\underline{pp}\ x\ [\ \underline{T}\ ,\ y\ ] \approx [\ \underline{F}\ ,\ y\ ] \qquad \text{and} \qquad \underline{pp}\ x\ [\ \underline{F}\ ,\ y\ ] \approx [\ \underline{F}\ ,\ xy\ ]\ .$$

Calculate :

$$\underline{pp}\ x\ [\ \underline{T}\ ,\ y\ ] \approx (\lambda uw \bullet [\ \underline{F}\ ,\ \underline{1st}w(\underline{rst}w)(u(\underline{rst}w))\ ])\ x\ [\ \underline{T}\ ,\ y\ ]$$

$$\approx (\lambda w \bullet [\ \underline{F}\ ,\ \underline{1st}w(\underline{rst}w)(x(\underline{rst}w))\ ])\ [\ \underline{T}\ ,\ y\ ]$$

$$\approx [\ \underline{F}\ ,\ \underline{1st}[\ \underline{T}\ ,\ y\ ](\underline{rst}[\ \underline{T}\ ,\ y\ ])(x(\underline{rst}[\ \underline{T}\ ,\ y\ ]))\ ]$$

$$\approx [\ \underline{F}\ ,\ \underline{T}\ y(xy)\ ] \approx [\ \underline{F}\ ,\ y\ ]\ .$$

The last step uses **VII-2.1**(a).

Skipping the almost identical middle steps in the other one, we get

$$\underline{pp}\ x\ [\ \underline{F}\ ,\ y\ ] \approx \cdots \approx [\ \underline{F}\ ,\ \underline{F}\ y(xy)\ ] \approx [\ \underline{F}\ ,\ xy\ ]\ .$$

Next we deduce

$$(\underline{pp}\ x)^n\ [\ \underline{F}\ ,\ y\ ] \approx [\ \underline{F}\ ,\ x^n y\ ] \qquad \text{and} \qquad (\underline{pp}\ x)^n\ [\ \underline{T}\ ,\ y\ ] \approx [\ \underline{F}\ ,\ x^{n-1}y\ ]\ ,$$

the latter only for $n > 0$. The left-hand identity is proved by induction on $n$, the right-hand one deduced from it, using **VII-2.6** with $i = n-1$ and $j = 1$ twice and once, respectively.

For the left-hand identity, when $n = 0$, this is trivial, in view of the fact that $A^0 B \approx B$. When $n = 1$, this is just the right-hand identity of the two proved above, in view of the fact that $A^1 \approx A$ . Inductively, with $n \geq 2$, we get

$$(\underline{pp}\ x)^n\ [\ \underline{F}\ ,\ y\ ] \approx (\underline{pp}\ x)^{n-1}(\underline{pp}\ x\ [\ \underline{F}\ ,\ y\ ]) \approx (\underline{pp}\ x)^{n-1}[\ \underline{F}\ ,\ xy\ ]$$
$$\approx [\ \underline{F}\ ,\ x^{n-1}(xy)\ ] \approx [\ \underline{F}\ ,\ x^n y\ ]\ .$$

(Since $xy$ isn't a variable, the penultimate step looks fishy, but actually, all these identities hold with $x$ and $y$ as names for arbitrary terms, not just variables.)

For the right-hand identity,

$$(\underline{pp}\ x)^n\ [\ \underline{T}\ ,\ y\ ] \approx (\underline{pp}\ x)^{n-1}(\underline{pp}\ x\ [\ \underline{T}\ ,\ y\ ]) \approx (\underline{pp}\ x)^{n-1}[\ \underline{F}\ ,\ y\ ] \approx [\ \underline{F}\ ,\ x^{n-1}y\ ]\ .$$

Now to prove Kleene's striking (see the next subsection) discovery, using the ($\beta$)-rule with the definition of $\underline{p}$, we see that, for all terms $A$,

$$\underline{p}\ A\ \approx\ \lambda yz \bullet \underline{rst}(A(\underline{pp}\ y)[\ \underline{T}\ ,\ z\ ])\ .$$

Thus

$$\underline{p}\,\overline{n}\ \approx\ \lambda yz{\bullet}\underline{rst}((\lambda uw{\bullet}u^n w)(\underline{pp}\,y)[\,\underline{T}\,,\,z\,])\ \approx\ \lambda yz{\bullet}\underline{rst}((\lambda w{\bullet}(\underline{pp}\,y)^n w)[\,\underline{T}\,,\,z\,])$$
$$\approx\ \lambda yz \bullet \underline{rst}((\underline{pp}\ y)^n[\ \underline{T}\ ,\ z\ ])\ \approx\ \lambda yz \bullet \underline{rst}[\ \underline{F}\ ,\ y^{n-1}z\ ])$$
$$\approx\ \lambda yz{\bullet}y^{n-1}z\ \approx\ \overline{n-1}\,.$$

**Exercise.** Show that $\underline{p}\ \overline{0}\ \approx \overline{0}$ .

**Exercise.** Do we have $\underline{p}\ \underline{s}\ \approx \underline{I}$ or $\underline{s}\ \underline{p}\ \approx \underline{I}$ ?

**Definition of $\underline{Y}$ .**

$$\underline{Y}\ := \lambda x \bullet (\lambda y \bullet x(yy))(\lambda y \bullet x(yy)) \ .$$

Of course, $x$ and $y$ are different from each other.

> *It is a very powerful concept, one that in a sense has inspired many mathematical developments, such as recursion theory, indeed also some literary productions, such as Hofstadter's popular book.*
>
> Erwin Engeler **[En]**

**VII-2.9** *For all terms $A$, there is a term $B$ such that $AB \approx B$ . In fact, the term $B = \underline{Y}\ A$ will do nicely for that.*

**Proof.** Using the $(\beta)$-rule twice, we see that

$$\underline{Y}\ A\ =\ (\ \lambda x \bullet (\lambda y \bullet x(yy))(\lambda y \bullet x(yy))\ )A\ \approx\ (\lambda y \bullet A(yy))(\lambda y \bullet A(yy))$$

$$\approx\ A((\lambda y \bullet A(yy))(\lambda y \bullet A(yy)))\ \approx\ A\ (\underline{Y}\ A)\ ,$$

as required. The last step used the '$\approx$' relation between the first and third terms in the display.

It is interesting that Curry's $\underline{Y}$ apparently fails to give either $A(\underline{Y}A) \overline{\succeq} \underline{Y}A$, or $\underline{Y}A \overline{\succeq} A(\underline{Y}A)$. It can be useful to replace it by Turing's

$$\underline{Z}\ :=\ (\lambda xy \bullet y(xxy))(\lambda xy \bullet y(xxy))\ ,$$

another so-called fixed point operator for which at least one does have $\underline{Z}A \overline{\succeq} A(\underline{Z}A)$ for all $A$ in $\Lambda$.

Recall that a *closed* term is one with no free variable occurrences.

**VII-2.10** *Let $x$ and $y$ be distinct variables, and let $A$ be a term with no free variables other than possibly $x$ and $y$. Let $F := \underline{Y}(\lambda yx \bullet A)$ . Then, for any closed term $B$, we have*

$$FB\ \approx\ A^{[y \to F]\,[x \to B]}\ .$$

The 'okayness' of the substitutions below follows easily from the restrictions on $A$ and $B$, which are stronger than strictly needed, but are satisfied in all the applications.

**Proof.** Using the basic property of $\underline{Y}$ from **VII-2.9** for the first step,

$$FB \;\approx\; (\lambda yx \bullet A)FB \;\approx\; (\lambda x \bullet A)^{[y\mapsto F]}B \;\approx\; (\lambda x \bullet A^{[y\mapsto F]})B \;\approx\; A^{[y\mapsto F][x\mapsto B]} \;.$$

**VII-2.11** *For all terms $G$ and $H$, there is a term $F$ such that*

$$F\,\overline{0} \;\approx\; G \quad \textit{and} \quad F\,\overline{n+1} \;\approx\; H[\,F\overline{n}\;,\;\overline{n}\,] \;\; \textit{for all natural numbers } n \;.$$

(My goodness, this looks a lot like primitive recursion!)

**Proof.** Let

$$F := \underline{Y}(\lambda yx \bullet A) \;,$$

as in the previous result, where $x$ and $y$ are distinct variables which do not occur in $G$ or $H$, and where

$$A \;:=\; (\underline{isz}\;x)\;G\;(H[\;y(\underline{p}x)\;,\;\underline{p}x\;]) \;.$$

Then, for any term $B$, using **VII-2.10** for the first step,

$$FB \;\approx\; A^{[y\mapsto F][x\mapsto B]} \;\approx\; (\underline{isz}\;B)\;G\;(H[\;F(\underline{p}B)\;,\;\underline{p}B\;]\;) \;.$$

First take $B = \overline{0}$. But $\underline{isz}\;\overline{0} \approx \underline{T}$ by **VII-2.7**, and $\underline{T}GJ \approx G$ by **VII-2.1**(a), so this time we get $F\,\overline{0} \;\approx\; G$ , as required.

Then take $B = \overline{n+1}$. But $\underline{isz}\;\overline{n+1} \approx \underline{F}$ by **VII-2.7**, and $\underline{F}GJ \approx J$ by **VII-2.1**(b), so here we get

$$F\,\overline{n+1} \;\approx\; H[\;F(\underline{p}\;\overline{n+1})\;,\;\underline{p}\;\overline{n+1}] \;\approx\; H[\;F\;\overline{n}\;,\;\overline{n}\;]\;,$$

as required, using **VII-2.8** for the second step.

## VII-3 So—what's going on?

Actually, I'm not completely certain myself. Let's review the various examples just presented.

At first we saw some terms which seemed to be modelling objects in propositional logic. Here are two slightly curious aspects of this. Firstly, $\underline{T}$ and $\underline{F}$ are presumably truth values in a sense, so come from the semantic side, whereas $\underline{\neg}$ , $\underline{\wedge}$ and $\underline{\vee}$ are more like syntactic objects. Having them all on the same footing does alter one's perceptions slightly, at least for non-CSers. Secondly, we're not surprised to see the latter versions of the connectives acting like functions which take truth values as input, and produce truth values as output. But everything's on a symmetric footing, so writing down a term like $\underline{F}\ \underline{\wedge}$ now seems like having truth values as functions which can take connectives as input, not a standard thing to consider. And $\underline{F}\ \underline{F}$ seems even odder, if interpreted as the truth value 'substituted into itself'!

But later we had terms $\overline{n}$ which seemed to represent numbers, not functions. However, two quick applications of the $\beta$-rule yield

$$\bar{n}AB \ \approx\ A^nB \ \approx\ A(A(A\cdots(AB))\cdots)) \ .$$

So if $A$ were thought of as representing a function, as explained a bit below, the term $\bar{n}$ may be thought of as representing that function which maps $A$ to its $n$-fold iterate $A^n$.

Now the $\bar{n}$, along with $\underline{s}$ and $\underline{p}$ as successor and predecessor functions, or at least terms representing them, give us the beginnings of a *numeral system* sitting inside $\Lambda$. The other part of that numeral system is $\underline{isz}$ , the "Is it 0?"–predicate. Of course predicates may be regarded as functions. And the final result goes a long way towards showing that terms exist to represent all primitive recursive functions. To complete the job, we just need to amplify **VII-2.11** to functions of more than one variable, and make some observations about composition. More generally, in the subsection after next, we show that *all* (possibly partial) recursive functions are definable in the $\lambda$-calculus. It is hardly surprising that no others are, thinking about Church's thesis.

**Exercise.** Show that $\overline{k}\ \overline{\ell}\ \approx\ \overline{\ell^k}$ . Deduce that $\overline{1}$ is alternatively interpretable as a combinator which defines the exponential function. (See the exercise before **VII-2.8** for the addition and multiplication functions.)

Show that $\overline{9}\ \overline{9}\ \overline{9}\ \overline{9}\ \approx\ \overline{n}$ , where $n > 2^m$, where $m$ is greater than the mass of the Milky Way galaxy, measured in milligrams !

It is likely clear to the reader that $[A\ ,\ B\ ]$ was supposed to be a term which represents an ordered pair. And we then produced ordered $n$-tuple

terms in $\Lambda$, as well as terms representing the projection functions.

So we can maybe settle on regarding each term in $\Lambda$ as representing a function in some sense, and the construction $(AB)$ as having $B$ fed in as input to the function represented by $A$, producing an 'answer' which again represents a function. But there is sense of unease (for some of us) in seeing what appears to be a completely self-contained system of functions, every one of which has the exact same set of all these functions apparently as both its domain and its codomain. (A caveat here—as mentioned below, the existence of terms without normal forms perhaps is interpretable as some of these functions being partial.) That begins to be worrisome, since except for the 1-element set, the set of all functions from $X$ to itself has strictly larger cardinality than $X$. But nobody said it had to be *all* functions. However it still seems a bit offputting, if not inconsistent, to have a bunch of functions, every one of which is a member of its own domain! (However, the formula for $\underline{Y}$ reflects exactly that, containing 'a function which is substituted into itself'—but $\underline{Y}$ was very useful in the last result, and that's only the beginning of its usefulness, as explained in the subsection after next. The example of a term with no normal form at the beginning of Subsection VII-2 was precisely 'a function which is substituted into itself'.) The "offputting" aspect above arises perhaps from the fact that, to re-interpret this stuff in a consistent way within axiomatic $1^{\underline{\text{st}}}$order set theory, some modification (such as abandonment) of the axiom of foundation seems to be necessary. See however Scott's constructions in Subsection VII-8 ahead.

In fact, this subject was carried on from about 1930 to 1970 in a very syntactic way, with little concern about whether there might exist mathematical models for the way $\Lambda$ behaved, other than $\Lambda$ itself, or better, the set of equivalence classes $\Lambda/\approx$ . But one at least has the Church-Rosser theorem, showing that, after factoring out by the equivalence relation $\approx$ , the whole thing doesn't reduce to the triviality of a 1-element set. Then around 1970 Dana Scott produced some interesting such models, and not just because of pure mathematical interest. His results, and later similar ones by others, are now regarded as being very fundamental in parts of computer science. See Subsections VII-7, VII-8 and VII-9 ahead.

Several things remain to be considered. It's really not the terms themselves, but rather their corresponding equivalence classes under $\approx$ which might be thought of as functions. The terms themselves are more like recipes for calculating functions. Of what does the calculation consist? Presumably

its steps are the reductions, using the three rules, but particularly ($\beta$) and ($\eta$). When is a calculation finished? What is the output? Presumably the answer is that it terminates when the normal form of the start term is reached, and that's the output. Note that the numerals $\overline{n}$ themselves are in normal form (with one exception). But what if the start term has no normal form? Aha, that's your infinite loop, which of course must rear its ugly head, if this scheme is supposed to produce some kind of general theory of algorithms. So perhaps one should consider the start term as a combination of both input data and procedure. It is a fact that a normal form will eventually be produced, if the start term actually has a normal form, by always concentrating on the leftmost occurrence of $\lambda$ for which an application of ($\beta$) or ($\eta$) can do a reduction (possibly preceded by an application of ($\alpha$) to change bound variables and make the substitution okay). This algorithm no doubt qualifies intuitively as one, involving discrete mechanical steps and a clear 'signal', if and when it's time to stop computing. The existence of more than one universal Turing machine, more than one **ATEN**-command for computing the universal function, etc. ... presumably correspond here to the possibility of other definite reduction procedures, besides the one above, each of which will produce the normal form of the start term (input), if the start term has one. As mentioned earlier, such reduction schemes are algorithms showing the semidecidability of the existence of a normal form. Later we'll see that this question is undecidable.

We have probably left far too late a discussion of what the rules ($\beta$) and ($\eta$) are doing. And indeed, what *is* $\lambda$ itself? It is often called the *abstraction*-operator. The symbol string '$\lambda x\bullet$' is to alert the computer (human or otherwise) that a function of the variable $x$ is about to be defined. And of course it's the *free* occurrences of $x$ in $A$ which give the 'formula' for the function which $\lambda x \bullet A$ is supposed to be defining.

So now the explanations of ($\beta$) and ($\eta$) are fairly clear :

The rule ($\beta$) just says that to evaluate the function above on $B$, i.e. $(\lambda x \bullet A)B$ , you substitute $B$ for the free occurrences of $x$ in $A$ (of course!).

And rule ($\eta$) just says that, if $x$ doesn't occur freely in $A$, then the function defined by $\lambda x \bullet (Ax)$ is just $A$ itself (of course—'the function obtained by evaluating $A$ on its argument!).

[As an aside, it seems to me to be not unreasonable to ask why one shouldn't change the symbol '$\lambda$' to a symbol '$\mapsto$'. After all, that's what we're talking about, and it has already been noted that the second basic

symbol '•' is quite unneeded, except for one of the abbreviations. So the string $(\lambda x \bullet A)$ would be replaced by $(x \mapsto A)$. Some things would initially be more readable for ordinary $\lambda$-inexperienced mathematicians. I didn't want to do that in the last section, because that would have given the game away too early. And I won't do it now, out of respect for tradition. Also we can think of '$\lambda$' as taking place in a formal language, whereas '$\mapsto$' is a concept from the metalanguage, so maybe that distinction is a good one to maintain in the notation. Actually, $\lambda$-philes will often use a sort of 'Bourbaki-$\lambda$' in their metalanguages !]

This has become rather long-winded, but it seems a good place to preview Subsection VII-6, and then talk about the history of the subject, and its successful (as well as aborted) applications.

There is a small but infinite subset of $\Lambda$ called the set of combinators (up to "$\approx$", just the set of closed terms), some of whose significance was discovered by Schonfinkel around 1920, without explicitly dealing with the $\lambda$-calculus. We shall treat it in some detail in Subsection VII-6, and hopefully explain more clearly than above about just which functions in some abstract sense are being dealt with by the subject, and what it was that Schonfinkel discovered.

Independently reinventing Schonfinkel's work a few years later, Curry attempted to base an entire foundations of mathematics on the combinators, as did Church soon after, using the $\lambda$-calculus, which he invented. But the system(s) proposed (motivated by studying very closely the processes of substitution occurring in Russell & Whitehead's Principia Mathematica) were soon discovered to be inconsistent, by Kleene and Rosser, students of Church. There will be nothing on this (largely abortive) application to foundations here (beyond the present paragraph). It is the case that, for at least 45 years after the mid-1930's when the above inconsistency was discovered (and possibly to the present day), there continued to be further attempts in the same direction by a small group, a subject known as illiative combinatory logic. [To get some feeling for this, take a look at [**Fi**], but start reading at Ch.1. Skip the Introduction, at least to begin, and also the Preface, or at least don't take too seriously the claims there, before having read the entire book and about 77 others, including papers of Peter Aczel, Solomon Fefferman and Dana Scott.] It's not totally unreasonable to imagine

a foundational system in which "Everything is a function!" might be attractive. After all, our 1st order set theory version of foundations is a system in which "Everything is a set!" (and Gödel seems to tell us that, if it *is* consistent, we can never be very sure of that fact). On the other hand, it is also hardly surprising that there might be something akin to Russell's paradox, which brought down Frege's earlier higher order foundations, but in an attempted system with the combinators. The famous *fixed point combinator* $\underline{Y}$ (see **VII-2.9**), or an analogue, played a major role in the Kleene-Rosser construction showing inconsistency.

> In current mathematics ... the notion of set is more fundamental than that of function, and the domain of a function is given before the function itself. In combinatory logic, on the other hand, the notion of function is fundamental; a set is a function whose application to an argument may sometimes be an assertion, or have some other property; its members are those arguments for which the application has that property. The function is primary; its domain, which is a set, is another function. Thus it is simpler to define a set in terms of a function than vice versa; but the idea is repugnant to many mathematicians, ...
>
> H.B. Curry **[BKK-ed]**

> What I always found disturbing about combinatory logic was what seemed to me to be a *complete lack* of conceptual continuity. There were no functions known to anyone else that had the extensive properties of the combinators *and allowed self-application*. I agree that people might wish to have such functions, but very early on the contradiction found by Kleene and Rosser showed there was trouble. What I cannot understand is why there was not more discussion of the question of *how* the notion of function that was behind the theory was to be made even mildly harmonious with the "classical" notion of function. The literature on combinatorial logic seems to me to be somehow silent on this point. Perhaps the reason was that the hope of "solving" the paradoxes remained alive for a long time—and may still be alive.
>
> D. Scott **[BKK-ed]**

And that is a good lead-in for a brief discussion of the history of the $\lambda$-calculus as a foundation for computability, indeed the very first foundation, beating out Turing machines by a hair. Around 1934, Kleene made great progress in showing many known computable functions to be definable

in the $\lambda$-calculus. A real breakthrough occurred when he saw how to do it for the predecessor function, our **VII-2.8**. It was based on Kleene's results that his supervisor, Church, first made the proposal that the intuitively computable functions be identified with the mathematically defined set of $\lambda$-*definable* functions. This is of course Church's Thesis, later also called the Church-Turing Thesis, since Turing independently proposed the Turing computable functions as the appropriate set within a few months, and gave a strong argument for this being a sensible proposal. Then he proved the two sets to be the same, as soon as he saw Church's paper. The latter paper proved the famous Church's Theorem providing a negative solution to Hilbert's Entscheidungsproblem, as had Turing also done independently. See the subsection after next for the definition of $\lambda$-definable and proof that all recursive functions are $\lambda$-definable.

Finally, I understand that McCarthy, in the late 1950's, when putting forth several fundamental ideas and proposing what have come to be known as *functional programming languages* (a bit of which is the **McSELF** procedures from earlier), was directly inspired by the $\lambda$-calculus [**Br-ed**]. This lead to his invention of LISP, the first such language (though it's not purely functional, containing, as it does, imperative features). It is said that all such languages include, in some sense, the $\lambda$-calculus, or even that they are all equivalent to the $\lambda$-calculus. As a miniature example, look at **McSELF** in [**CM**]. A main feature differentiating functional from imperative programming languages (of which **ATEN** is a miniature example) is that each program, when implemented, produces steps which (instead of altering a 'store', or a sequence of 'bins' as we called it) are stages in the calculation of a function, rather like the reduction steps in reducing a $\lambda$-term to normal form, or at least attempting to do so. Clearly we should expect there to be a theorem saying that there can be no algorithm for deciding whether a $\lambda$-term *has* a normal form. See the next subsection for a version of this theorem entirely within the $\lambda$-calculus.

Another feature differentiating the two types of languages seems to be a far greater use of the so-called recursive (self-referential) programs in the functional languages. In the case of our earlier miniature languages, we see that leaving that feature out of **McSELF** would destroy it (certainly many computable functions could not be programmed), whereas one of the main points of Subsection IV-9 was to see that any recursive command could be replaced by an ordinary **ATEN**-command.

We shall spend much time in the subsection after next with explaining in detail how to use the $\underline{Y}$-operator to easily produce terms which satisfy equations. This is a basic self-referential aspect of computing with the $\lambda$-calculus. In particular, it explains the formula which was just 'pulled out of a hat' in the proof of **VII-2.11**.

We shall begin the subsection after next with another, much simpler, technical idea, which implicitly pervades the last subsection. This is how the triple product $ABC$ can be regarded as containing the analogue of the if-then-else-construction which is so fundamental in **McSELF**.

First we present some crisp and edifying versions within the $\lambda$-calculus of familiar material from earlier in this work.

## VII-4 Non-examples and Non-calculability in $\Lambda$—undecidability.

Thinking of terms in $\Lambda$ as being, in some sense, algorithms, here are a few analogues of previous results related to the non-existence of algorithms/commands/Turing machines/recursive functions. All this clearly depends on the Church-Rosser theorem, since the results below would be manifestly false if, for example, all elements of $\Lambda$ had turned out to be related under "$\approx$" . Nothing later depends on this subsection.

**VII-4.1** (Undecidability of $\approx$ .) *There is no term $\underline{E} \in \Lambda$ such that, for all $A$ and $B$ in $\Lambda$,*

$$\underline{E}\ A\ B\ \approx\ \begin{cases} \underline{T} & \text{if } A \approx B\ ; \\ \underline{F} & \text{if } A \not\approx B\ . \end{cases}$$

**First Proof.** Suppose, for a contradiction, that $\underline{E}$ did exist, and define

$$u\ :=\ \lambda y \bullet \underline{E}(yy)\overline{0}\ \overline{1}\ \overline{0}\ .$$

By the $(\beta)$-rule, we get $uu \approx \underline{E}(uu)\overline{0}\ \overline{1}\ \overline{0}$ . Now either $uu \approx \overline{0}$ or $uu \not\approx \overline{0}$ .

But in the latter case, we get $uu\ \approx\ \underline{F}\ \overline{1}\ \overline{0}\ \approx\ \overline{0}$ , a contradiction.

And in the former case, we get $uu\ \approx\ \underline{T}\ \overline{1}\ \overline{0}\ \approx\ \overline{1}$ , contradicting uniqueness of normal form, which tells us that $\overline{1}\ \not\approx\ \overline{0}$.

**VII-4.2** (Analogue of Rice's Theorem.) *If $\underline{R} \in \Lambda$ is such that*

$$\forall A \in \Lambda\ \text{ either }\ \underline{R}\ A\ \approx\ \underline{T}\ \text{ or }\ \underline{R}\ A\ \approx\ \underline{F}\ ,$$

*then either* $\forall A \in \Lambda\ ,\ \underline{R}\ A\ \approx\ \underline{T}$ *or* $\forall A \in \Lambda\ ,\ \underline{R}\ A\ \approx\ \underline{F}$ .

**Proof.** For a contradiction, suppose we can choose terms $B$ and $C$ such that $\underline{R}B \approx \underline{T}$ and $\underline{R}C \approx \underline{F}$ . Then define

$$M \;:=\; \lambda y \bullet \underline{R}\; y\; C\; B \qquad \text{and} \qquad N \;:=\; \underline{Y}\; M.$$

By **VII-2.7**, we get $N \;\approx MN$ . Using the $(\beta)$-rule for the second $\approx$ ,

$$\underline{R}N \;\approx\; \underline{R}(MN) \;\approx\; \underline{R}(\underline{R}NCB)\ .$$

Since $\underline{R}N \approx$ either $\underline{T}$ or $\underline{F}$ , we get either

$$\underline{T} \;\approx\; \underline{R}(\underline{T}CB) \;\approx\; \underline{R}\,C \;\approx\; \underline{F}\ ,$$

or

$$\underline{F} \;\approx\; \underline{R}(\underline{F}CB) \;\approx\; \underline{R}\,B \;\approx\; \underline{T}\ ,$$

both of which are rather resounding contradictions to uniqueness of normal form.

**Second Proof of VII-4.1.** Again assuming $\underline{E}$ exists, fix any $B$ in $\Lambda$ and define $\underline{R} := \underline{E}B$. This immediately contradicts **VII-4.2**, by choosing any $C$ with $C \not\approx B$ , and calculating $\underline{R}B$ and $\underline{R}C$ .

**Second Corollary to VII-4.2.** (Undecidability of the existence of normal form.) *There is no $\underline{N} \in \Lambda$ such that*

$$\underline{N}\ A \;\approx\; \begin{cases} \underline{T} & \text{if } A \text{ has a normal form ;} \\ \underline{F} & \text{otherwise .} \end{cases}$$

This is immediate from knowing that some, but not all, terms do have a normal form.

**Third Proof of VII-4.1.** As an exercise, show that there is a ' 2-variable' analogue of Rice's Theorem : that is, prove the non-existence of a nontrivial $\underline{R_2}$ such that $\underline{R_2}AB$ always has normal form either $\underline{T}$ or $\underline{F}$ . Then **VII-4.1** is immediate.

**VII-5 Solving equations, and proving $\mathcal{RC} \iff \lambda$-definable.**

Looking back at the first two results in Subsection VII-2, namely

$$\underline{T}\ A\ B\ \approx\ A \quad \text{and} \quad \underline{F}\ A\ B\ \approx\ B\ ,$$

we are immediately reminded of if-then-else, roughly : 'a true condition says to go to $A$, but if false, go to $B$.'

Now turn to the displayed formula defining $A$ in the proof of **VII-2.11**, i.e.

$$A\ :=\ (\underline{isz}\ x)\ G\ (H[\ y(\underline{p}x)\ ,\ \underline{p}x\ ])\ .$$

This is a triple product, which is more-or-less saying

'if $x$ is zero, use $G$, but if it is non-zero, use $H[\ y(\underline{p}x)\ ,\ \underline{p}x\ ]$' .

This perhaps begins to explain why that proof works, since we are trying to get a term $F$ which, in a sense, reduces to $G$ when $x$ is zero, and to something involving $H$ otherwise. What still needs more explanation is the use of $\underline{Y}$ , and the form of the term $H[\ y(\underline{p}x)\ ,\ \underline{p}x\ ]$ . That explanation in general follows below; both it and the remark above are illustrated several times in the constructions further down.

The following is an obvious extension of **VII-2.10**.

**Theorem VII-5.1** *Let $z, y_1, \cdots, y_k$ be mutually distinct variables and let $A$ be any term in which no other variables occur freely. Define $F$ to be the closed term $\underline{Y}(\lambda z y_1 \cdots y_k \bullet A)$ . Then, for all closed terms $V_1, \cdots, V_n$ , we have*

$$FV_1 \cdots V_n\ \approx\ A^{[z \mapsto F][y_1 \mapsto V_1]} \ldots^{[y_k \mapsto V_k]}\ .$$

**Proof.** (This is a theorem, not a proposition, because of its importance, not the difficulty of its proof, which is negligible!) Using the basic property of $\underline{Y}$ from **VII-2.9** for the first step, and essentially **VII-1.2** for the other steps,

$$\begin{aligned} FV_1 \cdots V_k\ &\approx\ (\lambda z y_1 \cdots y_k \bullet A) F V_1 \cdots V_k \\ &\approx\ (\lambda y_1 \cdots y_k \bullet A)^{[z \mapsto F]} V_1 \cdots V_k\ \approx\ (\lambda y_1 \cdots y_k \bullet A^{[z \mapsto F]}) V_1 \cdots V_k \\ &\cdots\ \approx\ \cdots\ \approx A^{[z \mapsto F][y_1 \mapsto V_1]} \ldots^{[y_k \mapsto V_k]}\ . \end{aligned}$$

Why is this interesting? Imagine given an 'equation'

$$F\ \overline{n_1} \cdots \overline{n_k}\ \approx\ ---F------F---F---\ ,$$

where the right-hand side is something constructed using the $\lambda$-calculus, except that $F$ is an unknown term which we wish to find, and it may occur more than once on the right-hand side. We need to construct a term $F$ which satisfies the equation. The right-hand side will probably also have the numerals $\overline{n_1} \cdots \overline{n_k}$ appearing. Well, the above theorem tells us how to solve it. Just produce the term $A$ by inventing "$k+1$" distinct variables not occurring bound in the right-hand side above, and use them to replace the occurrences of $F$ and of the numerals in that right-hand side. That will produce a term, since that's what we meant by saying that the right-hand side was "constructed using the $\lambda$-calculus". Now the theorem tells us the formula for $F$ in terms of $A$.

Of course the theorem is more general, in that we can use any closed terms $V_i$ in place of numerals. But very often it's something related to a function of $k$-tuples of numbers where the technique is used. In the specific example of the proof of **VII-2.11**, the function $f$ to be represented can be written in the if-then-else-form as

"$f(x)$ , if $x = 0$, is $g$, but otherwise is $h(f(\mathrm{pred}(x)), \mathrm{pred}(x))$ ."

Here "pred" is the predecessor function. Also $g$ is a number represented by the term $G$ (which is presumably a numeral), and $h$ is a function of two variables represented by $H$ . This immediately motivates the formula for $A$ in the proof of **VII-2.11** .

What we called the "basic property of $\underline{Y}$ from **VII-2.9**" says that it is a so-called **fixed point operator**. There are many other possibilities for such an operator. But the detailed form of $\underline{Y}$ is always irrelevant to these constructions in existence proofs. However, when getting right down to the details of compiling a functional programming language, those details are undoubtedly essential, and there, presumably, some $\underline{Y}$'s are better than others. We gave another one, due to Turing, soon after the introduction of Curry's $\underline{Y}$ in Subsection VII-2.

**Definition of the numeral equality predicate term, $\underline{eq}$ .**

Here is a relatively simple example, producing a useful little term for testing equality of numerals (but only numerals—testing for $\approx$ in general is

unecidable, as we saw in the previous subsection!); that is, we require

$$\underline{eq}\ \overline{n}\ \overline{k}\ \approx\ \begin{cases} \underline{T} & \text{if } n = k\ ; \\ \underline{F} & \text{if } n \neq k\ . \end{cases}$$

An informal **McSELF**-like procedure for the actual predicate would be

$\mathsf{EQ}(n,k)$ $\Leftarrow$ if $n = 0$
then if $k = 0$
then true
else false
else if $k = 0$
then false
else $\mathsf{EQ}(n-1, k-1)$

(This is not really a **McSELF** procedure for two reasons—the equality predicate is a primitive in **McSELF**, and the values should be 1 and 0, not true and false. But it does show that we could get 'more primitive' by beginning **McSELF** only with 'equality to zero' as a unary predicate in place of the binary equality predicate.)

What the last display does is to tell us a defining equation which $\underline{eq}$ must satisfy:

$$\underline{eq}\ \overline{n}\ \overline{k}\ \approx\ (\underline{isz}\ \overline{n})((\underline{isz}\ \overline{k})\underline{T}\ \underline{F})((\underline{isz}\ \overline{k})\underline{F}(\underline{eq}(\underline{p}\ \overline{n})(\underline{p}\ \overline{k})))\ .$$

Thus we merely need to take

$$\underline{eq}\ :=\ \underline{Y}(\lambda zyx \bullet A)\ ,$$

where

$$A\ :=\ (\underline{isz}\ y)((\underline{isz}\ x)\underline{T}\ \underline{F})((\underline{isz}\ x)\underline{F}(z(\underline{p}\ y)(\underline{p}\ x)))\ .$$

[Purely coincidentally, the middle term in that triple product, namely $(\underline{isz}\ x)\underline{T}\ \underline{F}$ could be replaced by simply $\underline{isz}\ x$ , since $\underline{T}\ \underline{T}\ \underline{F} \approx \underline{T}$ and $\underline{F}\ \underline{T}\ \underline{F} \approx \underline{F}$ .]

### $\underline{\lambda}$-definability

First here is the definition. In the spirit of this work, we go directly to the case of (possibly) partial functions, rather than fooling around with totals first.

**Definition.** Let $D \subset \mathbf{N}^k$ and let $f : D \to \mathbf{N}$ . Say that the function $f$ *is* $\lambda$*-definable* if and only if there is a term $F$ in $\Lambda$ such that the term $F\overline{n_1}\cdots\overline{n}_k$ has a normal form for exactly those $(n_1, \cdots, n_k)$ which are in $D$, the domain of $f$, and, in this case, we have

$$F\ \overline{n_1}\cdots\overline{n_k}\ \approx\ \overline{f(n_1, \cdots, n_k)}\ .$$

We wish to prove that all (partial) recursive functions are $\lambda$-definable. This definitely appears to be a somewhat involved process. In particular, despite the sentiment expressed just before the definition, it seems most efficient to deal with the $\lambda$-definability of *total* recursive functions first. So the wording of the following theorem is meant to indicate we are ignoring all the tuples not in the domains. That is, no claim is made about the non-existence of normal forms of $H\overline{n_1}\cdots\overline{n_k}$ , for $(n_1, \cdots, n_k)$ not in the domain of $h$. This theorem is a main step for dealing with minimization (in proving all recursive functions to be $\lambda$-definable).

**Theorem VII-5.2** *If $g : D_g \to \mathbf{N}$ has a $\Lambda$-term which computes it on its domain $D_g \subset \mathbf{N}^k$, then so has $h$, where $h(n_1, \cdots, n_k)$ is defined to be*

$$\min\{\ell \mid \ell \geq n_1\ ;\ (m, n_2, \cdots, n_k) \in D_g \text{ for } n_1 \leq m \leq \ell \text{ and } g(\ell, n_2, \cdots, n_k) = 0\}.$$

The function $h$ has domain

$$D_h = \{(n_1, n_2, \cdots, n_k) \mid \exists \ell \text{ with } \ell \geq n_1\ ,\ g(\ell, n_2, \cdots, n_k) = 0$$
$$\text{and } (m, n_2, \cdots, n_k) \in D_g \text{ for } n_1 \leq m \leq \ell\ \}\ .$$

*In particular, if $g$ is total and $h$ happens to be total, $\lambda$-definability of $g$ implies $\lambda$-definability of $h$ .*

**Proof.** This is another example of using the fixed point operator, i.e. the term $\underline{Y}$ . Here is an informal **McSELF** procedure for $h$ :

$$\begin{array}{lll} h(n_1, \cdots, n_k) & \Leftarrow & \text{if } g(n_1, n_2, \cdots, n_k) = 0 \\ & & \text{then } n_1 \\ & & \text{else } h(n_1 + 1, n_2, \cdots, n_k) \end{array}$$

So, if $G$ is a $\Lambda$-term defining $g$ on its domain, then a defining equation for a term $H$ for $h$ is

$$H\overline{n_1}\cdots\overline{n_k}\ \approx\ (\underline{isz}\ (G\overline{n_1}\cdots\overline{n_k}))\overline{n_1}(H(\underline{s}\ \overline{n_1})\overline{n_2}\cdots\overline{n}_k)\ .$$

Thus we merely need to take

$$H \;:=\; \underline{Y}(\lambda z y_1 \cdots y_k \bullet A) \;,$$

where

$$A \;:=\; (\underline{isz}\;(Gy_1 \cdots y_k))y_1(z(\underline{s}\;y_1)y_2 \cdots y_k) \;.$$

It's surprising how easy this is, but 'that's the power of $\underline{Y}$ ' !

But to repeat: the non-existence of a normal form for $H\overline{n_1}\cdots\overline{n_k}$ away from the domain of $h$ is quite likely false! For the next theorem, we can actually give a counter-example to establish that the proof really only gives closure of the set of *total* $\lambda$-definable functions under composition, despite the fact that, as we'll see later, the result is true in the general case of (possibly strictly) partial functions.

**Theorem VII-5.3** *The set of total $\lambda$-definable functions is closed under composition. That is, given $\lambda$-definable total functions as follows :*
*$g$, of "$a$" variables ; and $h_1, \cdots, h_a$, each of "$b$" variables ; their composition*

$$(n_1, \cdots, n_b) \;\mapsto\; g(h_1(n_1, \cdots, n_b), - - -, h_a(n_1, \cdots, n_b))$$

*is also $\lambda$-definable. More generally, we can again assume the functions are partial and produce a $\Lambda$-term for the composition, using ones for the ingredients; but no claim can be made about lack of normal form away from the domain.*

**Example.** We have that the identity function, $g : \mathbf{N} \to \mathbf{N}$, is $\lambda$-defined by $G = \lambda x \bullet x$ . And $H = (\lambda x \bullet xx)(\lambda x \bullet xx)$ defines the empty function $h : \emptyset \to \mathbf{N}$. The composite $g \circ h$ is also the empty function. But the term $D_{1,1}GH$ constructed in the following proof does not $\lambda$-define it; in fact, it defines the identity function.

**Proof.** Let terms $G$ and $H_1, \cdots, H_a$ represent the given functions, so that

$$G\;\overline{m_1}\cdots\overline{m_a} \;\approx\; \overline{g(m_1, \cdots, m_a)}$$

and

$$H_i\;\overline{n_1}\cdots\overline{n_b} \;\approx\; \overline{h_i(n_1, \cdots, n_b)} \;,$$

each holding for those tuples for which the right-hand side is defined.

Now there is a term $D_{a,b}$ such that

$$D_{a,b}GH_1\cdots H_aN_1\cdots N_b \;\approx\; G(H_1N_1\cdots N_b)(H_2N_1\cdots N_b)---(H_aN_1\cdots N_b) \quad (*)$$

holds for *any* terms $G, H_1, \cdots, H_a, N_1, \cdots, N_b$ .

Assuming this for the moment, and returning to the $G, H_1, \cdots, H_a$ earlier, the term $D_{a,b}GH_1\cdots H_a$ will represent the composition. Just calculate :

$$\begin{aligned} D_{a,b}GH_1\cdots H_a\,\overline{n_1}\cdots\overline{n_b} \;&\approx\; G(H_1\,\overline{n_1}\cdots\overline{n_b})(H_2\,\overline{n_1}\cdots\overline{n_b})---(H_a\,\overline{n_1}\cdots\overline{n_b}) \\ &\approx\; G\;\overline{h_1(n_1,\cdots,n_b)}\;\overline{h_2(n_1,\cdots,n_b)}---\overline{h_a(n_1,\cdots,n_b)} \\ &\approx\; \overline{g(h_1(n_1,\cdots,n_b),---,h_a(n_1,\cdots,n_b))}\ . \end{aligned}$$

To prove $(*)$, we give two arguments, the first being facetious:

Firstly, it's just another one of those equations to be solved using the fixed point operator, except that the number of times the unknown term $D_{a,b}$ appears on the right-hand side of its defining equation is zero. So this is a bit of a phoney application, as we see from the second proof below. In any case, by this argument, we'll set

$$D_{a,b} \;:=\; \underline{Y}(\lambda zuy_1\cdots y_ax_1\cdots x_b \bullet A)\ ,$$

for a suitable set of "$2+a+b$" variables, where

$$A \;:=\; u(y_1x_1\cdots x_b)(y_2x_1\cdots x_b)---(y_ax_1\cdots x_b)\ .$$

The other argument is a straightforward calculation, directly defining

$$D_{a,b} \;:=\; \lambda uy_1\cdots y_ax_1\cdots x_b \bullet u(y_1x_1\cdots x_b)(y_2x_1\cdots x_b)---(y_ax_1\cdots x_b)\ .$$

This more than completes the proof.

The assertion in $(*)$ is one instance of a general phenomenon called *combinatorial completeness*, which we study in the next subsection. There will be a choice between two proofs there as well, exactly analogous to what we did above. The identities in **VII-2.5** are other examples of this phenomenon.

**Theorem VII-5.4** *Any total recursive function is λ-definable.*

**Proof.** We use the fact that any such function can be obtained from the starters discussed below, by a sequence of compositions and minimizations which use as ingredients (and produce) only *total* functions. See [**CM**]. The starters are shown to be λ-definable below, and **VII-5.3** deals with composition. So it remains only to use **VII-5.2** to deal with minimization. Suppose that $p : \mathbf{N}^{n+1} \to \mathbf{N}$ is a total function which is λ-defined by $H$. And assume that for all $(k_1, \cdots, k_n)$, there is a $k$ with $p(k, k_1, \cdots, k_n) = 0$. Define the total function $m_p : \mathbf{N}^n \to \mathbf{N}$ by

$$m_p(k_1, \cdots, k_n) \ := \ \min\{k \mid p(k, k_1, \cdots, k_n) = 0\} \ .$$

Then, if $h^{(g)}$ is the $h$ produced from $g$ in the statement of **VII-5.2**, it is straightforward to see that $m_p = h^{(p)} \circ (\mathrm{zero}_n, \pi_1, \cdots, \pi_n)$, where the last tuple of "$n$"-variable functions consists of the zero constant function and the projections, all λ-definable. But $h^{(p)}$ is λ-definable by **VII-5.2**. So, by closure under composition, the proof is complete.

Let us now have a discussion of the λ-definability of various simple functions, including the starters referred to in this last proof.

It is a total triviality to check that the constant function, mapping all numbers to zero, is defined by the term $\lambda x \bullet \overline{0}$ . We already know that the successor function is defined by $\underline{s}$.

The $i$th projection function is *not* λ-defined by $\underline{ith}$, because we are not bothering to convert to genuine multi-valued functions, defined on tuples, but rather treating them as successively defined using adjointness. (See the initial paragraph of the Digression in Subsection VII-7 for more explanation, if desired.) There is a λ-calculus formalism for going back-and-forth between the two viewpoints, called "currying" and "uncurrying", very handy for functional programming, I believe, but we don't need it here. The $i$th projection function *is* defined by $\lambda x_1 x_2 \cdots x_n \bullet x_i$ , as may be easily checked.

At this point, if we wish to use the starter functions just above, the proof that *every total recursive function is λ-definable* would be completed by showing that the set of λ-definable functions is closed under primitive recursion. The latter is simply the many-variable case of the curried version of **VII-2.11**, and can be safely left to the reader as yet another (by now mechanical) exercise with $\underline{Y}$.

On the other hand, we don't need to bother with primitive recursion if we go back to our original definition of the recursive functions, and also accept the theorem that a total recursive function can be built from starters using only *total* functions at all intermediate steps (as we did in this last proof). But we do need to check that the addition and multiplication functions, and the '$\geq$'-predicate, all of two variables, are $\lambda$-definable, since they were the original starters along with the projections. These are rather basic functions which everybody should see $\lambda$-defined. So we'll now do these examples of using the fixed point operator, plus one showing how we could also have gotten a different predecessor function this way. In each case we'll write down a suitable (informal) **McSELF**-procedure for the function (copying from earlier), then use it to write down the $\lambda$-calculus equation, convert that into a formula for what we've been calling $A$, and the job is then done as usual with an application of the fixed-point operator $\underline{Y}$ .

Here is that list of terms for basic functions mentioned just above, where we've just stolen the informal **McSELF** procedures from an earlier section.

**The addition function.** (See also the exercise before **VII-2.8**)

$$\begin{array}{lll} \mathsf{ADD}(m,n) & \Leftarrow & \text{if } n = 0 \\ & & \text{then } m \\ & & \text{else } \mathsf{ADD}(m+1, n-1) \end{array}$$

So the term $\underline{add}$ must satisfy

$$\underline{add}\ \overline{m}\ \overline{n}\ \approx\ (\underline{isz}\ \overline{n})\overline{m}(\underline{add}(\underline{s}\ \overline{m})(\underline{p}\ \overline{n}))\ .$$

And so we take $\underline{add}\ :=\ \underline{Y}(\lambda zxy \bullet A)$ where $A\ :=\ (\underline{isz}\ y)x(z(\underline{s}\ x)(\underline{p}\ y))$ .

**The multiplication function.** (See also the exercise before **VII-2.8**)

$$\begin{array}{lll} \mathsf{MULT}(m,n) & \Leftarrow & \text{if } n = 0 \\ & & \text{then } 0 \\ & & \text{else } \mathsf{ADD}(m, \mathsf{MULT}(m, n-1)) \end{array}$$

So the term $\underline{mult}$ must satisfy

$$\underline{mult}\ \overline{m}\ \overline{n}\ \approx\ (\underline{isz}\ \overline{n})\overline{0}(\underline{add}\ \overline{m}(\underline{mult}\ \overline{m}(\underline{p}\ \overline{n}))\ .$$

And so we take $\underline{mult}\ :=\ \underline{Y}(\lambda zxy \bullet A)$ where $A\ :=\ (\underline{isz}\ y)\underline{0}(\underline{add}\ x(z\ x(\underline{p}\ y))$ .

**The 'greater than or equal to' predicate.**

$$\begin{array}{lll} \mathsf{GEQ}(m,n) & \Leftarrow & \text{if } n = 0 \\ & & \text{then } 1 \\ & & \text{else} \quad \text{if } m = 0 \\ & & \qquad \text{then } 0 \\ & & \qquad \text{else } \mathsf{GEQ}(m-1, n-1) \end{array}$$

So the term $\underline{geq}$ must satisfy

$$\underline{geq}\ \overline{m}\ \overline{n} \ \approx\ (\underline{isz}\ \overline{n})\overline{1}((\underline{isz}\ \overline{m})\overline{0}(\underline{geq}(\underline{p}\ \overline{m})(\underline{p}\ \overline{n}))) \ .$$

And so we take

$$\underline{geq} \ :=\ \underline{Y}(\lambda zxy \bullet A)$$

where

$$A \ :=\ (\underline{isz}\ y)\underline{1}((\underline{isz}\ x)\underline{0}(z(\underline{p}\ x)(\underline{p}\ y))) \ .$$

This predicate takes values 0 and 1, rather than $\underline{F}$ and $\underline{T}$ . To make that alteration will be left as an exercise.

**The new, non-total, predecessor function.**

We get the new predecessor function PRED (undefined at 0 and otherwise mapping $n$ to $n-1$) by $\mathsf{PRED}(n) = \mathsf{PREPRED}(0,n)$ , where

$$\mathsf{PREPRED}(m,n) \ =\ \begin{cases} n-1 & \text{if } m < n \ ; \\ err & \text{if } m \geq n \ . \end{cases}$$

The latter has **McSELF**-procedure

$$\begin{array}{lll} \mathsf{PREPRED}(m,n) & \Leftarrow & \text{if } m+1 = n \\ & & \text{then } m \\ & & \text{else } \mathsf{PREPRED}(m+1, n) \end{array}$$

To distinguish them from the earlier, total, functions $\underline{p}$ and $\underline{pp}$, we'll name the terms which represent these functions as $\underline{q}$ and $\underline{qq}$. Then the term $\underline{qq}$ must satisfy

$$\underline{qq}\ \overline{m}\ \overline{n} \ \approx\ (\underline{eq}(\underline{s}\ \overline{m})\overline{n})\overline{m}(\underline{qq}(\underline{s}\ \overline{m})\ \overline{n}) \ .$$

And so we take

$$\underline{qq} \ :=\ \underline{Y}(\lambda zxy \bullet A)$$

where

$$A \;:=\; (\underline{eq}(\underline{s}\;x)y)x(z(\underline{s}\;x)\;y)\;.$$

And now for the predecessor itself :

$$\underline{q}\;\overline{n} \;\;\approx\;\; \underline{qq}\;\overline{0}\;\overline{n}\;, \quad \text{so define} \quad \underline{q} \;:=\; \lambda x \bullet \underline{qq}\;\overline{0}\;x\;.$$

These are simpler than $\underline{p}$ and $\underline{pp}$, aren't they?

Now let's get serious about partial recursive functions. The theorem works for any numeral system with a few basic properties, as may be seen by checking through the proofs above and below. So that's how it will be stated:

**Theorem VII-5.5.** *For any choice of numeral system, that is, of terms* $\underline{s}, \underline{p}, \underline{isz}$, *and* $\overline{n}$ *for each* $n \geq 0$, *which satisfy*

$$\underline{s}\;\overline{n} \;\approx\; \overline{n+1} \;\;;\;\; \underline{isz}\;\overline{0} \approx \underline{T} \;\;;\;\; \text{if } n > 0\,, \text{then } \underline{isz}\;\overline{n} \approx \underline{F} \text{ and } \underline{p}\;\overline{n} \;\approx\; \overline{n-1}\;,$$

*every (partial) recursive function is* $\lambda$*-definable.*

But we'll just prove it for the earlier Church numeral system, reviewed below. Were it not for needing a term which 'does a loop', i.e. has no normal form, in 'all the right places', we would at this point have done what was necessary to prove this theorem. For obtaining the needed result about the non-existence of normal forms, it is surprising how many technicalities (just below), and particularly what difficult syntactic theorems (see the proof of the theorem a few pages ahead), seem to be needed .

We begin with a list of new and old definitions, and then four lemmas, before completing the proof.

Confession. Just below there is a slight change from an earlier definition, and also a 'mistake'—a small fib, if you like. We come clean on this right at the end of this subsection, using \sf print. This seems the best way, and it doubles the profit, as we explain there. So most readers, if they discover one or both of these two minor anomalies, should just press on in good humour. But if that is a psychological impossibility, go to the end of the subsection to get relief.

Define inductively, for $\Lambda$-terms $X$ and $Y$ :

$$X^m Y \;:=\; \begin{cases} Y & \text{if } m = 0\;; \\ X^{m-1}(XY) & \text{if } m > 0\;. \end{cases}$$

Thus $X^m Y \;=\; X(X(\cdots\cdots(X(XY))\cdots)) \;=\; XC$ for $C \;=\; X^{m-1}Y$ .

Recall the definition $\overline{m} \;:=\; \lambda xy \bullet x^m y$ .

Note that $x^m y$, despite appearances, doesn't have the form $Ay$ for any $\Lambda$-term $A$, so we cannot apply $(\eta)$-reduction to reduce this to $\lambda x \bullet x^m$ . In fact, $\overline{m}$ is in normal form. (In both cases, $m = 1$ is an exception.) We have

$$\overline{m}\ A\ B \,\overline{\succeq}\, A^m B \ ,$$

as is easily checked for any terms $A$ and $B$, though you *do* have to check that $(x^m y)^{[x \mapsto A][y \mapsto B]} \;=\; A^m B$ , which is not entirely, though almost, a no-brainer.

Recall that $\underline{s} \;:=\; \lambda xyz \bullet (xy)(yz)$ . One can easily directly show that

$\underline{s}\ \overline{m}\ \overline{\succeq}\ \overline{m+1}$ , though it follows from the earlier fact that $\underline{s}\ \overline{m} \;\approx\; \overline{m+1}$ ,

since the right-hand side is in normal form. We have also

$$IX \;\overline{\succeq}\; X \qquad \text{and} \qquad KXY \;\overline{\succeq}\; X \ ,$$

where now $\underline{K} \;:=\; \lambda xy \bullet x$ is denoted just $K$ .

Define $D \;:=\; \lambda xyz \bullet z(Ky)x$ . Then we have

$$DAB\,\overline{0} \;\overline{\succeq}\; \overline{0}(KB)A \;\overline{\succeq}\; (KB)^0 A = A \ ,$$

and, for $i > 0$ ,

$$DAB\,\overline{i} \;\overline{\succeq}\; \overline{i}(KB)A \;\overline{\succeq}\; (KB)^i A = KBC \;\overline{\succeq}\; B \quad , \quad \text{with } C = (KB)^{i-1}A \ .$$

Define

$$T \;:=\; \lambda x \bullet D\overline{0}(\lambda uv \bullet u(x(\underline{s}v))u(\underline{s}v)) \ ,$$

and then define, for any $\Lambda$-terms $X$ and $Y$,

$$PXY \;:=\; TX(XY)(TX)Y \ .$$

If we only needed "$\approx$" and not "$\overline{\succeq}$" below, we could take $P$ as an actual term $\lambda xy \bullet Tx(xy)(Tx)y$ . Note that $P$ will never occur on its own, only in the form $PXY$ .

**Lemma A.** *For $X$ and $Y$ in $\Lambda$ with $XY \ \overline{\succeq}\ \overline{i}$ , we have*

$$PXY \ \overline{\succeq}\ \begin{cases} Y & \text{if } i = 0 \ ; \\ PX(\underline{s}Y) & \text{if } i > 0 \ . \end{cases}$$

*Also, in each case, the "$\overline{\succeq}$" may be realized by a sequence of $(\beta)$-reductions, the first of which obliterates the leftmost occurrence of $\lambda$ .*

**Proof.** Using, in the first step, the definitions of $PXY$ and of $T$, and a leftmost $(\beta)$-reduction,

$$PXY \ \overline{\succeq}\ D\overline{0}(\lambda uv\bullet u(X(\underline{s}v))u(\underline{s}v))(XY)(TX)Y$$

$$\overline{\succeq}\ D\overline{0}(\lambda uv \bullet u(X(\underline{s}v))u(\underline{s}v))\overline{i}(TX)Y \quad \cdots \qquad \text{(to be continued)}$$

Now when $i = 0$ this continues

$$\overline{\succeq}\ \overline{0}(TX)Y \qquad\qquad [\text{since } \ DAB\,\overline{0}\ \overline{\succeq}\ A]$$

$$\overline{\succeq}\ (TX)^0Y \ = \ Y \ .$$

When $i > 0$ it continues

$$\overline{\succeq}\ (\lambda uv \bullet u(X(\underline{s}v))u(\underline{s}v))(TX)Y \quad [\text{since } DAB\,\overline{i}\ \overline{\succeq}\ B]$$

$$\overline{\succeq}\ TX(X(\underline{s}Y))(TX)(\underline{s}Y) \ = \ PX(\underline{s}Y) \ ,$$

as required, completing the proof.

Now suppose that $g : \mathbf{N} \to \mathbf{N}$ and $h : \mathbf{N}^{n+1} \to \mathbf{N}$ are total functions, and that $G, H$ in $\Lambda$ are such that

$$G\overline{j} \ \overline{\succeq}\ \overline{g(j)} \quad \text{and} \quad H\overline{k}_1\overline{k}_2\cdots\overline{k}_n\overline{k} \ \overline{\succeq}\ \overline{h(k_1, k_2, \cdots k_n, k)}$$

for all $j, k_1, k_2, \cdots k_n, k$ in $\mathbf{N}$. Define $f : \mathrm{dom}(f) \to \mathbf{N}$ by

$$f(k_1, k_2, \cdots k_n) \ := \ g(\min\{\ k \mid h(k_1, k_2, \cdots k_n, k) = 0\ \}) \ ,$$

so

$$\mathrm{dom}(f) \ = \ \{\ (k_1, k_2, \cdots k_n) \mid \exists k \text{ with } h(k_1, k_2, \cdots, k_n, k) = 0\ \} \ .$$

**Lemma B.** *Given $m$ such that $h(k_1, k_2, \cdots k_n, \ell) \neq 0$ for $0 \leq \ell < m$ , we have, for these $\ell$,*

(i) $P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)\overline{\ell} \quad \overline{\succeq} \quad P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)\overline{\ell+1}$ ,

*where each such " $\overline{\succeq}$ " may be realized by a sequence of $(\beta)$-reductions at least one of which obliterates the leftmost occurrence of $\lambda$ ; and*

(ii) $P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)\overline{0} \quad \overline{\succeq} \quad P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)\overline{m}$ .

**Proof.** (ii) is clearly immediate from (i). As for the latter, with $X = H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n$ and $Y = \overline{\ell}$, we have

$$XY \;=\; H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n\overline{\ell} \;\overline{\succeq}\; \overline{h(k_1, k_2, \cdots k_n, \ell)} \;=\; (\text{say})\; \overline{i}\ ,$$

where $i > 0$ . And so

$$\begin{aligned} P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)\overline{\ell} \;&\overline{\succeq}\; P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)(\underline{s}\overline{\ell}) \quad \text{by Lemma } \mathbf{A} \text{ (second part)} \\ &\overline{\succeq}\; P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)\overline{\ell+1} \quad \text{since} \quad \underline{s}\overline{\ell} \;\overline{\succeq}\; \overline{\ell+1}\ . \end{aligned}$$

**Lemma C.** *If $(k_1, k_2, \cdots k_n)$ is such that there is a $k$ with $h(k_1, k_2, \cdots k_n, k) = 0$ , and we define*

$$m \;:=\; \min\{\ k \mid h(k_1, k_2, \cdots k_n, k) = 0\ \}\ ,$$

*then* $P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)\overline{0} \;\overline{\succeq}\; \overline{m}$ .

**Proof.** Using Lemma B(ii), and then using Lemma A (first part) with $X = H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n$ and $Y = \overline{m}$, which can be done since

$$XY \;=\; H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n\overline{m} \;\overline{\succeq}\; \overline{h(k_1, k_2, \cdots k_n, m)} \;=\; \overline{0}\ ,$$

we get

$$P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)\overline{0} \;\overline{\succeq}\; P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)\overline{m} \;\overline{\succeq}\; \overline{m}\ .$$

**Main Lemma D.** (a) *Let*

$$L \;:=\; \lambda x_1 \cdots x_n y \bullet P(Hx_1 \cdots x_n)y \;\text{ and }\; F \;:=\; \lambda x_1 \cdots x_n \bullet G(Lx_1 \cdots x_n \overline{0}) \;.$$

*Then*

$$\forall (k_1, \cdots, k_n) \in \mathrm{dom}(f) \;\text{ we have }\; F\overline{k}_1\overline{k}_2 \cdots \overline{k}_n \;\overline{\succeq}\; \overline{f(k_1, k_2, \cdots k_n)} \qquad (*)$$

(b) *For any $F$ satisfying $(*)$, define*

$$E \;:=\; \lambda x_1 \cdots x_n \bullet P(Hx_1 \cdots x_n)\overline{0}\; I(Fx_1 \cdots x_n) \;.$$

*Then* (i) *the reduction $(*)$ also holds with $E$ in place of $F$ ; and*

(ii) *for all $(k_1, \cdots, k_n) \notin \mathrm{dom}(f)$, there is an infinite sequence of $(\beta)$-reductions, starting with $E\overline{k}_1\overline{k}_2 \cdots \overline{k}_n$ , and such that, infinitely often, it is the leftmost $\lambda$ which is obliterated.*

**Proof.** (a) We have

$$\begin{aligned}
F\overline{k}_1\overline{k}_2 \cdots \overline{k}_n \;&\overline{\succeq}\; G(L\overline{k}_1\overline{k}_2 \cdots \overline{k}_n\overline{0}) && \text{[definition of } F\text{]} \\
&\overline{\succeq}\; G(P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)\overline{0}) && \text{[definition of } L\text{]} \\
\overline{\succeq}\; G\overline{m} \quad &\text{[where } m = \min\{\, k \mid h(k_1, k_2, \cdots k_n, k) = 0 \,\} \text{ by Lemma } \mathbf{C}\text{]} \\
\overline{\succeq}\; \overline{g(m)} \;&=\; \overline{g(\min\{\, k \mid h(k_1, k_2, \cdots k_n, k) = 0 \,\})} \;=\; \overline{f(k_1, k_2, \cdots k_n)} \;.
\end{aligned}$$

(b)(i) We have

$$\begin{aligned}
E\overline{k}_1\overline{k}_2 \cdots \overline{k}_n \;&\overline{\succeq}\; P(H\overline{k}_1\overline{k}_2 \cdots \overline{k}_n)\overline{0}I(F\overline{k}_1\overline{k}_2 \cdots \overline{k}_n) && \text{[definition of } E\text{]} \\
&\overline{\succeq}\; \overline{m}I(F\overline{k}_1\overline{k}_2 \cdots \overline{k}_n) && [m \text{ as above, by Lemma } \mathbf{C}] \\
&\overline{\succeq}\; I^m(F\overline{k}_1\overline{k}_2 \cdots \overline{k}_n) && [\text{since } \overline{m}XY \;\overline{\succeq}\; X^mY] \\
&\overline{\succeq}\; F\overline{k}_1\overline{k}_2 \cdots \overline{k}_n && [\text{since } IX \;\overline{\succeq}\; X] \\
&\overline{\succeq}\; \overline{f(k_1, k_2, \cdots k_n)} && [\text{by } (*) \text{ for } F] \;.
\end{aligned}$$

(b)(ii) We have

$$
\begin{aligned}
E\overline{k}_1\overline{k}_2\cdots\overline{k}_n \;\overline{\succeq}\; & P(H\overline{k}_1\overline{k}_2\cdots\overline{k}_n)\;\overline{0}\;I(F\overline{k}_1\overline{k}_2\cdots\overline{k}_n) && \text{[definition of } E\text{]}\\
\overline{\succeq}\; & P(H\overline{k}_1\overline{k}_2\cdots\overline{k}_n)\;\overline{1}\;I(F\overline{k}_1\overline{k}_2\cdots\overline{k}_n) && \text{[repeatedly applying}\\
\overline{\succeq}\; & P(H\overline{k}_1\overline{k}_2\cdots\overline{k}_n)\;\overline{2}\;I(F\overline{k}_1\overline{k}_2\cdots\overline{k}_n) && \text{Lemma } \mathbf{B}\text{(i), and}\\
\overline{\succeq}\; & P(H\overline{k}_1\overline{k}_2\cdots\overline{k}_n)\;\overline{3}\;I(F\overline{k}_1\overline{k}_2\cdots\overline{k}_n) && \text{noting that each step}\\
\overline{\succeq}\; & P(H\overline{k}_1\overline{k}_2\cdots\overline{k}_n)\;\overline{4}\;I(F\overline{k}_1\overline{k}_2\cdots\overline{k}_n) && \text{here is doable with}\\
\overline{\succeq}\; & P(H\overline{k}_1\overline{k}_2\cdots\overline{k}_n)\;\overline{5}\;I(F\overline{k}_1\overline{k}_2\cdots\overline{k}_n) && \text{at least one leftmost}\\
\overline{\succeq}\; & P(H\overline{k}_1\overline{k}_2\cdots\overline{k}_n)\;\overline{6}\;I(F\overline{k}_1\overline{k}_2\cdots\overline{k}_n) && (\beta)-\text{reduction, giving}\\
\overline{\succeq}\; & P(H\overline{k}_1\overline{k}_2\cdots\overline{k}_n)\;\overline{7}\;I(F\overline{k}_1\overline{k}_2\cdots\overline{k}_n) && \text{infinitely many,}\\
\overline{\succeq}\; & P(H\overline{k}_1\overline{k}_2\cdots\overline{k}_n)\;\overline{8}\;I(F\overline{k}_1\overline{k}_2\cdots\overline{k}_n) \quad \bullet\bullet\bullet && \text{as required.]}
\end{aligned}
$$

**Proof of Theorem VII-5.5**—*Any (partial) recursive function is λ-definable.* Let $f$ be such a function, and use Kleene's theorem (see [**CM**], Section IV) to write

$$f(k_1, k_2, \cdots k_n) \;=\; g(\min\{\; k \mid h(k_1, k_2, \cdots k_n, k) = 0 \;\})\;,$$

for primitive recursive functions $g$ and $h$. These two are, in particular, *total* recursive functions. So we can, by **VII-5.4**, find $G$ and $H$ in $\Lambda$ such that

$$G\overline{j} \;\approx\; \overline{g(j)} \qquad \text{and} \qquad H\overline{k}_1\overline{k}_2\cdots\overline{k}_n\overline{k} \;\approx\; \overline{h(k_1, k_2, \cdots k_n, k)}$$

for all $j, k_1, k_2, \cdots k_n, k$ in $\mathbf{N}$. But since the right-hand sides are already in normal form, by the normal form theorem we can change "$\approx$" to "$\overline{\succeq}$" in both places above. But now, using parts of Lemma **D**, first get an $F$ as in part (a), and then use it to get an $E$ as in part (b). Part (b)(i) gives the "$\approx$" equation required for $(k_1, \cdots, k_n) \in \mathrm{dom}(f)$ (and more, since it even gives a specific way to reduce $E\overline{k}_1\overline{k}_2\cdots\overline{k}_n$ to $\overline{f(k_1, k_2, \cdots k_n)}$, dependent on having similar reductions for $G$ and $H$—see the exercises below as well). For $(k_1, \cdots, k_n) \notin \mathrm{dom}(f)$, part (b)(ii) gives an infinite reduction sequence starting with $E\overline{k}_1\overline{k}_2\cdots\overline{k}_n$ , and such that, infinitely often, it is the leftmost $\lambda$ which is obliterated. Thus it is a "quasi-leftmost reduction" in the sense of [**Kl**], because of leftmost $(\beta)$-reductions occurring arbitrarily far out. Thus,

by Cor 5.13, p.293 of that thesis, $E\overline{k}_1\overline{k}_2\cdots\overline{k}_n$ has no normal form (in the *extensional* $\lambda$-calculus), as required.

It seems interesting how much is involved in this proof. I've not seen any essentially different proof; in fact I've not seen any other detailed proof at all in the extensional case. This one is essentially that in [**HS**], except they deal with the *intensional* $\lambda$-calculus. And they are acknowledged experts. One can well imagine manœuvering to avoid Kleene's theorem. But Klop's quasi-leftmost normalization theorem seems to be essential, and its proof apparently dates from only around 1980. It is stated as a problem by Hindley [**Hi**] in 1978. Furthermore, it seems always to be the extensional $\lambda$-calculus with which the CSers work in hard-core denotational semantics. I would assume that a definition of "computable" phrased in terms of the extensional $\lambda$-calculus was one which was long accepted before 1980. This may be a situation somewhat analogous to that with the Littlewood-Richardson rule from algebraic combinatorics/representation theory. Those two only stated the rule in the 1930's as an empirically observed phenomenon. A proof or proofs with large gaps appeared long before 1970, but only around then did acceptable proofs begin to appear. A joke I heard from Ian Macdonald has astronauts landing on the moon and (fortunately) returning, by use of technology dependent on the Littlewood-Richardson rule, apparently well before a genuine proof was known!

In any case, let me own up again—we have, in this write-up, depended on three basic, difficult syntactic properties of the extensional $\lambda$-calculus (unproved here) :

the Church-Rosser theorem and its consequence that normal forms are unique;

the normalization theorem, that when $M$ has a normal form $N$, the sequence of leftmost reductions starting with $M$ terminates with $N$ ;

and Klop's theorem above, implying that, if an infinite quasi-leftmost reduction starting from $M$ exists, then $M$ has no normal form (which theorem also says that we can weaken the word "leftmost" to "quasi-leftmost" in the preceding statement about those $M$'s which *do* have normal forms).

Coming clean. The minor change, from earlier definitions in VII-2, made just after the "Confession", is that here in VII-4 we defined $X^mY$ a bit differently. The new definition agrees with the old up to "$\approx$", so everything stated earlier still

holds using the new definition. Since they are defined using $x^m y$, the numerals $\overline{m}$ have now been changed slightly, so there are actually quite a few earlier results to which that comment applies. It just seemed to the author that in VII-2 it would avoid fuss if $X^m$ actually had a meaning on its own as a term. From the "Confession" onwards, that definition should be ignored—$X^m$ never occurs without being followed by a term $Y$.

The mistake referred to in the "Confession" is that, in the extensional $\lambda$-calculus, though all the other numerals are, the numeral $\overline{1}$ is NOT in normal form, despite the claim. It can be $(\eta)$-reduced to the term we are denoting as $I$. There is only one place where this affects the proofs above, as indicated in the paragraph after next.

But first let's double our money. The reader may have noticed that $(\eta)$-reduction never comes in anywhere in these proofs—it's always $(\beta)$-reduction. Furthermore, the numeral $\overline{1}$, in the intensional $\lambda$-calculus, IS in normal form. So, using the corresponding three syntactic theorems from just a few paragraphs above (whose proofs are actually somewhat easier in the intensional case), we have a perfectly good and standard proof that every recursive function is $\lambda$-definable in the intensional $\lambda$-calculus.

The only place where the non-normality of the numeral $\overline{1}$ is problematic, in the proofs above for the extensional $\lambda$-calculus, is in the proof of **VII-5.5**, where we go from asserting "$\approx$" to asserting "$\overline{\Sigma}$", when it happens that the right-hand side is $\overline{1}$. But just use the above result for the intensional $\lambda$-calculus to see that we can get defining terms for which these reductions exist. So that takes care of the small anomaly (and it did seem simplest to ignore it and to be a bit intellectually dishonest until now).

**Sketch of the proof that $\lambda$-definable $\Longrightarrow \mathcal{RC}$.**

This can be done following the pattern which we used much earlier in [**CM**] to show $\mathcal{BC} \Longrightarrow \mathcal{RC}$ (i.e. Babbage computable implies recursive). Let $A \in \Lambda$ . Define functions $f_{n,A} : D_{n,A} \to \mathbf{N}$ as follows:

$$D_{n,A} \;:=\; \{\; \vec{v} \mid A\overline{v}_1 \cdots \overline{v}_n \approx \overline{\ell} \;\text{ for some } \ell \in \mathbf{N} \;\} \;\subset\; \mathbf{N}^n \;;$$

and since, by Church-Rosser, such an $\ell$ is unique (given $\vec{v}$), define

$$f_{n,A}(\vec{v}) \;:=\; \ell \;.$$

Every $\lambda$-definable function has the form $f_{n,A}$ for some $(n, A)$ [though the function $f_{n,A}$ itself might not actually be definable using the term $A$, if $A$ is not chosen so that $A\overline{v}_1 \cdots \overline{v}_n$ "loops" for all $\vec{v} \notin D_{n,A}$ ] . However, we prove that *all* $f_{n,A}$ are recursive.

In fact, as in the case of $\mathcal{BC}$-computable functions (but using lower-case names so as not to confuse things), one is able to write

$$f_{n,A}(\vec{v}) \;=\; \mathsf{print}(\min\{h \mid \mathsf{kln}(g_A, \vec{v}, h) = 1 \;\}) \;.$$

The functions kln and print will be defined below in such a way that it is *manifestly believable* that they are primitive recursive. This will use Gödel numbering for terms in $\Lambda$ ; and $g_A$ above is the Gödel number of $A$ . To change "*believable*" to *true* is a tedious but unsubtle process, analogous to much earlier material, especially the proof of primitive recursiveness for $KLN$ and $PRINT$. The proof here for kln and print will be omitted. Via the primitive recursiveness of kln and print, the recursiveness of $f_{n,A}$ follows immediately from the above display. [Note that this will also give a new proof of Kleene's normal form theorem for recursive functions, using the $\lambda$-calculus in place of **ATEN**.]

So here at least are the needed definitions. The argument using the definitions below works in the intensional $\lambda$-calculus, just as well as in our case, the extensional $\lambda$-calculus.

As for Gödel numbering of terms, define, somewhat arbitrarily, using the coding of finite strings of natural numbers from the Appendix before **IV-3** in **[CM]** :

$$G\ddot{o}d(x_i) =< 1, i > \;;\; G\ddot{o}d(AB) =< 2, G\ddot{o}dA, G\ddot{o}dB > \;;\; G\ddot{o}d(\lambda x_i \bullet A) =< 3, i, G\ddot{o}dA > \;.$$

Now define kln to be the relation for which

$$\mathsf{kln}(g, \vec{v}, h) = 1 \iff \exists A \in \Lambda \text{ satisfying the following :}$$

(i) $g \;=\; G\ddot{o}d(A)$ ;
(ii) $A\overline{v}_1 \cdots \overline{v}_n$ has a finite leftmost reduction leading to a term of the form $\overline{\ell}$;

(iii) $h$ is the code of the history of that leftmost reduction :
that is, if the reduction is

$$A\overline{v}_1 \cdots \overline{v}_n = B_1 \ \overline{\succeq} \ B_2 \ \overline{\succeq} \ B_3 \ \overline{\succeq} \ \cdots \ \overline{\succeq} \ B_k = \overline{\ell} \ ,$$

then $h =< b_1, \cdots, b_k >$, where $b_i = G\ddot{o}d(B_i)$ .

Finally $\mathsf{print} : \mathbf{N} \to \mathbf{N}$ may be suitably defined (see the next exercise) so that

$$\mathsf{print}(< b_1, \cdots, b_k >) = \ell \ \text{ if } \ b_k = G\ddot{o}d(\overline{\ell}) \ .$$

**Exercises to define print.**

Take $\overline{\ell}$ explicitly to be $\lambda x_2 \bullet (\lambda x_1 \bullet x_2{}^{\ell} x_1)$.

(i) Show that

$$G\ddot{o}d(\overline{3}) \ = \ < 3, 2, < 3, 1, < 2, < 1, 2 >, < 2, < 1, 2 >, < 2, < 1, 2 >, < 1, 1 >>>>>> \ .$$

(ii) Find a primitive recursive function $\Phi$ so that $\Phi(< 1, 1 >) = 0$ and $\Phi(< a, b, x >) = 1 + \Phi(x)$ for all $a, b$ and $x$.

(iii) Define $\Psi(k) \ := \ \Phi(CLV(CLV(k, 3), 3))$ . Prove : $\forall \ell, \ \Psi(G\ddot{o}d(\overline{\ell})) = \ell$ .

(iv) Defining

$$\mathsf{print}(x) \ := \ \Psi(CLV(x, CLV(x, 0))) \ ,$$

check that you have a primitive recursive function with the property displayed just before the exercise.

Both the $\lambda$-calculus and the theory of combinators were originally developed as foundations for mathematics before digital computers were invented. They languished as obscure branches of mathematical logic until rediscovered by computer scientists. It is remarkable that a theory developed by logicians has inspired the design of both the hardware and software for a new generation of computers. There is an important lesson here for people who advocate reducing support for 'pure' research: the pure research of today defines the applied research of tomorrow.

•
•
•

$\cdots$ David Turner proposed that Schönfinkel and Curry's combinators could be used as machine code for computers for executing functional programming languages. Such computers could exploit mathematical properties of the $\lambda$-calculus $\cdots$

•
•
•

We thus see that an obscure branch of mathematical logic underlies important developments in programming language theory, such as:

(i) The study of fundamental questions of computation.

(ii) The design of programming languages.

(iii) The semantics of programming languages.

(iv) The architecture of computers.

Michael Gordon [**Go1**]

## VII-6 Combinatorial Completeness and the Invasion of the Combinators.

Let $\Omega$ be a set with a binary operation, written as juxtaposition. We shall be discussing such objects a lot, and that's what we'll call them, rather than some subname of hemi-demi-semi-quasiloopoid/groupoid/algebraoid, or indeed applicative set, though the latter is suggestive of what we are trying to understand here. The set $\Omega^{\Omega^n}$ consists of all functions of "$n$" variables from $\Omega$, taking values in $\Omega$, i.e. functions $\Omega^n \to \Omega$ . It has a rather small subset consisting of those functions 'algebraically definable just using the operation'—for example,

$$(\omega_1, \omega_2, \omega_3) \mapsto (\omega_2((\omega_3\omega_3)(\nu\omega_1)))\omega_2 \quad ,$$

where $\nu$ is some fixed element of $\Omega$. [We'll express this precisely below.] The binary operation $\Omega$ is said to be combinatorially complete if and only if every such "algebraically definable" function can be given as left multiplication by at least one element of $\Omega$ (i.e. 'is representable'). For example, there would have to be an element $\zeta$ such that, for all $(\omega_1, \omega_2, \omega_3)$ we have

$$\zeta\omega_1\omega_2\omega_3 \;=\; (\omega_2((\omega_3\omega_3)(\nu\omega_1)))\omega_2 \; .$$

We are using the usual convention here on the left, that is,

$$\zeta\omega_1\omega_2\omega_3 \;:=\; ((\zeta\omega_1)\omega_2)\omega_3 \; .$$

Note that the cardinality of $\Omega^{\Omega^n}$ is strictly bigger than that of $\Omega$ except in the trivial case that $\Omega$ has only one element, so we cannot expect this definition to lead anywhere without some restriction on which functions are supposed to be realizable by elements $\zeta$ .

We shall show quite easily in the next section that $\Lambda/\approx$ is combinatorially complete, where the binary operation is the one inherited from $\Lambda$. Then we introduce an ostensibly simpler set $\Gamma$, basically the combinators extended with variables. This produces the original combinatorially complete binary operation, namely $\Gamma/\sim$ , due to Schonfinkel, who invented the idea. He didn't prove it this way, since the $\lambda$-calculus hadn't been invented yet, but we do it in the second section below by showing that the two binary operations $\Lambda/\approx$ and $\Gamma/\sim$ are isomorphic. So we don't really have two different examples here, but the new description using $\Gamma$ is simpler in many respects.

Reduction algorithms for "$\sim$" in $\Gamma$ have actually been used in the design of computers, when the desire is for a machine well adapted to functional programming languages [**Go1**].

**Combinatorial Completeness of $\Lambda/\approx$ .**

First let's give a proper treatment of which functions we are talking about, that is, which are *algebraically definable*. The definition below is rather formal. It makes proving facts about these functions easy by induction on that inductive definition. However, for those not hidebound by some constructivist philosophy, there are simpler ways of giving the definition—"...the smallest set closed under pointwise multiplication of functions, and containing all the projections and all constant functions...". See the axiomatic construction of combinatorially complete binary operations near the beginning of Subsection VII-8 for more on this.

If $\Theta$ is any set, then the free binary operation generated by $\Theta$, where we use $*$ to denote the operation, is the smallest set, $FREE(\Theta)$, of non-empty finite strings of symbols from $\Theta \cup \{ \ ( \ , \ * \ , \ ) \ \}$ (a disjoint union of course!) such that each element of $\Theta$ is such a (length 1) string, and the set is closed under $(g, h) \mapsto (g{*}h)$ . So it consists of strings such as $((\theta_3{*}((\theta_1{*}\theta_2){*}\theta_4)){*}\theta_3)$.

Now suppose given a binary operation on $\Omega$ as in the introduction, so here the operation is denoted by juxtaposing. Let $\{v_1, v_2, \cdots\}$ be a sequence of 'variables', all distinct, and disjoint from $\Omega$, and, for each $n \geq 1$, take $\Theta$ to be the set $\Omega \ \cup \ \{v_1, v_2, \cdots, \ v_n\}$.

Now define a function

$$FUN = FUN_n \ : \ FREE(\Omega \ \cup \ \{v_1, v_2, \cdots, v_n\}) \ \rightarrow \Omega^{\Omega^n}$$

by mapping each element of $\Omega$ to the corresponding constant function; mapping each $v_i$ to the $i$th projection function; and finally requiring that

$$FUN(A * B) \ = \ FUN(A)FUN(B) \ .$$

On the right-hand side, we're using the operation in $\Omega$ . (So actually $FUN$ is the unique morphism of binary operations which acts as specified on the length 1 strings, the generators of the free binary operation, and where the binary operation in $\Omega^{\Omega^n}$ is pointwise multiplication using the given operation in $\Omega$.) Then the functions algebraically definable using that operation are

defined to be those which are in the image of the $FUN_n$ for some $n \geq 1$ . In the example beginning this section, the function given is

$$FUN_3 \; [ \; (v_2 * ((v_3 * v_3) * (\nu * v_1))) * v_2 \; ] \; .$$

So let's formalize the earlier definition:

**Definition.** The binary operation $\Omega$ is *combinatorially complete* if and only if, for all $n$ and for all $f \in FREE(\Omega \cup \{ v_1 , v_2 , \cdots , v_n \})$ , there is a $\zeta \in \Omega$ such that, for all $(\omega_1, \omega_2, \cdots, \omega_n)$ we have

$$\zeta\omega_1\omega_2 \cdots \omega_n \; = \; FUN_n(f)(\omega_1, \omega_2, \cdots, \omega_n) \; .$$

**Theorem VII-6.1.** *The binary operation on* $\Lambda/\approx$ *, induced by the juxtaposition operation on* $\Lambda$*, is combinatorially complete.*

**Proof.** Fix $n$, and proceed by induction on $f$ in the definition above. The initial cases require us to find $\zeta$'s which work for the constant functions and for the projections, something which we did in the last section. For the inductive step, let $f = (g * h)$ , where we assume that we have $\zeta_g$ and $\zeta_h$ which work for $g$ and $h$ respectively. But now the right-hand side in the definition above is just

$$FUN_N(g * h)(\omega_1, \omega_2, \cdots, \omega_n) \; =$$

$$FUN_N(g)(\omega_1, \omega_2, \cdots, \omega_n) FUN_N(h)(\omega_1, \omega_2, \cdots, \omega_n) \; =$$

$$(\zeta_g\omega_1\omega_2 \cdots \omega_n)(\zeta_h\omega_1\omega_2 \cdots \omega_n) \; .$$

So we must be able to solve the following equation for the unknown $\zeta$, where $\zeta_g$ and $\zeta_h$ are fixed, and the equation should hold for all $(\omega_1, \omega_2, \cdots, \omega_n)$ :

$$\zeta\omega_1\omega_2 \cdots \omega_n \; = \; (\zeta_g\omega_1\omega_2 \cdots \omega_n)(\zeta_h\omega_1\omega_2 \cdots \omega_n) \; .$$

Perhaps your initial reaction is to do this as just another application of the fixed-point operator; and that will work perfectly well, but is overkill, since the unknown occurs exactly zero times on the right-hand side. A straightforward solution is simply

$$\zeta \; = \; \lambda x_1 x_2 \cdots x_n \bullet (\zeta_g x_1 x_2 \cdots x_n)(\zeta_h x_1 x_2 \cdots x_n) \; .$$

This is a bit sloppy, confusing work in $\Lambda$ with work in $\Lambda/\approx$ . We should have said that $\zeta$ is the equivalence class of $Z$, where, if $Z_g$ and $Z_h$ are elements in the equivalence classes $\zeta_g$ and $\zeta_h$ , respectively, we define

$$Z \;=\; \lambda x_1 x_2 \cdots x_n \bullet (Z_g x_1 x_2 \cdots x_n)(Z_h x_1 x_2 \cdots x_n) \ .$$

Verification is a straightforward $\lambda$-calculation, which is a bit quicker if you apply **VII-1.2** .

**Combinators.**

Let $S$ and $K$ be two distinct symbols, disjoint from our original sequence $x_1, x_2, \cdots$ of distinct variables (which, though it won't come up directly, are quite distinct from the variables $v_i$ of the last section).

**Definition.** The binary operation $\Gamma$ is written as juxtaposition, and is defined to be the free binary operation generated by the set $\{S, K, x_1, x_2, \cdots\}$. Its elements are called by various names : *combinatory terms* [**Sö**][**Ko**], *combinations* [**Sö**], **CL-***terms* [**Ba**], *combinatorial expressions* [**Go**]. We won't need a name for them as individuals. The combinators are the elements of the subset of $\Gamma$ which is the free binary operation generated by $\{S, K\}$. Thus combinators are just suitable strings of $S$'s and $K$'s and lots of brackets. We shall again use the convention that in $\Gamma$, the string $P_1 P_2 P_3 \cdots P_n$ really means $((\cdots((P_1 P_2)P_3)\cdot\cdots\cdot)P_n)$ .

Finally, let $\sim$ be the equivalence relation on $\Gamma$ generated by the following conditions, where we are requiring them to hold for *all* elements $A, B, A_1$, etc. in $\Gamma$ :

$$SABC \sim (AC)(BC) \quad ; \quad KAB \sim A \quad ;$$
$$A_1 \sim B_1 \ \text{ and } \ A_2 \sim B_2 \quad \Longrightarrow \quad A_1 A_2 \sim B_1 B_2 \ ;$$

and

$$Ax \sim Bx \ \Longrightarrow \ A \sim B \ , \text{ if the variable } x \text{ appears in neither } A \text{ nor } B \ .$$

**Remarks.** (i) See the proof of **VII-6.12** below for a more explicit description of $\sim$ .

(ii) Don't be fooled by the 'argument', that the last (extensionality) condition is redundant, which tells you to just multiply on the left by $K$ and apply the second and third conditions : the point is that $K(Ax)$ is a lot different than $KAx$.

(iii) The requirement that $x$ isn't in $A$ or $B$ cannot be removed from the last condition defining $\sim$ . For we have $K(xx)x \sim xx$, but $K(xx) \not\sim x$. If the latter was false, it would follow from the results below in several ways that $K(BB) \sim B$ for any $B$. Taking $B = I = SKK$, it would follow that $KI \sim I$, and then multiplying on the right by $A$, that $I \sim A$ for any $A$. This would show that $\sim$ relates any two elements. That contradicts the main result below that the $\sim$-classes are in 1-1 correspondence with the $\approx$-classes from $\Lambda$ : The Church-Rosser theorem tells us that there are tons of $\approx$-classes.

(iv) Apologies to the $\lambda$-experts for avoiding a number of arcane subtleties, both here and previously. In particular, whether to assume that last condition, or something weaker such as nothing at all, occupies a good deal of the literature. This is the great *intensionality versus extensionality* debate.

> To make the ascent from mathematics to logic is to pass from the object language to the metalanguage, or, as it might be said without jargon, to stop toying around and start believing in something ... A function could be a scheme for a type of process which would become definite when presented with an argument ... Two functions that are extensionally the same might be 'computed', however, by quite different processes ... The mixture of abstract objects needed would obviously have to be very rich, and I worry that it is a quicksand for foundational studies ... Maybe after sufficient trial and error we can come to agree that intensions have to be believed in, not just reconstructed, but I have not yet been able to reach that higher state of mind.
>
> Dana Scott ([**SAJM-ed**], pp. 157-162)

But, for the intended audience here, that debate is simply an unnecessary complication on first acquaintance with the subject, I believe.

**Theorem VII-6.2.** *The binary operation on* $\Gamma/\sim$ *, induced by the juxtaposition operation on* $\Gamma$*, is combinatorially complete.*

More examples arising from combinatorial completeness.

The top half of the following table is already familiar. On all lines, that the $\lambda$-version gives the effect is immediate. The reader might enjoy the following exercises :

(1) Show that the definition gives the effect; that is, work directly with combinator identities rather than the $\lambda$-calculus.

(2) Use the $\lambda$-versions of the ingredients in the definition column, and reduce that $\lambda$-term to normal form, which should be the entry in the 3rd column, up to re-naming bound variables.

It should be understood that the variables in each entry of the third column are distinct from each other.

| combinator | definition | normal $\lambda$-version | effect |
|---|---|---|---|
| $K$ | | $\lambda xy \bullet x$ | $K\alpha\beta \trianglerighteq \alpha$ |
| $S$ | | $\lambda xyz\bullet(xz)(yz)$ | $S\alpha\beta\gamma \trianglerighteq (\alpha\gamma)(\beta\gamma)$ |
| $I$ | $SKK$ | $\lambda x \bullet x$ | $I\alpha \trianglerighteq \alpha$ |
| $B$ | $S(KS)K$ | $\lambda xyz\bullet x(yz)$ | $B\alpha\beta\gamma \trianglerighteq \alpha(\beta\gamma)$ |
| $W$ | $SS(KI)$ | $\lambda xy\bullet xyy$ | $W\alpha\beta \trianglerighteq \alpha\beta\beta$ |
| $C$ | $S(BBS)(KK)$ | $\lambda xyz\bullet xzy$ | $C\alpha\beta\gamma \trianglerighteq \alpha\gamma\beta$ |
| $G$ | $B(BS)B$ | $\lambda xyzw\bullet x(yw)(zw)$ | $G\alpha\beta\gamma\delta \trianglerighteq \alpha(\beta\delta)(\gamma\delta)$ |
| $E$ | $B(BW(BC))(BB(BB))$ | $\lambda xyzw\bullet x(yz)(yw)$ | $E\alpha\beta\gamma\delta \trianglerighteq \alpha(\beta\gamma)(\beta\delta)$ |

**Exercises.** (i) Find a combinator whose normal $\lambda$-version is $\lambda xy \bullet yx$ .
(i) Try to find a combinator whose normal $\lambda$-version is the term

$$\underline{ADD} \;=\; \lambda uvxy \bullet (ux)(vxy) \;.$$

Recall that $\underline{ADD}\;\overline{k}\;\overline{\ell} \;=\; \overline{k+\ell}$ , from the exercise before **VII-2.8**, where it had a different name. Referring to the second part of that exercise, find also a combinator for multiplication.

As mentioned earlier, the first proof of **VII-6.2** will not be direct. Rather we'll show that $\Gamma/\sim$ is isomorphic to $\Lambda/\approx$ . That is, there is a bijective function between them which preserves the binary operations. Since combinatorial completeness is clearly invariant up to isomorphism, by **VII-6.1** this is all we need (and of course it gives us the extra information that we haven't, strictly speaking, produced a *new* example of combinatorial completeness).

To do this we need only define functions

$$\Psi : \Gamma \to \Lambda \quad \text{and} \quad \Phi : \Lambda \to \Gamma$$

such that the following five results hold :

**VII-6.3.** *If $P \sim Q$ then $\Psi(P) \approx \Psi(Q)$.*

**VII-6.4.** *If $A \approx B$ then $\Phi(A) \sim \Phi(B)$.*

**VII-6.5.** *For all $P \in \Gamma$, we have $\Phi\Psi(P) \sim P$.*

**VII-6.6.** *For all $A \in \Lambda$, we have $\Psi\Phi(A) \approx A$.*

**VII-6.7.** *For all $P$ and $Q$ in $\Gamma$, we have $\Psi(PQ) \approx \Psi(P)\Psi(Q)$.*

That this suffices is elementary general mathematics which the reader can work out if necessary. The first two give maps back and forth between the sets of equivalence classes, and the second two show that those maps are inverse to each other. The last one assures us that the maps are morphisms.

**Definition of $\Psi$.** Define it to be the unique morphism of binary operations which maps generators as follows : all the variables go to themselves; $K$ goes to $\underline{K} := \underline{T} := \lambda xy \bullet x$; and $S$ goes to $\underline{S} := \lambda xyz \bullet (xz)(yz)$.

**Remarks.** (i) Since $\Psi$, by definition, preserves the operations, there is nothing more needed to prove **VII-6.7** (which is understated—it holds with $=$ , not just $\approx$ .

(ii) The sub-binary operation of $\Lambda$ generated by $\underline{S}$ , $\underline{K}$ and all the variables is in fact freely generated by them. This is equivalent to the fact that $\Psi$ is injective. But we won't dwell on this or prove it, since it seems not to be useful in establishing that the map induced by $\Psi$ on equivalence classes is injective. But for concreteness, the reader may prefer to identify $\Gamma$ with that subset of $\Lambda$, namely the image of $\Psi$ . So you can think of the combinators as certain kinds of closed $\lambda$-expressions, closed in the sense of having no free variables.

The main job is to figure out how to simulate, in $\Gamma$, the abstraction operator in $\Lambda$.

**Definition of $\mu x \bullet$ in $\Gamma$ .** For each variable $x$ and each $P \in \Gamma$, define $\mu x \bullet P$ as follows, by induction on the structure of $P$:

$$\text{If } P \text{ is an atom other than } x, \text{ define } \mu x \bullet P \ := \ KP \ .$$

$$\text{Define } \mu x \bullet x \ := \ SKK \ := \ I \ .$$

$$\text{Define } \mu x \bullet (QR) \ := \mu x \bullet QR \ := \ S(\mu x \bullet Q)(\mu x \bullet R) \ .$$

**Definition of $\Phi$ .** This is again inductive, beginning with the atoms, i.e. variables $x$ :

$$\Phi(x) := x \quad ; \quad \Phi(AB) := \Phi(A)\Phi(B) \quad ; \quad \Phi(\lambda x \bullet A) := \mu x \bullet \Phi(A) \ .$$

It should be (but seldom is) pointed out that the correctness of this definition depends on unique readability of the strings which make up the set $\Lambda$ .

The first result is the mirror image of the last part of that definition.

**VII-6.8.** *For all $P \in \Gamma$, we have* $\Psi(\mu x \bullet P) \ \approx \ \lambda x \bullet \Psi(P)$ .

**Proof.** Proceed by induction on $P$.

When $P$ is the variable $x$, the left-hand side is

$$\Psi(\mu x \bullet x) \ = \ \Psi(SKK) \ = \ \underline{S}\ \underline{K}\ \underline{K}\ ;$$

whereas the right-hand side is $\lambda x \bullet x$ . When applied to an arbitrary $B \in \Lambda$ these two 'agree' (as suffices even with $B$ just a variable):

$$\underline{S}\ \underline{K}\ \underline{K}\ B \ \approx \ \underline{K}\ B\ (\underline{K}\ B) \ \approx \ B \approx \ (\lambda x \bullet x)B \ .$$

The middle $\approx$ is just **VII-2.1**(a), since $\underline{K} = \underline{T}$, and the first one is **VII-2.5**. [Notice the unimpressive fact that we could have defined $I$ to be $SKZ$ for *any* $Z \in \Gamma$ .]

When $P$ is an atom other than $x$, the left-hand side is

$$\Psi(KP) \ = \ \Psi(K)\Psi(P) \ = \ \underline{K}\ \Psi(P) \ .$$

But if $x$ is not free in $A$ [and it certainly isn't when we take $A = \Psi(P)$ here], for any $B$ we have

$$\underline{K}\ A\ B \ \approx \ A \ \approx \ (\lambda x \bullet A)B \ .$$

Finally, when $P$ is $QR$, where the result is assumed for $Q$ and $R$, the left-hand side is

$$\Psi(S(\mu x \bullet Q)(\mu x \bullet R)) \ = \ \underline{S}\ \Psi(\mu x \bullet Q)\Psi(\mu x \bullet R) \ \approx \ \underline{S}\ (\lambda x \bullet \Psi(Q))(\lambda x \bullet \Psi(R)) \ .$$

The right-hand side is $\lambda x \bullet \Psi(Q)\Psi(R)$ . Applying these to suitable $B$ yields respectively (use **VII-2.5** again) :

$$\Psi(Q)^{[x \mapsto B]}\Psi(R)^{[x \mapsto B]} \quad \text{and} \quad (\Psi(Q)\Psi(R))^{[x \mapsto B]} \ ,$$

so they agree, completing the proof.

Any $\lambda$-expert reading the previous proof and next result will possibly find them irritating. We have made extensive use of extensionality in the last proof. But the results actually hold (and are very important in more encyclopædic versions of this subject) with '$\approx$' replaced by '$\approx_{\text{in}}$', where the latter is defined using only rules ($\alpha$) and ($\beta$), and congruence [that is, drop rule ($\eta$)]. So an exercise for the reader is to find proofs of these slightly more delicate facts.

**Proof of VII-6.6**. Proceed by induction on the structure of $A$ :
When $A$ is a variable $x$, we have $\Psi\Phi(x) = \Psi(x) = x$ .
When $A = BC$, where the result holds for $B$ and $C$,

$$\Psi\Phi(BC) \;=\; \Psi(\Phi(B)\Phi(C)) \;=\; \Psi\Phi(B)\Psi\Phi(C) \;\approx\; BC \ .$$

Finally, when $A = \lambda x \bullet B$, where the result holds for $B$, using **VII-6.8**,

$$\Psi\Phi(\lambda x \bullet B) = \Psi(\mu x \bullet \Phi(B)) \;\approx\; \lambda x \bullet \Psi\Phi(B) \;\approx\; \lambda x \bullet B \ ,$$

as required.

**Proof of (most of) VII-6.5**. Proceed by induction on the structure of $P$: The inductive step is trivial, as in the second case above.
When $A$ is a variable, this is also trivial, as in the first case above.
When $A = K$, it is a straightforward calculation : For any $B$,

$$\Phi\Psi(K)B \;=\; \Phi(\underline{K})B \;=\; \Phi(\lambda x \bullet (\lambda y \bullet x))B \;=\; (\mu x \bullet (\mu y \bullet \Phi x))B \;=$$

$$(\mu x \bullet Kx)B \;=\; S(\mu x \bullet K)(\mu x \bullet x)B \;=\; S(KK)IB \;\sim$$

$$KKB(IB) \;\sim\; K(SKKB) \;\sim\; K(KB(KB)) \;\sim\; KB \ .$$

Taking $B$ to be a variable, we get $\Phi\Psi(K) \;\sim\; K$ , as required.
When $A = S$, there is undoubtedly a similar, but very messy, calculation. But the author doubts whether he will ever have the energy to write it out, or, having done so, much confidence that it is free of error. So we postpone the remainder of the proof until just after that of **VII-6.13** below. But we have the good luck that nothing between here and there depends on the

present result, **VII-6.5**. [The method there also gives an alternative to the above calculation for showing $\Phi\Psi(K) \sim K$ .]

For $\lambda$-experts we are avoiding some delicate issues here by imposing extensionality. A quick calculation shows that $\Phi(\Psi(K)) = S(KK)I$ —in fact, just drop the $B$ in the first two displayed lines above to see this—but $S(KK)I$ and $K$ are not related under the equivalence relation '$\sim_{\text{in}}$' defined as with '$\sim$' except that extensionality (the last condition) is dropped. One needs a version of the Church-Rosser theorem to prove this.

**Proof of VII-6.3**. By the definition of $\sim$ , and basics on equivalence relations (see the proof of **VII-6.12** ahead for what I mean by this, if necessary), it suffices to prove one fact for each of the four conditions generating the relation $\sim$ , those facts being

$$\Psi(SABC) \approx \Psi(AC(BC)) \quad ; \quad \Psi(KAB) \approx \Psi(A) \quad ;$$

$$\Psi(A_1) \approx \Psi(B_1) \text{ and } \Psi(A_2) \approx \Psi(B_2) \implies \Psi(A_1A_2) \approx \Psi(B_1B_2) \ ;$$

and

$\Psi(Ax) \approx \Psi(Bx) \Rightarrow \Psi(A) \approx \Psi(B)$, if the variable $x$ appears in neither $A$ nor $B$.

The latter two are almost trivial, and the first two are simple consequences of earlier identities : Using **VII-2.1**(a),

$$\Psi(KAB) \ = \ \Psi(K)\Psi(A)\Psi(B) \ = \ \underline{T}\ \Psi(A)\Psi(B) \ \approx \ \Psi(A) \ ,$$

as required. Using **VII-2.5**,

$$\Psi(SABC) \ = \ \Psi(S)\Psi(A)\Psi(B)\Psi(C) \ = \ \underline{S}\ \Psi(A)\Psi(B)\Psi(C) \ \approx$$

$$\Psi(A)\Psi(C)(\Psi(B)\Psi(C)) \ = \ \Psi(AC(BC)) \ ,$$

as required.

The proof of **VII-6.4** seems to be somewhat more involved, apparently needing the following results.

**VII-6.9.** *If $x$ isn't in $P$, then $\mu x \bullet P \ \sim \ KP$ .*

**Proof.** Proceeding by induction on $P$, the atomic case doesn't include $P = x$, so we get equality, not just $\sim$ . The inductive step goes as follows : For any $T$,

$$(\mu x \bullet QR)T \ = \ S(\mu x \bullet Q)(\mu x \bullet R)T \ \sim \ S(KQ)(KR)T \ \sim$$

$$KQT(KRT) \ \sim \ QR \ \sim \ K(QR)T \ ,$$

and so, $\mu x \bullet QR \ \sim \ K(QR)$ , as required.

**VII-6.10.** *If $P \sim Q$ then $\mu x \bullet P \sim \mu x \bullet Q$.*

**Proof.** By the definition of $\sim$ , and basics on equivalence relations, it suffices to prove one fact for each of the four conditions generating the relation $\sim$ , those facts being

$$\mu x \bullet SABC \sim \mu x \bullet AC(BC) \quad ; \quad \mu x \bullet KAB \sim \mu x \bullet A \quad ;$$

$\mu x \bullet A_1 \sim \mu x \bullet B_1$ and $\mu x \bullet A_2 \sim \mu x \bullet B_2 \implies \mu x \bullet A_1 A_2 \sim \mu x \bullet B_1 B_2$ ;

and finally, if the variable $z$ appears in neither $A$ nor $B$,

$$\mu x \bullet Az \sim \mu x \bullet Bz \implies \mu x \bullet A \sim \mu x \bullet B \ .$$

However we first eliminate need to deal with the last of these when $z = x$ by proving the case of the entire result where $x$ is not in $P$ or $Q$. In that case, using **VII-6.9**,

$$\mu x \bullet P \ \sim \ KP \ \sim \ KQ \ \sim \ \mu x \bullet Q \ .$$

As for the last fact when $z \neq x$, for any $R$ we have

$$(\mu x \bullet Az)R \ = \ S(\mu x \bullet A)(\mu x \bullet z)R \ \sim \ (\mu x \bullet A)R((\mu x \bullet z)R) \ \sim$$

$$(\mu x \bullet A)R(KzR) \ \sim \ (\mu x \bullet A)Rz \ .$$

The same goes with $B$ replacing $A$, so taking $R$ as a variable different from $z$ and which is not in $A$ or $B$, we cancel twice to get the result.

For the second fact above, we have, for any $C$,

$$(\mu x \bullet KAB)C \;=\; S(\mu x \bullet KA)(\mu x \bullet B)C \;\sim\; (\mu x \bullet KA)C((\mu x \bullet B)C) \;\sim$$

$$S(\mu x \bullet K)(\mu x \bullet A)C((\mu x \bullet B)C) \;\sim\; (\mu x \bullet K)C((\mu x \bullet A)C)((\mu x \bullet B)C) \;\sim$$

$$KKC((\mu x \bullet A)C)((\mu x \bullet B)C) \;\sim\; K((\mu x \bullet A)C)((\mu x \bullet B)C) \;\sim\; (\mu x \bullet A)C \;,$$

as suffices.

The first one is similar but messier.

The third fact is quick:

$$\mu x \bullet A_1A_2 \;=\; S(\mu x \bullet A_1)(\mu x \bullet A_2) \;\sim\; S(\mu x \bullet B_1)(\mu x \bullet B_2) \;=\; \mu x \bullet B_1B_2 \;.$$

**VII-6.11.** *The variables occurring in $\mu x \bullet P$ are exactly those other than $x$ which occur in $P$.*

**Proof.** This is just a book-keeping exercise, by induction on $P$.

**VII-6.12.** *$R \sim P$ implies $R^{[x \to Q]} \sim P^{[x \to Q]}$*

**Proof.** A basis for this and a couple of earlier proofs is the following explicit description of the relation '$\sim$' :

$R \sim P \iff$ there is a finite sequence of ordered pairs of elements of $\Gamma$, whose last term is the pair $(R, P)$, and each of whose terms has at least one of the following seven properties. It is :

(1) $(A, A)$, for some $A$ ; or

(2) $(B, A)$, where $(A, B)$ occurs earlier in the sequence; or

(3) $(A, C)$, where both $(A, B)$ and $(B, C)$ occur earlier for some $B$; or

(4) $(KAB\ ,\ A)$, for some $A$ and $B$ ; or

(5) $(SABC\ ,\ (AC)(BC))$, for some $A, B$ and $C$ ; or

(6) $(A_1A_2\ ,\ B_1B_2)$, where both $(A_1, B_1)$ and $(A_2, B_2)$ occur earlier; or

(7) $(A, B)$, where $(Az, Bz)$ occurs earlier in the sequence, and where $z$ is some variable not occurring in $A$ or $B$.

[This is true because the condition above does define an equivalence relation which satisfies the conditions defining '$\sim$' , and any pair that is related under any relation which satisfies the conditions defining '$\sim$', is also necessarily related under this relation just above.]

Call any sequence of ordered pairs as above a *verification* for $R \sim P$.

Now proceed by contradiction, assuming that $R \sim P$ has a shortest possible verification among all pairs for which the result fails for some $x$ and some $Q$—shortest in the sense of sequence length.

Then the pair $(R, P)$ cannot have any of the forms as in (1) to (5) with respect to its "shortest verification", because $(R^{[x \to Q]}, P^{[x \to Q]})$ would have the same form (with respect to another verification in the case of (2) and (3), by "shortest").

It also cannot have the form in (6), since (again by "shortest") we could concatenate verifications for $A_1^{[x \to Q]} \sim B_1^{[x \to Q]}$ and $A_2^{[x \to Q]} \sim B_2^{[x \to Q]}$ to prove that $(A_1 A_2)^{[x \to Q]} \sim (B_1 B_2)^{[x \to Q]}$ .

So $(R, P)$ must have the form in (7), i.e. we have $(R, P) = (A, B)$ where $(Az, Bz)$ occurs earlier in that shortest sequence, for some $z$ not in $A$ or $B$; and yet $A^{[x \to Q]} \not\sim B^{[x \to Q]}$ .

Now necessarily $z \neq x$, since otherwise $x$ does not occur in $A$ or $B$, so

$$A^{[x \to Q]} \;=\; A \;\sim\; B = B^{[x \to Q]} \;.$$

Note that if $w$ is a variable not occurring anywhere in a verification, applying $^{[y \to w]}$ for any $y$ to all terms in the pairs in the verification will produce another verification.

Now choose the variable $w$ different from $x$, so that $w$ is not in $Q$ nor in any term in pairs in the above "shortest verification". Applying $^{[z \to w]}$ to everything up to the $(Az, Bz)$ term in that verification shows that $Aw \sim Bw$ by a derivation shorter than the "shortest" one above. Thus we have $(Aw)^{[x \to Q]} \sim (Bw)^{[x \to Q]}$ ; that is, $A^{[x \to Q]} w \sim B^{[x \to Q]} w$ . But now we can just erase $w$ since it doesn't occur in $A^{[x \to Q]}$ nor in $B^{[x \to Q]}$, yielding the contradiction $A^{[x \to Q]} \sim B^{[x \to Q]}$ , i.e. $R^{[x \to Q]} \sim P^{[x \to Q]}$ , and completing the proof.

[Of course this isn't really a proof by contradiction; rather it is one by induction on the length of the shortest verification for $P \sim Q$.]

**VII-6.13.** *For all $P \in \Gamma$, we have $(\mu x \bullet P)x \sim P$, and, more generally* (noting that 'okayness' of substitution is not an issue in $\Gamma$) *the following analogue of the $(\beta)$-rule:*

$$(\mu x \bullet P)Q \;\sim\; P^{[x \to Q]} \;.$$

[Note that the first part plus 'extensionality' quickly give the direct analogue in $\Gamma$ for the $(\eta)$-rule from $\Lambda$, namely

$$\mu x \bullet Rx \ \sim \ R \qquad \text{if } x \text{ is not in } R \ .$$

To see this, just apply the left side to $x$.]

**Proof.** Proceeding by induction on $P$ for the first identity, the atomic case when $P = x$ just uses $Ix \sim x$ . When $P$ is an atom other than $x$,

$$(\mu x \bullet P)x \ = \ KPx \ \sim \ P \ .$$

For the inductive step,

$$(\mu x \bullet QR)x \ = \ S(\mu x \bullet Q)(\mu x \bullet R)x \ \sim \ (\mu x \bullet Q)x((\mu x \bullet R)x) \ \sim \ QR \ .$$

Then the second identity follows using the first one, using **VII-6.12**, and using the previous fact in **VII-6.11** about $x$ not occurring in $\mu x \bullet P$ :

$$(\mu x \bullet P)Q \ = \ ((\mu x \bullet P)x)^{[x \to Q]} \ \sim \ P^{[x \to Q]} \ .$$

**Completion of the proof of VII-6.5**. To prove the remaining fact, namely $\Phi\Psi(S) \ \sim \ S$, first we show that $\Phi\Psi(S)BCD \ \sim \ SBCD$ for any $B, C, D$ . This is straightforward, using **VII-6.13** three times :

$$\Phi\Psi(S)BCD \ = \ \Phi(\underline{S})BCD \ = \ (\mu x \bullet (\mu y \bullet (\mu z \bullet xz(yz))))BCD \ \sim$$

$$(\mu y \bullet (\mu z \bullet xz(yz)))^{[x \to B]}CD \ = \ (\mu y \bullet (\mu z \bullet Bz(yz)))CD \ \sim$$

$$(\mu z \bullet Bz(yz))^{[y \to C]}D \ = \ (\mu z \bullet Bz(Cz))D \ \sim$$

$$(Bz(Cz))^{[z \to D]} \ = \ BD(CD) \ \sim \ SBCD \ .$$

Now just take $B, C, D$ as three distinct variables and cancel.

**VII-6.14.** *If $y$ doesn't occur in $Q$ and $x \neq y$, then*

$$(\mu y \bullet P)^{[x \to Q]} \ \sim \ \mu y \bullet (P^{[x \to Q]}) \ .$$

**Proof.** Proceeding by induction on $P$, the atomic cases are as follows:

$\underline{P = y}$ : Both sides give $I$ .

$\underline{P = x}$ : Both sides give $KQ$, up to $\sim$ . (Use **VII-6.9** .)

$\underline{P \text{ is any other atom}}$ : Both sides give $KP$ .

The inductive step uses nothing but definitions :

$$(\mu y \bullet TR)^{[x \to Q]} \;=\; (S(\mu y \bullet T)(\mu y \bullet R))^{[x \to Q]} \;=$$

$$S(\mu y \bullet T)^{[x \to Q]}(\mu y \bullet R)^{[x \to Q]} \;\sim\; S(\mu y \bullet T^{[x \to Q]})(\mu y \bullet R^{[x \to Q]}) \;=$$

$$\mu y \bullet (T^{[x \to Q]} R^{[x \to Q]}) \;=\; \mu y \bullet ((TR)^{[x \to Q]}) \; .$$

**VII-6.15.** *If $x$ is not free in $A$, then it does not occur in $\Phi(A)$ .*

**Proof.** This is an easy induction on $A$, using **VII-6.11** and definitions.

**VII-6.16.** *If $A^{[x \to B]}$ is okay, then $\Phi(A^{[x \to B]}) \;\sim\; \Phi(A)^{[x \to \Phi(B)]}$ .*

**Proof.** Proceeding by induction on $A$, the atomic cases are as follows:

$\underline{A = x}$ : Both sides give $\Phi(B)$ .

$\underline{A = y \neq x}$ : Both sides give $\Phi(y)$ .

The inductive steps are as follows:

(I) $\underline{A = CD}$ : Since $C^{[x \to B]}$ and $D^{[x \to B]}$ are both okay as long as $(CD)^{[x \to B]}$ is,

$$\Phi((CD)^{[x \to B]}) \;=\; \Phi(C^{[x \to B]})\Phi(D^{[x \to B]}) \;\sim\; \Phi(C)^{[x \to \Phi(B)]}\Phi(D)^{[x \to \Phi(B)]} \;=$$

$$(\Phi(C)\Phi(D))^{[x \to \Phi(B)]} \;=\; \Phi(CD)^{[x \to \Phi(B)]} \; .$$

(II) $\underline{A = \lambda x \bullet C}$ : We have,

$$\Phi((\lambda x \bullet C)^{[x \to B]}) \;=\; \Phi(\lambda x \bullet C) \;=\; \mu x \bullet \Phi(C) \; ,$$

whereas, using **VII-6.11**,

$$(\Phi(\lambda x \bullet C))^{[x \to \Phi(B)]} \;=\; (\mu x \bullet \Phi(C))^{[x \to \Phi(B)]} \;=\; \mu x \bullet \Phi(C) \; .$$

(III) $\underline{A = \lambda y \bullet C \;\text{ for } y \neq x}$ : The first '$\sim$' below uses the inductive hypothesis and **VII-6.10**. The second one uses **VII-6.14**, observing that, by **VII-6.15**, the variable $y$ doesn't occur in $\Phi(B)$ because it is not free in $B$, the

latter being the case because $(\lambda y \bullet C)^{[x \to B]}$ is okay.

$$\Phi((\lambda y \bullet C)^{[x \to B]}) \;=\; \Phi(\lambda y \bullet (C^{[x \to B]})) \;=\; \mu y \bullet (\Phi(C^{[x \to B]})) \;\sim$$
$$\mu y \bullet (\Phi(C)^{[x \to \Phi(B)]}) \;\sim\; (\mu y \bullet \Phi(C))^{[x \to \Phi(B)]} \;=\; (\Phi(\lambda y \bullet C))^{[x \to \Phi(B)]} \,.$$

**VII-6.17.** *If $y$ isn't in $P$, then $\mu x \bullet P \;\sim\; \mu y \bullet (P^{[x \to y]})$.*
[Note that this is the direct analogue in $\Gamma$ for the $(\alpha)$-rule from $\Lambda$.]

**Proof.** We leave this to the reader—a quite straightforward induction on $P$, using only definitions directly.

**Proof of VII-6.4**. Proceeding inductively, and looking at the inductive procedure in the definition of $\Lambda$ first : If $A_1 \approx A_2$ and $B_1 \approx B_2$, then

$$\Phi(A_1 B_1) \;=\; \Phi(A_1)\Phi(B_1) \;\sim\; \Phi(A_2)\Phi(B_2) \;=\; \Phi(A_2 B_2) \,,$$

and

$$\Phi(\lambda x \bullet A_1) \;=\; \mu x \bullet \Phi(A_1) \;\sim\; \mu x \bullet \Phi(A_2) \;=\; \Phi(\lambda x \bullet A_2) \,,$$

using **VII-6.10** in this last line.

So the result will follow from three facts, corresponding to the three basic reduction laws in the definition of $\approx$ :

$(\eta)'$ $\quad \Phi(\lambda x \bullet Ax) \;\sim\; \Phi(A)$ if $x$ is not free in $A$ .

$(\beta)'$ $\quad \Phi((\lambda x \bullet A)B) \;\sim\; \Phi(A^{[x \to B]})$ if $A^{[x \to B]}$ is okay .

$(\alpha)'$ $\quad \Phi(\lambda x \bullet A) \;\sim\; \Phi(\lambda y \bullet A^{[x \to y]})$ if $y$ is not free in $A$ and $A^{[x \to y]}$ is okay.

Twice below we use **VII-6.15**.

To prove $(\eta)'$, for any $Q$, we have

$$\Phi(\lambda x \bullet Ax)Q \;=\; (\mu x \bullet \Phi(A)x)Q \;\sim\; (\Phi(A)x)^{[x \to Q]} \;=\; \Phi(A)Q \,,$$

as suffices, using **VII-6.13** for the middle step.

To prove $(\beta)'$, the left-hand side is

$$(\mu x \bullet \Phi(A))\Phi(B) \;\sim\; \Phi(A)^{[x \to \Phi(B)]} \;\sim\; \Phi(A^{[x \to B]}) \,,$$

using **VII-6.13** and **VII-6.16** .

To prove $(\alpha)'$, the left-hand side is

$$\mu x \bullet \Phi(A) \ \sim \ \mu y \bullet (\Phi(A)^{[x \to y]}) \ \sim \ \mu y \bullet \Phi(A^{[x \to y]}) \ ,$$

which is the right-hand side, using, respectively, [**VII-6.17** plus $y$ not occurring in $\Phi(A)$], and then using **VII-6.16** .

This completes the proof that the combinators do produce a combinatorially complete binary operation, not different from that produced by the $\lambda$-calculus (in our so-called extensional context).

Schonfinkel's original proof is different, and not hard to follow. Actually with all this intricacy laid out above, we can quickly give his direct proof that $\Gamma/\sim$ is combinatorially complete. More-or-less just copy the proof that $\Lambda/\approx$ is combinatorially complete, replacing '$\lambda x\bullet$' everywhere by '$\mu x\bullet$' , as follows.

For the constant function of '$n$' variables with value $P$, pick distinct variables $y_i$ not in $P$ and let

$$\zeta \ = \ \mu y_1 \bullet \mu y_2 \bullet \cdots \bullet \mu y_n \bullet P \ .$$

For the $i$th projection, let $\zeta \ = \ \mu x_1 \bullet \mu x_2 \bullet \cdots \bullet \mu x_n \bullet x_i$ .
For the inductive step, let

$$\zeta \ = \ \mu x_1 \bullet \mu x_2 \bullet \cdots \bullet \mu x_n \bullet (\zeta_g x_1 x_2 \cdots x_n)(\zeta_h x_1 x_2 \cdots x_n) \ ,$$

as in the proof of **VII-6.1** .

Schonfinkel's theorem is a bit more general than this, as below. The ideas in the proof of the substantial half are really the same as above, so the main thing will be to set up the machinery. We'll leave out details analogous to those in the build-up above.

**Theorem VII-6.18.** (Schonfinkel) *Let $\Omega$ be any binary operation* (written as juxtaposition with the usual bracket conventions). *Then $\Omega$ is combinatorially complete if and only if it has elements $\kappa$ and $\sigma$ such that, for all $\omega_i \in \Omega$, we have*

$$\kappa\omega_1\omega_2 \ = \ \omega_1 \qquad \textit{and} \qquad \sigma\omega_1\omega_2\omega_3 \ = \ (\omega_1\omega_3)(\omega_2\omega_3) \ .$$

For example, as we saw above, when $\Omega = \Gamma/\sim$ , we could take $\kappa$ and $\sigma$ to be the equivalence classes $[K]_\sim$ and $[S]_\sim$ , respectively.

**Proof.** Assume it is combinatorially complete, and with $F = FUN_2(v_1)$ and $G = FUN_3((v_1 * v_3) * (v_1 * v_3))$, let $\kappa$ and $\sigma$ respectively be corresponding elements $\zeta$ from the definition of combinatorial completeness. Thus, as required,

$$\kappa\omega_1\omega_2 \;=\; F(\omega_1,\omega_2) \;=\; \omega_1 \quad \text{and} \quad \sigma\omega_1\omega_2\omega_3 \;=\; G(\omega_1,\omega_2,\omega_3) \;=\; (\omega_1\omega_3)(\omega_2\omega_3)\;.$$

For the converse, here is some notation :

$$\Omega \;\subset\; \Omega_+^{(0)} = FREE(\Omega) \;\subset\; \Omega_+ = FREE(\Omega \cup \{v_1, v_2, \cdots\}) \;=\; \bigcup_{n\geq 0} \Omega_+^{(n)}\;,$$

where

$$\Omega_+^{(n)} \;:=\; FREE(\Omega \cup \{v_1, v_2, \cdots, v_n\})\;.$$

As before, the operation in $\Omega_+$ will be written $*$ , to contrast with juxtaposition in $\Omega$ (and the $v_i$'s are distinct and disjoint from $\Omega$) .

Note that morphisms $\chi : \Omega_+ \to \Omega$ are determined by their effects on the set of generators, $\Omega \cup \{v_1, v_2, \cdots\}$ . We'll only consider those which map each element of $\Omega$ to itself, so when restricted to $\Omega_+^{(0)}$, all such $\chi$ agree. Furthermore, we have, with $A_i \in \Omega$ ,

$$FUN_n(f)(A_1, \cdots, A_n) \;=\; \chi(f^{[v_1\mapsto A_1][v_2\mapsto A_2]\cdots[v_n\mapsto A_n]})\;.$$

This is because, for fixed $(A_1, \cdots, A_n)$, each side is a morphism $\Omega_+^{(n)} \to \Omega$, "of $f$" , and they agree on generators.

Now, for variables $y = v_i$ and $P \in \Omega_+$, define $\mu y \bullet P \in \Omega_+$ inductively much as before :

$$\mu y \bullet P \;:=\; k * P \quad \text{for } P \in \Omega \text{ or if } P \text{ is a variable } v_j \neq y\;;$$

$$\mu y \bullet y \;:=\; s * k * k \;\;;$$

$$\mu y \bullet (Q * R) \;:=\; s * (\mu y \bullet Q) * (\mu y \bullet R) \;\;.$$

It follows as before that the variables occurring in $\mu y \bullet P$ are precisely those other than $y$ in $P$. Also, define an equivalence relation on $\Omega_+$ by

$$a \sim b \;\iff\; \forall \chi\;,\; \chi(a) = \chi(b)$$

(referring to $\chi$ which map elements of $\Omega$ by the identity map). Then we have, proceeding by induction on $P$ as in **VII-6.13**,

$$(\mu y \bullet P) * y \ \sim \ P \ ,$$

and, substituting $Q$ for $y$,

$$(\mu y \bullet P) * Q \ \sim \ P^{[y \to Q]} \ ,$$

and finally, by induction on $n$,

$$(\mu y_1 \bullet \mu y_2 \bullet \cdots \mu y_n \bullet P) * Q_1 * \cdots * Q_n \ \sim \ P^{[y_1 \to Q_1][y_2 \to Q_2] \cdots [y_n \to Q_n]} \ .$$

The proof is then completed easily : Given $f \in \Omega_+^{(n)}$, define

$$\zeta \ := \ \chi(\mu v_1 \bullet \mu v_2 \bullet \cdots \mu v_n \bullet f) \ \in \ \Omega \ .$$

Then, for all $A_i \in \Omega$ , we have $A_i = \chi(A_i)$ , so

$$\zeta A_1 \cdots A_n \ = \ \chi(\mu v_1 \bullet \mu v_2 \bullet \cdots \mu v_n \bullet f)\chi(A_1) \cdots \chi(A_n) \ =$$

$$\chi((\mu v_1 \bullet \mu v_2 \bullet \cdots \mu v_n \bullet f) * A_1 * \cdots * A_n) \ =$$

$$\chi(f^{[v_1 \to A_1][v_2 \to A_2] \cdots [v_n \to A_n]}) \ = \ FUN_n(f)(A_1, \cdots, A_n) \ ,$$

as required.

## VII-7 Models for $\lambda$-Calculus, and Denotational Semantics.

This will be a fairly sketchy introduction to a large subject, mainly to provide some motivation and references. Then, in the last two subsections, we give many more details about a particular family of $\lambda$-models.

As far as I can determine, there is still not complete agreement on a single best definition of what is a 'model for the $\lambda$-calculus'. For example, not many pages along in **[Ko]**, you already have a choice of at least **ten** definitions, whose names correspond to picking an element from the following set and erasing the commas and brackets :

$$\{ \mathsf{proto} \ , \ \mathsf{combinatorial} \ , \ \mathsf{Meyer} \ , \ \mathsf{Scott} \ , \ \mathsf{extensional} \} \times \{ \mathsf{lambda} \} \times \{ \mathsf{algebra} \ , \ \mathsf{model} \} \ .$$

And that's only the beginning, as further along we have "categorical lambda models", etc., though here the first adjective refers to the method of construction, rather than an axiomatic definition. In view of all this intricacy (and relating well to the simplifications of the last subsection), we shall consider in detail only what are called extensional lambda models . The definition is quite simple : Such an object is any structure $(D\ ,\ \cdot\ ,\ \kappa\ ,\ \sigma)$ , where " $\cdot$ " is a binary operation on the set $D$, which set has specified distinct elements $\kappa$ and $\sigma$ causing it to be combinatorially complete as in **VII-6.18**, and the operation must satisfy

(extensionality) *for all* $\alpha$ *and* $\beta$ $\quad [(\forall\gamma\ \alpha\cdot\gamma = \beta\cdot\gamma) \implies \alpha = \beta\ ]$ .

Notice that, using extensionality twice and then three times, the elements $\kappa$ and $\sigma$ are unique with respect to having their properties

$$\kappa\omega_1\omega_2 \ = \ \omega_1 \qquad \text{and} \qquad \sigma\omega_1\omega_2\omega_3 \ = \ (\omega_1\omega_3)(\omega_2\omega_3)\ .$$

So they needn't be part of the structure, just assumed to exist, rather like the identity element of a group. That is, just assume combinatorial completeness (along with extensionality), and forget about any other structure.

The other 10+ definitions referred to above are similar (once one becomes educated in the conventions of this type of model theory/equational-combinatorial logic, and can actually determine what is intended—but see the lengthy digression in small print beginning two paragraphs below). Effectively each replaces extensionality with a condition that is strictly weaker (and there are many interesting models that are not extensional, and which in some cases are apparently important for the application to denotational semantics.) So the reader should keep in mind that the extensional case, which we emphasize except in the small print just below, is not the entire story. Much more on the non-extensional case can be found in the references. Except for first reading the digression below, the clearest treatment I know is that in **[HS]**, if you want to learn more on models in general, for the $\lambda$-calculus and for combinators.

One missing aspect in our definition above is this : Where did the $\lambda$-abstraction operator get to, since, after all, it is the $\lambda$-calculus we are supposed to be modelling? Several of the other definitions address this directly. Below for several paragraphs we discuss a way of arriving at a general sort of definition of the term $\lambda$-*model*, a little different from others of which I

am aware, but easily seen to be mathematically equivalent to the definitions usually given. In particular, this will indicate how to get maps from the $\lambda$-calculus, $\Lambda$, into any model, and quite explicitly into extensional ones as above, one such map for each assignment of variables.

## A (not entirely) Optional Digression on $\lambda$- models.

To remind you of a very basic point about sets and functions, there is a 1-1 *adjointness* correspondence as follows, where $A^B$ denotes the set of all functions with (domain, codomain) the pair of sets $(B, A)$ :

$$\begin{array}{ccc} A^{B\times C} & \longleftrightarrow & (A^C)^B \\ f & \vdash\!\!\!\rightarrow & [b \mapsto [c \mapsto f(b,c)]] \\ [(b,c) \mapsto g(b)(c)] & \leftarrow\!\!\!\dashv & g \end{array}$$

[Perhaps the $\lambda$-phile would prefer $\lambda bc \bullet f(b,c)$ and $\lambda(b,c) \bullet gbc$ .] Taking $B = C = A$, this specializes to a bijection between the set of all binary operations on $A$ and the set $(A^A)^A$. In particular, as is easily checked, the *extensional* binary operations on $A$ correspond to the *injective* functions from $A$ to $A^A$ .

Let us go back to some of the vague remarks of Subsection VII-3, and try to puzzle out a rough idea of what the phrase "model for $\lambda$-calculus" ought to mean, as an object in ordinary naive set-theoretic mathematics, avoiding egregious violations of the axiom of foundation. We'll do this without requiring the reader to master textbooks on model theory in order to follow along. It seems to the author that the approach below, certainly not very original, is more natural than trying to force the round peg of $\lambda$-calculus into the square hole of something closely similar to models for $1^{\underline{\text{st}}}$order theories.

We want some kinds of sets, $D$, for which each element in $D$ can play a second role as also somehow representing a function from $D$ to itself. As noted, the functions so represented will necessarily form a very small subset, say $[D \!\rightharpoonup\! D]$, of $D^D$, very small relative to the cardinality of $D^D$.

The simplest way to begin to do this is to postulate a surjective function $\phi$, as below :

$$\phi \;:\; D \;\rightarrow\; [D \!\rightharpoonup\! D] \;\subset\; D^D \;.$$

So we are given a function from $D$ to the set of all its self-functions, we name that function's image $[D \!\rightharpoonup\! D]$, and use $\phi$ as the generic name for this surjective adjoint. It is the adjoint of a multiplication $D \times D \rightarrow D$, by the remarks in the first paragraph of this digression.

Now let $\Lambda^{(0)}$ be the set of all *closed terms* in $\Lambda$, those without free variables. A model, $D$, such as we seek, will surely involve at least a function $\Lambda^{(0)} \rightarrow D$ with good properties with respect to the structure of $\Lambda^{(0)}$ and to our intuitive notion of what that structure is supposed to be modelling; namely, function *application* and function *abstraction*. As to the former, the obvious thing is to have the map be a morphism between the binary operations on the two sets.

Because $\Lambda$ is built by structural induction, it is hard to define anything related to $\Lambda$ without using induction on structure. Furthermore, by first fixing, for each variable $x$, an element $\rho(x) \in D$, we'd expect there to be maps (one for each "assignment" $\rho$)

$$\rho_+ \ : \ \Lambda \to D \ ,$$

which all agree on $\Lambda^{(0)}$ with the one above. (So their restrictions to $\Lambda^{(0)}$ are all the same map.) These $\rho_+$'s should map each variable $x$ to $\rho(x)$ . Thinking about application and abstraction (and their formal versions in $\Lambda$), we are led to the following requirements.

$$\rho_+ \ : \ x \ \mapsto \ \rho(x) \ ;$$
$$\rho_+ \ : \ MN \ \mapsto \ \phi(\rho_+(M))(\rho_+(N)) \ ;$$
$$\rho_+ \ : \ \lambda x \bullet M \ \mapsto \ \psi(d \mapsto \rho_+^{[x \mapsto d]}(M)) \ .$$

The middle display is the obvious thing to do for a 'semantic' version of application. It's just another way of saying that $\rho_+$ is a morphism of binary operations.

But the bottom display needs plenty of discussion. Firstly, the assignment $\rho^{[x \mapsto d]}$ is the assignment of variables which agrees with $\rho$, except that it assigns $d \in D$ to the variable $x$. (This is a bit like substitution, so our notation reflects that, but is deliberately different, $^{[x \mapsto d]}$ rather than $^{[x:=d]}$). For the moment, let $\psi(f)$ vaguely mean "some element of $D$ which maps under $\phi$ to $f$". Thus, the bottom display says that $\lambda x \bullet M$, which we intuit as "the function of $x$ which $M$ gives", should be mapped by $\rho_+$ to the element of $D$ which is "somehow represented" (to quote our earlier phrase) by the function $D \to D$ which sends $d$ to $\rho_+^{[x \mapsto d]}(M)$ .

Inductively on the structure of $M$, it is clear from the three displays that, for all $\rho$, the values $\rho_+(M)$ are completely determined, at least once $\psi$ is specified (if the displays really make sense).

And it is believable that $\rho_+(M)$ will be independent of $\rho$ for closed terms $M$. In fact, one would expect to be able to prove the following :

$$\forall M, \forall \rho, \forall \rho' \ [\forall x \ [x \text{ is free in } M \ \Rightarrow \rho(x) = \rho'(x)] \implies \ \rho_+(M) = \rho'_+(M)] \ .$$

This clearly includes the statement about closed $M$.

We'd also want to prove that terms which are related under the basic relation $\approx$ (or rather its non-extensional version $\approx_{\text{in}}$) will map to the same element in $D$.

But before we get carried away trying to construct these proofs, there is one big problem with the bottom display in the triple specification of $\rho_+$'s values :

How do we know, on the right-hand side, that the function $d \mapsto \rho_+^{[x \mapsto d]}(M)$ is actually in the subset $[D \rightharpoonup D]$ ??

This must be dealt with by getting more specific about what the model $(D, \phi, \psi)$ can be, particularly about $\psi$ . Playing around with the difficulty just above, we find that, besides wanting $(\phi \circ \psi)(f) = f$ for all $f \in [D \rightharpoonup D]$, we can solve the difficulty as long as (i) $D$ is combinatorially complete, and (ii) the composite the other way, namely $\psi \circ \phi$, is in $[D \rightharpoonup D]$ .

So now, after all this motivation, here is the definition of $\lambda$-model or model for the $\lambda$-calculus which seems the most natural to me :

**Definition of $\lambda$-model.** It is a structure $(D\ ;\ \cdot\ ;\ \psi)$ such that :
(i) $(D\ ;\ \cdot)$ is a combinatorially complete binary operation with more than one element ;
(ii) $\psi : [D \multimap D] \to D$ is a right inverse for the surjective adjoint $\phi : D \to [D \multimap D]$ of the multiplication in (i), such that $\psi \circ \phi \in [D \multimap D]$ .

Before going further, we should explain how our earlier definition (on which these notes will entirely concentrate after this subsection) is a special case. As noted in the first paragraph of this digression, if the binary operation is extensional, then $\phi$ is injective. But then it is bijective, so it has a unique right inverse $\psi$, and this is a 2-sided inverse. But then the last condition is automatic, since the identity map of $D$ is in $[D \multimap D]$ by combinatorial completeness. So an extensional, combinatorially complete, and non-trivial binary operation is automatically a $\lambda$-model in a unique way, as required.

It is further than we wish to go in the way of examples and proofs, but it happens to be an interesting fact that there exist combinatorially complete binary operations for which the number of $\psi$'s as in (ii) of the definition is zero, and others where there are more than one, even up to isomorphism. In other words, $\lambda$-models are more than just combinatorially complete binary operations—there is extra structure whose existence and uniqueness is far from guaranteed.

Let's now state the theorem which asserts the existence of the maps $\rho_+$ talked about earlier.

**Theorem.** *Let $(D\ ;\ \cdot\ ;\ \psi)$ be a $\lambda$-model. Then there is a unique collection*

$$\{\ \rho_+ :\ \Lambda \to D \mid \rho : \{x_1, x_2, \cdots\} \to D\ \}$$

*for which the following hold :*
(1) $\rho_+(x_i) = \rho(x_i)$ *for all* $i$ ;
(2) $\rho_+(MN) = \rho_+(M) \cdot \rho_+(N)$ *for all terms* $M$ *and* $N$ ;
(3) (i) *The map* $(d \mapsto \rho_+^{[x \mapsto d]}(M))$ *is in* $[D \multimap D]$ *for all* $\rho, x$, *and* $M$ ; *and*
(ii) $\rho_+(\lambda x \bullet M) = \psi(d \mapsto \rho_+^{[x \mapsto d]}(M))$ *for all* $x$ *and* $M$ .

As mentioned earlier, the uniqueness of the $\rho_+$ is pretty clear from (1), (2) and (3)(ii). And the following can also be proved in a relatively straightforward manner, inductively on structure: the dependence of $\rho_+(M)$ on only the values which $\rho$ takes on variables free in $M$, the invariance of $\rho_+$ with respect to the basic equivalence relation "$\approx$" on terms, and several other properties. But these proofs are not just one or two lines, particularly carrying out the induction to establish existence of the $\rho_+$ with property (3)(i). Combinatorial completeness has a major role. We won't give the proof, but the following example should help to make it clear how that induction goes.

**Example.** This illustrates how to reduce calculation of the $\rho_+$ to statement (1) of the theorem, using (2) and (3)(ii), and then why the three needed cases of (3)(i) hold.

$$\rho_+(\lambda xyz \bullet yzx) = \psi(d \mapsto \rho_+^{[x\mapsto d]}(\lambda yz \bullet yzx)) = \psi(d \mapsto \psi(c \mapsto \rho_+^{[x\mapsto d][y\mapsto c]}(\lambda z \bullet yzx)))$$

$$= \psi(d \mapsto \psi(c \mapsto \psi(b \mapsto \rho_+^{[x\mapsto d][y\mapsto c][z\mapsto b]}(yzx)))) = \psi(d \mapsto \psi(c \mapsto \psi(b \mapsto cbd))) \ .$$

But we want to see, as follows, why each application of $\psi$ makes sense, in that it applies to a function which is actually in $[D \mathrel{-\!\!\!\rightsquigarrow} D]$ . Using combinatorial completeness, choose elements $\epsilon, \zeta_1, \zeta_2$ and $\zeta_3$ in $D$ such that, for all $p, q, r, s$ and $t$ in $D$, we have

$$\epsilon p = (\psi \circ \phi)(p) \ \ ; \ \ \zeta_1 pqr = prq \ \ ; \ \ \zeta_2 pqrs = p(qsr) \ \ ; \ \ \zeta_3 pqrst = p(qrst) \ .$$

[Note that a neat choice for $\epsilon$ is $\psi(\psi \circ \phi)$ .]

Now

$$b \mapsto cbd = \zeta_1 cdb = \phi(\zeta_1 cd)(b) \ ,$$

so the innermost one is okay; it is $\phi(\zeta_1 cd) \in [D \mathrel{-\!\!\!\rightsquigarrow} D]$ . But then

$$c \ \ \mapsto \ \ \psi(b \mapsto cbd) = \psi(\phi(\zeta_1 cd)) = \epsilon(\zeta_1 cd) = \zeta_2 \epsilon \zeta_1 dc = \phi(\zeta_2 \epsilon \zeta_1 d)(c) \ ,$$

so the middle one is okay; it is $\phi(\zeta_2 \epsilon \zeta_1 d) \in [D \mathrel{-\!\!\!\rightsquigarrow} D]$ . But then

$$d \ \ \mapsto \ \ \psi(c \mapsto \psi(b \mapsto cbd) = \psi(\phi(\zeta_2 \epsilon \zeta_1 d)) = \epsilon(\zeta_2 \epsilon \zeta_1 d) = \zeta_3 \epsilon \zeta_2 \epsilon \zeta_1 d = \phi(\zeta_3 \epsilon \zeta_2 \epsilon \zeta_1)(d) \ ,$$

so the outer one is okay; it is $\phi(\zeta_3 \epsilon \zeta_2 \epsilon \zeta_1) \in [D \mathrel{-\!\!\!\rightsquigarrow} D]$ .

**Meyer-Scott approach.**

The element $\epsilon$ from the last example determines $\psi \circ \phi$, and therefore $\psi$, since $\phi$ is surjective. It is easy to show that, for all $a$ and $b$ in $D$ ,

$$\epsilon ab = ab \qquad \text{and} \qquad [\forall c, \ ac = bc] \implies \epsilon a = \epsilon b \quad :$$

$$\epsilon \cdot a \cdot b = (\psi \circ \phi)(a) \cdot b = \phi((\psi \circ \phi)(a))(b) = (\phi \circ \psi \circ \phi)(a)(b) = (\phi)(a)(b) = a \cdot b \ .$$

$$[\forall c, \ a \cdot c = b \cdot c] \quad \text{i.e.} \ \ \phi(a) = \phi(b) \qquad \implies \qquad \psi(\phi(a)) = \psi(\phi(b)) \quad \text{i.e.} \ \ \epsilon \cdot a = \epsilon \cdot b \ .$$

Using these two facts, one can quickly recover the properties of $\psi$. Thus our definition of $\lambda$-model can be redone as a triple $(D \ ; \ \cdot \ ; \ \epsilon)$ with (ii) replaced by the two properties just above. A minor irritant is that $\epsilon$ is not unique. That can be fixed by adding the condition $\epsilon\epsilon = \epsilon$. So this new definition is arguably simpler than the one given, but I think less motivatable; but the difference between them is rather slight.

**Scholium on the common approach.**

The notation $\rho_+(M)$ is very non-standard. Almost every other source uses the notation

$$[\![M]\!]_\rho^{\mathcal{M}} \ \ \text{where} \ \ \mathcal{M} = (D \ ; \ \cdot) \ .$$

Both in logic and in denotational semantics (and perhaps elsewhere in computer science), the heavy square brackets $[\![ \ ]\!]$ seem to be ubiquitous for indicating 'a semantic version of

the syntactic object inside the $[\![\ ]\!]$'s'. As mentioned earlier, the vast majority of sources for $\lambda$-*model* (and for weaker (more general) notions such as $\lambda$-*algebra* and *proto-*$\lambda$-*algebra*) express the concepts' definitions by lists of properties of the semantic function $[\![\ ]\!]$. And, of course, you'll experience a heavy dosage of "$\models$" and "$\vdash$". Again **[Hind-Seld]**, Ch. 11 (and Ch. 10 for models of combinatory logic), is the best place to start, esp. pp.112-122, for the list of properties, first the definition then derived laws.

Given such a definition, however weakly motivated, one can get to our definition—more quickly than the other way round (the theorem above embellished)—by defining $\psi$ using

$$\psi(\phi(a)) \ := \ \rho_+^{[y\mapsto a]}(\lambda x \bullet yx) \ .$$

Note that $\lambda x \bullet yx \approx y$, but $\lambda x \bullet yx \not\approx_{\text{in}} y$, as expected from this. The properties of $\rho_+$ vaguely alluded to above assure one that

$$\phi(a) = \phi(b) \quad [\text{i.e. } \forall d,\ ad = bd] \quad \Longrightarrow \quad \rho_+^{[y\mapsto a]}(\lambda x \bullet yx) = \rho_+^{[y\mapsto b]}(\lambda x \bullet yx) \ ,$$

whereas $\rho_+^{[y\mapsto a]}(y) \neq \rho_+^{[y\mapsto b]}(y)$ clearly, if $a \neq b$ !

The combinatorial completeness in our definition will then follow from Schonfinkel's theorem, noting that $\sigma := \rho_+(\lambda xyz \bullet (xz)(yz))$ and $\kappa := \rho_+(\underline{T})$ (for any $\rho$) have the needed properties.

Again, from the viewpoint which regards $[\![\ ]\!]$, or $\rho_+$, as primary, the element $\epsilon$ previous would be defined by $\epsilon := \rho_+(\lambda xy \bullet xy)$ for any $\rho$ ;—hardly surprising in view of the law $\epsilon ab = ab$ .

Finally, going the other way, a quick definition of $\rho_+$ for an *extensional* $D$ is just the composite

$$\rho_+ \ = \ (\Lambda \ \xrightarrow{\Phi} \ \Gamma \ \xrightarrow{\rho_*} \ D) \ ,$$

where we recall from VII-6 that $\Gamma$ is the free binary operation on $\{S, K, x_1, x_2, \cdots\}$ and the map $\Phi$ is the one inducing the isomorphism between $\Lambda/\approx$ and $\Gamma/\sim$ , and here, $\rho_*$ is defined to be the unique morphism of binary operations taking each $x_i$ to $\rho(x_i)$, and taking $S$ and $K$ to the elements $\sigma$ and $\kappa$ from Schonfinkel's theorem (unique by extensionality).

**Term model.**

The set of equivalence classes $\Lambda/\approx_{\text{in}}$ is a combinatorially complete non-trivial binary operation. It is not extensional, but would be if we used "$\approx$" in place of "$\approx_{\text{in}}$". The canonical way to make it into a $\lambda$-model, using our definition here, is to define

$$\psi(\phi([M]_{\approx_{\text{in}}})) \ := \ [\lambda x \bullet Mx]_{\approx_{\text{in}}} \qquad \text{for any } x \text{ not occurring in } M \ .$$

It will be an exercise for you to check that this is well-defined, and satisfies (ii) in the definition of $\lambda$-*model*. Notice that, as it should, if we used "$\approx$" in place of "$\approx_{\text{in}}$", the displayed formula would just say that $\psi \circ \phi$ is the identity map.

**What does a $\lambda$-model do?**

(1) For some, using the Meyer-Scott or 'our' definition, it is sufficient that it provides an interesting genus of mathematical structure, one for which finding the 'correct' definition

had been a decades-long effort, and for which finding examples more interesting than the term models was both difficult and important. See the next subsection.

(2) For computer scientists, regarding the $\lambda$-calculus as a super-pure functional programming language, the functions $[\![\ ]\!]$ are the denotational semantics of that language. See the rest of this subsection, the table on p.155 of [**St**], and the second half of Subsection VII-9.

(3) For logicians, $[\![\ ]\!]$ provides the semantics for a putative form of proto-logic. Or, coming down from that cloud, one may think of $[\![\ ]\!]$ as analogous to the "Tarski definition of truth" in standard $1^{\underline{\text{st}}}$order logic. Again, read on !

**Back to regular programming.**

Why should one wish to produce models other than the term models? (There are at least four of those, corresponding to using " $\approx$ " or " $\approx_{\text{in}}$ ", and to restricting to closed terms or allowing terms with free variables. But the closed term models actually don't satisfy our definition above of $\lambda$-model, just one of the weaker definitions vaguely alluded to earlier.) As far as I can tell, besides the intrinsic mathematical interest, the requirements of denotational semantics include the need for individual objects inside the models with which one can work with mathematical facility. The syntactic nature of the elements in the term models dictate against this requirement. Further requirements are that one can manipulate with the models themselves, and construct new ones out of old with gay abandon. See the remarks below for more on this, as well as Subsection VII-9.

So we shall content ourselves, in the next subsection, 'merely' to go through Scott's original construction, which (fortunately for our pedagogical point of view) did produce *extensional* models. First I'll make some motivational and other remarks about the computer science applications. This remaining material also ties in well with some of the rather vague comments in VII-3.

The remainder of this subsection contains what I have gleaned from an unapologetically superficial reading of some literature on denotational semantics. To some extent, it may misreprepresent what has been really in the minds of the pioneers/practitioners of this art. I would be grateful for corrections/criticisms from any knowledgeable source. See Subsection VII-9 for some very specific such semantics of the language **ATEN** from the main part of this paper, and for the semantics of the $\lambda$-calculus itself.

We did give what we called the semantics of **BTEN** right after the syn-

tactic definition of that language. As far as I can make out, that was more like what the CSers would call an "operational semantics", rather than a denotational semantics. The reason is that we referred to a 'machine' of sorts, by talking about "bins" and "placing numbers in bins after erasing the numbers already there". That is a pretty weak notion of a machine. But to make the "meaning" of a language completely "machine independent" is one of the main criteria for an adequate denotational semantics.

See also the following large section on Floyd-Hoare logic. In it, we use a kind of operational semantics for several command languages. These assign, to a (command, input)-pair, the entire sequence of states which the pair generates. So that's an even more 'machine-oriented' form of operational semantics than the above input/output form. On the other hand, Floyd-Hoare logic itself is sometimes regarded as a form of semantic language specification, much more abstract than the above forms. There seems to have been (and still is?) quite a lot of controversy as to whether the denotational viewpoint is preferable to the above forms, especially in giving the definitions needed to make sense of questions of soundness and completeness of the proof systems in F-H logic. But we are getting ahead of ourselves.

Along with some of the literature on F-H logic, as well as some of Dana Scott's more explanatory productions, the two CS textbooks [**Go**] and [**Al**] have been my main sources. The latter book goes into the mathematical details (see the next subsection) somewhat more than the former.

In these books, the authors start with very small imperative programming languages : **TINY** in [**Go**], and "simple language" in Ch.5 of [**Al**]. These are clearly very close to **ATEN/BTEN** from early in this work. They contain some extras which look more like features in, say, **PASCAL**. Examples of "extras" are :

(1) the command "OUTPUT $E$" in [**Go**] ;

(2) the use of "BEGIN... END" in [**Al**], this being simply a more readable replacement for brackets ;

(3) the SKIP-command in [**Al**], which could be just $\mathsf{whdo}(0 = 1)(\text{any } C)$ or $x_0 \leftarrow\!\!\!\div\, x_0$ from **ATEN** .

So we may ask : which purely mathematical object corresponds to our mental image of a bunch of stored natural numbers, with the number called

$v_i$ stored in bin number $i$ ? That would be a function

$$\sigma \ : \ IDE \ \rightarrow \ VAL \ \ ,$$

where $\sigma$ is called a state, $IDE$ is the (syntactic) set of identifiers, and $VAL$ is the (semantic) 'set' of values.

Now $IDE$ for an actual command language might, for example, be the set of all finite strings of symbols from the 62-element symbol set consisting of the 10 decimal digits and the $(26 \times 2)$ upper- and lower-case letters of the Roman alphabet, with the proviso that the string begin with a letter (and with a few exclusions, to avoid words such as "while" and "do" being used as identifiers). For us in **BTEN**, the set $IDE$ was simply the set $\{x_0, x_1, x_2, \cdots\}$ of variables. (In each case we of course have a countably infinite set. And in the latter case, there would be no problem in changing to a set of finite strings from a finite alphabet, for example the strings $x_{|||\cdots||}$ . Actually, as long as the subscripts are interpreted as strings of [meaningless?] digits, as opposed to Platonic integers, we already have a set of finite strings from a finite alphabet.)

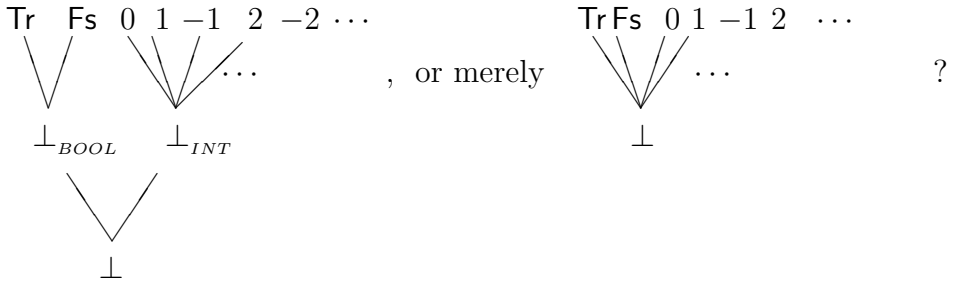
In referring above to $VAL$, we used " 'set' " rather than "set", because this is where $\lambda$-calculus models begin to come in, or what are called domains in this context. For us, $VAL$ was just the set $\mathbf{N}= \{0, 1, 2, \cdots\}$ of all natural numbers. And $\sigma$ was the function mapping $x_i \in IDE$ to $v_i \in VAL$. But for doing denotational semantics with a real, practical language, it seems to be essential that several changes including the following are made :
Firstly, the numbers might be more general, such as allowing all integers (negatives as well).
Secondly, $VAL$ should contain something else, which is often called $\perp$ in this subject. It corresponds more-or-less to our use of "*err*" much earlier.
Thirdly, we should have a (rather weak) partial order $\sqsubseteq$ on the objects above (much more on this in the next subsection). As to the actual ordering here, other than requiring $b \sqsubseteq b$ for all $b$ (part of the definition of *partial order*), we want $\perp \sqsubseteq c$ for all numbers $c \in VAL$ (but no other relation $\sqsubseteq$ between elements as above). So $\sqsubseteq$ is a kind of 'how much information?' partial order.
Finally, for a real command language, $VAL$ would contain other semantic objects such as the truth values Tr and Fs. So, for example, in [**Al**], p.34, we see definitions looking like

$$Bv \ = \ INT + BOOL + Char + \cdots \ \ ,$$

and

$$VAL \;=\; Bv + [VAL + VAL] + [VAL \times VAL] + [VAL \rightharpoonup VAL] \quad .$$

Later there will be more on that last 'equation' and similar ones. Quite apart from the $\cdots$'s, **[Al]**'s index has no reference to $Char$ (though one might guess), and its index has double the page number, p.68, for $Bv$, which likely stands for 'basic values'. Also, even using only numbers and truth values, there are some decisions to make about the poset $VAL$ ; for example, should it contain




In any case, it is crucial that we have at least the following :

(1) Some syntactically defined sets, often $IDE$, $EXP$ and $COM$, where the latter two respectively are the sets of expressions and commands.

(2) Semantically defined 'domains' such as $VAL$ and perhaps

$$STATE \;=\; [IDE \rightharpoonup VAL] \;,$$

where the right-hand side is a subposet of the poset of all functions from $IDE$ to the underlying set of the structure $VAL$.

(3) Semantically defined functions

$$\mathcal{E} \;:\; EXP \;\rightarrow\; [STATE \rightharpoonup VAL] \;,$$

and

$$\mathcal{C} \;:\; COM \;\rightarrow\; [STATE \rightharpoonup STATE] \;.$$

The situation in **[Go]** is a bit messier (presumably of necessity) because $\perp$ is not used as a value, but separated out, called sometimes "unbound" and sometimes "error", but systematically.

Be that as it may, let us try to explain $\mathcal{E}$ and $\mathcal{C}$ in the case of our simple language **BTEN/ATEN**. Much more detail on this appears two subsections ahead.

Here $EXP$ is the set of terms and quantifier-free formulae from the language of $1^{\underline{\text{st}}}$order number theory. Then $\mathcal{E}$ is that part of the Tarski definition of truth as applied to the particular interpretation, $\mathbf{N}$, of that $1^{\underline{\text{st}}}$order language which

(1) assigns to a term $t$, in the presence of a state $\underline{v}$, the natural number $t^{\underline{v}}$ . That is, adding in the totally uninformative 'truth value' $\perp$,

$$\mathcal{E}[t](\underline{v}) \ := \ t^{\underline{v}} \ .$$

(But here we should add that $t^{\underline{v}} = \perp$ if any $v_i$, with $x_i$ occurring in $t$, is $\perp$.)

(2) and assigns to a formula $G$, in the presence of a state $\underline{v}$, one the truth values. That is,

$$\mathcal{E}[G](\underline{v}) \ := \ \begin{cases} \mathsf{Tr} & \text{if } G \text{ is true at } \underline{v} \ ; \\ \mathsf{Fs} & \text{if } G \text{ is false at } \underline{v} \ ; \\ \perp & \text{if any } v_i, \text{ with } x_i \text{ free in } G, \text{ is } \perp \ . \end{cases}$$

Also, $COM$ is the set of commands in **ATEN** or **BTEN**. And $\mathcal{C}$ assigns to a command $C$, in the presence of a state $\underline{v}$, the new state $||C||(\underline{v})$ . See the beginning of Subsection VII-9 for more details.

So it appears that there is no great difference between the 'operational' semantics already given and this denotational semantics, other than allowing $\perp$ as a 'totally undefined number' or as a 'totally uninformative truth value'. But such a remark unfairly trivializes denotational semantics for several reasons. Before expanding on that, here is a question that experts presumably have answered elsewhere.

What are the main impracticalities of trying to bypass the whole enterprise of denotational semantics as follows ?

(1) Give, at the syntactic level, a translation algorithm of the relevant practical imperative programming language back into **ATEN**. For example, in the case of a self-referential command (i.e. recursive program), I have already done that in **IV-9** of this work. I can imagine that $GOTO$-commands would present some problems, but presumably not insuperable ones.

(2) Then use the simple semantics of **ATEN** to do whatever needs to be done, such as trying to prove that a program never loops, or that it is correct

according to its specifications, or that an F-H proof system is complete, or, indeed, such as implementing the language.

This is the sort of thing mentioned by Schwartz [**Ru-ed**] pp. 4-5, but reasons why it is not pursued seem not to be given there. Certainly one relevant remark is that, when one (as in the next section on F-H logic) generalizes **ATEN** by basing it on an arbitrary $1^{\underline{\text{st}}}$order language and its semantics on an arbitrary interpretation of that language, the translation as in (1) may no longer be possible. And so, even when the interpretation contains the natural numbers in a form which permits Gödel numbering for coding all the syntax, the translation might lack some kind of desirable naturality. And its efficiency would be a problem for sure.

This is becoming verbose, but there are still several matters needing explanation, with reference to why denotational semantics is of interest.

First of all, it seems clear that the richer 'extras' in real languages such as PASCAL and ALGOL60, as opposed to the really basic aspects already seen in these simpler languages, are where the indispensibility of denotational semantics really resides. (As we've emphasized, though these extras make programming a tolerable occupation, they don't add anything to what is programmable in principle.) Examples here would be those referred to in (1) just above—self-referential or recursive commands, and *GOTO*-statements (which are also self-referential in a different sense), along with declarations and calls to procedures with parameters, declarations of variables (related to our $\mathcal{B} - \mathcal{E}$-command in **BTEN**), declarations of functions, etc. Here we are completely ignoring parallelism and concurrent programming, which have become very big topics in recent years.

But I also get the impression that *mechanizing* the projects which use denotational semantics is a very central aspect here. See the last chapter of [**Al**], where some of it is made *executable*, in his phraseology. The mathematical way in which we had originally given the semantics of **BTEN** is inadequate for this. It's not *recursive* enough. It appears from [**Go**] and [**Al**], without being said all that explicitly, that one aspect of what is really being done in denotational semantics is to translate the language into a form of the $\lambda$-calculus, followed by perhaps some standard maps (like the $\rho_+$ earlier) of the latter into one or another "domain". So the metalanguage of the first half of this semantics becomes quite formalized, and (it appears to me), is a pure functional (programming?) language. (Perhaps other pure func-

tional programming languages don't need so much denotational semantics (beyond that given for the $\lambda$-calculus itself) since they are already in that form?)

For example, lines -5 and -6 of p. 53 of [**Al**] in our notation above and applied to **BTEN** become

$$\mathcal{C}[\mathbf{ite}(F)(C)(D)](\sigma) \ := \ (\text{if } \mathcal{E}[F](\sigma) = \mathsf{Tr}, \text{ then } \mathcal{C}[C], \text{ else } \mathcal{C}[D])(\sigma) \ ,$$

for all formulas $F$, commands $C$ and $D$, and states $\sigma$ . At first, this appears to be almost saying nothing, with the if-then-else being semanticized in terms of if-then-else. But note how the right-hand side is a statement from the informal version of **McSELF**, in that what follows the "then" and "else" are not commands, but rather function values. In a more formal $\lambda$-calculus version of that right-hand side, we would just write a triple product, as was explained in detail at the beginning of Subsection VII-5.

Even more to the point here, lines -3 and -4 of p. 53 of [**Al**] in our notation above and applied to **BTEN** become

$$\mathcal{C}[\mathbf{whdo}(F)(C)](\sigma) \ := \ (\text{if } \mathcal{E}[F](\sigma) = \mathsf{Tr}, \text{then } \mathcal{C}[\mathbf{whdo}(F)(C)] \circ \mathcal{C}[C], \text{else } Id)(\sigma) \ ,$$

First notice how much more compact this is than the early definition in the semantics of **BTEN**. And again, the "if-then-else" on the right-hand side would be formalized as a triple product. But much more interestingly here, we have a self-reference, with the left-hand side appearing buried inside the right-hand side. So here we need to think about solving equations. The $\underline{Y}$-operator does that for us systematically in the $\lambda$-calculus, which is where the right-hand side resides, in one sense. The discussion of least fixed points ending the next subsection is clearly relevant. A theorem of Park shows that, at least in Scott's models from the next subsection, the same result is obtained from Tarski's 'least fixed points' operator in classical lattice theory (see **VII-8.12** ending the next subsection), as comes from the famous $\underline{Y}$-operator of Curry within $\lambda$-calculus.

We shall return to denotational semantics after the more purely mathematical subsection to follow on Scott's construction of $\lambda$-calculus models for doing this work.

**VII-8 Scott's Original Models.**

We wish to approach from an oblique angle some of the fundamental ideas of Dana Scott for constructing extensional models of the $\lambda$-calculus. There are many other ways to motivate this for mathematics students. In any case, presumably he won't object if this approach doesn't coincide exactly with his original thought processes.

One of several basic ideas here, in a rather vague form, is to employ some mathematical structure on $A$ in order to construct such a model with underlying set $A$. We wish to define a (relatively small) subset of $A^A$, which we'll call $[A \multimap A]$, and a bijection between it and $A$ itself. The corresponding extensional binary operation on $A$ (via the adjointness discussed in the first paragraph of the digression in the last subsection) we hope will be combinatorially complete; that is, all its algebraically definable functions should be representable. And of course, representable implies algebraically definable almost by definition. To pick up on the idea again, can we somehow build a structure so that $[A \multimap A]$ turns out to be precisely the 'structure-preserving' functions from $A$ to itself? If this were the case, and the structure-preserving functions satisfied a few simple and expected properties (with respect to composition particularly), then the proof of combinatorial completeness would become a fairly simple formality :

representable $\Longrightarrow$ algebraically definable

$\Longrightarrow$ structure-preserving $\Longrightarrow$ representable .

In the next several paragraphs, the details of how such an object would give an extensional model for the $\lambda$-calculus are given in an axiomatic style, leaving for later 'merely' the questions of what "structure" to use, and of self-reflection.

**Details of essentially the category-theoretic approach.**

Assume the following:

Data

Given a collection of 'sets with structure', and, for any two such objects, $A$ and $B$, a subset $[A \multimap B] \subset B^A$ of 'structure preserving functions'. Given also a canonical structure on both the Cartesian product $A \times B$ and on $[A \multimap B]$ .

Axioms

(1) The *ith* projection is in $[A_1 \times \cdots \times A_n \multimap A_i]$ for $1 \leq i \leq n$ ; in

particular, taking $n = 1$, the identity map, $id_A$, is in $[A \rightsquigarrow A]$ .

(2) If $f \in [A \rightsquigarrow B]$ and $g \in [B \rightsquigarrow C]$ then the composition $g \circ f$ is always in $[A \rightsquigarrow C]$ .

(3) The diagonal map $\delta$ is in $[A \rightsquigarrow A \times A]$ , where $\delta(a) := (a, a)$ .

(4) Evaluation restricts to a map $ev$ in $[[A \rightsquigarrow B] \times A \rightsquigarrow B]$ , where $ev(f, a) = f(a)$.

(5) If $f_1 \in [A_1 \rightsquigarrow B_1]$ and $f_2 \in [A_2 \rightsquigarrow B_2]$ then the $f_1 \times f_2$ is necessarily in $[A_1 \times A_2 \rightsquigarrow B_1 \times B_2]$ , where $(f_1 \times f_2)(a_1, a_2) := (f_1(a_1), f_2(a_2))$.

(6) All constant maps are "structure preserving".

(7) The adjointness bijection, from $A^{B \times C}$ to $(A^C)^B$, maps $[B \times C \rightsquigarrow A]$ into $[B \rightsquigarrow [C \rightsquigarrow A]]$ .

*All this will be relatively easy to verify, once we've chosen the appropriate "structure" and "structure preserving maps", to make the following work, which is more subtle. That choosing will also be motivated by the application to denotational semantics.*

Self-reflective Object

Now suppose given one of these objects $A$, and a mutually inverse pair of bijections which are structure preserving :

$$\phi \in [A \rightsquigarrow [A \rightsquigarrow A]] \quad \text{and} \quad \psi = \phi^{-1} \in [[A \rightsquigarrow A] \rightsquigarrow A] \ .$$

We want to show how such a "self-reflective" object may be made canonically into an extensional combinatorially complete binary operation.

Define the multiplication in $A$ from $\phi$, using adjointness—see the first paragraph of the digression in the last subsection—and so it is the following composite :

$$\begin{array}{rccccc} mult \ : & A \times A & \stackrel{\phi \times id}{\longrightarrow} & [A \rightsquigarrow A] \times A & \stackrel{ev}{\longrightarrow} & A \\ & (x, y) & \mapsto & (\phi(x), y) & \mapsto & \phi(x)(y) := x \cdot y \end{array}$$

By (1),(2),(4) and (5), this map, which we'll name $mult$, is in $[A \times A \rightsquigarrow A]$.

Now $[A \rightsquigarrow A]$ contains $id_A$ and all constant maps, and is closed under pointwise multiplication of functions as follows :

$$\begin{array}{ccccccc} A & \stackrel{\delta}{\longrightarrow} & A \times A & \stackrel{f \times g}{\longrightarrow} & A \times A & \stackrel{mult}{\longrightarrow} & A \\ x & \mapsto & (x,x) & \mapsto & (f(x), g(x)) & \mapsto & f(x) \cdot g(x) \end{array}$$

The fact that $\phi$ is injective implies that *mult* is extensional, as we noted several times earlier. First we shall check the 1-variable case of combinatorial completeness of *mult*.

**Definitions.** (More-or-less repeated from much earlier.)

Say that $f \in A^A$ is *1-representable* when there is a $\zeta \in A$ such that $f(a) = \zeta \cdot a$ for all $a \in A$ .

Define the set of *1-algebraically definable* functions in $A^A$ to be the smallest subset of $A^A$ containing $id_A$ and all constant functions, and closed under pointwise multiplication of functions.

*1-combinatorial completeness* of $A$ is the fact that the two notions just above coincide.

Its proof is now painless in the form

1-representable $\Longrightarrow$ 1-algebraically definable

$\Longrightarrow$ structure-preserving $\Longrightarrow$ 1-representable .

The first implication is because a 1-representable $f$ as in the definition is the pointwise multiplication of (the constant function with value $\zeta$) times (the identity function).

The second implication is the fact noted above that the set $[A \rightharpoonup\!\!\!\!\!\sim\ A]$ is an example of a set containing $id_A$ and all constant functions, and closed under pointwise multiplication of functions.

(Of course, the equation in the definition of 1-representability can be rewritten as $f = \phi(\zeta)$ , so the equivalence of 1-representability with structure preserving becomes pretty obvious.)

The third implication goes as follows :

Given $f \in [A \rightharpoonup\!\!\!\!\!\sim\ A]$, define $\zeta$ to be $\psi(f)$ . then

$$\zeta \cdot a = \psi(f) \cdot a = \phi(\psi(f))(a) = f(a) \ ,$$

as required.

Now we shall check the 2-variable case of combinatorial completeness, and leave the reader to pump this up into a proof for any number of variables.

**Definitions.**

Say that $f \in A^{A\times A}$ is *2-representable* when there is a $\zeta \in A$ such that $f(b,c) = (\zeta \cdot b) \cdot c$ for all $b$ and $c$ in $A$ .

Define the set of *2-algebraically definable* functions in $A^{A\times A}$ to be the smallest subset of $A^{A\times A}$ containing both projections and all constant functions, and closed under pointwise multiplication of functions.

*2-combinatorial completeness* of $A$ is the fact that the two notions just above coincide.

Its proof is much as in the 1-variable case :

2-representable $\Longrightarrow$ 2-algebraically definable

$\Longrightarrow$ structure-preserving $\Longrightarrow$ 2-representable .

The first implication is because a 2-representable $f$ as in the definition is a suitably sequenced pointwise multiplication of the constant function with value $\zeta$ and the two projections.

The second implication is the fact that the set $[A \times A \multimap A]$ is an example of a set containing the projections and all constant functions, and closed under pointwise multiplication of functions. The latter is proved by composing:

$$A \times A \quad \stackrel{\delta}{\longrightarrow} \quad (A \times A) \times (A \times A) \quad \stackrel{f\times g}{\longrightarrow} \quad A \times A \quad \stackrel{mult}{\longrightarrow} \quad A$$

$$(b,c) \quad \mapsto \quad ((b,c),(b,c)) \quad \mapsto \quad (f(b,c),g(b,c)) \quad \mapsto \quad f(b,c)\cdot g(b,c)$$

The third implication goes as follows :

If $f \in [A \times A \multimap A]$, the composite $\psi \circ adj(f)$ is in $[A \multimap A]$, using axiom (7) for the first time. By the part of the 1-variable case saying that *structure preserving* implies *1-representable*, choose $\zeta$ so that, for all $b \in A$, we have $\psi(adj(f)(b)) = \zeta \cdot b$ . Then

$$\zeta \cdot b \cdot c = \psi(adj(f)(b)) \cdot c = \phi(\psi(adj(f)(b)))(c) = adj(f)(b)(c) = f(b,c) \ ,$$

as required.

So now we must figure out what kind of structure will work, and how we might produce a self-reflective object.

The next idea has already occurred in the verbose discussion of denotational semantics of the previous subsection : the structure referred to above

maybe should be a partial order with some extra properties (so that, in the application, it intuitively coincides with 'comparing information content').

Now I believe (though it's not always emphasized) that an important part of Scott's accomplishment is not just to be first to construct a $\lambda$-model (and do so by finding a category and a self-reflective object in it), but also to show how to start with an individual from a rather general species of posets

(in the application, from { numbers , truth values , $\perp$ } at least) ,
and show how to embed it (as a poset) into an extensional $\lambda$-model.

So let's start with any poset $(D, \sqsubseteq)$ and see how far we can get before having to impose extra properties. Recall that the definition of *poset* requires

(i) $a \sqsubseteq b$ and $b \sqsubseteq c$ implies $a \sqsubseteq c$ ; and

(ii) $a \sqsubseteq b$ and $b \sqsubseteq a$ if and only if $a = b$ .

Temporarily define $[D \multimap D]$ to consist of all those functions $f$ which preserve order; that is

$$d \sqsubseteq e \implies f(d) \sqsubseteq f(e) \ .$$

More generally this defines $[D \multimap E]$ where $E$ might not be the same poset as $D$.

Now $[D \multimap D]$ contains all constant functions, so we can embed $D$ into $[D \multimap D]$ by $\phi_D : d \mapsto (d' \mapsto d)$ . This $\phi_D : D \to [D \multimap D]$ will seldom be surjective. It maps each $d$ to the constant function with value $d$.

Next suppose that $D$ has a minimum element called $\perp$ . Then we can map $[D \multimap D]$ back into $D$ by $\psi_D : f \mapsto f(\perp)$ . It maps each function to its minimum value.

It is a trivial calculation to show that $\psi_D \circ \phi_D = id_D$, the identity map of $D$. By definition, this shows $D$ to be a *retract* of $[D \multimap D]$ . (Note how this is the reverse of the situation in the digression of the last subsection, on the general definition of $\lambda$-model, where $[D \multimap D]$ was a retract of $D$.) In particular, $\phi_D$ is injective (obvious anyway) and $\psi_D$ is surjective.

The set $[D \multimap D]$ itself is partially ordered by

$$f \sqsubseteq g \quad \iff \quad \forall d \ , \ f(d) \sqsubseteq g(d) \ .$$

This actually works for the set of *all* functions from any set to any poset.

How do $\phi_D$ and $\psi_D$ behave with respect to the partial orders on their domains and codomains? Very easy calculations show that they do preserve the orders, and thus we have

$$\phi_D \in [D \multimap [D \multimap D]] \quad \text{and} \quad \psi_D \in [[D \multimap D] \multimap D] \ .$$

So $D$ is a retract of $[D{-\!\!\rightarrow} D]$ *as a poset*, not just as a set. But $[D{-\!\!\rightarrow} D]$ is usually too big—the maps above are not inverses of each other, only injective and surjective, and one-sided inverses of each other, as we already said.

Now here's another of Scott's ideas : whenever you have a retraction pair $D \overset{\hookrightarrow}{\twoheadleftarrow} E$ , you can automatically produce another retraction pair

$$[D{-\!\!\rightarrow} D] \quad \overset{\hookrightarrow}{\twoheadleftarrow} \quad [E{-\!\!\rightarrow} E] \quad .$$

If $(\phi, \psi)$ is the first pair and $(\phi', \psi')$ the second pair, then the formulae defining the latter are

$$\phi'(f) \;:=\; \phi \circ f \circ \psi \quad \text{and} \quad \psi'(g) \;:=\; \psi \circ g \circ \phi \quad .$$

Again a very elementary calculation shows that this is a retraction pair.

[As an aside which is relevant to your further reading on this subject, notice how what we've just done is purely 'arrow-theoretic' : the only things used are associativity of composition and behaviour of the identity morphisms. It's part of *category theory*. In fact Lambek has, in a sense, identified the theory of combinators, and of the typed and untyped $\lambda$-calculi, with the theory of *cartesian closed categories*. The "closed" part is basically the situation earlier with the axioms (1) to (7) where the set of morphisms between two objects in the category is somehow itself made into an object in the category, an object with good properties.]

Now it is again entirely elementary to check that, for the category of posets and order-preserving maps, the maps $\phi'$ and $\psi'$ do in fact preserve the order.

Back to the earlier situation in which we have the poset $D$ as a retract of the poset $E = [D{-\!\!\rightarrow} D]$, the construction immediately above can be iterated : Define

$$D_0 := D \quad , \quad D_1 := [D_0{-\!\!\rightarrow} D_0] \quad , \quad D_2 := [D_1{-\!\!\rightarrow} D_1] \quad , \quad \text{etc. } \cdots \; .$$

Then the initial retraction pair $D_0 \overset{\hookrightarrow}{\twoheadleftarrow} D_1$ gives rise by the purely category-theoretic construction to a sequence of retraction pairs $D_n \overset{\hookrightarrow}{\twoheadleftarrow} D_{n+1}$ . We'll denote these maps as $(\phi_n, \psi_n)$ .

Ignoring the surjections for the moment, we have

$$D_0 \overset{\phi_0}{\hookrightarrow} D_1 \overset{\phi_1}{\hookrightarrow} D_2 \overset{\phi_2}{\hookrightarrow} \cdots \; .$$

Wouldn't it be nice to be able to 'pass to the limit', producing a poset $D_\infty$ as essentially a union. And then to be able to just set $n = \infty = n+1$, and say that we'd get canonical *bijections* back-and-forth between $D_\infty$ and $[D_\infty \!\!\rightharpoonup\!\!\!\!\!\rightarrow D_\infty]$ ??? (The first $\infty$ is $n+1$, and the second one is $n$, so to speak.) After all, that was the original objective here !! In fact, going back to some of the verbosities in Subsection VII-3, this looks as close as we're likely to get to a mathematical situation in which we have a non-trivial structure $D_\infty$ which can be identified in a natural way with its set of (structure-preserving!) self-maps. The binary operation on $D_\infty$ would of course come from the adjointness discussed at the beginning of this subsection; that is,

$$a_\infty \cdot b_\infty \;:=\; (\phi_\infty(a_\infty))(b_\infty) \quad ,$$

where $\phi_\infty$ is the isomorphism from $D_\infty$ to $[D_\infty \!\!\rightharpoonup\!\!\!\!\!\rightarrow D_\infty]$ .

Unfortunately, life isn't quite that simple; but really not too complicated either. The Scott construction which works is not to identify $D_\infty$ as some kind of direct limit of the earlier displayed sequence—thinking of the $\phi$'s as actual inclusions, that would be a nested union of the $D_n$'s—but rather to define $D_\infty$ as the inverse limit, using the sequence of surjections

$$D_0 \overset{\psi_0}{\twoheadleftarrow} D_1 \overset{\psi_1}{\twoheadleftarrow} D_2 \overset{\psi_2}{\twoheadleftarrow} \cdots \quad .$$

To be specific, let

$$D_\infty \;:=\; \{\; (d_0, d_1, d_2, \cdots) \mid \text{for all } i, \text{ we have } d_i \in D_i \text{ and } \psi_i(d_{i+1}) = d_i \;\} \quad .$$

Partially order $D_\infty$ by

$$(d_0, d_1, d_2, \cdots) \sqsubseteq (e_0, e_1, e_2, \cdots) \iff \text{for all } i, \text{ we have } d_i \sqsubseteq e_i \quad .$$

This is easily seen to be a partial order (even on the set of *all* sequences, i.e. on the infinite product $\Pi_{i=0}^{\infty} D_i$ ).

[Notice that, as long as $D_0$ has more than one element, the set $D_\infty$ will be uncountably infinite in cardinality. So despite this all being motivated by very finitistic considerations, it has drawn us into ontologically sophisticated mathematics.]

And there's another complication, but one of a rather edifying nature (in my opinion). In order to make the following results actually correct, we need

to go back and change *poset* to complete lattice and *order preserving map* to continuous map. The definitions impose an extra condition on both the objects and the morphisms.

**Definition.** A *complete lattice* is a poset $(D, \sqsubseteq)$, where every subset of $D$ has a *least upper bound*. Specifically, if $A \subset D$, there is an element $\ell \in D$ such that

(i) $a \sqsubseteq \ell$ for all $a \in A$ ; and

(ii) for all $d \in D$, if $a \sqsubseteq d$ for all $a \in A$, then $\ell \sqsubseteq d$ .

It is immediate that another least upper bound $m$ for $A$ must satisfy both $\ell \sqsubseteq m$ and $m \sqsubseteq \ell$ , so the least upper bound is unique. We shall use $\sqcup A$ to denote it. In particular, a complete lattice always has a least element $\sqcup\emptyset$, usually denoted $\perp$ .

**Definition.** A function $f : D \to E$ between two complete lattices is called *continuous* if and only if $f(\sqcup A) = \sqcup f(A)$ for all *directed subsets* $A$ of $D$, where $f(A)$ is the usual image of the subset under $f$. Being *directed* means that, for any two elements $a$ and $b$ of $A$, there is a $c \in A$ with $a \sqsubseteq c$ and $b \sqsubseteq c$.

It follows for continuous $f$ that $f(\perp_D) = \perp_E$. Also, such an $f$ preserves order, using the fact that, if $x \sqsubseteq y$, the $x \sqcup y := \sqcup\{x, y\} = y$ . If $f$ is bijective, we call it an *isomorphism*. Its inverse is automatically continuous.

**Definition.** The set $[D \multimap E]$ is defined to consist of all continuous maps from $D$ to $E$.

**Scott's Theorem.** *Let $D$ be any complete lattice. Then there is a complete lattice $D_\infty$ which is isomorphic to $[D_\infty \multimap D_\infty]$, and into which $D$ can be embedded as a sublattice.*

For the theorem to be meaningful, the set $[D_\infty \multimap D_\infty]$ is made into a complete lattice as indicated in the first exercise below. In view of that exercise, this theorem is exactly what we want, according to our elementary axiomatic development in the last several pages.

For Scott's theorem, we'll proceed to outline two proofs in the form of sequences of exercises. First will be a rather "hi-fallutin' " proof, not much different than Lawvere's suggestion in **[La-ed]** p.179, but less demanding of sophistication about categories on the reader's part (as opposed to *categorical sophistication*, which must mean sophistication about one and only one thing,

up to isomorphism!).

Both proofs use the following exercise.

**Ex. VII-8.1.** (a) Verify the seven axioms for the category of complete lattices and continuous maps, first verifying (and doing part (b) below simultaneously) that $D \times E$ and $[D \!\rightsquigarrow\! E]$ are complete lattices whenever $D$ and $E$ are, using their canonical orderings as follows :

$$(d,e) \sqsubseteq_{D\times E} (d',e') \iff d \sqsubseteq_D d' \text{ and } e \sqsubseteq_E e' \ ;$$
$$f \sqsubseteq_{[D\rightsquigarrow E]} g \iff f(d) \sqsubseteq_E g(d) \text{ for all } d \in D \ .$$

*The subscripts on the "* $\sqsubseteq$ *" will be omitted in future, as they are clear from the context, and the same for the subscripts on the "* $\sqcup$ *" below.*

(b) Show also that the least upper bounds in these complete lattices are given explicitly by

$$\sqcup_{D\times E} A \ = \ (\sqcup_D p_1(A), \sqcup_E p_2(A)) \quad \text{where the } p_i \text{ are the projections, and}$$
$$(\sqcup_{[D\rightsquigarrow E]} A)(x) \ = \ \sqcup_E \{f(x) | f \in A\} \ .$$

(c) Verify also that all the maps $\phi_n$ and $\psi_n$ are continuous.

(d) Show that $D_\infty$ and $\prod_{i=0}^{\infty} D_i$ are complete lattices, and the inclusion of the former into the latter is continuous.

(e) Show that all the maps $\theta_{ab} : D_a \to D_b$ are continuous, including the cases when some subscripts are $\infty$, where the definitions are as follows (using $\theta$ so as not to show any favouritism between $\phi$ and $\psi$) :

When $a = b$, use the identity map.

When $a < b < \infty$, use the obvious composite of $\phi_i$'s.

When $b < a < \infty$, use the obvious composite of $\psi_i$'s.

Let $\theta_{\infty,b}(d_0, d_1, d_2, \cdots) \ := \ d_b$ .

Let $\theta_{a,\infty}(x) \ := \ (\theta_{a,0}(x), \theta_{a,1}(x), \cdots, \theta_{a,a-1}(x), x, \theta_{a,a+1}(x), \cdots)$ .

Note that this last element *is* in $D_\infty$ . See also **VII-8.6** .

**Ex. VII-8.2.** Show that $D_\infty$ satisfies the 'arrow-theoretic' definition of being 'the' inverse limit (in the category of complete lattices and continuous maps) of the system

$$D_0 \xleftarrow{\psi_0} D_1 \xleftarrow{\psi_1} D_2 \xleftarrow{\psi_2} \cdots$$

with respect to the maps $\theta_{\infty n} : D_\infty \to D_n$ . That is, given a complete lattice $E$ and continuous maps $\eta_n : E \to D_n$ such that $\eta_n = \psi_n \circ \eta_{n+1}$ for all $n$, there is a unique continuous map $\eta_\infty : E \to D_\infty$ such that $\eta_n = \theta_{\infty n} \circ \eta_\infty$ for all $n$.

**Ex. VII-8.3.**(General category theory)

(a) Show quite generally from the arrow-theoretic definition that, given an infinite commutative ladder

$$\begin{array}{ccccccc} A_0 & \xleftarrow{\alpha_0} & A_1 & \xleftarrow{\alpha_1} & A_2 & \xleftarrow{\alpha_2} & \cdots \\ \zeta_0 \downarrow & & \zeta_1 \downarrow & & \zeta_2 \downarrow & & \\ B_0 & \xleftarrow{\beta_0} & B_1 & \xleftarrow{\beta_1} & B_2 & \xleftarrow{\beta_2} & \cdots \end{array}$$

(i.e. each square commutes) in which both lines have an inverse limit, there is a unique map

$$\zeta_\infty \ : \ \varprojlim(A_i, \alpha_i) \ \longrightarrow \ \varprojlim(B_i, \beta_i)$$

for which $\beta_{\infty n} \circ \zeta_\infty \ = \ \zeta_n \circ \alpha_{\infty n}$ for all $n$ .

Here,

$$\varprojlim(A_i, \alpha_i) \ \text{ together with its maps } \ \alpha_{\infty n} \ : \ \varprojlim(A_i, \alpha_i) \ \to \ A_n$$

is any choice of an inverse limit (in the arrow-theoretic sense) for the top line in the display, and similarly for the bottom line.

(b) Furthermore, as long as $\zeta_n$ is an isomorphism for all sufficiently large $n$, then the map $\zeta_\infty$ is also an isomorphism (that is, a map with a 2-sided inverse with respect to composition). And so, *inverse limits are unique up to a unique isomorphism.*

**Crucial Remark.** We have such a commutative ladder

$$\begin{array}{ccccccccc} D_0 & \xleftarrow{\psi_0} & D_1 & \xleftarrow{\psi_1} & D_2 & \xleftarrow{\psi_2} & D_3 & \xleftarrow{\psi_3} & \cdots \\ \text{who cares?} \downarrow & & = \downarrow & & = \downarrow & & = \downarrow & & \\ \text{whatever} & \longleftarrow & [D_0 \rightharpoonup\!\!\!\!\!\sim\ D_0] & \xleftarrow{\psi_1} & [D_1 \rightharpoonup\!\!\!\!\!\sim\ D_1] & \xleftarrow{\psi_2} & [D_2 \rightharpoonup\!\!\!\!\!\sim\ D_2] & \xleftarrow{\psi_3} & \cdots \end{array}$$

Now, $D_\infty$, by **VII-8.2**, is the inverse limit of the top line, so, by **VII-8.3**(b), to show that $D_\infty \cong [D_\infty \rightharpoonup\!\!\!\!\!\sim\ D_\infty]$ , it remains only to prove the following :

**Ex. VII-8.4.** Show directly from the arrow-theoretic definition that

$[D_\infty \!\!\rightarrowtail\!\!\!\!\!\rightsquigarrow D_\infty]$ together with its maps $\eta_{\infty n} : [D_\infty \!\!\rightarrowtail\!\!\!\!\!\rightsquigarrow D_\infty] \to [D_n \!\!\rightarrowtail\!\!\!\!\!\rightsquigarrow D_n]$ ,

defined by $\eta_{\infty n}(f) = \theta_{\infty n} \circ f \circ \theta_{n\infty}$ , is the inverse limit of the lower line,

$\varprojlim([D_i \!\!\rightarrowtail\!\!\!\!\!\rightsquigarrow D_i], \psi_{i+1})$ , in the category of complete lattices and continuous maps.

**Remark.** The last three exercises complete the job, but the first and last may be a good challenge for most readers. They are likely to involve much of the material in the following more pedestrian approach to proving Scott's theorem (which is therefore not really a different proof), and which does exhibit the isomorphisms $D_\infty \overleftrightarrows [D_\infty \!\!\rightarrowtail\!\!\!\!\!\rightsquigarrow D_\infty]$ quite explicitly.

**Ex. VII-8.5.** (a) Show that $(\phi_0 \circ \psi_0)(f) \sqsubseteq f$ for all $f \in D_1$ .
(b) Deduce that, for all $n$, we have $(\phi_n \circ \psi_n)(f) \sqsubseteq f$ for all $f \in D_{n+1}$ .

The next exercise is best remembered as : "up, then down, (or, right, then left) always gives the identity map" whereas : "down, then up, (or left, then right) never gives a larger element" We're thinking of the objects as lined up in the usual way :

$$D_0 \qquad D_1 \qquad D_2 \qquad D_3 \qquad D_4 \quad \cdots\cdots \qquad D_\infty$$

**Ex. VII-8.6.** (a) Show that
(i) if $b \geq a$, then $\theta_{ba} \circ \theta_{ab} = \mathrm{id}_{D_a}$ ;
(ii) if $a \geq b$, then $(\theta_{ba} \circ \theta_{ab})(x) \sqsubseteq x$ for all $x$ .
This, and the next part, include the cases when some subscripts are $\infty$ .

(b) More generally, deduce that, if $b < a$ and $b < c$, then

$$(\theta_{bc} \circ \theta_{ab})(x) \sqsubseteq \theta_{ac}(x) \quad \text{for all } x \in D_a \ ,$$

whereas, for all other $a, b$ and $c$, the maps $\theta_{bc} \circ \theta_{ab}$ and $\theta_{ac}$ are actually equal.

Starting now we shall have many instances of $\bigsqcup_n d_n$. In every case, the sequence of elements $d_n$ form a chain with respect to $\sqsubseteq$, and so a directed set, and thus the least upper bound does exist by completeness of $D$. Checking the 'chaininess' will be left to the reader.

**Definitions.** Define $\phi_\infty \ : \ D_\infty \ \to \ [D_\infty \!\rightharpoonup\! D_\infty]$ by

$$\phi_\infty(x) \ := \ \bigsqcup_k (\theta_{k,\infty} \circ (\theta_{\infty,k+1}(x)) \circ \theta_{\infty,k}) \ .$$

Define $\psi_\infty \ : \ [D_\infty \!\rightharpoonup\! D_\infty] \ \to \ D_\infty$ by

$$\psi_\infty(f) \ := \ \bigsqcup_n \theta_{n+1,\infty}(\theta_{\infty,n} \circ f \circ \theta_{n,\infty}) \ .$$

**Ex. VII-8.7.** Show that all the following are continuous :
(i) $\phi_\infty(x) : D_\infty \ \to \ D_\infty$ for all $x \in D_\infty$ (so $\phi_\infty$ is well-defined) ;
(ii) $\phi_\infty$ ; and
(iii) $\psi_\infty$ .

**Ex. VII-8.8.** Prove that, for all $n, k < \infty$, and all $f \in D_{k+1}$, we have

$$\bigsqcup_n \theta_{n+1,\infty}(\theta_{k,n} \circ f \circ \theta_{n,k}) \ = \ \theta_{k+1,\infty}(f) \ ,$$

perhaps first checking that

$$\theta_{n+1,\infty}(\theta_{k,n} \circ f \circ \theta_{n,k}) \begin{cases} \sqsubseteq \ \theta_{k+1,\infty}(f) & \text{for } n \le k \ ; \\ = \ \theta_{k+1,\infty}(f) & \text{for } n \ge k \ . \end{cases}$$

(Oddly enough, this is not meaningful for $n$ or $k$ 'equal' to $\infty$ .)

**Ex. VII-8.9.** Show that

$$\theta_{\infty,k+1}(\bigsqcup_n \theta_{n+1,\infty}(z_{n+1})) \ = \ z_{k+1} \ ,$$

if $\{z_r \in D_r\}_{r>0}$ satisfies $\psi_r(z_{r+1}) = z_r$ .

**Ex. VII-8.10.** Show that, for all $x \in D_\infty$ ,

$$\bigsqcup_k (\theta_{k+1,\infty} \circ \theta_{\infty,k+1})(x) \ = \ x \ .$$

**Ex. VII-8.11.** Show that, for all $f \in [D_\infty \!\rightharpoonup\! D_\infty]$ ,

$$\bigsqcup_k (\theta_{k,\infty} \circ \theta_{\infty,k} \circ f \circ \theta_{k,\infty} \circ \theta_{\infty,k}) \ = \ f \ .$$

Using these last four exercises, we can now complete the more mundane of the two proofs of Scott's theorem, by directly calculating that $\phi_\infty$ and $\psi_\infty$ are mutually inverse :

For all $x \in D_\infty$ ,

$$\psi_\infty(\phi_\infty(x)) \;=\; \psi_\infty(\bigsqcup_k(\theta_{k,\infty} \circ (\theta_{\infty,k+1}(x)) \circ \theta_{\infty,k})) \qquad \text{(definition of } \phi_\infty)$$

$$= \bigsqcup_k \psi_\infty(\theta_{k,\infty} \circ (\theta_{\infty,k+1}(x)) \circ \theta_{\infty,k}) \qquad \text{(since } \psi_\infty \text{ is continuous)}$$

$$= \bigsqcup_k \bigsqcup_n \theta_{n+1,\infty}(\theta_{\infty,n} \circ \theta_{k,\infty} \circ (\theta_{\infty,k+1}(x)) \circ \theta_{\infty,k} \circ \theta_{n,\infty}) \qquad \text{(definition of } \psi_\infty)$$

$$= \bigsqcup_k \bigsqcup_n \theta_{n+1,\infty}(\theta_{k,n} \circ (\theta_{\infty,k+1}(x)) \circ \theta_{n,k}) \qquad \text{(by } \mathbf{VII-8.6}\text{(b))}$$

$$= \bigsqcup_k \theta_{k+1,\infty}(\theta_{\infty,k+1}(x)) \;=\; x\,, \quad \text{as required, by } \mathbf{VII-8.8} \text{ and } \mathbf{VII-8.10}.$$

For all $f \in [D_\infty \!\rightharpoonup\! D_\infty]$ ,

$$\phi_\infty(\psi_\infty(f)) \;=\; \bigsqcup_k(\theta_{k,\infty} \circ (\theta_{\infty,k+1}(\psi_\infty(f))) \circ \theta_{\infty,k}) \qquad \text{(definition of } \phi_\infty)$$

$$= \bigsqcup_k(\theta_{k,\infty} \circ (\theta_{\infty,k+1}(\bigsqcup_n \theta_{n+1,\infty}(\theta_{\infty,n} \circ f \circ \theta_{n,\infty}))) \circ \theta_{\infty,k}) \qquad \text{(definition of } \psi_\infty)$$

$$= \bigsqcup_k(\theta_{k,\infty} \circ (\theta_{\infty,k} \circ f \circ \theta_{k,\infty}) \circ \theta_{\infty,k}) \;=\; f \text{ as required, by } \mathbf{VII-8.11}.$$

The penultimate "=" uses **VII-8.9** with $z_{r+1} = \theta_{\infty,r} \circ f \circ \theta_{r,\infty}$ ; and that result is applicable because, rather trivially,

$$\psi_r(\theta_{\infty,r} \circ f \circ \theta_{r,\infty}) \;=\; \theta_{\infty,r-1} \circ f \circ \theta_{r-1,\infty}\ ,$$

employing the definition of $\psi_r$ and **VII-8.6** .

The following fundamental result of Tarski about fixed points in complete lattices will be useful in the next subsection. Later we relate this to the fixedpoint combinator $\underline{Y}$ of Curry, called the *paradoxical combinator* by him.

**Theorem VII-8.12.** *Let $D$ be a complete lattice, and consider operators $\Omega \in [D \multimap D]$. Define*

$$\mathsf{fix} \ : \ [D \multimap D] \ \to D \quad \textit{by} \quad \mathsf{fix}(\Omega) := \bigsqcup_n \Omega^n(\bot_D) \ .$$

*Then $\mathsf{fix}(\Omega)$ is the minimal fixed point of $\Omega$, and $\mathsf{fix}$ is itself continuous. In particular, any continuous operator on a complete lattice has at least one fixed point.*

**Proof.** That $\mathsf{fix}$ is well-defined should be checked. As with all our other uses of $\bigsqcup_n$, the set of elements form a chain with respect to $\sqsubseteq$, and so a directed set, and the least upper bound does exist by completeness of $D$. To prove 'chaininess', we have

$$\bot_D \sqsubseteq \Omega(\bot_D) \ \text{ implies immediately that } \ \Omega^n(\bot_D) \sqsubseteq \Omega^{n+1}(\bot_D) \ .$$

As for the fixed point aspect, using continuity of $\Omega$ ,

$$\Omega(\mathsf{fix}(\Omega)) \ = \ \Omega(\bigsqcup_n \Omega^n(\bot_D)) \ = \ \bigsqcup_n \Omega^{n+1}(\bot_D) \ = \ \mathsf{fix}(\Omega) \ ,$$

as required, again using continuity of $\Omega$ .

As to the minimality, if $\Omega(d) = d$, then $\bot \sqsubseteq d$ gives $\Omega^n(\bot) \sqsubseteq \Omega^n(d) = d$ for all $n$, so

$$\mathsf{fix}(\Omega) \ = \ \bigsqcup_n \Omega^n(\bot_D) \ \sqsubseteq \ \bigsqcup_n \Omega^n(d) \ = \ \bigsqcup_n d \ = \ d \ ,$$

as required.

Finally, given a directed set $\mathbf{O} \subset [D \multimap D]$, we have

$$\mathsf{fix}(\bigsqcup_{\Omega \in \mathbf{O}} \Omega) \ = \ \bigsqcup_n (\bigsqcup_{\Omega \in \mathbf{O}} \Omega)^n(\bot) \ = \ \bigsqcup_n (\bigsqcup_{\Omega \in \mathbf{O}} \Omega^n)(\bot)$$

$$= \ \bigsqcup_n \bigsqcup_{\Omega \in \mathbf{O}} (\Omega^n(\bot)) \ = \ \bigsqcup_{\Omega \in \mathbf{O}} \bigsqcup_n (\Omega^n(\bot)) \ = \ \bigsqcup_{\Omega \in \mathbf{O}} \mathsf{fix}(\Omega) \ ,$$

as required, where justifications of the three middle equalities are left to the reader. And so, $\mathsf{fix}$ is continuous.

## VII-9 Two (not entirely typical) Examples of Denotational Semantics.

We shall write out in all detail what presumably ought to be the denotational semantics of **ATEN** (earlier used to define computability), then illustrate it with a few examples. In the second half, we re-write the maps $\rho_+$ in the style of "denotational semantics for the $\lambda$-calculus", finishing with several interesting theorems about $\rho_+$ when the domain is $D_\infty$, which also gives practice calculating in that domain.

### Denotational semantics of ATEN.

Since *machine readability* and *executability* seem to be central concerns here, the formulas will all be given very technically, and we'll even begin with a super-formal statement of the syntax (though not exactly in the standard BNF-style).

Here it all is, in one largely human-unreadable page. See the following remarks.

$BRA := \{\,)\,,\,(\,\}$ ,which merely says that we'll have lots of brackets, despite some CSers' abhorrence.

$v \in IDE := \{\, x \mid * \,||\, x \,||\, v* \,\}$ which says, e.g. that the 'real' $x_3$ is $x***$, and $x_0$ is just $x$.

$$s,t \in EXP := \{\, BRA \mid IDE \mid + \mid \times \mid 0 \mid 1 \,||\, v \mid 0 \mid 1 \,||\, (s+t) \mid (s \times t) \,\}\,.$$

$$F,G \in EXP' := \{\, BRA \mid EXP \mid < \mid \approx \mid \neg \mid \wedge \,||\, s < t \mid s \approx t \,||\, \neg F \mid (F \wedge G) \,\}\,.$$

$$C,D \in COM := \{\, BRA \mid IDE \mid EXP \mid EXP' \mid \twoheadleftarrow \mid ; \mid \mathbf{whdo} \,||\, v \twoheadleftarrow t \,||\, (C;D) \mid \mathbf{whdo}(F)(C) \,\}\,.$$

$$VAL \text{ “}=\text{”} \{\perp_{\mathbf{N}}\} \cup \mathbf{N} \quad ; \quad BOOL \text{ “}=\text{”} \{\, \perp_{BOOL}\,,\, \mathsf{Tr}\,,\, \mathsf{Fs}\,\} \quad ; \quad STATE = [IDE \rightharpoonup VAL]\,.$$

$$\mathcal{E} : EXP \to [STATE \rightharpoonup VAL] \qquad \text{defined by}$$

$$\mathcal{E}[\![v]\!](\sigma) := \sigma(v) \quad ; \quad \mathcal{E}[\![0]\!](\sigma) := 0_{\mathbf{N}} \quad ; \quad \mathcal{E}[\![1]\!](\sigma) := 1_{\mathbf{N}} \quad ;$$

$$\mathcal{E}[\![(s+t)]\!](\sigma) := \mathcal{E}[\![s]\!](\sigma) +_{\mathbf{N}} \mathcal{E}[\![t]\!](\sigma) \quad ; \quad \mathcal{E}[\![(s \times t)]\!](\sigma) := \mathcal{E}[\![s]\!](\sigma) \cdot_{\mathbf{N}} \mathcal{E}[\![t]\!](\sigma) \quad .$$

(Note that $\perp_{\mathbf{N}}$ is produced when it is either of the inputs for $+_{\mathbf{N}}$ or for $\cdot_{\mathbf{N}}$ .)

$$\mathcal{E}' \ : \ EXP' \ \rightarrow \ [STATE \rightsquigarrow BOOL] \qquad \text{defined by}$$

saying firstly that $\mathcal{E}'[\![F]\!](\sigma) :=\perp_{BOOL}$ if either $F$ is an atomic formula involving a term $s$ with $\mathcal{E}[\![s]\!](\sigma) :=\perp_{\mathbf{N}}$ , or if $F$ is built using $\neg$ or $\wedge$ using a formula $G$ for which $\mathcal{E}'[\![G]\!](\sigma) :=\perp_{BOOL}$ ; otherwise

$$\mathcal{E}'[\![s < t]\!](\sigma) := \text{if } \mathcal{E}[\![s]\!](\sigma) <_{\mathbf{N}} \mathcal{E}[\![t]\!](\sigma), \text{ then } \mathsf{Tr}, \text{ else } \mathsf{Fs} \ ;$$

$$\mathcal{E}'[\![s \approx t]\!](\sigma) := \text{if } \mathcal{E}[\![s]\!](\sigma) = \mathcal{E}[\![t]\!](\sigma), \text{ then } \mathsf{Tr}, \text{ else } \mathsf{Fs} \ ;$$

$$\mathcal{E}'[\![\neg F]\!](\sigma) := \text{if } \mathcal{E}'[\![F]\!](\sigma) = \mathsf{Fs}, \text{ then } \mathsf{Tr}, \text{ else } \mathsf{Fs} \ ;$$

$$\mathcal{E}'[\![(F \wedge G)]\!](\sigma) := \text{if } \mathcal{E}'[\![F]\!](\sigma) = \mathsf{Tr} \text{ and } \mathcal{E}'[\![G]\!](\sigma) = \mathsf{Tr}, \text{ then } \mathsf{Tr}, \text{ else } \mathsf{Fs} \ .$$

$$\mathcal{C} \ : \ COM \ \rightarrow \ [STATE \rightsquigarrow STATE] \qquad \text{defined by}$$

$$\mathcal{C}[\![v \Leftarrow t \ ]\!](\sigma) := \sigma^{[v \mapsto \mathcal{E}[\![t]\!](\sigma)]} \quad \text{where } \sigma^{[v \mapsto \alpha]} \text{ agrees with } \sigma, \text{ except } v \mapsto \alpha \ ;$$

$$\mathcal{C}[\![(C; D)]\!] := \mathcal{C}[\![D]\!] \circ \mathcal{C}[\![C]\!] \ ;$$

$$\mathcal{C}[\![\mathbf{whdo}(F)(C)]\!] := \mathsf{fix}(f \mapsto (\sigma \mapsto (\text{if } \mathcal{E}'[\![F]\!](\sigma) = \mathsf{Tr}, \text{ then } (f \circ \mathcal{C}[\![C]\!])(\sigma), \text{ else } \sigma \ ))) \ ,$$

where the fixpoint operator, $\mathsf{fix} \ : \ [D \rightsquigarrow D] \ \rightarrow D$, has $D \ = \ [STATE \rightsquigarrow STATE]$ .

**Remarks.** The first five displays are the syntax. All but the first give a set of strings in the usual three stages: { first, all symbols to be used || next, which are the atomic strings || and finally, how the 'production' of new strings is done (induction on structure)}. To the far left are 'typical' member(s) of the string set to be defined, those being then used in the production and also on lines further down. [Where whole sets of strings are listed on the left of the symbols-to-be-used listing and other times as well, a human interpreter thinks of those substrings as single symbols, when they appear inside the strings being specified on that line. For example, $(x***+x**)$ has ten symbols, but a human ignores the brackets and thinks of three, i.e. $x_3 + x_2$—and maybe even as a single symbol, since, stretching it a bit, however complicated, a term occurring inside a formula is intuited as a single symbol in a sense, as a component of the formula. And similarly, a term or a formula within a command is psychologically a single symbol. Sometimes the phrase "immediate constituents" is used for this, in explaining below the inductive nature of the semantic function definitions.]

The last of the five lines is the syntax of **ATEN** itself. The third and fourth lines give the syntax of the assertion language, first terms, then (quantifier-free) formulas, in $1^{\underline{\text{st}}}$order number theory. We've used names

which should remind CSers of the word "expression", but they might better be called $TRM(= EXP)$ and $FRM(= EXP')$ from much earlier stuff here.

Often, CSers would lump $EXP$ and $EXP'$ together into a single syntactic category. Why they do so is not unmysterious to me; I am probably missing some considerable subtlety. But I am well aware of the confusion that tendency in earlier CS courses causes for average students when I teach them logic. They initially find it strange that I should make a distinction between strings in a formal language which we intuit as standing for objects, and other strings which we intuit as standing for statements! On the other hand, I myself find it strange to think of $(3 < 4) + ((5 = 2) + 6)$ as any kind of expression! (and not because $5 \neq 2$) Another inscrutability for me is that many of the technicalities on the previous page would normally be written down in exactly the opposite order in a text on programming languages! While I'm in the mood to confess a mentality completely out of synch with the CSers, here's another bafflement which some charitable CSer will hopefully straighten me out on. Within programs in, say, PASCAL, one sees the *begin—end* and use of indentation as a replacement for brackets, which makes things much more readable (if *any* software could ever be described as (humanly) readable). But there seems to be a great desire to avoid brackets, and so avoid non-ambiguity, in writing out the syntax of languages. Instead, vague reference is made to parsers. I can appreciate a desire sometimes to leave open different possibilities. A good example is the $\lambda$-calculus, where using $A(B)$ rather than $(AB)$ in the basic syntactic setup is a possibility, and probably better for psychological purposes, but not for economy. And the semantics is largely independent of the parsing. But when precision is of utmost importance, I cannot understand this tendency to leave things vague. Of course, there still remains a need to specify algorithms for deciding whether a string (employing the basic symbols of the language) is actually in the language. But surely that's a separate issue.

The sixth display gives information about the semantic domains needed. We haven't said what $VAL$ actually "is" (hence the "="), other than that it contains as elements $\perp$ and all the natural numbers. Below we come clean on that. The use of $[\ \ \multimap\ \ ]$ is some indication that these semantic sets have some structure, actually a partial order.

Finally, the three semantic functions, corresponding to terms, formulae, and commands, are given. We have been super-strict in distinguishing notationally between the symbol "+" and the actual operation "$+_{\mathbf{N}}$" on natural numbers, and similarly for "$\cdot_{\mathbf{N}}$" and "$<_{\mathbf{N}}$". Some elementary confusions revolve around this point, in the presence of ambiguous notation, which point otherwise might seem overly fussy. We have, in the same vein, used our usual "$\approx$" as the formal equality symbol, to distinguish it from the actual relation of sameness. Each of the three semantic functions is defined by structural induction on the productions defined at the right-hand ends of the corresponding syntactic sets. As mentioned in defining $\mathcal{E}$, the operations $+_{\mathbf{N}}$ and

$\cdot_{\mathbf{N}}$ produce $\perp_{\mathbf{N}}$ if either or both of their inputs is $\perp_{\mathbf{N}}$ . And from the definition of $\mathcal{E}'$, the relation $<_{\mathbf{N}}$ has no need to concern itself with comparing $\perp_{\mathbf{N}}$ with anything.

As for $\mathcal{E}$, we're just saying how, using $\mathcal{E}$'s adjoint, a term together with a state will yield a number, exactly as in Tarski's definition of truth. In fact,

$\mathcal{E}[\![t]\!](\sigma)$ is just another name for $t^{\underline{v}}$, where $\underline{v}$ is identified with that state $\sigma$ mapping each $x_n$ to $v_n$.

See [**LM**], beginning of Ch.6.

As for $\mathcal{E}'$, we're just saying how, after taking its adjoint, a formula together with a state will yield a truth value, exactly as in Tarski's definition of truth. In fact,

$Tr$ and $Fs$ for $\mathcal{E}'[\![F]\!](\sigma)$ are just other ways of saying whether $\underline{v} \in F^V$ or not.

See [**LM**] for $F^V$. Alternatively, $\underline{v} \in F^V$ says "$F$ is true at $\underline{v}$ from $V$".

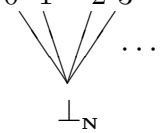
As for $\mathcal{C}$, we're just saying how, after taking its adjoint, a command together with a state will yield another state, exactly as in our original definition of the semantics of **BTEN**. In fact,

$\mathcal{C}[\![C]\!](\sigma)$ is just another name for $||C||(\underline{v})$; i.e. $\mathcal{C}[\![C]\!]$ is another name for $||C||$ .

The right-hand sides in the definitions of $\mathcal{E}'$ and $\mathcal{C}$ use an (if-then-else) in the spirit of **McSelf** and should be taken that way. (See also the discussion just below the diagram in the next paragraph.) In [**Al**] Ch.5, a program in PASCAL is written out soon after specifying the denotational semantics of his small language. Presumably LISP would be just as good. Such a program is called an *implementer*. Another one of my confusions is the question of why implementing a simple language inside a very complicated practical language is *useful*. (I can see where it would be *fun*.) Perhaps it is related to the fact that the implementer is just a *single* program in that complex language, a single program in which one can develop considerable confidence. I am certainly not (yet?) the one to write an implementation of **ATEN** in any real programming language.

Note that for the final clause, giving the denotation of the **whdo**-command, we need the discussion of fixed points from the end of the previous subsection. This specification of the **whdo**-command comes from the remarks at the end of Subsection VII-7, and is evidently far more 'implementable' than that in the originally specified semantics of **BTEN**. If not for that one command, this whole exercise would presumably be regarded as rather sterile. Indeed, as mentioned earlier, it's when applied to much more complicated (e.g.

ALGOL-like) languages that this impression of sterility dissipates. The need for anything like a self-reflective domain as constructed in the previous subsection is unclear. But we do at least need that $D = [STATE \multimap STATE]$ is a lattice for which Tarski's theorem on minimum fixpoint operators works. That follows as long as $STATE$ is a complete lattice, which itself follows as long as $VAL$ is. Therefore we complete the unfinished business on the previous page by specifying

$VAL :=$ the flat lattice  and $BOOL$ similarly.



There are a couple of points worth adding, to shore up the semantic definitions. Suppose we were dealing with **BTEN** rather than **ATEN**, so we had to give the semantics of the command $\mathbf{ite}(F)(C)(D)$, that is, the (*if-then-else*)-command. In the style previous this would be

$$\mathcal{C}[\![\mathbf{ite}(F)(C)(D)]\!](\sigma) \;:=\; \text{if } \mathcal{E}'[\![F]\!](\sigma) = \mathsf{Tr},\ \text{then } \mathcal{C}[\![C]\!](\sigma),\ \text{else } \mathcal{C}[\![D]\!](\sigma)\ .$$

To my complete bafflement, [**Al**], top of p.12 would object strenuously to this, insisting that it must be

$$\mathcal{C}[\![\mathbf{ite}(F)(C)(D)]\!](\sigma) \;:=\; (\text{if } \mathcal{E}'[\![F]\!](\sigma) = \mathsf{Tr},\ \text{then } \mathcal{C}[\![C]\!],\ \text{else } \mathcal{C}[\![D]\!])(\sigma)\ .$$

Once again, I am in need of a kind CSer to straighten me out on a subtle point. Furthermore, the definition in [**Go**], p.51 (C3) , seems not to be consistent in terms of the domains involved with the earlier definitions there, though that seems to be fixable by extending the $\mathsf{cond}$-function there in a certain way. But perhaps again I need a tutorial. In any case here is a re-written version of our definition more in that style, which is similar to [**St**], p.196. (He has the added complication of "side-effects" to deal with, but in their absence, his definition reduces to exactly the following, at least up to currying/uncurrying.) Further down we do the same for the **whdo**-command, to which similar bafflements on my part apply with respect to the versions in [**Al**] and [**Go**]. Re-define (without really changing the definition)

$$\mathcal{C}[\![\mathbf{ite}(F)(C)(D)]\!] \;:=\; \mathit{con} \circ (\mathcal{E}'[\![F]\!] \times \mathcal{C}[\![C]\!] \times \mathcal{C}[\![D]\!]) \circ \mathit{ddg}\ ,$$

where

$$ddg : STATE \to STATE \times STATE \times STATE \quad ; \quad \sigma \mapsto (\sigma, \sigma, \sigma)$$

is the double diagonal map, and

$$con : BOOL \times STATE \times STATE \to STATE$$

$$(Tr, \sigma, \tau) \mapsto \sigma \quad ; \quad (Fs, \sigma, \tau) \mapsto \tau \quad ; \quad (\perp_{BOOL}, \sigma, \tau) \mapsto \quad \perp_{STATE} \quad ;$$

is the conditional. (This has turned out a bit simpler than in the above references, with no need to fool around with defining a $\star$-operator, partly because we are not insisting on currying everything, with its attendant contortions.)

In any case, this isolates the facts that we definitely need *ddg* and *con* to be continuous, which they are, but otherwise no further comment is needed.

Now we can re-write the *while-do* semantic definition also in this style :

$$\mathcal{C}[\![\mathbf{whdo}(F)(C)]\!] \;:=\; fix\ [\ f \ \mapsto\ con \circ (\mathcal{E}'[\![F]\!] \times (f \circ \mathcal{C}[\![C]\!]) \times Id) \circ ddg\ ]\ .$$

**Some examples of the definition.**

The first three examples below are utterly simple programs in **ATEN**, to the point of being silly. And the purpose of denotational semantics, though not crystal clear to me, is certainly *not* to be able to write out particular examples. But the ones below should serve to better familiarize us with how the definition works, and, more importantly, to give some confidence that the answers are coming out 'right'.

(1) If $C = \mathbf{whdo}(x_0 \approx x_0)(D)$ for some command $D$, we obviously realize an infinite loop no matter what is in the bins to start, i.e. for any initial state. Let's figure out $\mathcal{C}[\![C]\!]$. Using the definition, we get

$$\mathsf{fix}(f \mapsto (\sigma \mapsto (f \circ \mathcal{C}[\![D]\!])(\sigma))) \;=\; \mathsf{fix}(f \mapsto f \circ \mathcal{C}[\![D]\!])\ ,$$

since certainly $\mathcal{E}'[\![x_0 \approx x_0]\!](\sigma) = Tr$ for all $\sigma$ . Now define a function $f_0$ in $[STATE \multimap STATE]$ by $f_0(\sigma) = \perp$ for all $\sigma$ . It is very clear that $f_0 \sqsubseteq f$ for all $f \in [STATE \multimap STATE]$. Also $f_0 \circ g = f_0$ for any function $g$ in $[STATE \multimap STATE]$ , since $f_0$ is a constant function. So $f_0$ is the required answer, that is, the minimal fixed point of the operator which is 'composition on the left with $\mathcal{C}[\![D]\!]$'. And surely $f_0$ *should be* the element

in the domain $[STATE \rightsquigarrow STATE]$ which denotes a program that always loops ! A better name for that function is $\perp_{[STATE \rightsquigarrow STATE]}$. The previous $\perp$ is $\perp_{STATE} = \perp_{[IDE \rightsquigarrow VAL]}$, namely the function which maps all $v \in IDE$ to $\perp_{VAL}$ , which Dana Scott refers to as "the undefined".

(2) If $C = \mathbf{whdo}(x_0 < 1)(x_0 \doteqdot x_0 + 1)$ , we see that any initial state is unchanged by the command, except when bin 0 contains zero. In the latter case, the zero in bin 0 is changed to 1, and then the process terminates. First here are three preliminary calculations:

$$\mathcal{E}[\![x_0 + 1]\!](\sigma) = \mathcal{E}[\![x_0]\!](\sigma) +_{\mathbf{N}} \mathcal{E}[\![1]\!](\sigma) = \sigma(x_0) +_{\mathbf{N}} 1_{\mathbf{N}} \ .$$

$$\mathcal{C}[\![x_0 \doteqdot x_0 + 1]\!](\sigma) = \sigma^{[x_0 \ \mapsto \ \mathcal{E}[\![x_0+1]\!](\sigma)]} = \sigma^{[x_0 \ \mapsto \ 1_{\mathbf{N}} +_{\mathbf{N}} \sigma(x_0)]} \ .$$

$$\mathcal{E}'[\![x_0 < 1]\!] \ = \ \begin{cases} Tr & \text{if } \sigma(x_0) <_{\mathbf{N}} 1_{\mathbf{N}} \ ; \\ Fs & \text{otherwise .} \end{cases}$$

Let's figure out $\mathcal{C}[\![C]\!]$. Using the definition, we get

$$\mathsf{fix}(f \mapsto (\sigma \mapsto (\text{if } \mathcal{E}'[\![x_0 < 1]\!](\sigma) = \mathsf{Tr}, \text{then } (f \circ \mathcal{C}[\![x_0 \doteqdot x_0 + 1]\!])(\sigma), \text{else } \sigma \ )))$$

$$= \ \mathsf{fix}(f \mapsto (\sigma \mapsto \begin{cases} f(\sigma^{[x_0 \ \mapsto \ 1_{\mathbf{N}}]}) & \text{if } \sigma(x_0) = 0_{\mathbf{N}} \ ; \\ \sigma & \text{if } \sigma(x_0) \neq 0_{\mathbf{N}} \ . \end{cases}$$

Now let $f_1$ be *any* fixed point of the latter operator. So

$$f_1(\sigma) = \begin{cases} f_1(\sigma^{[x_0 \ \mapsto \ 1_{\mathbf{N}}]}) & \text{if } \sigma(x_0) = 0_{\mathbf{N}} \ ; \\ \sigma & \text{if } \sigma(x_0) \neq 0_{\mathbf{N}} \ . \end{cases}$$

But the lower line then determines the upper line, and we conclude that there is only one fixed point in this case (no minimization needed!), given by

$$f_1(\sigma) = \begin{cases} \sigma^{[x_0 \ \mapsto \ 1_{\mathbf{N}}]} & \text{if } \sigma(x_0) = 0_{\mathbf{N}} \ ; \\ \sigma & \text{if } \sigma(x_0) \neq 0_{\mathbf{N}} \ . \end{cases}$$

Well, this last function is exactly the one the command was supposed to compute, i.e. 'if necessary, change the 0 in bin zero to a 1, and do nothing else', so our definition is doing the expected here as well.

(3) If $C = \mathbf{whdo}(x_0 < 1)(x_1 \doteqdot x_1)$ , we see that any initial state is unchanged by the command, except when bin zero contains 0. In the latter case, the program does an infinite loop.

First here are the three preliminary 'calculations':

$$\mathcal{E}[\![x_1]\!](\sigma) \;=\; \sigma(x_1)\ ;$$

$$\mathcal{C}[\![x_1 \mathrel{=\!\!\div} x_1]\!](\sigma) \;=\; \sigma^{[x_1 \,\mapsto\, \mathcal{E}[\![x_1]\!](\sigma)]} \;=\; \sigma^{[x_1 \,\mapsto\, \sigma(x_1)]} = \sigma\ ,$$

hardly surprising; and, as before

$$\mathcal{E}'[\![x_0 < 1]\!] \;=\; \begin{cases} Tr & \text{if } \sigma(x_0) <_{\mathbf{N}} 1_{\mathbf{N}}\ ; \\ Fs & \text{otherwise}\ . \end{cases}$$

From the definition, $\mathcal{C}[\![C]\!]$ is given by

$$\mathsf{fix}(f \mapsto (\sigma \mapsto (\text{if } \mathcal{E}'[\![x_0 < 1]\!](\sigma) = \mathsf{Tr}, \text{then } (f \circ \mathcal{C}[\![x_1 \mathrel{=\!\!\div} x_1]\!])(\sigma), \text{else } \sigma\ )))$$

$$=\; \mathsf{fix}(f \mapsto (\sigma \mapsto \begin{cases} f(\sigma) & \text{if } \sigma(x_0) = 0_{\mathbf{N}}\ ; \\ \sigma & \text{if } \sigma(x_0) \neq 0_{\mathbf{N}}\ . \end{cases}$$

Suppose that $f_2$ is a fixed point of the latter operator. So

$$f_2(\sigma) \;=\; \begin{cases} f_2(\sigma) & \text{if } \sigma(x_0) = 0_{\mathbf{N}}\ ; \\ \sigma & \text{if } \sigma(x_0) \neq 0_{\mathbf{N}}\ . \end{cases}$$

The top line says nothing, and any such function *is* a fixed point. Thus, clearly $f_3$ is the minimal fixed point, where

$$f_3(\sigma) \;=\; \begin{cases} \perp & \text{if } \sigma(x_0) = 0_{\mathbf{N}}\ ; \\ \sigma & \text{if } \sigma(x_0) \neq 0_{\mathbf{N}}\ . \end{cases}$$

Once again, this last function is exactly the one the command was supposed to compute, i.e. 'loop if bin 0 has a 0, otherwise, terminate after doing nothing'. So our definition is doing the expected, giving at least a little reinforcement to our confidence in the technicalities.

(4) Now we give a more extended (and possibly interesting) example, using the obvious non-recursive algorithm that calculates the factorial function. In [**Al**], p.55, this is also an example, with very abbreviated explanation, so the reader may be interested to compare. A major contributor to the much more detail here (besides kindness to the reader) is the fact that **ATEN** is so primitive. The simple language in [**Al**] at least has subtraction and negative integers, which **ATEN** doesn't. But the details below are probably

instructive, even in the direction of writing an general algorithm to translate programs in that language into commands in **ATEN**.

We would like to use the 'command'

$$x_0 \dot{\leftarrow} 1 \ ; \ \mathbf{whdo}(0 < x_1)(x_0 \dot{\leftarrow} x_1 \times x_0 \ ; \ \text{“}x_1 \dot{\leftarrow} x_1 - 1\text{”}) \ .$$

After approximately "$v_1$" cycles, this should terminate, leaving the number "$v_1$!" in bin zero (where the natural number $v_i$ is the initial content of bin number $i$, i.e. it is $\sigma(x_i)$ below). And below we demonstrate that the semantic definitions prove this fact. The quotation marks are there because $x_1 - 1$ is not a term. We take "$x_1 \dot{\leftarrow} x_1 - 1$" to be an abbreviation for

$$x_2 \dot{\leftarrow} 0 \ ; \ \mathbf{whdo}(\neg x_2 + 1 \approx x_1)(x_2 \dot{\leftarrow} x_2 + 1) \ ; \ x_1 \dot{\leftarrow} x_2 \ ,$$

omitting associativity brackets for ";". So this produces that predecessor function which is undefined on 0.

So we've got a fairly lengthy job, much of which will be left as exercises for the reader.

First show that

$$\mathcal{C}[\![\mathbf{whdo}(\neg x_2{+}1 \approx x_1)(x_2 \dot{\leftarrow} x_2{+}1)]\!] = \mathsf{fix}(f \mapsto (\sigma \mapsto \begin{cases} f(\sigma^{[x_2 \mapsto \sigma(x_2)+1]}) & \text{if } \sigma(x_2)+1 \neq \sigma(x_1) \ ; \\ \sigma & \text{if } \sigma(x_2)+1 = \sigma(x_1) \ . \end{cases}$$

Then argue (by induction on $\sigma(x_1) - \sigma(x_2)$ in the first case) that the minimal fixed point here is given by

$$\sigma \mapsto \begin{cases} \sigma^{[x_2 \mapsto \sigma(x_1)-1]} & \text{if } \sigma(x_2) < \sigma(x_1) \ ; \\ \bot & \text{if } \sigma(x_2) \geq \sigma(x_1) \ . \end{cases}$$

Now argue that

$$\mathcal{C}[\![\text{“}x_1 \dot{\leftarrow} x_1 - 1\text{”}]\!] \ = \ \sigma \mapsto \begin{cases} \sigma^{[x_2 \mapsto \sigma(x_1)-1][x_1 \mapsto \sigma(x_1)-1]} & \text{if } \sigma(x_1) \neq 0 \ ; \\ \bot & \text{if } \sigma(x_1) = 0 \ . \end{cases}$$

Next argue that, if $E = (x_0 \dot{\leftarrow} x_1 \times x_0 \ ; \ \text{“}x_1 \dot{\leftarrow} x_1 - 1\text{”})$ , then

$$\mathcal{C}[\![E]\!] \ = \ \sigma \mapsto \begin{cases} \sigma^{[x_0 \mapsto \sigma(x_1)\sigma(x_2)][x_1 \mapsto \sigma(x_1)-1][x_2 \mapsto \sigma(x_1)-1]} & \text{if } \sigma(x_1) \neq 0 \ ; \\ \bot & \text{if } \sigma(x_1) = 0 \ . \end{cases}$$

The next step is yet another application of the definition of the semantic function on **whdo**-commands to yield

$$\mathcal{C}[\![\mathbf{whdo}(0 < x_1)(E)]\!] = \mathsf{fix}(f \mapsto (\sigma \mapsto \begin{cases} \sigma^{[x_0 \mapsto \sigma(x_1)\sigma(x_2)][x_1 \mapsto \sigma(x_1)-1][x_2 \mapsto \sigma(x_1)-1]} & \text{if } \sigma(x_1) \neq 0 \ ; \\ \sigma & \text{if } \sigma(x_1) = 0 \ . \end{cases}$$

A more subtle argument than the earlier analogues then yields the minimal fixed point as

$$\sigma \mapsto \begin{cases} \sigma^{[x_0 \mapsto \sigma(x_1)!\sigma(x_0)][x_1 \mapsto 0][x_2 \mapsto 0]} & \text{if } \sigma(x_1) \neq 0 \ ; \\ \sigma & \text{if } \sigma(x_1) = 0 \ . \end{cases}$$

Finally , we can easily see that

$$\mathcal{C}[\![x_0 \leftarrow 1 \ ; \ \mathbf{whdo}(0 < x_1)(E)]\!] \ = \ \sigma \mapsto \begin{cases} \sigma^{[x_0 \mapsto \sigma(x_1)!][x_1 \mapsto 0][x_2 \mapsto 0]} & \text{if } \sigma(x_1) \neq 0 \ ; \\ \sigma^{[x_0 \mapsto 1]} & \text{if } \sigma(x_1) = 0 \ . \end{cases}$$

This is the required result—our original command computes a function which puts "$\sigma(x_1)!$" into bin zero (to mix up the two ways of thinking about states), and which does irrelevant things to bins 1 and 2.

**Denotational semantics of $\Lambda$**

This consists of super-formal statements of the syntax from the very beginning, about 100 pages back, and of the specifications in the definition of $\lambda$-*model*. See the first half of this subsection for an analogue and an explanation of the syntax specification in the first three lines below. Here it is, again largely human-unreadable, but perhaps more mechanically translatable for writing an implementation. See Ch. 8 of **[St]** for much more on this.

$$BRA \;:=\; \{\ )\ ,\ (\ \}\quad .$$

$$v \in IDE \;:=\; \{\ x \mid\ *\ \|\|\ x\ \|\|\ v*\ \}\quad .$$

$$A, B \in \Lambda = EXP \;:=\; \{\ BRA \mid IDE \mid \lambda \mid\ \bullet\ \|\|\ v\ \|\|\ (AB) \mid (\lambda v \bullet A)\ \}\ .$$

$$VAL \;=\; \text{any } \lambda\text{−model}\ .$$
$$ENV \;=\; [IDE \to VAL]\ .$$

For $\rho \in ENV$, $v \in IDE$, and $d \in VAL$, define $\rho^{[v \mapsto d]} \in ENV$ to agree with $\rho$ except it maps the variable ('identifier') $v$ to $d$ .

$$\mathcal{E}\ :\ EXP\ \to\ [ENV \to VAL] \qquad \text{is defined by}$$
$$\mathcal{E}[\![v]\!](\rho) := \rho(v)\ \ ;$$
$$\mathcal{E}[\![(AB)]\!](\rho) := \mathcal{E}[\![A]\!](\rho) \cdot \mathcal{E}[\![B]\!](\rho)\quad ;$$
$$\mathcal{E}[\![(\lambda v \bullet A)]\!](\rho) := \psi(d\ \mapsto\ \mathcal{E}[\![A]\!](\rho^{[v \mapsto d]})\quad .$$

Note that the last three lines are just the three basic properties, in a messier notation, describing how what we called $\rho_+$ behaves in a $\lambda$-model. Note also that the last line is only meaningful in this structural inductive definition with an accompanying inductive proof that the map to which we are applying $\psi$ is in fact in $[VAL \rightharpoonup VAL]$ .

The other two fundamental proofs wanted here are of the facts that

$$(*) \qquad A \approx B \qquad \Longrightarrow \qquad \mathcal{E}[\![A]\!] = \mathcal{E}[\![B]\!]\ ;\ \text{i.e.}\ \ \rho_+(A) = \rho_+(B)\ \ \forall \rho\ \ ;$$

and that

$$(**) \quad \mathcal{E}[\![A^{[x \mapsto B]}]\!](\rho) = \mathcal{E}[\![A]\!](\rho^{[x \mapsto \mathcal{E}[\![B]\!](\rho)]})\ ;\ \text{i.e.}\ \ \rho_+(A^{[x \mapsto B]}) = \rho_+^{[x \mapsto \rho_+(B)]}(A)\ .$$

None of these proofs requires any great cleverness.

Later we discuss how, after passing to equivalence classes, and when Scott's $D_\infty$ is $VAL$, the map on $EXP/\approx$ coming from $\rho_+$ is not injective, but that it becomes so when restricted to the set of equivalence classes of closed terms which have a normal form; i.e. closed normal form terms all map under this denotational semantics to different objects in the $\lambda$-model, this being independent of $\rho$.

We'll now go back to the handier $\rho_+$ notation.

Proving the theorem of David Park below gives an opportunity for illustrations of this "denotational semantics of the $\lambda$-calculus" (and especially for calculations to increase one's familiarity with the innards of Scott's $D_\infty$).

Recall that Curry's fixpoint (or "paradoxical") combinator $\underline{Y}$ is a closed term, so it is unambiguous to define

$$Y \;:=\; \rho_+(\underline{Y}) \;\in\; D_\infty \quad ,$$

i.e. it is independent of the particular $\rho$ used.

**Theorem VII-9.1.** *Under the canonical isomorphism*

$$[[D_\infty \!\!\rightharpoonup\!\!\rightarrow D_\infty] \!\!\rightharpoonup\!\!\rightarrow D_\infty] \quad \overset{\cong}{\longleftrightarrow} \quad D_\infty \ ,$$

*the Tarski fixpoint operator,* $\mathsf{fix}$ *(from* **VII-8.12***), corresponds to* $Y$*, the image of Curry's fixpoint combinator. Three equivalent expressions of this are*

$$\phi_\infty(Y) = \mathsf{fix} \circ \phi_\infty \quad ; \quad Y = \psi_\infty(\mathsf{fix} \circ \phi_\infty) \quad ; \quad \mathsf{fix} = \phi_\infty(Y) \circ \psi_\infty \ .$$

We shall give the proof as a sequence of exercises, ending with establishing the first of those three. The fact that the three are equivalent is staggeringly trivial, via $\psi_\infty = \phi_\infty^{-1}$ . The only two proofs of the theorem that I have seen are the scanned image of Park's succinct typescript from 1976-78, and the proof in **[Wa]** (which we give, with proofs of basic identities he uses).

This proof depends on the explicit $\phi_0$ and $\psi_0$ (but not any explicit $D_0$) from Scott's building of $D_\infty$ from $D_0$, as exposited here in VII-8. Park sketches how a different choice of $(\phi_0, \psi_0)$ gives a different answer. So this theorem definitely does not generalize as is, to an arbitrary extensional $\lambda$-model

with compatible complete lattice structure. Later we use Park's theorem a couple of times to give other explicit results about the denotational semantics of $\Lambda$, including another fixpoint operator, though not Turing's fixpoint combinator (which the reader might like to think about).

First here are some useful exercises and notation.

**Definition.** Abbreviate $\theta_{n,\infty} \circ \theta_{\infty,n}$ to just $\pi_n : D_\infty \to D_\infty$ .

Then $\pi_n$ is a projection operator (i.e. $\pi_n \circ \pi_n = \pi_n$), whose image is a sublattice of $D_\infty$ which is isomorphic to $D_n$, and often identified with $D_n$ .

Much of the literature makes the notation even more succinct, denoting $\pi_n x$ as $x_n$ . This can be very efficient, but a bit confusing for us beginners. Firstly, it's rather more natural in mathematics to use $x_n$ as the $n$th component of $x$, that is, to write $x$ as $(x_1, x_2, x_3, \cdots)$ . So this latter $x_n$ would be in $D_n$, not in $D_\infty$ . To avoid conflict with the literature, we won't use $x_n$ at all. Another source of confusion with this subscripting is that an expression such as $y_{n+1}(x_n)$ sits in one's consciousness more naturally as an element of $D_{n+1} = [D_n \!\rightsquigarrow\! D_n]$ acting on an element of $D_n$ to produce an element of $D_n$ . But in fact it must (and does in the literature) mean what in the above defined notation is $\phi_\infty(\pi_{n+1} y)(\pi_n x)$ . (The two confusable meanings are of course closely related—see for example **Ex. 8** below.) This last expression is becoming unwieldy, and we'll at least be able to simplify it to $\pi_{n+1} y \cdot \pi_n x$ .

**Definition.** Let "$\cdot$" be the binary operation on $D_\infty$ adjoint to $\phi_\infty$ ; that is,

$$a \cdot b \;:=\; \phi_\infty(a)(b) \ .$$

This is the binary operation from earlier in the general theory of $\lambda$-models.

(Perhaps annoying the experts) for a bit more clarity we continue with juxtaposition for the operation in $\Lambda$, but use "$\cdot$" in $D_\infty$ . Thus, a basic property of the maps from denotational semantics is

$$\rho_+(\underline{A}\ \underline{B}) \;=\; \rho_+(\underline{A}) \cdot \rho_+(\underline{B}) \qquad \text{for all } \underline{A} \text{ and } \underline{B} \text{ in } \Lambda \ .$$

Readers who wish to go further with this and consult the literature will need to accustom themselves to two 'identifications' which we are avoiding for conceptual clarity (but giving ourselves messier notation) :

(i) identifying $\Lambda$ with its image $\rho_+(\Lambda) \subset D_\infty$, which *is* ambiguous on *non-closed* terms in $\Lambda$ , and also ambiguous in another sense 'up to $\approx$' ;

(ii) identifying $D_n$ with its image $\pi_n(D_\infty) \subset D_\infty$, that is, the image of the map $\theta_{n,\infty}$ .

**General Exercises on $D_\infty$.**

**1.** Show, for $n \leq k$ and all $a \in D_\infty$, that $\pi_n a \sqsubseteq \pi_k a \sqsubseteq a$ .

**2.** Prove, for all $a \in D_\infty$, that $a = \bigsqcup_n \pi_n a$ .

**3.** Using the definition of $\phi_k$ for $k > 0$, show inductively on $k$ that, for all $c \in D_0$,

$$\theta_{k,\infty} \circ \theta_{0,k+1}(c) \circ \theta_{\infty,k} = \theta_{0,\infty} \circ \phi_0(c) \circ \theta_{\infty,0} : D_\infty \to D_\infty .$$

**4.** Deduce from **3** and the explicit definition of $\phi_\infty$ as the *lub* of a non-decreasing sequence that, for all $a, b$ in $D_\infty$ ,

$$(\pi_0 a) \cdot b = \theta_{0,\infty}(\phi_0(\theta_{\infty,0}(a))(\theta_{\infty,0}(b))) .$$

**5.** Deduce from **4** and Scott's version of $\phi_0$ that $(\pi_0 a) \cdot b = \pi_0 a$ .

**6.** Deduce from **5** and from $\pi_0 \perp = \perp$ that $\perp \cdot \perp = \perp$ .

**7.** As in **3**, show inductively on $k \geq n$ that, for all $a \in D_\infty$ ,

$$\theta_{k,\infty} \circ \theta_{\infty,k+1}(\pi_{n+1} a) \circ \theta_{\infty,k} = \theta_{n,\infty} \circ \theta_{\infty,n+1}(a) \circ \theta_{\infty,n} : D_\infty \to D_\infty .$$

**8.** Deduce from **7** and the explicit definition of $\phi_\infty$ as the *lub* of a non-decreasing sequence that, for all $a, b$ in $D_\infty$ ,

$$(\pi_{n+1} a) \cdot b = \theta_{n,\infty}(\theta_{\infty,n+1}(a)(\theta_{\infty,n}(b))) .$$

(And so $(\pi_{n+1} a) \cdot b \in \pi_n D_\infty =$ "$D_n$" .)

**9.** Combining **8** with $\theta_{\infty,n}(\pi_n b) = \theta_{\infty,n}(b)$ , deduce that

$$(\pi_{n+1} a) \cdot b = (\pi_{n+1} a) \cdot (\pi_n b) .$$

**10.** Using **9** and **1** , deduce that, for all $k \geq n$,

$$(\pi_{n+1} a) \cdot b = (\pi_{n+1} a) \cdot (\pi_k b) .$$

**11.** Using the definition of $\phi_\infty$, and $a \cdot \perp = \phi_\infty(a)(\perp)$ , show that, for all $a \in D_\infty$ , we have

$$\pi_0 a \ \sqsubseteq \ \pi_0(a \cdot \perp) \ .$$

(Actually, equality holds here, but is unneeded below.)

Now let's come back to the situation of Park's theorem. Again to be perhaps overly fussy, we shall use $x$ as the name for a 'general' element of $D_\infty$. And also (to keep $\Lambda$ and $D_\infty$ distinct) underline all the elements in $\Lambda$, including variables. We fix a 'general' variable $\underline{x}$ , and always use a (so-called environment) $\rho$ which maps $\underline{x}$ to $x$. Fix another variable $\underline{y} \in \Lambda$, and define Curry's combinator explicitly as

$$\underline{Y} \ := \ \lambda\underline{x} \bullet (\lambda\underline{y} \bullet \underline{x}(\underline{y} \ \underline{y}))(\lambda\underline{y} \bullet \underline{x}(\underline{y} \ \underline{y})) \ = \ \lambda\underline{x} \bullet \underline{X} \ \underline{X} \ ,$$

where

$$\underline{X} \ := \ \lambda\underline{y} \bullet \underline{x}(\underline{y} \ \underline{y}) \ .$$

Define elements of $D_\infty$ by

$$Y \ := \ \rho_+(\underline{Y}) \qquad [\text{independent of } \rho] \ ,$$

$$X \ := \ \rho_+(\underline{X}) \qquad [\text{depending only on } \rho(\underline{x}) = x] \ .$$

**Remaining Exercises to prove Park's theorem.**

**12.** Show $\underline{X} \ \underline{z} \ = \ \underline{x}(\underline{z} \ \underline{z})$ for any variable $\underline{z}$ in $\Lambda$ .

**13.** Deduce that $X \cdot z \ = \ x \cdot (z \cdot z)$ for any $z \in D_\infty$ .

**14.** Show $\underline{Y} \ \underline{x} \ = \ \underline{X} \ \underline{X}$ in $\Lambda$ .

**15.** Deduce that $Y \cdot x \ = \ X \cdot X$
[for any $x \in D_\infty$ , but note that $X$ 'depends on $x$'].

**16.** Deduce from $\pi_0 a \sqsubseteq a$ for all $a \in D_\infty$, combining **13, 6** and **11**, that

$$\pi_0 X \ \sqsubseteq \ x \cdot \perp \quad .$$

**17.** Deduce from **13, 5** and **16** that

$$X \cdot \pi_0 X \ = \ x \cdot (\pi_0 X \cdot \pi_0 X) \ \sqsubseteq \ x \cdot (x \cdot \perp) \ .$$

**18.** Using **17** as the initial case, and noting from **10** that

$$\pi_{n+1}X \cdot \pi_{n+1}X \;=\; \pi_{n+1}X \cdot \pi_n X \;,$$

show by induction on $n$ that

$$X \cdot \pi_n X \;=\; x \cdot (\pi_n X \cdot \pi_n X) \;\sqsubseteq\; \phi_\infty(x)^{n+2}(\bot) \;\;.$$

**19.** Using the definitions of "$\cdot$" and of $\mathsf{fix}$, deduce from **2, 15** and **18** that

$$\phi_\infty(Y)(x) \;\sqsubseteq\; \mathsf{fix}(\phi_\infty(x)) \;.$$

**20.** Recalling that $\underline{Y}$ is a fixpoint combinator, show that $\phi_\infty(Y)(x)$ is a fixed point of $\phi_\infty(x)$ .

**21.** Deduce from **20** and the basic property of $\mathsf{fix}$ (in **VII-8.12**) that

$$\mathsf{fix}(\phi_\infty(x)) \;\sqsubseteq\; \phi_\infty(Y)(x) \;.$$

**22.** Combine **19** and **21** to get Park's theorem :

$$\phi_\infty(Y) \;=\; \mathsf{fix} \circ \phi_\infty \;\;.$$

**Quick proofs of the crucial later exercises.**

**16.** $\pi_0 X \;\sqsubseteq\; \pi_0(X \cdot \bot) \;\sqsubseteq\; X \cdot \bot \;=\; x \cdot (\bot \cdot \bot) \;=\; x \cdot \bot \;\;.$

**17.** $X \cdot \pi_0 X \;=\; x \cdot (\pi_0 X \cdot \pi_0 X) \;=\; x \cdot \pi_0 X \;\sqsubseteq\; x \cdot (x \cdot \bot) \;\;.$

**18.** The equality is **13**, and, by the hint, the left-hand side for the inductive step in the inequality is

$$x \cdot (\pi_{n+1}X \cdot \pi_n X) \;\sqsubseteq\; x \cdot (X \cdot \pi_n X) \;\sqsubseteq\; x \cdot (\phi_\infty(x)^{n+2}(\bot)) \;=\; \phi_\infty(x)^{n+3}(\bot) \;\;.$$

**19.**

$$\phi_\infty(Y)(x) \;=\; Y \cdot x \;=\; X \cdot X \;=\; X \cdot \bigsqcup_n \pi_n X$$

$$=\; \bigsqcup_n X \cdot \pi_n X \;\sqsubseteq\; \bigsqcup_n \phi_\infty(x)^{n+2}(\bot) \;=\; \bigsqcup_n \phi_\infty(x)^n(\bot) \;=\; \mathsf{fix}(\phi_\infty(x)) \;.$$

**20.**

$$\phi_\infty(x)(\phi_\infty(Y)(x)) \;=\; x{\cdot}(Y{\cdot}x) \;=\; \rho_+(\underline{x}(\underline{Y}\,\underline{x})) \;=\; \rho_+(\underline{Y}\,\underline{x}) \;=\; Y{\cdot}x \;=\; \phi_\infty(Y)(x)\,.$$

**11.**

$$\pi_0(a\cdot\perp) \;=\; \theta_{0\infty}(\theta_{\infty 0}(\phi_\infty(a)(\perp))) \;=\; \theta_{0\infty}\theta_{\infty 0}\bigsqcup_k(\theta_{k\infty}\circ\theta_{\infty,k+1}(a)\circ\theta_{\infty k})(\perp_\infty)$$

$$=\; \bigsqcup_k\theta_{0\infty}(\theta_{k0}(\theta_{\infty,k+1}(a)(\perp_k))) \;\sqsupseteq\; \theta_{0\infty}(\theta_{00}(\theta_{\infty,1}(a)(\perp_0)))$$

$$=\; \theta_{0\infty}(\psi_0(\theta_{\infty,1}(a))) \;=\; \theta_{0\infty}(\theta_{\infty,0}(a)) \;=\; \pi_0 a\,.$$

The step after the inequality uses the definition of $\psi_0$ . To strengthen the inequality to equality (which is needed further down), one shows that all terms in the *lub* just before the " $\sqsupseteq$ " agree with $\pi_0 a$, by a slightly longer argument. For example, with $k=1$, we get down to

$$\theta_{0\infty}(\theta_{10}(\theta_{\infty,2}(a)(\perp_1))) \;=\; \theta_{0\infty}(\psi_0(\theta_{\infty,2}(a)(\phi_0(\perp_0)))) \;=\; \theta_{0\infty}((\psi_0{\circ}\theta_{\infty,2}(a){\circ}\phi_0)(\perp_0))$$

$$=\; \theta_{0\infty}(\psi_1(\theta_{\infty,2}(a))(\perp_0)) \;=\; \theta_{0\infty}(\psi_0(\psi_1(\theta_{\infty,2}(a)))) \;=\; \theta_{0\infty}(\theta_{\infty,0}(a)) \;=\; \pi_0 a\,.$$

Our early example of a $\Lambda$-term with no normal form, namely

$$(\lambda x\bullet xx)(\lambda x\bullet xx)\;,$$

is another interesting case for the denotational semantics of $\Lambda$ using Scott's $D_\infty$ . The answer is actually that

$$\rho_+((\lambda x\bullet xx)(\lambda x\bullet xx)) \;=\; \perp\;,$$

a theorem of Scott, derived below using Park's theorem. (This tries to say that the term carries 'no information at all'.) The reader is challenged to use the definitions directly to show this, and see how much is involved. One approach is to attempt to show, for all $d\in D_\infty$ , that

$$\rho_+((\lambda x\bullet xx)(\lambda x\bullet xx)) \;\sqsubseteq\; d\;.$$

The trick we use is the following : Let

$$I := \rho_+(\underline{I}) := \rho_+(\lambda x\bullet x)\;.$$

Using the evident fact that $\underline{I}\ \underline{A} \approx \underline{A}$ for all $\underline{A} \in \Lambda$, it is clear that $I \cdot d = d$ for all $d \in D_\infty$ ; that is, every element of $D_\infty$ is a fixed point of $\phi_\infty(I)$, which is the identity map of $D_\infty$ . In particular, the minimum fixed point of $\phi_\infty(I)$ is certainly $\perp$ . And so, from Park, we see that

$$\rho_+(\underline{Y}\ I) \ = \ \perp \ .$$

Thus it suffices to prove that

$$\underline{Y}\ I \ \approx \ (\lambda x \bullet xx)(\lambda x \bullet xx) \ .$$

Letting $C := \lambda y \bullet x(yy)$, we have $\underline{Y} = \lambda x \bullet CC$, and, for any closed $A$, we immediately $(\beta)$-reduce $\underline{Y}A$ to $C_AC_A$ , where $C_A := C^{[x \to A]} = \lambda y \bullet A(yy)$. But

$$C_I \ = \ \lambda y \bullet I(yy) \approx \ \lambda y \bullet yy \ ,$$

so

$$\underline{Y}\ I \ \approx \ C_IC_I \ \approx \ (\lambda y \bullet yy)(\lambda y \bullet yy) \ ,$$

as required.

**Exercise.** Show that $\rho_+((\lambda x \bullet xxx)(\lambda x \bullet xxx)) \ = \ \perp$ .

As mentioned earlier, by $(*)$ from about 7 pages back , we can only expect $\rho_+$ to be possibly injective after passing to equivalence classes under $\approx$ . And it certainly won't be on non-closed terms in general unless $\rho$ itself is injective. But for closed terms without normal forms, it still isn't injective in general, as we see further down. However, we do have the following 'faithfulness of the semantics' for terms with normal forms.

**Theorem VII-9.2.** *If $E$ and $F$ are distinct normal form terms in $\Lambda$* (i.e. they are not related by a change of bound variables)*, and continuing with Scott's $D_\infty$ as our domain, for some $\rho$, we have $\rho_+(E) \neq \rho_+(F)$ . In particular, the restriction of $\rho_+$ to closed normal form terms* (which restriction is independent of $\rho$) *is injective.*

This will be almost immediate from another quite deep syntactical result whose proof will be omitted:

**Böhm's Theorem VII-9.3.** (See [**Cu**], p.156) *If $E$ and $F$ are as in the previous theorem, and $y_1, \cdots, y_t$ are the free variables in $EF$, then there are terms $G_1, \cdots, G_t$, and $H_1, \cdots, H_k$ for some $k$, such that for some distinct variables $u$ and $v$*

$$E^{[\vec{y}\to\vec{G}]}H_1\cdots H_k uv \;\approx\; u \quad \text{and} \quad F^{[\vec{y}\to\vec{G}]}H_1\cdots H_k uv \;\approx\; v \;.$$

To deduce the earlier theorem, proceed by contradiction, and use the property labelled $(**)$ at the beginning of this section to see easily that

$$\forall\rho \;,\; \rho_+(E) = \rho_+(F) \quad \Longrightarrow \quad \forall\rho \;,\; \rho_+(E^{[\vec{y}\to\vec{G}]}) = \rho_+(F^{[\vec{y}\to\vec{G}]}) \;.$$

But then the left-hand sides in Böhm's theorem map to the same thing under all the maps $\rho_+$ , contradicting the fact that $\rho_+(u) \neq \rho_+(v)$ for some $\rho$ (since $\rho$ can take any values we want on variables).

To illustrate the 'serious' non-injectiveness of the semantics related to terms without normal forms, here is a striking theorem of Wadsworth [**Wa**].

**Theorem VII-9.4.** *Let $\underline{M}$ be any closed $\Lambda$-term with a normal form. Then there is a closed term $\underline{N}$ with no normal form* (so, of course, $\underline{M} \not\approx \underline{N}$) *such that $\rho_+(\underline{M}) \;=\; \rho_+(\underline{N})$ .*

**Proof.** Firstly, let us establish that it suffices to prove this for the single example $\underline{M} = \underline{I} := \lambda x \bullet x$ , where $\rho_+(\underline{I}) = I$ corresponds to the identity map $D_\infty \to D_\infty$ under $\phi_\infty$ . So assume (as we shall prove below) that $\rho_+(\underline{J}) = \rho_+(\underline{I})$ for some closed term $\underline{J}$ with no normal form. Given $\underline{M}$ as in the theorem, let $\underline{M}'$ be its normal form. Find the variable occurrence furthest to the right in the string $\underline{M}'$ (and say $x$ is that variable). Now replace that single occurrence $x$ by $(\underline{I}\; x)$ and call the result $\underline{M}''$ . Similarly replace it by $(\underline{J}\; x)$ and call the result $\underline{N}$ . Since $\underline{M}'$ is normal, the sequence of leftmost reductions for $\underline{N}$ is simply the same (infinite) sequence for $\underline{J}$ applied to that subterm. So $\underline{N}$ has no normal form. Also

$$\rho_+(\underline{M}) \;=\; \rho_+(\underline{M}') \;=\; \rho_+(\underline{M}'') \quad ,$$

since $\underline{M} \;\approx\; \underline{M}' \;\approx\; \underline{M}''$ , the latter because $\underline{I}\; x \approx x$ . But also

$$\rho_+(\underline{M}'') \;=\; \rho_+(\underline{N}) \;,$$

completing this part of the proof. To see this last equality, note that, in the notation way back in **VII-1.1**, it is easy to see that

$$\text{if } \ \rho_+(A) \ = \ \rho_+(B) \ , \ \text{ then } \ \rho_+(SAT) \ = \ \rho_+(SBT) \ .$$

So take $A$ and $B$ to be $\underline{I}$ and $\underline{J}$, respectively, in $SAT = \underline{M}''$ and $SBT = \underline{N}$.

Thus it remains to establish the particular case $\underline{M} = \underline{I}$ . Here we'll give $\underline{J} = \underline{N}$ quite explicitly. Let

$$\underline{F} \ := \ \lambda f x y \bullet x(f y) \ , \quad \text{and take} \qquad \underline{J} \ := \ \underline{Y} \ \underline{F} \ ,$$

where $\underline{Y}$ is again Curry's fixpoint combinator. We shall show quite generally in the paragraphs after next that, for any closed $\underline{A} \in \Lambda$ ,

$$\underline{F} \ \underline{A} \ \approx \ \underline{A} \implies \ \rho_+(\underline{A}) \ = \ \rho_+(\underline{I}) \qquad\qquad (*)$$

(It is very easy, but irrelevant, to see that $\underline{F} \ \underline{I} \ \approx \ \underline{I}$ .) However, the fact that $\underline{F} \ \underline{J} \ \approx \ \underline{J}$ is immediate from the fixpoint property of Curry's combinator, so the above will give us $\rho_+(\underline{J}) \ = \ \rho_+(\underline{I})$ , the main conclusion of the theorem.

First let's prove the other required property of $\underline{J}$ , namely that it has no normal form. With $\underline{Y} := \lambda x \bullet C_x C_x$, where $C_x := \lambda y \bullet x(yy)$ and

$$C_A \ := \ C_x^{[x \mapsto A]} \ = \ \lambda y \bullet A(yy) \ \ ,$$

the leftmost reduction of $\underline{J} = \underline{Y} \ \underline{F}$ goes as follows :

$$\begin{aligned}
\underline{Y} \ \underline{F} \ \overline{\succeq} \ C_{\underline{F}} C_{\underline{F}} \ &= \ (\lambda y \bullet \underline{F}(yy)) C_{\underline{F}} \ \overline{\succeq} \ \underline{F}(C_{\underline{F}} C_{\underline{F}}) \\
&\overline{\succeq} \ \lambda x y \bullet x((C_{\underline{F}} C_{\underline{F}}) y) \ \overline{\succeq} \ \cdots\cdots \\
&\overline{\succeq} \ \lambda x y \bullet x((\lambda x y \bullet x((C_{\underline{F}} C_{\underline{F}}) y)) y) \ \overline{\succeq} \ \cdots\cdots \\
\overline{\succeq} \ \lambda x y \bullet x((\lambda x y \bullet x((\lambda x y \bullet x((C_{\underline{F}} C_{\underline{F}}) y)) y)) y) \ &\overline{\succeq} \ - - - - - - - \ .
\end{aligned}$$

This sequence does not terminate, as required.

The subtlest part comes now, in proving $(*)$ above, by manipulations due to Wadsworth which seem fiendishly clever to me! So assume that $\underline{A}$ is any closed term with $\underline{F} \ \underline{A} \ \approx \ \underline{A}$ , and we shall show that $A = I$, where $A := \rho_+(\underline{A})$ . For all $\underline{B}$ and $\underline{C}$ in $\Lambda$, we have

$$\underline{A} \ \underline{B} \ \underline{C} \ \approx \ \underline{F} \ \underline{A} \ \underline{B} \ \underline{C} \ = \ (\lambda f x y \bullet x(f y)) \underline{A} \ \underline{B} \ \underline{C} \ \approx \ \underline{B} \ (\underline{A} \ \underline{C})$$

Applying $\rho_+$ , we get, for all $B$ and $C$ in $D_\infty$ ,

$$A \cdot B \cdot C \;=\; B \cdot (A \cdot C) \qquad\qquad (**)$$

To show that $A = I$, we'll now use only $(**)$. It suffices to show that $\pi_n A = \pi_n I$ for all $n$, by induction on $n$, involving the projections $\pi_n$ introduced back in the discussion of Park's theorem. Here is a repeat of three of the exercises from back there, plus a list of four new general exercises for the reader to work on. But if needed, see after the end below of the present theorem's proof for some hints which make their proofs completely mechanical.

**5**. $$(\pi_0 a) \cdot b \;=\; \pi_0 a \ .$$

**9**. $$(\pi_{n+1} a) \cdot b \;=\; (\pi_{n+1} a) \cdot (\pi_n b) \ .$$

**11**. $$\pi_0 a \;=\; \pi_0 (a \cdot \bot) \ .$$

**23**. $$\pi_n \bot \;=\; \bot \ .$$

**24**. $$\bot \cdot x = \;\bot \ .$$

**25**. $$I \cdot C = \; C \ .$$

**26**. $$(\pi_{n+1} a) \cdot b \;=\; \pi_n (a \cdot \pi_n b) \ .$$

In both initial cases and also in the inductive case, we'll use the extensionality of $D_\infty$ twice.

The initial case $n = 0$ :

Using **5**, then **5**, then **11**, for all $B, x$ and $y$ in $D_\infty$ ,

$$\pi_0 B \cdot x \cdot y \;=\; \pi_0 B \cdot y \;=\; \pi_0 B \;=\; \pi_0 (B \cdot \bot) \ .$$

Thus, with $B = I$, using the above, then **25**,

$$\pi_0 I \cdot x \cdot y \;=\; \pi_0 (I \cdot \bot) \;=\; \pi_0 \bot \ .$$

With $B = A$, using the above, then **11**, then $(**)$, then **24**,

$$\pi_0 A \cdot x \cdot y \;=\; \pi_0 (A \cdot \bot) \;=\; \pi_0 (A \cdot \bot \cdot \bot) \;=\; \pi_0 (\bot \cdot (A \cdot \bot)) \;=\; \pi_0 \bot \ .$$

So $\pi_0 A = \pi_0 I$, by extensionality.

The initial case $n = 1$ :

Using **26**, then **25**, then idempotency of $\pi_0$, then **5**,

$$\pi_1 I \cdot x \cdot y \;=\; \pi_0(I \cdot \pi_0 x) \cdot y \;=\; \pi_0(\pi_0 x) \cdot y \;=\; \pi_0 x \cdot y \;=\; \pi_0 x \;.$$

Using **26**, then **5**, then **11**, then $(**)$, then **5**, then idempotency of $\pi_0$,

$$\pi_1 A \cdot x \cdot y \;=\; \pi_0(A \cdot \pi_0 x) \cdot y \;=\; \pi_0(A \cdot \pi_0 x)$$

$$=\; \pi_0(A \cdot \pi_0 x \cdot \bot) \;=\; \pi_0(\pi_0 x \cdot (A \cdot \bot)) \;=\; \pi_0(\pi_0 x) \;=\; \pi_0 x \;.$$

So $\pi_1 A = \pi_1 I$, by extensionality.

The inductive case :

Assume inductively that $\pi_{n+1} A = \pi_{n+1} I$. Using **26**, then **25**, then idempotency of $\pi_{n+1}$, then **9**,

$$\pi_{n+2} I \cdot x \cdot y \;=\; \pi_{n+1}(I \cdot \pi_{n+1} x) \cdot y \;=\; \pi_{n+1}(\pi_{n+1} x) \cdot y \;=\; \pi_{n+1} x \cdot y \;=\; \pi_{n+1} x \cdot \pi_n y \;.$$

Now using **26**, then the inductive assumption, then **26**, then **25**, then idempotency of $\pi_n$,

$$\pi_n(A \cdot \pi_n y) \;=\; \pi_{n+1} A \cdot y \;=\; \pi_{n+1} I \cdot y \;=\; \pi_n(I \cdot \pi_n y) \;=\; \pi_n(\pi_n y) \;=\; \pi_n y \;.$$

So, using **26**, then **26**, then $(**)$, then **9**, then **26**, then idempotency of $\pi_{n+1}$, then the display just above,

$$\pi_{n+2} A \cdot x \cdot y \;=\; \pi_{n+1}(A \cdot \pi_{n+1} x) \cdot y \;=\; \pi_n(A \cdot \pi_{n+1} x \cdot \pi_n y)$$

$$=\; \pi_n(\pi_{n+1} x \cdot (A \cdot \pi_n y)) \;=\; \pi_n(\pi_{n+1} x \cdot \pi_n(A \cdot \pi_n y))$$

$$=\; \pi_{n+1}(\pi_{n+1} x) \cdot \pi_n(A \cdot \pi_n y) \;=\; \pi_{n+1} x \cdot \pi_n(A \cdot \pi_n y) \;=\; \pi_{n+1} x \cdot \pi_n y \;.$$

So, once again by extensionality, $\pi_{n+2} A = \pi_{n+2} I$ .

Thus **VII-9.4** is now finally proved.

As for the new exercises used above, if you haven't done them already, here are statements which reduce them to mechanical checks :

As for **23**, this just amounts to the fact that

$$\bot \;:=\; \bot_{D_\infty} \;=\; (\bot_0, \bot_1, \bot_2, \cdots) \;,$$

since the right-hand side is clearly the smallest element in $D_\infty$ .

As for **24**, the argument is the same as for **6**.

As for **25**, this is immediate from $\underline{I}\;\underline{C} \;\approx\; \underline{C}$ .

**27**. Using the definition of the maps $\psi_j$, show by induction on $k \geq n$ that for all $a \in D_\infty$ and $y \in D_n$ , we have

$$\theta_{kn}(\theta_{\infty,k+1}(a)(\theta_{nk}(y))) \;=\; \theta_{\infty,n+1}(a)(y) \;.$$

As for **26**, use **27** with $y = \theta_{\infty n}(b)$ in the last step just below to see that

$$\pi_n(a{\cdot}\pi_n b) \;=\; \pi_n(\phi_\infty(a)(\pi_n b)) \;=\; \theta_{n\infty}{\circ}\theta_{\infty n}(\bigsqcup_k \theta_{k\infty}(\theta_{\infty,k+1}(a)(\theta_{\infty,k}{\circ}\theta_{n\infty}{\circ}\theta_{\infty n}(b))))$$

$$=\; \theta_{n\infty}(\bigsqcup_k \theta_{kn}(\theta_{\infty,k+1}(a)(\theta_{nk}\circ\theta_{\infty n}(b)))) \;=\; \theta_{n\infty}(\theta_{\infty,n+1}(a)(\theta_{\infty n}(b))) \;.$$

But the latter is $(\pi_{n+1}a)\cdot b$ by **8** .

Quite a bit easier is producing an example of two distinct elements of $\Lambda/\approx$ , both closed, and *neither* with normal form, which give the same element of $D_\infty$ under $\rho_+$ . Such elements are the classes of $\underline{Y}$ and $\underline{Y'} = \underline{Y}\;\underline{G}$, where $\underline{Y}$ is again Curry's fixpoint combinator, and

$$\underline{G} \;:=\; \lambda uv \bullet v(uv) \;.$$

The argument is quite elegant :

Firstly, it's becoming slightly embarrassing how many syntactic results we are leaving the reader to look up, but here is the final one:

**Proposition VII-9.5.** $\underline{Y'} \not\approx \underline{Y}$ *and neither has a normal form.*
See Böhm p. 179 in **[St-ed]**.

To show that $\rho_+\underline{Y'} \;=\; \rho_+\underline{Y}$ , first we establish a nice result of independent interest.

**Proposition VII-9.6.**

$$\forall \underline{Z} \in \Lambda \ , \ [\ \underline{G}\ \underline{Z} \ \approx \ \underline{Z} \ \iff \ \forall \underline{A} \in \Lambda \ , \ \ \underline{A}\ (\underline{Z}\ \underline{A}) \ \approx \ \underline{Z}\ \underline{A}\ ] \ .$$

*That is, $\underline{Z}$ is a fixed point of $\underline{G}$ if and only if it is a fixpoint operator.*

**Proof.** First note that

$$\underline{G}\ \underline{Z} \ = \ (\lambda uv \bullet v(uv))\underline{Z} \ \approx \ \lambda v \bullet v(\underline{Z}v) \qquad\qquad (*)$$

As for $\Longrightarrow$: Using the assumption, then $(*)$, then $\beta$-reduction,

$$\underline{Z}\ \underline{A} \ \approx \ \underline{G}\ \underline{Z}\ \underline{A} \ \approx \ (\lambda v \bullet v(\underline{Z}v))\underline{A} \ \approx \ \underline{A}\ (\underline{Z}\ \underline{A}) \ \ .$$

As for $\Longleftarrow$: Using $(*)$, then the assumption with $\underline{A} = v$, then $\eta$-reduction,

$$\underline{G}\ \underline{Z} \ \approx \ \lambda v \bullet v(\underline{Z}v) \ \approx \ \lambda v \bullet \underline{Z}v \ \approx \ \underline{Z} \ \ .$$

Now abbreviate $\rho_+\underline{Y'}$ and $\rho_+\underline{Y}$ to $Y'$ and $Y$, respectively.

**Proposition VII-9.7.** *$\underline{Y'}$ is a fixpoint operator ; and so*

$$A \cdot (Y' \cdot A) = Y' \cdot A \ \text{ for all } \ A \in D_\infty \ .$$

**Proof.** Use 9.6$\Longrightarrow$, after calculating using the fact that $\underline{Y}$ is a fixpoint operator :

$$\underline{G}\ \underline{Y'} \ = \ \underline{G}(\underline{Y}\ \underline{G}) \ \approx \ \underline{Y}\ \underline{G} \ = \ \underline{Y'} \ .$$

**Corollary of Park's Theorem VII-9.8.**

$$Y \cdot A \ \sqsubseteq \ Y' \cdot A \quad \text{ for all } \ A \in D_\infty \ .$$

**Proof.** Both $Y \cdot A$ and $Y' \cdot A$ are fixed by $A \cdot$ . But Park says that $Y \cdot A$ is the smallest of all elements in $D_\infty$ fixed by $A \cdot$ .

**Corollary VII-9.9** (of $\Longleftarrow$ in 9.6).

$$\underline{G}\ \underline{Y} \ \approx \ \underline{Y} \ \text{ and so } \ G \cdot Y \ = \ Y$$

Thus $Y$ is a fixed point of $G \cdot$ . And so $Y \cdot G \ \sqsubseteq \ Y$, i.e. $Y' \ \sqsubseteq \ Y$, since $Y \cdot G$ is the smallest fixed point of $G \cdot$ . Thus we get

**Corollary VII-9.10**

$$Y' \cdot A \sqsubseteq Y \cdot A \quad \text{for all} \quad A \in D_\infty \ .$$

And now finally

**Corollary VII-9.11** (of 9.10 and 9.8)

$$Y' \cdot A = Y \cdot A \quad \text{for all} \quad A \in D_\infty \ .$$

Thus, by extensionality, we have what we want, namely

$$Y' = Y \quad \text{that is,} \quad \rho_+ \underline{Y'} = \rho_+ \underline{Y} \ .$$

We should finish with what seems to be the canonical example used whenever anything related to 'recursive computing' needs a simple illustration, namely, the factorial function. And then again, maybe we shouldn't, though just seeing how complicated or otherwise a $\Lambda$-term is needed to programme the factorial function is a bit interesting!

## References


[**Al**] Allison, Lloyd *A Practical introduction to denotational semantics.* Cambridge U. Press, Cambridge, 1989.

[**Ba**] Barendregt, H. P. *The Lambda Calculus : its syntax and semantics.* North-Holland, Amsterdam, 1984.

[**BKK-ed**] Barwise, J., Keisler, H.J. and Kunen, K. *The Kleene Symposium.* North-Holland, Amsterdam, 1980.

[**Bö-ed**] Böhm, C. *$\lambda$-calculus and computer science theory : Proceedings.* LNCS # 37, Springer-Verlag, Berlin, 1975.

[**Br-ed**] Braffort, Paul. *Computer Programming and Formal Systems.* North-Holland, Amsterdam, 1963.

[**Ch**] Church, Alonzo *The Calculi of Lambda-Conversion.* Princeton U. Press, Princeton, 1941.

[**CM**] Hoffman, P. *Computability for the Mathematical.* this website, 2005.

[**Cu**] Curry, H. B., Hindley, J. R. and Seldin, J. P. *Combinatory Logic, Vol. II.* North-Holland, Amsterdam, 1972.

[**En**] Engeler, Erwin, et al. *The Combinatory Programme.* Birkhäuser, Basel, 1995.

[**Fi**] Fitch, F. B. *Elements of Combinatory Logic.* Yale U. Press, New Haven, 1974.

[**Go**] Gordon, M. J. C. *Denotational Description of Programming Languages.* Springer-Verlag, Berlin, 1979.

[**Go1**] Gordon, M. J. C. *Programming Language Theory and its Implementation.* Prentice Hall, New York, 1988.

[**Hi**] Hindley, J. R. *Standard and Normal Reductions.* Trans.AMS, 1978.

[**HS**] Hindley, J. R. and Seldin, J. P. *Introduction to Combinators and the $\lambda$-calculus.* London Mathematical Society Student Texts # 1, Cambridge U. Press, Cambridge, 1986.

[**HS-ed**] Hindley, J. R. and Seldin, J. P. *To H. B. Curry : Essays on Combinatory Logic, lambda-calculus, and formalism.* Academic Press, London, 1980.

[**Kl**] Klop, J. W. *Combinatory Reduction Systems.* Mathematisch Centrum, Amsterdam, 1980.

[**Ko**] Koymans, C. P. J. *Models of the lambda calculus.* CWI Tract, Amsterdam, 1984.

[**La-ed**] Lawvere, F. W. *Toposes, Algebraic Geometry and Logic.* LNM # 274, Springer-Verlag, Berlin, 1972.

[**LS**] Lambek, J. and Scott, P. J. *Introduction to higher order categorical logic.* Cambridge U. Press, Cambridge, 1986.

[**LM**] Hoffman, P. *Logic for the Mathematical.* this website, 2003.

[**Pe**] Penrose, R. *The Emperor's New Mind.* Oxford U. Press, Oxford, 1989.

[**Ru-ed**] Rustin, Randall *Formal Semantics of Programming Languages.* Prentice-Hall, N.J., 1972.

[**SAJM-ed**] Suppes, P.—Henkin, L.—Athanase, J.—Moisil, GR. C. *Logic, Methodology and Philosophy of Science IV.* North-Holland, Amsterdam, 1973.

[**St-ed**] Steel, T.B. *Formal Language Description Languages for Computer Programming.* North-Holland, Amsterdam, 1966.

[**Sö**] Stenlund, Sören *Combinators, $\lambda$-terms, and Proof Theory.* Reidel Pub. Co., Dordrecht, 1972.

[**St**] Stoy, Joseph *Denotational Semantics : the Scott-Strachey approach to programming language theory.* MIT Press, Cambridge, Mass., 1977.

[**Wa**] Wadsworth, Christopher P. *The Relation between Computational and Denotational Properties for Scott's $D_\infty$-Models of the Lambda-calculus.* SIAM J. Comput. 5(3) 1976, 488-521.