Theoretical Computer Science 1 (1975) 125–159.

CALL-BY-NAME, CALL-BY-VALUE AND THE λ-CALCULUS

G. D. PLOTKIN

Department of Machine Intelligence, School of Artificial Intelligence, University of Edinburgh, Edinburgh, United Kingdom

Communicated by R. Milner
Received 1 August 1974

Abstract. This paper examines the old question of the relationship between ISWIM and the λ-calculus, using the distinction between call-by-value and call-by-name. It is held that the relationship should be mediated by a standardisation theorem. Since this leads to difficulties, a new λ-calculus is introduced whose standardisation theorem gives a good correspondence with ISWIM as given by the SECD machine, but without the *letrec* feature. Next a call-by-name variant of ISWIM is introduced which is in an analogous correspondence with the usual λ-calculus. The relation between call-by-value and call-by-name is then studied by giving simulations of each language by the other and interpretations of each calculus in the other. These are obtained as another application of the continuation technique. Some emphasis is placed throughout on the notion of operational equality (or contextual equality). If terms can be proved equal in a calculus they are operationally equal in the corresponding language. Unfortunately, operational equality is not preserved by either of the simulations.

1. Introduction

Our intention is to study call-by-value and call-by-name in the setting of the lambda-calculus which was first used to explicate programming language features by Landin [5, 6, 7]. To this end, for each calling mechanism we set up a programming language and a formal calculus and then show how each determines the other. After that we give simulations of the call-by-value programming language by the call-by-name one and vice versa — this also provides interpretations of each calculus in the other one.

If the terms of the λ-calculus (we have in mind the λK-β calculus for the moment) are regarded as rules, with a reduction relation showing how they may be carried out and indeed with a normal order reduction sequence capturing, in deterministic fashion, all possible normal forms, then we have already pretty well determined a programming language.

On the other hand, the language can be regarded as giving true equations between programs (= terms of the calculus). Informally, one program equals another, operationally, if it can be substituted for the other in all contexts without "changing

the results" (cf. [9, 17]). From this point of view a calculus can be correct with respect to the programming language.

As primary example of a λ-calculus based programming language we consider ISWIM [5, 7], without the recursion operator or any syntactic sugar. It has an operational semantics which is given by the SECD machine. As primary example of a calculus, take the λK-$\beta\delta$ calculus [2] (the δ-rules make the comparison easier).

Unfortunately, the two are hardly in accord.

(1) Sometimes the SECD machine stops when it should either go on to give a normal form or should not terminate, according to normal order reduction. This is because ISWIM does not simplify procedure bodies.

(2) Sometimes the SECD machine never stops when, according to normal order reduction, it ought to. This is because ISWIM calls its arguments by value.

So one has to look for other programming language/calculus pairs. Wegner [19] gives a machine which gives a language corresponding to normal order reduction for the λK-β calculus, regarding (1) and (2) as bad points of the SECD machine. McGowan [8] gives a call-by-value machine and a corresponding altered normal order reduction sequence (where value = normal form); thus regarding (1) as a bad point of the SECD machine, but accepting (2). However he does not give a call-by-value calculus.

Our intention is to study programming mechanisms and so we accept the SECD machine and look for the corresponding calculus — called λ_v in the text. The notion of value is changed to that induced by the SECD machine, and a normal order reduction sequence theorem is given, which establishes a good correspondence between λ_v and ISWIM. In this way we hope to have shown that ISWIM is more than a specification of some characterless reduction sequence. Rather, as well as being computationally natural, it gives rise to an interesting calculus. Its correspondence with this calculus shows it to be less order of reduction dependent than its definition shows.

To study call-by-name, we define a call-by-name ISWIM, corresponding to a certain modification of the SECD machine, which *keeps* the above notion of value, and show that the usual λK-$\beta\delta$ calculus can be regarded as the call-by-name calculus. This substantiates folklore.

In both cases the calculi are seen to be correct from the point of view of the programming languages.

Finally, as mentioned above, we give simulations of call-by-value by call-by-name and vice versa. These use the continuation technique as developed in [3, 10, 12, 15]. It turns out that the simulations also give interpretations of each formal system in the other one, but unfortunately they do not preserve operational equality.

From a practical point of view, one can, using λ_v, look for optimal or improved evaluation mechanisms as in [17]. See also [16] for similar work on McCarthy systems of recursive definitions.

In a future paper we will discuss the relation between the evaluation mechanisms considered here and some denotational semantics in the style of Scott and Strachey [14].

2. Technical preliminaries

The set of λ-calculus terms is determined by a set of *variables* $x, y, z, \ldots$, a disjoint set of *constants* $a, b, \ldots$ and *improper symbols*, λ, (,and).

It is defined inductively by:

(1) Any variable is a term.

(2) Any constant is a term.

(3) If x is a variable and M is a term then (λxM) is a term.

(4) If M and N are terms, so is (MN).

A term of the form (λxM) is an *abstraction*; one of the form (MN) is a combination. A term is a *value* iff it is not a combination. In general the set of variables will be infinite, although the set of constants need not be. The set of variables is called Variables. Similar conventions are used throughout. We will also use variables and constants as metavariables ranging over variables and constants respectively. $M = N$ means that M and N are identical terms. We say that M_i is in *position i* in $(M_1\ M_2)$ $(i = 1, 2)$.

If M is a term, it has a set $FV(M)$ of free variables and a set $BV(M)$ of bound variables. These are defined inductively by:

(1) $FV(x) = \{x\}$; $FV((MN)) = FV(M) \cup FV(N)$; $FV((\lambda xM)) = FV(M) \backslash \{x\}$.

(2) $BV(x) = \emptyset$; $BV((MN)) = BV(M) \cup BV(N)$; $BV((\lambda xM)) = BV(M) \cup \{x\}$.

A term is *closed* iff $FV(M) = \emptyset$, otherwise it is *open*. The *size* $|M|$ of a term is defined inductively by:

$$|x| = |a| = 1; \quad |(\lambda xM)| = |M| + 1; \quad |(MN)| = |M| + |N|.$$

Given an infinite list $x_1, \ldots$ of distinct variables, the substitution prefix is defined inductively by:

$$[M/x]\, x = M; \quad [M/x]\, y = y \text{ (if } x \neq y);$$
$$[M/x]\, a = a;$$
$$[M/x]\, (NN') = ([M/x]\, N\, [M/x]\, N');$$
$$[M/x]\, (\lambda xN) = (\lambda xN); \quad [M/x]\, (\lambda yN) = \lambda z\, [M/x]\, [z/y]\, N, \text{ if } x \neq y,$$

where z is the variable defined by:

(1) If $x \notin FV(N)$ or $y \notin FV(M)$ then $z = y$.

(2) Otherwise, z is the first variable in the list $x_1, x_2, \ldots$ such that $z \notin FV(N) \cup FV(M)$.

That this is a good definition is shown in [2] where other properties of the substitution prefix can be found.

The relation, $=_\alpha$, of alphabetic equivalence, is defined inductively by:

(1) $x =_\alpha x$ and $a =_\alpha a$.

(2) If $M =_\alpha M'$ and $N =_\alpha N'$ then $(MN) =_\alpha (M'N')$.

(3) If $M =_\alpha [x/y]\, M'$, where either $x = y$ or $x \notin FV(M')$ then $(\lambda xM) =_\alpha (\lambda yM')$.

In general we will only be interested in proving terms alphabetically equivalent. For this reason it does not matter which set of variables is used when defining terms

or which list when defining the substitution prefix. However on occasion it will be convenient to specify these more fully and prove syntactic equality; this will not result in any loss of generality.

In Section 3, nil will be used for the empty sequence and : for concatentation. Given a set X, X^* is the set of sequences of members of X.

Given sets X and Y, $(X \xrightarrow{p} Y)$ is the set of partial functions from X to Y. If $f \in (X \xrightarrow{p} Y)$, Dom (f) is its domain, that is Dom $(f) = \{x \in X | \langle x, y \rangle \in f$, for some $y \in Y\}$. Expressions using partial functions are defined iff the functions are defined at the indicated arguments. They are equal $(=)$ iff they are both undefined, or are both defined and have the same value. They are alphabetically equivalent $(=_\alpha)$ under similar conditions.

Given a relation $\rightarrow$ (usually using an infix notation), $\xrightarrow{n}$ is its nth power $(n \geqslant 0)$, $\xrightarrow{+}$ is its transitive closure, and $\xrightarrow{*}$ is its transitive reflexive closure.

Occasionally we shall prove something by "lexicographic induction" on, say, a pair $\langle m, n \rangle$ of integer indices. The ordering $\leqslant$ used is this: $\langle m, n \rangle \leqslant \langle m', n' \rangle$ iff $m < m'$ or else $m = m'$ and $n \leqslant n'$. Such proofs can be replaced by nested ordinary inductions.

3. ISWIM

We are going to define ISWIM without letrec, without any syntactic sugar and without any imperative features. Abstract syntax will also be ignored. Its set of *programs* is just the set of closed terms, as defined in Section 2, given some infinite set of variables and a set of constants.

Example. The constants are $\Delta_n (n \geqslant 0)$, Succ, Pred, and Zero. The variables are, say, $x_1, x_2, \ldots$.

To complete the definition of the programming language, we will specify the evaluation function Eval_V: Programs $\xrightarrow{p}$ Programs. This is defined relative to an interpretation of the constants, given by a function:

$$\text{Constapply: Constants} \times \text{Constants} \xrightarrow{p} \text{Closed Values.}$$

Constapply will give the δ-rules for λ_V, and it is for this reason that we do not take Constapply in Constants $\times$ Closed Values $\xrightarrow{p}$ Closed Values, as does Landin, since this would not lead to δ-rules in the sense of Curry [2]. From a practical point of view this seems to lose few possibilities. By making a few alterations we could have allowed the range of Constapply to be Closed Terms.

An important special case is when Constants is a disjoint union of functional constants, $\mathcal{F}$-Constants, and basic constants, $\mathcal{B}$-Constants, and Constapply can be regarded as being in $\mathcal{F}$-Constants $\times$ $\mathcal{B}$-Constants $\xrightarrow{p}$ Closed Values.

Example. (contd.) With the constants and variables as above, set $\mathcal{F}$-Constants = {Succ, Pred, Zero} and $\mathcal{B}$-Constants = $\{\Delta_n | n \geqslant 0\}$ and define Constapply by:

$$\text{Constapply (Succ, } \Delta_n) = \Delta_{n+1},$$
$$\text{Constapply (Pred, } \Delta_{n+1}) = \Delta_n,$$
$$\text{Constapply (Zero, } \Delta_0) = (\lambda x\,(\lambda y x)) \text{ and}$$
$$\text{Constapply (Zero, } \Delta_{n+1}) = (\lambda x(\lambda y y)) \qquad (n \geqslant 0).$$

The official definition of Eval_V now requires the SECD machine. It should be remarked that this is too tedious to work with directly and so we will give an equivalent definition of Eval_V immediately afterwards, prove it equivalent and from then on use only the simpler one.

The SECD machine is given by a set, Dumps, of states and a transition function $\Rightarrow$. $\text{Eval}_V(M)$ is obtained by loading M with a function Load: Programs $\to$ Dumps, running the machine till it stops and unloading it with a partial function Unload: Dumps $\xrightarrow{P}$ Programs.

Dumps is specified via definitions of the sets: Closures, Environments, Controlstrings, and Stacks.

Closures and Environments are defined inductively by:

(1) If $x_1, \ldots, x_n$ are distinct variables and Cl_i $(i = 1, n)$ are closures, $\{\langle x_i, \text{Cl}_i\rangle | i = 1, n\}$ is in Environments $(n \geqslant 0)$.

(2) If E is an environment and M is a term such that $FV(M) \subseteq \text{Dom}(E)$, then $\langle M, E\rangle$ is a closure.

Our definition differs in a few ways from that of Landin. The main difference is that in (2) we allow M to be any term, rather than just a λ-expression; closures of the form $\langle a, \emptyset\rangle$ will perform for us the function the corresponding constants do for Landin and the other possibilities allow call-by-name versions of the SECD machine.

$E\{\text{Cl}/x\}$ is the unique environment E' such that $E'(y) = E(y)$, if $y \neq x$ and $E'(x) = \text{Cl}$ (Cl ∈ Closures).

The function Real: Closures $\to$ Terms is defined inductively by:

$$\text{Real}(\langle M, E\rangle) = [\text{Real}(E(x_1))/x_1] \ldots [\text{Real}(E(x_n))/x_n]\, M,$$

where

$$FV(M) = \{x_1, \ldots, x_n\}.$$

It gives the term "represented" by a closure, and will be used to define Unload.

Now, Controlstrings = (Terms ∪ {ap})*, where $ap \notin$ Terms, and Stacks = Closures*.

The function FV is extended to Controlstrings by:

$$FV(ap) = \emptyset; \qquad FV(C_1 : \ldots : C_n) = \bigcup_{i=1}^{n} FV(C_i) \quad (n \geqslant 0).$$

Finally, Dumps is defined inductively by:

(1) nil is a dump.

(2) If S is a stack, E an environment, C a control string such that $FV(C) \subseteq \text{Dom}(E)$, and D is a dump then $\langle S, E, C, D\rangle$ is a dump

The transition function $\Rightarrow$, in Dumps $\xrightarrow{P}$ Dumps is defined by:

(1) $\langle \mathrm{Cl}: S, E, \mathrm{nil}, \langle S', E', C', D' \rangle \rangle \Rightarrow \langle \mathrm{Cl}: S', E', C', D' \rangle$,

(2) $\langle S, E, x: C, D \rangle \Rightarrow \langle E(x): S, E, C, D \rangle$,

(3) $\langle S, E, a: C, D \rangle \Rightarrow \langle \langle a, \emptyset \rangle: S, E, C, D \rangle$,

(4) $\langle S, E, (\lambda x M): C, D \rangle \Rightarrow \langle \langle (\lambda x M), E \rangle: S, E, C, D \rangle$,

(5) $\langle \langle (\lambda x M), E' \rangle: \mathrm{Cl}: S, E, ap: C, D \rangle \Rightarrow \langle \mathrm{nil}, E'\{\mathrm{Cl}/x\}, M, \langle S, E, C, D \rangle \rangle$,

(6) $\langle \langle a, E' \rangle : \langle b, E'' \rangle : S, E, ap: C, D \rangle \Rightarrow \langle \langle \mathrm{Constapply}(a, b), \emptyset \rangle: S, E, C, D \rangle$,

(7) $\langle S, E, (MN): C, D \rangle \Rightarrow \langle S, E, N: M: ap: C, D \rangle$.

Now Load and Unload are defined by:

$$\mathrm{Load}(M) = \langle \mathrm{nil}, \emptyset, M, \mathrm{nil} \rangle$$

$$\mathrm{Unload}(\langle \mathrm{Cl}, \emptyset, \mathrm{nil}, \mathrm{nil} \rangle) = \mathrm{Real}(\mathrm{Cl}).$$

The evaluation function can now be defined by:

$\mathrm{Eval}_V(M) = N$ iff $\mathrm{Load}(M) \overset{*}{\Rightarrow} D$, and $N = \mathrm{Unload}(D)$ for some dump D.

Example. (contd.) One can now easily define all partial recursive functions. As recursion operator one should not use $Y = \lambda f((\lambda x f(xx))(\lambda x f(xx)))$, but rather, $Z = \lambda f((\lambda x f(\lambda z x x z))(\lambda x f(\lambda z x x z)))$, [11, 19]. The point is that $\mathrm{Eval}_V(YM)$ is always undefined — as can be shown using Theorem 1 — and Z is designed to avoid this difficulty as the reader will find if he tries, say, to define addition.

In the light of Section 6, it seems quite possible that ISWIM with letrec can be translated into ISWIM without letrec.

As remarked above, this definition of Eval_V is rather too clumsy to work with directly and we prefer to use a function eval_V: Programs $\xrightarrow{P}$ Programs with a simple recursive definition, which uses substitution rather than closures. This has the informal definition:

$$\mathrm{eval}_V(a) = a; \mathrm{eval}_V(\lambda x M) = \lambda x M$$

$$\mathrm{eval}_V(MN) = \begin{cases} \mathrm{eval}_V([N'/x]\, M') \text{ (if } \mathrm{eval}_V(M) = (\lambda x M') \text{ and } \mathrm{eval}_V(N) = N') \\ \mathrm{Constapply}(a, b) \text{ (if } \mathrm{eval}_V(M) = a \text{ and } \mathrm{eval}_V(N) = b) \end{cases}$$

Formally, we define the predicate "M has value N at time t" by induction on t, for closed terms M and N.

(1) a has value a at time 1; $(\lambda x M)$ has value $(\lambda x M)$ at time 1.

(2) If M has value $(\lambda x M')$ at time t, N has value N' at time t' and $[N'/x]\, M'$ has value L at time t'', then (MN) has value L at time $t+t'+t''+1$.

(3) If M has value a at time t and N has value b at time t' then, if Constapply (a, b) is defined, (MN) has value Constapply (a, b) at time $t+t'+1$.

One then sees that if M has values N, N' at time t, t' then $N = N'$ and $t = t'$. Consequently this is a good definition of a partial function:

$\mathrm{eval}_V(M) = N$ iff M has value N at some time.

The t in the above definition will give a good handle for inductive proofs. Clearly, if $M =_\alpha M'$, then $\text{eval}_V(M)$ has value N at time t iff for some $N' =_\alpha N$, M' has value N' at time t.

It is also clear, from this definition, that, if $\text{eval}_V(M)$ exists it is a closed value, which justifies our value terminology, given the naturalness of including open values among the set of values.

The next theorem states that eval_V and Eval_V are the same functions to within alphabetic equivalence.

Theorem 1. *For any program* M, $\text{Eval}_V(M) =_\alpha \text{eval}_V(M)$.

This theorem justifies our making (after we have proved it) the mathematically convenient decision to take the definition of eval_V as the definition also of Eval_V, rather than the one via the SECD machine. Results asserted later for this definition will then also hold for the one using the SECD machine to within alphabetic equivalence which is all that really matters.

The proof of Theorem 1 requires three lemmas.

The notions of a value closure and value environment are defined inductively by:

(1) A closure $\langle M, E\rangle$ is a *value* closure iff M is an abstraction or a constant, and E is a value environment.

(2) An environment, E, is a *value* environment, iff for every variable, x, in $\text{Dom}(E)$, $E(x)$ is a value closure.

Value closures correspond fairly closely to Landin's closures.

Lemma 1. *Suppose* $\langle \lambda y M, E\rangle$ *and* $\langle N, E'\rangle$ *are value closures,* $\text{Real}(\langle \lambda y M, E\rangle) =_\alpha (\lambda x M')$ *and* $\text{Real}(\langle N, E'\rangle) =_\alpha N'$. *Then* $\text{Real}(\langle M, E\{\langle N, E'\rangle/y\}\rangle) =_\alpha [N'/x]M'$.

We omit the straightforward proof of this lemma.

Lemma 2. *Suppose* E *is a value environment and* $\langle M, E\rangle$ *is a closure and* M'' *is the value of* $\text{Real}(\langle M, E\rangle)$ *at time* t. *Then, for all* S, E, C *and* D, *with* $FV(C) \subseteq \text{Dom}(E)$ *and some* $t' \geq t$, $\langle S, E, M: C, D\rangle \overset{t'}{\Rightarrow} \langle\langle M', E'\rangle: S, E, C, D\rangle$ *where* $\langle M', E'\rangle$ *is a value closure and* $\text{Real}(\langle M', E'\rangle) =_\iota M''$.

Proof. By induction on t.

Case 1. M is a constant. Here

$$\text{Real}(\langle M, E\rangle) = M = M'' \text{ and } t = 1.$$

As

$$\langle S, E, M: C, D\rangle \Rightarrow \langle\langle M, \emptyset\rangle: S, E, C, D\rangle$$

we can take $\langle M', E'\rangle = \langle M, \emptyset\rangle$ and $t' = 1$.

Case 2. M is an abstraction. Here

$$M'' = \text{Real}(\langle M, E\rangle) \text{ and } t = 1.$$

As

$$\langle S, E, M: C, D\rangle \Rightarrow \langle\langle M, E\rangle: S, E, C, D\rangle$$

we can take $\langle M', E'\rangle = \langle M, E\rangle$ and $t' = 1$.

Case 3. M is a variable. Here

$$M'' = \text{Real}(E(M)) \text{ and } t = 1.$$

As

$$\langle S, E, M: C, D\rangle \Rightarrow \langle E(M): S, E, C, D\rangle$$

we can take $\langle M', E'\rangle = E(M)$ and $t' = 1$.

Case 4. $M = (M_1\ M_2)$ is a combination. Then

$$\text{Real}(\langle M, E\rangle) = \text{Real}(\langle M_1, E\rangle)\,\text{Real}(\langle M_2, E\rangle) = N_1\ N_2 \text{ say.}$$

Subcase 1. $(\lambda x N_3)$ is the value of N_1 at time t_1, N_4 is the value of N_2 at time t_2, M'' is the value of $[N_4/x]\,N_3$ at time t_3 and $t = t_1+t_2+t_3+1$.

Then by induction hypothesis there are $t_i' \geqslant t_i$ $(i = 1, 2)$ such that

$$\begin{aligned}\langle S, E, (M_1\ M_2): C, D\rangle &\Rightarrow \langle S, E, M_2: M_1: ap: C, D\rangle\\ &\overset{t_2'}{\Rightarrow}\langle\langle M_2', E_2'\rangle: S, E, M_1: ap: C, D\rangle\\ &\overset{t_1'}{\Rightarrow}\langle\langle M_1', E_1'\rangle: \langle M_2', E_2'\rangle: S, E, ap: C, D\rangle,\end{aligned}$$

where

$$\text{Real}(\langle M_i', E_i'\rangle) = {}_\alpha\lambda x N_3 \text{ and } \text{Real}(\langle M_2', E_2'\rangle) = {}_\alpha N_4,$$

and the $\langle M_i', E_i'\rangle$ are value closures.

Here $M_1' = \lambda y M_3'$ for some M_3', and

$$\text{Real}(\langle M_3', E_1'\,\{\langle M_2', E_2'\rangle/y\}\rangle) = {}_\alpha[N_4/x]\,N_3 \text{ (Lemma 1).}$$

Now,

$$\begin{aligned}\langle\langle M_1', E_1'\rangle: \langle M_2', E_2'\rangle: S, E, ap: C, D\rangle &\Rightarrow \langle \text{nil}, E_1'\,\{\langle M_2', E_2'\rangle/y\},\\ &\qquad M_3', \langle S, E, C, D\rangle\rangle\\ &\overset{t_3'}{\Rightarrow}\langle\langle M', E'\rangle, E_1'\,\{\langle M_2', E_2'\rangle/x\},\\ &\qquad \text{nil}, \langle S, E, C, D\rangle\rangle\\ &\Rightarrow \langle\langle M', E'\rangle: S, E, C, D\rangle\rangle\end{aligned}$$

where, by the induction hypothesis, $\text{Real}(\langle M', E'\rangle)$ is to within alphabetic equivalence the value of $\text{Real}(\langle M_3', E_1'\,\{\langle M_2', E_2'\rangle/y\}\rangle)$ at time $t_3 \leqslant t_3'$ and $\langle M', E'\rangle$ is a value closure. Taking $t' = t_1'+t_2'+t_3'+3$ concludes this subcase.

Subcase 2. a is the value of N_1 at time t_1, b is the value of N_2 at time t_2, and Constapply $(a, b) = M''$ and $t = t_1+t_2+1$.

By induction hypothesis there are $t_i' \geqslant t_i$ $(i = 1, 2)$ and value environments

E_i' $(i = 1, 2)$ such that

$$\langle S, E, (M_1\ M_2): C, D\rangle \Rightarrow \langle S, E, M_2: M_1: ap: C, D\rangle$$
$$\overset{t_2'}{\Rightarrow}\langle\langle b, E_2'\rangle: S, E, M_1: ap: C, D\rangle$$
$$\overset{t_1'}{\Rightarrow}\langle\langle a, E_1'\rangle: \langle b, E_2'\rangle: S, E, ap: C, D\rangle$$
$$\Rightarrow \langle\langle M'', \emptyset\rangle: S, E, C, D\rangle$$

and taking $t' = t_1'+t_2'+2$ and $\langle M', E'\rangle = \langle M'', \emptyset\rangle$ concludes the proof.

If $D \overset{t}{\Rightarrow} D'$ where D' does not have the form $\langle \mathrm{Cl}, \emptyset, \mathrm{nil}, \mathrm{nil}\rangle$ and $D' \not\Rightarrow D''$ for any D'', then D is said to *hit an error state* (viz. D').

Lemma 3. *Suppose E is a value environment and $\langle M, E\rangle$ is a closure. If* Real $(\langle M, E\rangle)$ *has no value at any $t' \leqslant t$, then either for all S, C, D, with $FV(C) \subseteq$ Dom (E) $\langle S, E, M: C, D\rangle$ hits an error state or else $\langle S, E, M: C, D\rangle \overset{t}{\Rightarrow} D'$ for some D'. $(t \geqslant 1)$.*

Proof. By induction on t. For $t = 1$, the result is obvious. Otherwise, Real $(\langle M, E\rangle)$, and so M, must be a combination, $(M_1\ M_2)$, say.
Then

$$\langle S, E, (M_1M_2): C, D\rangle \Rightarrow \langle S, E, M_2: M_1: ap: C, D\rangle.$$

If Real $(\langle M_2, E\rangle)$ has no value at any time $\leqslant (t-1)$, the result follows by applying the induction hypothesis to $\langle M_2, E\rangle$.

Otherwise, suppose M_2'' is the value of Real $(\langle M_2, E\rangle)$ at time $t_2 \leqslant (t-1)$. By Lemma 1,

$$\langle S, E, M_2: M_1: ap: C, D\rangle \overset{t_2'}{\Rightarrow} \langle\langle M_2', E_2'\rangle: S, E, M_1: ap: C, D\rangle$$

where $t_2' \geqslant t_2$, $\langle M_2', E_2'\rangle$ is a value closure, and Real $(\langle M_2', E_2'\rangle) = {}_\alpha M_2''$. If $t_2' \geqslant (t-1)$ we are finished and so we may suppose that $t_2' < (t-1)$.

If Real $(\langle M_1, E\rangle)$ has no value at any time $\leqslant (t-1-t_2')$ the result follows by applying the induction hypothesis to $\langle M_1, E\rangle$.

Otherwise, suppose M_1'' is the value of Real $(\langle M_1, E\rangle)$ at time $t_1 \leqslant (t-1-t_2')$. By Lemma 1,

$$\langle\langle M_2', E_2'\rangle: S, E, M_1: ap: C, D\rangle \overset{t_1'}{\Rightarrow} \langle\langle M_1', E_1'\rangle: \langle M_2', E_2'\rangle: S, E, ap: C, D\rangle,$$

where $t_1' \geqslant t_1$, $\langle M_1', E_1'\rangle$ is a value closure and Real $(\langle M_1', E_1'\rangle) = {}_\alpha M_1''$. If $t_1' \geqslant \geqslant (t-1-t_2')$ we are finished and so we may suppose that $t_1' < (t-1-t_2')$. The argument now splits into three cases.

Case 1. $M_1' = (\lambda x M_3)$.

Now,

$$\langle\langle \lambda x M_3, E_1'\rangle: \langle M_2', E_2'\rangle: S, E, ap: C, D\rangle \Rightarrow$$
$$\langle \mathrm{nil}, E_1'\{\langle M_2', E_2'\rangle/x\}, M_3, \langle S, E, C, D\rangle\rangle.$$

If $t = (t_1'+t_2'+2)$ we are finished.

Supposing otherwise and letting $E_3' = E_1'\{\langle M_2', E_2'\rangle/x\}$, if Real $(\langle M_3', E_3'\rangle)$ has no value at any time $\leqslant (t-t_1'-t_2'-2)$, the result follows from the induction hypothesis. Otherwise, if it has a value at such a time, that value must be to within alphabetic equivalence the value of Real $(\langle M, E\rangle)$ (as in the proof of Lemma 2), which therefore has a value at time $t_2+t_1+t_3+2 \leqslant t_2'+t_1'+(t-t_1'-t_1'-2)+2 = t$ contradicting the original assumption.

Case 2. M_1' and M_2' are constants and Constapply (M_1', M_2') is defined. Here Real $(\langle M, E\rangle)$ has a value a time $t_1+t_2+2 \leqslant (t_1'+t_2'+2) \leqslant t$, contradicting the original assumption.

Case 3. For all other possibilities for M_1' and M_2' an error stop occurs.

Proof of Theorem 1. Suppose $\text{eval}_V(M) = M''$. Then, at some time t, M'' is the value of M at time t. By Lemma 2,

$$\langle \text{nil}, \emptyset, M, \text{nil}\rangle \overset{+}{\Rightarrow} \langle\langle M', E'\rangle, \emptyset, \text{nil}, \text{nil}\rangle,$$

where Real $(\langle M', E'\rangle) =_\alpha M''$. So $\text{Eval}_V(M) =_\alpha M''$.

Suppose, on the other hand, that M has no value at any time. Then by Lemma 3, either $\langle \text{nil}, \emptyset, M, \text{nil}\rangle$ hits an error state or else for every t there is a D such that $\langle \text{nil}, \emptyset, M, \text{nil}\rangle \overset{t}{\Rightarrow} D$. In either case $\text{Eval}_V(M)$ is also not defined, concluding the proof.

4. The λ_V calculus

A suitable λ_V calculus is obtained by simply restricting the β rule in the λK-$\beta\delta$ calculus induced by Constapply. Explicitly the λ_V calculus has formulae of the form $M = N$ where M and N are terms. Given a function Constapply, its rules are as follows:

I1. $(\lambda xM) = (\lambda y\,[y/x]\,M)\,(y \notin FV(M))$. ($\alpha$-rule)

2. $(\lambda xM)\,N = [N/x]\,M$ (if N is a value). (β-rule)

3. $(ab) =$ Constapply (a, b) (if this is defined). (δ-rule)

II1. $M = M$

2. $$\frac{M = N \quad N = L}{M = L}$$

3. $$\frac{M = N}{N = M}$$

III1. $$\frac{M = N}{MZ = NZ}, \quad \frac{M = N}{ZM = ZN}$$

2. $$\frac{M = N}{\lambda xM = \lambda xN}$$

$\lambda_V \vdash M = N$ means that $M = N$ is provable by the above rules.

$\lambda_V \vdash M \geqslant N$ means that $M = N$ is provable by the above rules without using II3.

In [4] Goodman develops what seems to be the corresponding call-by-value combinatory logic.

First we develop some elementary theory for the λ_V calculus.

Theorem 1. *If $\lambda_V \vdash M = N$ and L is a value then $\lambda_V \vdash [L/x]\,M = [L/x]\,N$.*

Proof. By induction on the number of steps in the proof of $M = N$. The proof splits into cases according to the last rule used in the proof of $M = N$, and is similar to the proof of Theorem 3 in § 3.E of [2].

The supposition that L is a value is necessary. For example, if $L = (\lambda xxx)\,(\lambda xxx)$, $M = (\lambda x\,(\lambda xx))\,(x)$ and $N = (\lambda xx)$ then $\lambda_V \vdash M = N$, by a β-reduction, but it is not the case that $\lambda_V \vdash [L/x]\,M = [L/x]\,N$, as can be seen from the Church–Rosser theorem (to be proved) as $[L/x]\,M$ can reduce only to an α-equivalent term and $[L/x]\,N = N$ is in λ_V normal form (i. e. has no call-by-value β-redex or δ-redex (see below)) and is not α-equivalent to $[L/x]\,M$. So free variables should be thought of as ranging over values and not arbitrary terms; in a model one would expect that they would be interpreted as being universally quantified over a restricted domain.

Theorem 2. (Church–Rosser theorem). *If $\lambda_V \vdash M_1 \geqslant M_i$ $(i = 2, 3)$ then for some M_4, $\lambda_V \vdash M_i \geqslant M_4$ $(i = 2, 3)$.*

Proof. The main tool is the parallel reduction relation, $\geqslant_1$, defined below. It is easily shown that $\lambda_V \vdash M \geqslant N$ iff there are $N_1, \ldots, N_n$ such that $M = N_1$, $\lambda_V \vdash N_i \geqslant {}_1N_{i+1}$ $(1 \leqslant i \leqslant (n-1))$ and $N_n = {}_\alpha N$. Then using Lemma 5, a straightforward case analysis proves that if $M_1 \geqslant {}_1M_i$ $(i = 1, 2)$ then for some M_4, M_5, $M_2 \geqslant {}_1M_4$, $M_3 \geqslant {}_1M_5$ and $M_4 = {}_\alpha M_5$. Then the theorem follows by a simple induction. This is the method of Tait [1]. The details are both routine and omitted.

As usual this theorem has the consequence that $\lambda_V \vdash M = N$ iff there is a Z such that $\lambda_V \vdash M \geqslant Z$ and $\lambda_V \vdash N \geqslant Z$. A call-by-value normal form is a term with no call-by-value β-redex and no δ-redex, where such a β-redex is a term of the form $(\lambda xM)\,N$, with N a value and a δ-redex is a term of the form (ab) with Constapply (a, b) defined. It is easy to see that if $\lambda_V \vdash M \geqslant N$ and M is in call-by-value normal form then $M = {}_\alpha N$.

Alterations to λ_V generally result in the Church–Rosser theorem failing. This happens if, for example, one changes the definition of value to normal form, in the usual λK-$\beta\delta$ sense, as would be natural with the kind of reduction sequence used by McGowan [8]. Then the the lemma analogous to Lemma 5 below fails as the property of having a normal form is not preserved under substitution. For

a counterexample to the Church-Rosser theorem take $M_1 = (\lambda x\,(\lambda yz)\,(x\,(\lambda xxx)))\,(\lambda xxx)$, $M_2 = z$ and $M_3 = (\lambda yz)\,((\lambda xxx)\,(\lambda xxx))$.

Similarly if an η rule is added the theorem fails as the rule does not preserve the property of being a value and so the appropriate version of Lemma 5 does not hold. A counterexample to the Church-Rosser theorem is given by taking $M_1 = (\lambda xy)\,(\lambda x\,((\lambda xxx)\,(\lambda xxx))\,x)$, $M_2 = y$ and $M_3 = (\lambda xy)\,((\lambda xxx)\,(\lambda xxx))$.

Next comes our analogue of the Curry standardisation theorem. This gives a normal order reduction sequence and ties in with Eval_V. To this end, definitions of left reduction, parallel reduction and standard reduction sequences (s.r. sequences) are needed. Our proof will avoid explicit mention of redexes.

Left reduction, $\underset{V}{\rightarrow}$ is the least relation between terms such that:

1. $(\lambda x M)\,N \underset{V}{\rightarrow} [N/x]\,M$, (when N is a value).
2. $(ab) \underset{V}{\rightarrow}$ Constapply (a, b) (when defined).
3. If $M \underset{V}{\rightarrow} M'$ then $(MN) \underset{V}{\rightarrow} (M'N)$.
4. If $M \underset{V}{\rightarrow} M'$ then $(NM) \underset{V}{\rightarrow} (NM')$, when N is a value.

Note that $\underset{V}{\rightarrow}$ is a partial function and if $M \underset{V}{\rightarrow} N$ then M is not a value. Informally if $M \underset{V}{\rightarrow} N$ then N is gotten from M by reducing the leftmost redex, not in the scope of a λ.

Lemma 1. *If* $M =_\alpha M' \underset{V}{\rightarrow} N'$, then there is an N such that $M \underset{V}{\rightarrow} N =_\alpha N'$.

We omit the proof, which is quite straightforward.

To define parallel reduction, we use a little formal system, whose formulae have the form $M \geqslant_1 N$, where M and N are terms and whose rules are:

II. $\dfrac{M \geqslant_1 M'\ N \geqslant_1 N'}{(\lambda x M)\,N \geqslant_1 [N'/x]\,M'}$ (if N is a value)

2. $(ab) \geqslant_1$ Constapply (a, b) (when defined)

III. $M \geqslant_1 M$

2. $\dfrac{M \geqslant_1 M'}{(\lambda x M) \geqslant_1 (\lambda x M')}$

3. $\dfrac{M \geqslant M'\ N \geqslant N'}{(MN) \geqslant_1 (M'N')}$

$\lambda'_V \vdash M \geqslant_1 N$ means that $M \geqslant_1 N$ can be proved using the above rules; when no confusion arises we just write $M \geqslant_1 N$ instead.

We introduce a size measure on proofs, corresponding to the "implicit" number of β and δ reductions. The definition is by induction on the evident "number of proof steps" measure and is divided into cases according to the last rule applied. We set $n(x, M)$ to be the number of free occurrences of x in M.

I1. If the proof of $M \geqslant {}_1M'$ has reduction size p_M and that of $N \geqslant {}_1N'$, p_N then the proof has reduction size $p_M + n(x, M')\,p_N + 1$.

I2. The proof has reduction size 1.

II1. The proof has reduction size 0.

II2. The proof has the same reduction size as the proof of $M \geqslant {}_1M'$.

II3. If the proof of $M \geqslant {}_1M'$ has reduction size p_M and of $N \geqslant {}_1N'$ has reduction size p_N, the proof has reduction size $p_M + p_N$.

Note that if a proof of $M \geqslant {}_1N$ has reduction size 0 then $M = N$.

Lemma 2. *If* $M = {}_\alpha M' \geqslant {}_1N'$ *then there is an* N *such that* $M \geqslant {}_1N = {}_\alpha N'$.

Lemma 3. $\lambda_V \vdash M \geqslant N$ *iff there are* $M_1, \ldots, M_k$ *such that* $M = M_1 \geqslant {}_1 \ldots \geqslant {}_1 M_k = {}_\alpha N$.

We omit the quite straightforward proofs of these lemmas.

Standard reduction sequences (s.r. sequences) are defined inductively by:

1. Any variable, x, or constant, a, is a s.r. sequence.
2. If $N_2, \ldots, N_k$ is a s. r. sequence and $N_1 \underset{V}{\rightarrow} N_2$ then $N_1, \ldots, N_k$ is a s. r. sequence.
3. If $N_1, \ldots, N_k$ is a s.r. sequence, so is $(\lambda x N_1), \ldots, (\lambda x N_k)$.
4. If $M_1, \ldots, M_j$ and $N_1, \ldots, N_k$ are s.r. sequences, so is $(M_1\,N_1), \ldots, (M_j\,N_1), \ldots, (M_j\,N_k)$.

Lemma 4. *If* $N_1, \ldots, N_k$ *is a* s.r. *sequence and* $M_1 = {}_\alpha N_1$, *then there is a* s.r. *sequence* $M_1, \ldots, M_k$ *such that* $M_k = {}_\alpha N_k$.

Again, the proof is both straightforward and omitted.

We are aiming to prove:

Theorem 3. (Standardisation theorem) $\lambda_V \vdash M \geqslant N$ *iff there is a* s.r. *sequence* $N_1, \ldots, N_k$ *such that* $M = N_1$ *and* $N_k = {}_\alpha N$.

Lemma 5. *If there is a proof of* $M \geqslant {}_1M'$ *of reduction size* p_M *and of* $N \geqslant {}_1N'$ *of reduction size* p_N *where* N *is a value then there is one of* $[N/x]\,M \geqslant {}_1L$, *where* $L = {}_\alpha [N'/x]\,M'$ *of reduction size* $\leqslant p_M + n(x, M')\,p_N$.

Proof. By induction on the size of M and by cases according to the last rule applied in the proof.

I1. Here $M = (\lambda y M_1)\,M_2$, there is a proof of $M_1 \geqslant M_1'$ with reduction size p_{M_1}, one of $M_2 \geqslant {}_1M_2'$ of reduction size p_{M_2}, $p_M = p_{M_1} + n(y, M_1')\,p_{M_2} + 1$ and $M' = [M_2'/y]\,M_1'$.

Subcase 1. $x = y$. By induction hypothesis, there is a proof of $[N/x]\,M_2 \geqslant {}_1L_2$ of reduction size $\leqslant p_{M_2}+n(x, M_2')\,p_N$ where $L_2 = {}_\alpha[N'/x]\,M_2'$.

Hence there is a proof of $[N/x]\,M \geqslant {}_1\,[L_2/x]\,M_1'$ of reduction size

$$\leqslant p_{M_1}+n(x, M_1')\,(p_{M_2}+n(x, M_2')\,p_N)+1 = p_M\ +n(x, M_1')\,n(x, M_2')\,p_N$$
$$= p_M+n(x, [M_2'/x]\,M_1')\,p_N = p_M+n(x, M')\,p_N.$$

Finally,

$$\begin{aligned}[N'/x]\,M' &= [N'/x]\,[M_2'/x]\,M_1'\\ &= {}_\alpha\,[[N'/x]\,M_2'/x]\,M_1'\\ &= {}_\alpha\,[L_2/x]\,M_1'.\end{aligned}$$

so we can take $L = [L_2/x]\,M_1'$.

Subcase 2. $x \neq y$. Let z be the variable such that either $y \notin FV(N)$ and $z = y$ or, otherwise $x \notin FV(M_1)$ and $z = y$ or, otherwise, z is the first variable in the list $x_1, x_2, \ldots$ such that $z \notin FV(N) \cup FV(M_1)$. One easily sees from the induction hypothesis that for some L_0, $[z/y]\,M_1 \geqslant {}_1L_0$ has a proof of reduction size p_{M_1}, where $L_0 = {}_\alpha\,[z/y]\,M_1'$. Therefore, by another application of the induction hypothesis, for some L_1, $[N/x]\,[z/y]\,M_1 \geqslant {}_1L_1$ has a proof of reduction size $\leqslant p_{M_1}+n(x, L_0)\,p_N$ where $L_1 = {}_\alpha[N'/x]\,[z/y]\,M_1'$.

Next, by the induction hypothesis, for some L_2 there is a proof of $[N/x]\,M_2 \geqslant {}_1L_2$ of reduction size $\leqslant p_{M_2}+n(x, M_2')\,p_N$ where $L_2 = {}_\alpha[N'/x]\,M_2'$.

Putting all this together, we find that $(\lambda z\,[N/x]\,[z/y]\,M_1)\,([N/x]\,M_2) \geqslant {}_1[L_2/z]\,L_1$ has a proof of reduction size

$$p \leqslant (p_{M_1}+n(x, L_0)\,p_N)+n(z, L_1)\,(p_{M_2}+n(x, M_2')\,p_N)+1.$$

But

$$[N/x]\,M = (\lambda z\,[N/x]\,[z/y]\,M_1)\,([N/x]\,M_2),$$
$$\begin{aligned}[L_2/z]\,L_1 &= {}_\alpha[[N'/x]\,M_2'/z]\,[N'/x]\,[z/y]\,M_1'\\ &= {}_\alpha[N'/x]\,[M_2'/z]\,[z/y]\,M_1\\ &= {}_\alpha[N'/x]\,[M_2'/y]\,M_1'\\ &= {}_\alpha[N'/x]\,M'.\end{aligned}$$

and

$$\begin{aligned}&p_{M_1}+n(x, L_0)\,p_N+n(z, L_1)\,(p_{M_2}+n(x, M_2')\,p_N)+1\\ &= (p_{M_1}+n(z, L_1)\,p_{M_2}+1)+(n(x, L_0)+n(z, L_1)\,n(x, M_2'))\,p_N\\ &= (p_{M_1}+n(z, [N'/x]\,[z/y]\,M_1')p_{M_2}+1)+(n(x, [z/y]\,M_1')+n(z, L_1)\times\\ &\quad\times n(x, M_2'))\,p_N\\ &= (p_{M_1}+n(z, [z/y]\,M_1')\,p_{M_2}+1)+(n(x, M_1')+n(z, L_1)\,n(x, M_2'))\,p_N\\ &= (p_{M_1}+n(y, M_1')\,p_{M_2}+1)(n(x, M_1')+n(y, M_1')\,n(x, M_2'))\,p_N\\ &= p_M+n(x, [M_2'/y]\,M_1')\,p_N\\ &= p_M+n(x, M')\,p_N.\end{aligned}$$

Taking $L = [L_2/z]\, L_1$, this subcase is concluded.

I2. Here $M = (a, b)$, $M' =$ Constapply (a, b), and since $FV(M') = \emptyset$, the conclusion is immediate.

II1. Here $M' = M$ and $p_M = 0$. We proceed by induction on the structure of M'.

Subcase 1. $M \neq x$ and M is a variable or constant. Immediate.

Subcase 2. $M = x$. Immediate with $L = N' = [N'/x]\, x$ as then $n(x, M) = 1$.

Subcase 3. $M = (M_1\ M_2)$. By induction hypothesis, there are for some L_i proofs of

$$[N/x]\, M_i \geqslant {}_1L_i = {}_\alpha[N'/x]\, M_i$$

of reduction size

$$\leqslant n(x, M_i)\, p_N \quad \text{for } i = 1, 2.$$

The conclusion is immediate, taking $L = (L_1\ L_2)$.

Subcase 4. $M = (\lambda y M_1)$. If $y = x$ the conclusion is immediate. Otherwise, with z as in case I1, subcase 2, there is by induction hypothesis, a proof for some L_1 of

$$[N/x]\, [z/y]\, M_1 \geqslant {}_1L_1 = {}_\alpha[N/x]\, [y/z]\, M_1$$

of reduction size

$$\leqslant n(x, M_1')\, p_N = n(x, M)\, p_N.$$

The conclusion follows, taking $L = (\lambda z L_1)$.

II2. Here $M = (\lambda y M_1)$, $M' = (\lambda y M_1')$ and $M_1 \geqslant {}_1M_1'$ has a proof of reduction size p_M. If $y = x$, the conclusion is immediate. Otherwise the proof is like II1, subcase 4.

II3. Here $M = (M_1\ M_2)$, $M' = (M_1'\ M_2')$, and $M_i \geqslant M_i'$ has a proof of reduction size p_{M_i} for $i = 1, 2$. So, by induction hypothesis, there are proofs for some L_i of

$$[N/x]\, M_i \geqslant {}_1L_i = {}_\alpha[N'/x]\, M_i'$$

of reduction size

$$\leqslant p_{M_i} + n(x, M_i')\, p_N \quad (i = 1, 2).$$

Taking $L = (L_1\ L_2)$, we find a proof of

$$[N/x]\, M \geqslant {}_1L = {}_\alpha[N'/x]\, M'$$

of reduction size

$$\leqslant (p_{M_1} + n(x, M_1')\, p_N) + (p_{M_2} + n(x, M_2')\, p_N) = p_M + n(x, M')\, p_N.$$

This concludes the proof.

Lemma 6. *If $M \geqslant {}_1N$ where M is a combination and N is a constant or variable then $M \xrightarrow[v]{+} N$.*

Proof. By induction on the reduction size p_M of the proof of $M \geqslant {}_1N$. If the last rule applied in the proof is I2 the result is immediate, otherwise it must be I1 and

so $M = (\lambda x M_1)\, M_2$, $M_i \geqslant {}_1M_i'$ has a proof of reduction size p_{M_i} $(i = 1, 2)$, $N = [M_2'/x]\, M_1'$ and $p_M = p_{M_1} + n(x, M_1')\, p_{M_2} + 1$.

By Lemma 4, for some L, there is a proof of $[M_2/x]\, M_1 \geqslant {}_1L$ of reduction size

$$\leqslant p_{M_1} + n(x, M_1')\, p_{M_2} < p_M,$$

where $L = {}_\alpha[M_2'/x]\, M_1' = N$ (and so $L = N$). Hence,

$$M \underset{V}{\rightarrow} [M_2/x]\, M_1 \overset{+}{\underset{V}{\rightarrow}} N \quad \text{(by induction hypothesis)},$$

if $[M_2/x]\, M_1$ is a combination, and otherwise $[M_2/x]\, M_1 = N$ when the result is immediate.

Lemma 7. *If $M \geqslant {}_1M'$ has a proof of reduction size p_M, where M is a combination and M' is an abstraction, then for some abstractions L, L', $M \overset{+}{\underset{V}{\rightarrow}} L$ and $L \geqslant {}_1L'$ has a proof of reduction size $< p_M$ and $L' = {}_\alpha M'$.*

Proof. By induction on the reduction size, p_M, of the proof of $M \geqslant {}_1M'$. If the last rule applied in this proof is I2 we are finished, otherwise it must be I1, then much as in the proof of Lemma 5, we find an N, L'' such that $M \underset{V}{\rightarrow} N$, $N \geqslant {}_1L''$ has a proof of reduction size $< p_M$ and $L' = {}_\alpha M'$. If N is an abstraction, we are finished. Otherwise, by induction hypothesis, there are L, L' such that $N \overset{+}{\underset{V}{\rightarrow}} L$, $L \geqslant {}_1L'$ has a proof of reduction size $< p_M$ and $L' = {}_\alpha L'' = {}_\alpha M'$, concluding the proof.

Lemma 8. *If $M \geqslant {}_1M' \underset{V}{\rightarrow} M''$, then there are K, K' such that $M \overset{+}{\underset{V}{\rightarrow}} K \geqslant {}_1K' = {}_\alpha M''$.*

Proof. By lexicographic induction on $\langle p_M, |M| \rangle$ where p_M is the reduction size of the proof of $M \geqslant {}_1M'$ and divided into cases according to the last rule used in that proof.

I1. This case is straightforward and is quite similar to that of Lemma 6.

I2. This case is impossible.

II1. This case is trivial.

II2. This case is impossible.

II3. Here $M = (M_1\, M_2)$, $M' = (M_1'\, M_2')$, $M_i \geqslant M_i'$ has a proof of reduction size p_{M_i}, for $i = 1, 2$ and $p_M = p_{M_1} + p_{M_2}$. The proof divides into cases according to why $M' \underset{V}{\rightarrow} M''$.

1. Here $M' = (\lambda x M_3')\, M_2'$, $M'' = [M_2'/x]\, M_3'$ and M_2' is a value.

Since $M_1 \geqslant_1 (\lambda x M_3')$ we find in all cases, using Lemma 7, abstractions N and N' such that $M_1 \overset{*}{\underset{V}{\rightarrow}} N \geqslant {}_1N' = {}_\alpha M_1'$.

Since $M_2 \geqslant {}_1M_2'$, a value, we find in all cases, using Lemmas 6 and 7, values L and L' such that $M_2 \overset{*}{\underset{V}{\rightarrow}} L \geqslant {}_1L' = {}_\alpha M_2'$. It can be seen that putting $N = \lambda y N_1$, for some K', $NL \underset{V}{\rightarrow} [L/y]\, N_1 \geqslant {}_1K' = {}_\alpha M''$.

Hence, taking $[L/y]\,N_1 = K$.

$$(M_1\,M_2) \xrightarrow[V]{*} (NM_2) \xrightarrow[V]{*} (NL) \xrightarrow[V]{} K \geqslant {}_1K' = {}_\alpha M'',$$

as required.

2. Here for some constants a and b. $M_1' = a$ and $M_2' = b$.

So by Lemma 6,

$$(M_1\,M_2) \xrightarrow[V]{*} (aM_2) \xrightarrow[V]{*} (ab) \xrightarrow[V]{} \text{Constapply } (a, b),$$

and we can take $K = K' = M''$.

3. Here $M_1' \xrightarrow[V]{} M_1''$ and $M'' = (M_1''\,M_2)$.

By induction hypothesis there are K_1, K_1' such that

$$M_1 \xrightarrow[V]{+} K_1 \geqslant {}_1K_1' = {}_\alpha M_1''.$$

Therefore $(M_1\,M_2) \xrightarrow[V]{+} (K_1\,M_2) \geqslant {}_1(K_1'\,M_2') = {}_\alpha M_2'$ and taking $K = (K_1\,M_2)$, $K' = (K_1'\,M_2')$ concludes this case.

4. Here, M_1' is a value, $M_2' \xrightarrow[V]{} M_2''$ and $M'' = (M_1'\,M_2'')$. By Lemmas 6 and 7 there are values L, L' such that

$$M_1 \xrightarrow[V]{*} L \geqslant {}_1L' = {}_\alpha M_1'.$$

By induction hypothesis there are K_2, K_2' such that

$$M_2 \xrightarrow[V]{+} K_2 \geqslant {}_1K_2' = {}_\alpha M_2''.$$

Therefore,

$$(M_1\,M_2) \xrightarrow[V]{*} (LM_2) \xrightarrow[V]{+} (LK_2) \geqslant {}_1(L'K_2') = {}_\alpha M'',$$

and taking $K = (LK_2)$ and $K' = (L'K_2')$ concludes the proof.

Lemma 9. *If $M_1, \ldots, M_j$ is a reduction sequence and $M \geqslant {}_1M_1$ then there is a* s.r. *sequence $N_1 \ldots N_k$ such that $M = N_1$ and $N_k = {}_\tau M_j$.*

Proof. By lexicographic induction on $\langle j, p_M, |M| \rangle$ (where p_M is the reduction size of the proof of $M \geqslant {}_1M_1$) and by cases on the last rule used in the proof.

I1. Here $M = (\lambda x K_1)\,K_2$, $K_i \geqslant {}_1K_i'$ has a proof of reduction size p_{M_i}, for $i = 1, 2$, $M_1 = [K_2'/x]\,K_1'$ and $p_M = p_{M_1} + n(x, K_2')\,p_{M_2} + 1$.

Now for some L, $[K_2/x]\,K_1 \geqslant {}_1L$ has a proof of reduction size $\leqslant p_{M_1} + n(x, K_2) \times \times p_{M_2} < p_M$ and $L = {}_\alpha M_1$.

By Lemma 4, there is a reduction sequence $L_1, \ldots, L_j$ such that $L = L_1$ and $L_j = {}_\alpha M_j$. By induction hypothesis, there is a s. r. sequence $N_2, \ldots, N_k$ such that $N_2 = [K_2/x]\,K_1$, and $N_k = {}_\alpha L_j = {}_\alpha M_j$. The result follows immediately, taking $N_1 = M$.

I2. Here one takes $N_1 = M$, $N_i = M_{i-1}$ $(i = 2, \ldots, j+1)$ and $k = j+1$.

II1. Take $j = k$ and $N_i = M_i$.

II2. Here there is a s. r. sequence $M_1', \ldots, M_j'$ and $M_i = (\lambda x M_i')$ (for $i = 1, j$) and for some M', $M = \lambda x M'$ and $M' \geqslant {}_1M'$.

By induction hypothesis, there is a s.r. sequence $N_1' \ldots N_k'$ such that $M' = N_1'$ and $N_k' = {}_\alpha M_j'$. The result follows with $N_i = (\lambda x N_i')$ $(i = 1, k)$.
II3. There are two subcases; either $M_2, \ldots, M_j$ is a s.r. sequence and $M_1 \underset{V}{\to} M_2$ or else there are s.r. sequences $K_1 \ldots K_k, L_1 \ldots L_l$ such that $M_1, \ldots, M_j$ is just $(L_1\ K_1), \ldots, (L_l\ K_1), \ldots, (L_l\ K_k)$. In the first subcase, the proof is similar to that in Case II1, using Lemma 8 and then Lemma 4 and the induction hypothesis. In the second subcase, the proof is similar to that in Case II2, but uses the induction hypothesis twice.

Proof of Theorem 3. Clearly if there is a s.r. sequence $N_1, \ldots, N_k$ such that $M = N_1$ and $N_k = {}_\alpha N$, $\lambda_V \vdash M \geqslant N$. Conversely, suppose $\lambda_V \vdash M \geqslant N$, then by Lemma 3, there are $L_1 \ldots L_l$ such that $M = L_1 \geqslant_1 \ldots \geqslant_l L_l = {}_\alpha N$. We proceed by induction on l.

If $l = 1$, we are finished. Otherwise, there is a s.r. sequence $K_2 \ldots K_k$ such that $L_2 = K_1$ and $K_k = {}_\alpha L_l = {}_\alpha N$. As $M \geqslant {}_1 K_1$, the result follows at once from Lemma 9.

One can now define a normal order reduction sequence and show that this reaches a (call-by-value) normal form, iff one exists, just as in [2]. It is more relevant, however, to note this corollary:

Corollary 1. *For any term M, $\lambda_V \vdash M \geqslant N$ for some value N iff $M \underset{V}{\overset{*}{\to}} N'$ for some value N'.*

Proof. Clearly if $M \underset{V}{\overset{*}{\to}} N'$ for some value N' then $\lambda_V \vdash M \geqslant N'$. In the other direction suppose $\lambda_V \vdash M \geqslant N$ for a value N. By the Standardisation Theorem there is a s.r. sequence $N_1 \ldots, N_k$ such that $M = N_1$ and $N_k = N'$. Let N' be the first value in the sequence. Then $M \underset{V}{\overset{*}{\to}} N'$ as required.

The next theorem, by tying Eval_V in with $\underset{V}{\to}$, allows us to see the connections between λ_V and Eval_V.

Theorem 4. *$\mathrm{Eval}_V(M) = N$ iff $M \underset{V}{\overset{*}{\to}} N$, (for closed M and a value N).*

Proof. First, suppose M has value N at time t. If M is a constant or an abstraction, the result is immediate. Otherwise, $M = (M_1\ M_2)$ and either M_1 and M_2 have values a and b at times t_1 and t_2, respectively, $N = \mathrm{Constapply}\ (a, b)$ and $t = t_1 + t_2 + 1$, or, otherwise, M_1 has value $(\lambda x M_1')$ at time t_1, M_2 has value M_2' at time t_2, $[M_2'/x]\ M_1'$ has value N at time t_3 and $t = t_1 + t_2 + t_3 + 1$. In the first case, by induction hypothesis, $M_1 \underset{V}{\overset{*}{\to}} a$, $M_2 \underset{V}{\overset{*}{\to}} b$ and so $M \underset{V}{\overset{*}{\to}} (aM_2) \underset{V}{\overset{*}{\to}} (ab) \underset{V}{\to} N$. In the second case, by the induction hypothesis,

$$M_1 \underset{V}{\overset{*}{\to}} (\lambda x M_1'),\ M_2 \underset{V}{\overset{*}{\to}} M_2',\ [M_2'/x]\ M_1' \overset{*}{\to} N \text{ and so } M \underset{V}{\overset{*}{\to}}$$
$$(\lambda x M_1')\ M_2' \underset{V}{\to} [M_2'/x]\ M_1' \underset{V}{\overset{*}{\to}} N.$$

Second, suppose that $M \xrightarrow[V]{*} N$. We proceed by induction on the (unique) n such that $M \xrightarrow[V]{n} N$. The case $n = 0$ is easy; so suppose $n > 0$ and there is a sequence $(K_1\ L_1), \ldots, (K_n\ L_n),\ N$ where $M = (K_1\ L_1)$, $(K_i\ L_i) \xrightarrow[V]{} (K_{i+1}\ L_{i+1})$ (for $i = 1, n$) and $(K_n\ L_n) \xrightarrow[V]{} N$. Let K_{n_0} be the first value in the sequence $K_1, \ldots, K_n$ — there must be one since N is a value and so K_n must be one too. Then $L_i = L_1$ (if $1 \leqslant i \leqslant n_0$), and $K_1 \xrightarrow[V]{*} K_{n_0}$. Next let L_{n_1} be the first value in the sequence $L_{n_0}, \ldots, L_n$ — certainly L_n is a value, so L_{n_1} exists. Then $L_{n_0} \xrightarrow[V]{*} L_{n_1}$.

If $K_{n_0} = (\lambda x K'_{n_0})$ then if $n_0 \leqslant i \leqslant n_1$, $K_i = K_{n_0}$, and, further, $[L_{n_1}/x]\ K'_{n_0} \xrightarrow[V]{*} N$. Then by the induction hypothesis

$$\mathrm{Eval}_V(K_1) = (\lambda x K'_{n_0}),\ \mathrm{Eval}_V(L_{n_0}) = L_{n_1} \text{ and } \mathrm{Eval}_V([L_{n_1}/x]\ K'_{n_0}) = N.$$

Therefore,

$$\mathrm{Eval}_V(M) = N.$$

If, on the other hand, K_{n_0} is a constant, so is L_{n_1}, $n = n_1$, and $N = \mathrm{Constapply}\ (K_n, L_n)$. Then, by the induction hypothesis,

$$\mathrm{Eval}_V(K_1) = K_n,\ \mathrm{Eval}_V(L_1) = L_n \text{ and so } \mathrm{Eval}_V(M) = N,$$

concluding the proof.

Thus our clipped version of the programming language ISWIM is indeed determined by a standard reduction sequence as outlined in the introduction. As a corollary, we will give some weaker relations between λ_V and ISWIM, not involving the concept of a s.r. sequence.

Corollary 2. 1. *There is a value N such that $\lambda_V \vdash M = N$ iff $\mathrm{Eval}_V(M)$ is defined, for closed M.*

2. *Suppose Constants = $\mathcal{F}$-Constants ∪ $\mathcal{B}$-Constants as mentioned above. With each closed M we may associate two partial binary (say) functions f_M and g_M in $\mathcal{B}$-Constants$^2 \xrightarrow{P}$ $\mathcal{B}$-Constants by:*

$$f_M(a, b) = c \text{ iff } \mathrm{Eval}_V(Mab) = c, \text{ for any } \mathcal{B}\text{-Constant } c,$$

and $$g_M(a, b) = c \text{ iff } \lambda_V \vdash Mab = c, \text{ for any } \mathcal{B}\text{-Constant } c.$$

Then $f_M = g_M$.

Proof 1. From Theorem 4 $\lambda_V \vdash M = \mathrm{Eval}_V(M)$ when $\mathrm{Eval}_V(M)$ is defined. Conversely, suppose $\lambda_V \vdash M = N$ for some value N. Then, by the Church–Rosser theorem there is a term L such that $\lambda_V \vdash M \geqslant L$ and $\lambda_V \vdash N \geqslant L$. As N is a value, so must L be. So by Corollary 1, $M \xrightarrow[V]{*} K$ for some value K and so, by Theorem 4, $\mathrm{Eval}_V(M)$ is defined.

2. First, note that it follows from the Church–Rosser theorem that g_M is a well-defined partial function. Using Theorem 4, we see that

$$f_M(a, b) = c \text{ implies } \mathrm{Eval}_V(Mab) = c \text{ implies } \lambda_V \vdash Mab = c.$$

Conversely, using the Church–Rosser theorem, Corollary 1 and Theorem 4, we see that:

$$g_M(a, b) = c \text{ implies } \lambda_V \vdash Mab = c \text{ implies } \lambda_V \vdash Mab \geqslant c \text{ implies}$$
$$\text{Eval}_V(Mab) = c,$$

concluding the proof.

The second part of the Corollary says that λ_V and Eval_V assign the same functions of basic constants (to basic constants) to closed terms. If, as seems reasonable, one regards ISWIM as being given by such assignments rather than Eval_V, then we see that it is completely determined by λ_V without any reference to s. r. sequences.

Next we consider in what sense those equations provable in λ_V are true (for Eval_V). We assume that the constants are divided up into basic and functional ones.

Clearly if $M = N$ is true, then M and N are equal in any context, that is, with an obvious notation, for any context $C[\,]$, $C[M] = C[N]$ is true. Also, if $M = N$ is true and both M and N are closed either $\text{Eval}_V(M)$ and $\text{Eval}_V(N)$ are both undefined or else they are both defined and one is a given basic constant iff the other is.

We are going to take these necessary conditions as being also sufficient; the emphasis on basic constants seems appropriate, for one would want, for example, $(\lambda x \text{ succ } x)$ to equal succ. The notion of context can safely be kept informal, $C[\,]$ can be regarded as a term with a "hole", $C[N]$ is the term obtained by filling the hole with N. Note that $\lambda_V \vdash M = N$ implies $\lambda_V \vdash C[M] = C[N]$.

Definition. $M \approx_V N$ iff for any context $C[\,]$ such that $C[M]$ and $C[N]$ are closed, $\text{Eval}_V(C[M])$ and $\text{Eval}_V(C[N])$ are either both undefined, or else are both defined and one is a given basic constant iff the other is.

One can check that $\approx_V$ is indeed the largest relation satisfying the above conditions, and is an equivalence relation. Further, if $M \approx_V N$, for closed terms M and N, then with the notation of Corollary 2.2 $f_M = g_N$.

Theorem 5. *If* $\lambda_V \vdash M = N$ *then* $M \approx_V N$.

Proof. Suppose $\lambda_V \vdash M = N$ and $C[M]$ and $C[N]$ are closed.

If $\text{Eval}_V(C[M])$, say, is defined, and so is a value,

$$\lambda_V \vdash C[N] = C[M] = \text{Eval}_V(C[M]),$$

and so $\text{Eval}_V(C[N])$ is defined by Corollary 2.1.

Suppose $\text{Eval}_V(C[M])$ and $\text{Eval}_V(C[N])$ are both defined and the former, say, is a basic constant. As $\lambda_V \vdash \text{Eval}_V(C[M]) = \text{Eval}_V(C[N])$, the result follows from the Church–Rosser theorem, concluding the proof.

So we can regard λ_V as being consistent. It is, however, not complete. Let us say that M has *order zero* if it is closed and has no (call-by-value) value. Then if M and N are of order zero, $M \approx_V N$. We outline the proof. Let us say that terms M and M' *correspond* iff they are identical, apart from occurrences of order zero terms

at homologous positions. Then, if M and M' correspond and $M' \xrightarrow[V]{} N'$, either M and N' correspond or else, for some N, $M \xrightarrow[V]{} N$ and N and N' correspond. The result is then immediate.

In a similar way, one can show that $\lambda xx(\lambda yxy) \approx {}_V\lambda xxx$. This is an instance of a valid restricted form of η-reduction which it would be interesting to add to λ_V.

5. Call-by-name

In this section we proceed at a rather rapid rate through results for call-by-name quite analogous to those of the previous section.

The language of our call-by-name calculus is that of ISWIM. Since simple lambda calculus based programming languages, such as PAL [19], GEDANKEN [11], etc. all use call-by-value, we feel free to define Eval_N directly, in analogy to the recursive definition of Eval_V. It is an interesting exercise to define an appropriate version of the SECD machine, and prove the result analogous to Theorem 3.1.

Assuming a Constapply as given, Eval_N has this informal definition:

$\mathrm{Eval}_N(a) = a$; $\mathrm{Eval}_N(\lambda xM) = (\lambda xM)$.

$\mathrm{Eval}_N(MN) = \mathrm{Eval}_N([N/x]\,M')$ (if $\mathrm{Eval}_N(M) = (\lambda xM)$).

$\mathrm{Eval}_N(MN) = \mathrm{Constapply}(a, b)$ (if $\mathrm{Eval}_N(M) = a$ and $\mathrm{Eval}_N(N) = b$).

We leave to the reader the formal definition of Eval_N via a definition of "M has (call-by-name) value N at time t". Clearly, if it exists, $\mathrm{Eval}_N(M)$ is a value.

Our λ_N calculus is just the appropriate λK-$\beta\delta$ calculus. Explicitly, its formulas have the form $M = N$, where M and N are terms. Its rules are:

II. $(\lambda xM) = (\lambda y\,[y/x]\,M)\,(y \notin FV(M))$ (α-reduction).

2. $(\lambda xM)\,N = [N/x]\,M$ (β-reduction).

3. $(ab) = \mathrm{Constapply}(a, b)$ (if this is defined) (δ-reduction).

III. $M = M$

2. $\dfrac{M = N \quad N = L}{M = L}$

3. $\dfrac{M = N}{N = M}$

III. $\dfrac{M = N}{(MZ) = (NZ)}$, $\dfrac{M = N}{(ZM) = (ZN)}$

2. $\dfrac{M = N}{(\lambda xM) = (\lambda xN)}$

$\lambda_N \vdash M = N$ means that $M = N$ is provable by the above rules.

$\lambda_N \vdash M \geq N$ means that $M = N$ is provable using any of the above rules except II3.

From [2], we know that if $\lambda_N \vdash M = N$ then $\lambda_N \vdash [L/x]M = [L/x]N$, for any term L, and so the free variables can be interpreted as universally quantified; we also know that the Church–Rosser theorems holds. Next we proceed to a formulation of the standardisation theorem, again without any reference to redexes.

The relation $\underset{N}{\rightarrow}$ of left reduction is the least relation such that:

(1) $(\lambda x M)N \underset{N}{\rightarrow} [N/x]M$

(2) $(ab) \underset{N}{\rightarrow}$ Constapply (a, b) (when defined)

(3) If $M \underset{N}{\rightarrow} M'$ then $(MN) \underset{N}{\rightarrow} (M'N)$

(4) If M is a constant or variable and $N \underset{N}{\rightarrow} N'$ then $(MN) \underset{N}{\rightarrow} (MN')$.

Clearly left reduction is a partial function. If $M \underset{N}{\rightarrow} N$ then M is not a value.

Standard reduction sequences (s.r. sequences) are defined inductively by:

1. A variable or a constant is a s.r. sequence.

2. If $N_2, \ldots, N_k$ is a s.r. sequence and $N_1 \underset{N}{\rightarrow} N_2$ then $N_1, \ldots, N_k$ is a s.r. sequence.

3. If $N_1, \ldots, N_k$ is a s.r. sequence, so is $(\lambda x N_1), \ldots, (\lambda x N_k)$.

4. If $M_1, \ldots, M_j$ and $N_1, \ldots, N_k$ are s.r. sequences, so is $(M_1 N_1), \ldots, (M_j N_1), \ldots, (M_j N_k)$.

Theorem 1. (Standardisation theorem). $\lambda_N \vdash M \geqslant N$ *iff there is a* s.r. *sequence* $N_1, \ldots, N_k$ *such that* $M = N_1$ *and* $N_k = {}_\alpha N$.

We omit the proof. As remarked by Morris [9], it can be obtained by modifying the proof in Curry [2] to consider the δ-rules too. It can also be obtained by analogy with our method for λ_V.

The proofs of all following theorems and remarks are omitted. They are analogous to those of the corresponding ones in the previous section.

Corollary 1. *For any term* M, $\lambda_N \vdash M \geqslant N$ *for some value* N *iff* $M \underset{N}{\overset{*}{\rightarrow}} N'$ *for some value* N'.

Theorem 2. $\mathrm{Eval}_N(M) = N$ *iff* $M \underset{N}{\overset{*}{\rightarrow}} N$, *for closed* M.

Corollary 2. 1. *There is a value* N *such that* $\lambda_N \vdash M \geqslant N$ *iff* $\mathrm{Eval}_N(M)$ *is defined.*

2. *If the constants are divided into basic and functional constants, and we associate partial binary (say) functions* f_M *and* g_M *with a closed term,* M, *by:*

$$f_M(a, b) = c \text{ iff } \mathrm{Eval}_N(Mab) = c, \text{ for any basic constant } c,$$

and

$$g_M(a, b) = c \text{ iff } \lambda_N \vdash Mab = c, \text{ for any basic constant } c,$$

then $f_M = g_M$.

Assume now that the constants are divided into basic and functional ones.

Definition. $M \approx_N N$ iff for every context, $C[\,]$ where $C[M]$ and $C[N]$ are closed, $\mathrm{Eval}_N(C[M])$ and $\mathrm{Eval}_M(C[N])$ are either both defined or both undefined and in the former case either both are the same B-constant or both are not B-constants.

Theorem 3. *If* $\lambda_N \vdash M = N$ *then* $M \approx_N N$.

Again, although consistent, λ_N is by no means complete. If we say a closed term M is of order zero iff it has no call-by-name value then if M and N are order zero terms, $M \approx_N N$. It is also true that $\lambda x.(x(\lambda y.xy)) \approx_N \lambda x.xxx$, and a more general form of η-reduction is valid.

6. Simulations and translations

Our object here is to show that call-by-value can be simulated by call-by-name, and vice versa. It is known that some aspects of call-by-name can be fairly easily simulated by call-by-value; for example the call-by-name conditional can be simulated by the call-by-value one [7] and the term Z, mentioned previously, provides a good recursion operator, [19]. However these "protecting by a λ" techniques do not seem extendable to a complete simulation and it is fortunate that the technique of continuations is available. These have been used to provide denotational semantics for languages with call-by-value [15], to give definitional interpreters whose defined languages are independent of the order of evaluation of the defining language [10, 12] and to show that a deletion implementation strategy does not reduce the possible functions [3]. It turns out that this work easily provides us with a simulation of call-by-value by call-by-name. Some modification is required for a simulation in the other direction, which is based on a definitional interpreter, of the above sort, for a call-by-name language [13].

We begin with a simulation of call-by-value by call-by-name. Given a call-by-value language with its $\mathrm{Constapply}_V$, Eval_V and λ_V, we consider the call-by-name language whose variables are those of the given language together with three others, $\varkappa$, α and β say, and whose list of variables for the substitution prefix is that of the given language. Its Constapply will be given in a little while. First the term simulation map $M \mapsto \overline{M}$ sending terms in the call-by-value language to the call-by-name language is given by the recursive definition:

$$\overline{a} = \lambda\varkappa.(\varkappa a)$$

$$\overline{x} = \lambda\varkappa.(\varkappa x)$$

$$\overline{\lambda x.M} = \lambda\varkappa.(\varkappa(\lambda x.\overline{M}))$$

$$\overline{MN} = \lambda\varkappa.(\overline{M}(\lambda\alpha.\overline{N}(\lambda\beta.\alpha\beta\varkappa))).$$

$\mathrm{Constapply}_N$ is given by:

$$\mathrm{Constapply}_N(a, b) = \overline{\mathrm{Constapply}_V(a, b)}$$

Let $\mathcal{L}$ be the call-by-value language under consideration, $\mathcal{L}'$ be the call-by-name one and $\mathcal{L}''$ be the call-by-value language whose constants, variables, variable list and Constapply are those of $\mathcal{L}'$. On occasion use of the $\mathcal{L}$'s as superscripts or prefixes will avoid ambigiuty.

An auxiliary function, Ψ sending values to values is defined by:

$$\Psi(a) = a; \Psi(x) = x; \Psi(\lambda x M) = (\lambda x \overline{M}).$$

We intend to prove the following three theorems:

Theorem 1. (Indifference). $\mathrm{Eval}_N(\overline{M}(\lambda xx)) = \mathrm{Eval}_V(\overline{M}(\lambda xx))$. *for any program M.*

Theorem 2. (Simulation). $\Psi(\mathrm{Eval}_V(M)) = \mathrm{Eval}_N(\overline{M}(\lambda xx))$, for any program M.

Theorem 3. (Translation) *If* $\lambda_V^{\mathcal{L}} \vdash M = N$ *then* $\lambda_V^{\mathcal{L}''} \vdash \overline{M} = \overline{N}$ *and then* $\lambda_N \vdash \overline{M} = \overline{N}$. *The second but not the first implication is reversible.*

Notice that the simulation maps used for programs, that is closed terms, in Theorem 2 are different from that used for terms in general.

The first theorem is just a reworking of the "defining-language-order-of-evaluation-independent" definitional interpreter. The second is, essentially, due to Fischer, modified by an application of Theorem 1. The following corollary, shows that Eval_N gives all the functions (on basic constants) Eval_V does.

Corollary 1. *If* $f_M^{\mathcal{L}}$ *is the function assigned a closed term M as in Corollary* 4.2.2 *and* $f_N^{\mathcal{L}'}$ *the function assigned* $N = \lambda x \lambda y (\overline{Mxy}(\lambda xx))$ *as in Corollary* 5.2.2 *then* $f_M^{\mathcal{L}} = f_N^{\mathcal{L}'}$. *(Assuming the constants are divided into basic and functional ones.)*

Proof. $f_N^{\mathcal{L}'}(a, b) = c$ iff $\mathrm{Eval}_N(Nab) = c$

iff $\mathrm{Eval}_N(\overline{Mab}(\lambda xx)) = c$ (by Lemma 1 below)

iff $\Psi(\mathrm{Eval}_V(Mab)) = c$ (by Theorem 2)

iff $f_M^{\mathcal{L}'}(a, b) = c$.

Notice that since $\mathrm{Constapply}_N \neq \mathrm{Constapply}_V$, our simulations are not interpretation independent. It would be interesting to find general conditions on $\mathrm{Constapply}_V$ which would allow an interpretation independent simulation.

Unfortunately, although operational inequality is preserved, equality is not:

Corollary 2. 1. *For any terms, M, N, if* $\overline{M} \approx_N \overline{N}$ *then* $M \approx_V N$.

2. *For any closed terms M, N, if* $\overline{M}I \approx_N \overline{N}I$ *then* $M \approx_V N$, *where* $I = (\lambda xx)$.

3. *Neither of the converses of 1 or 2 hold.*

Proof 1. Suppose $M \not\approx_V N$. Then there are closed terms, $C[M]$ and $C[N]$ such that either one of $\mathrm{Eval}_V(C[M])$, $\mathrm{Eval}_V(C[N])$ is defined and the other not or else both

are defined, one is a basic constant and the other is not, or else they are both defined but are different basic constants. Now there is a context $D[\,]$ such that $\overline{C[M]}\,I = D\overline{[M]}$ and $\overline{C[N]}\,I = D\overline{[N]}$, where $I = (\lambda xx)$. Then it follows that $\bar{M} \approx_N \bar{N}$ by applying Theorem 2 to the terms $C[M]$ and $C[N]$.

2. Suppose $M \not\approx_V N$. If one of $\mathrm{Eval}_V(M)$, $\mathrm{Eval}_V(N)$ is defined and the other not then, by Theorem 2, the same applies to $\mathrm{Eval}_N(\bar{M}I)$ and $\mathrm{Eval}_N(\bar{N}I)$ and then $\bar{M}I \not\approx_N \bar{N}I$; otherwise both are defined and, as we have $\lambda\varkappa\varkappa\,(\bar{M}I) \approx_N \bar{M}$ and similarly for N, we cannot have $\bar{M}I \approx_N \bar{N}I$ as otherwise we would have $\bar{M} \approx_N \bar{N}$ contradicting 1.

3. Take $M = \lambda y\lambda xx\,(yx)$, $N = \lambda y\lambda xx\,(y\,(\lambda xxz))$. Then $M \approx_V N$ but $\bar{M}I \not\approx_N \bar{N}I$ and so we also have $\bar{M} \not\approx_N \bar{N}$, concluding the proof.

If the reader examines some examples of programs M_0 and the sequences $M_0 \xrightarrow[V]{} M_1 \xrightarrow[V]{} \ldots$ and $\bar{M}_0\,I = N_0 \xrightarrow[N]{} N_1 \to \ldots$ he will find that the $\xrightarrow[N]{}$'s consist of a sequence of "administrative" reductions followed by a "proper" reduction corresponding to a $\xrightarrow[V]{}$ followed by more administrative reductions and so on. The term I does not figure in any reductions until an N_n has been reached corresponding to an M_m which is a value. So we will define an infix operation: such that $M{:}K$ is the result of performing all the administrative reductions on $\bar{M}K$, and so $\bar{M}K \xrightarrow[N]{+} \bar{M}{:}K$ as is shown by Lemma 2 below. With the help of Lemma 1 it can be seen that the result, N_j, of the proper reduction $\bar{M}_j{:}I \xrightarrow[N]{} N_j$ corresponding to $M_0 \xrightarrow[V]{} M_1$ is itself the result of some administrative reductions on $\bar{M}_1\,I$ and so in general we will expect that if $M \xrightarrow[V]{} M'$ then $M{:}K \xrightarrow[\Lambda]{+} M'{:}K$, as shown in Lemma 3 below. One now has a good picture of the sequence $N_0 \xrightarrow[N]{} N_1 \to \ldots$ in terms of $M_0 \xrightarrow[V]{} M_1 \to \ldots$ and this, together with some information on error stops, given by Lemma 4 below, gives a proof of Theorem 2. Since all the $\xrightarrow[N]{}$'s are also $\xrightarrow[V]{}$'s we also have one of Theorem 1.

Lemma 1. $[\Psi(N)/x]\,\bar{M} = \overline{[N/x]\,M}$ *(if N is a value and $x \notin \{\varkappa, \alpha, \beta\}$).*

Proof. By induction on the size of M.

Case 1. If M is a constant a, then

$$[\Psi(N)/x]\,\bar{a} = [\Psi(N)/x]\,(\lambda\varkappa\varkappa a) = (\lambda\varkappa\varkappa a) = \overline{[N/x]\,a}$$

Case 2. $M = x$.

$$\text{L. S.} = [\Psi(N)/x]\,\lambda\varkappa\varkappa x = \lambda\varkappa\varkappa\Psi(N)\ (\varkappa \notin FV(N))$$
$$= \bar{N} = \overline{[N/x]\,x}.$$

$M = y$. Trivial.

Case 3. M is a combination, $(M_1\ M_2)$.

$$\text{L. S.} = [\Psi(N)/x]\,(\lambda\varkappa\bar{M}_1(\lambda\alpha\bar{M}_2(\lambda\beta\alpha\beta\varkappa))).$$
$$= (\lambda\varkappa\,[\Psi(N)/x]\,\bar{M}_1(\lambda\alpha\,[\Psi(N)/x]\,\bar{M}_2(\lambda\beta\alpha\beta\varkappa))) \quad (\text{as } x \notin \{\varkappa, \alpha, \beta\} \text{ and } \alpha, \varkappa \notin FV(\Psi(N))),$$
$$= (\lambda\varkappa\,\overline{[N/x]\,M}_1(\lambda\alpha\,\overline{[N/x]\,M}_2(\lambda\beta\alpha\beta\varkappa))) \quad (\text{by induction hypothesis}),$$
$$= \overline{[N/x]\,M_1[N/x]\,M_2}$$
$$= \text{R. S.}$$

Case 4. M is an abstraction, $(\lambda y M_1)$. When $y = x$, the result is immediate. Otherwise,

$$\begin{aligned}
\text{L. S.} &= [\Psi(N)/x]\,(\lambda\varkappa\varkappa\,(\lambda y \bar{M}_1)) \\
&= \lambda\varkappa\,(\varkappa\lambda z\,([\Psi(N)/x]\,[z/y]\,\bar{M}_1)) && (x \neq \varkappa,\ \varkappa \notin FV(N), \text{ and with the usual conditions on } z) \\
&= \lambda\varkappa\,(\varkappa\lambda z\,([\Psi(N)/x]\,\overline{[z/y]\,M_1})) && \text{(by induction hypothesis)} \\
&= \lambda\varkappa\,(\varkappa\lambda z\,\overline{[N/x]\,[z/y]\,M_1}) && \text{(by induction hypothesis)} \\
&= \overline{\lambda z\,[N/z]\,[z/y]\,M_1} \\
&= \overline{[N/x]\,\lambda y M_1}, \text{ concluding the proof.}
\end{aligned}$$

Next we define the convenient infix operation, :, in $\mathcal{L}$-Closed Terms $\times$ $\mathcal{L}'$-Terms $\to$ $\mathcal{L}'$-Terms.

$$\begin{aligned}
N{:}K &= K\,\Psi(N) && (N \text{ is a closed value}) \\
MN{:}K &= M{:}(\lambda\alpha\bar{N}\,(\lambda\beta\alpha\beta K)) && (M \text{ is not a value}) \\
MN{:}K &= N{:}(\lambda\beta\Psi(M)\,\beta K) && (M, \text{ but not } N, \text{ is a value}) \\
MN{:}K &= \Psi(M)\,\Psi(N)\,K && (M \text{ and } N \text{ are values}).
\end{aligned}$$

In the following, results asserted for $\to$. are intended to be asserted for both $\underset{V}{\to}$ and $\underset{N}{\to}$. In the former case we mean $\mathcal{L}''$.

Lemma 2. If K *is a closed value then* $\bar{M}K \overset{+}{\to} M{:}K$, *for any term* M, $(\alpha, \beta, \varkappa \notin FV(K))$.

Proof. By induction on the size of M and cases on the definition of :.

1. M is a value.

$$\begin{aligned}
\bar{M}K &= (\lambda\varkappa\varkappa\Psi(M))\,K \\
&\to K\Psi(M) \\
&= M{:}K.
\end{aligned}$$

2. $M = (M_1\,M_2)$.

Subcase 1. M_1 is not a value.

$$\begin{aligned}
\bar{M}K &= (\lambda\varkappa\bar{M}_1(\lambda\alpha\bar{M}_2(\lambda\beta\alpha\beta\varkappa)))\,K \\
&\to \bar{M}_1(\lambda\alpha\bar{M}_2(\lambda\beta\alpha\beta K)) \\
&\overset{+}{\to} M_1{:}(\lambda\alpha\bar{M}_2(\lambda\beta\alpha\beta K)) && \text{(by induction hypothesis)} \\
&= M{:}K
\end{aligned}$$

Subcase 2. M_1 is a value and M_2 is not.

$$\begin{aligned}
\bar{M}K &\overset{+}{\to} M_1{:}(\lambda\alpha\bar{M}_2(\lambda\beta\alpha\beta K)) = (\lambda\alpha\bar{M}_2(\lambda\beta\alpha\beta K))\,\Psi(M_1) \\
&\to \bar{M}_2(\lambda\beta\Psi(M_1)\,\beta K) \overset{*}{\to} M{:}K && \text{(by induction hypothesis).}
\end{aligned}$$

Subcase 3. M_1 and M_2 are values.

$$\begin{aligned}
\bar{M}K &\overset{+}{\to} \bar{M}_2{:}(\lambda\beta\Psi(M_1)\,\beta K) && \text{(as in Subcase 2)} \\
&\overset{+}{\to} \Psi(M_1)\,\Psi(M_2)\,K.
\end{aligned}$$

Lemma 3. *If* $M \underset{V}{\to} N$ *then* $M{:}K \overset{+}{\to} N{:}K$ (*if* K *is a closed value and* M *and* N *are terms*).

Proof. By induction on the size of M and by cases according to the definition of $\underset{V}{\to}$.

1. $M = (ab)$ and $N = \text{Constapply}\ (a, b)$
$M{:}K = abK \to \text{Constapply}_N(a, b)\ K = \bar{N}K$
$\overset{+}{\to} N{:}K$ (by Lemma 2).

2. $M = (\lambda x M_1)\ M_2$ and $N = [M_2/x]\ M_1$ and M_2 is a value.
$M{:}K = \Psi(\lambda x M_1)\ \Psi(M_2)\ K = (\lambda x \bar{M}_1)\ \Psi(M_2)\ K$
$\to [\Psi(M_2)/x]\ \bar{M}_1\ K$
$= \overline{[M_2/x]\ M_1}\ K = \bar{N}K$
$\overset{+}{\to} N{:}K$, (by Lemma 1).

3. $M = M_1\ M_2$, M_1 is a value, $M_2 \underset{V}{\to} N_2$ and $N = (M_1\ N_2)$.
$M{:}K = M_2{:}(\lambda\beta\Psi(M_1)\ \beta K)$
$\to N_2{:}(\lambda\beta\Psi(M_1)\ \beta K)$ (by induction hypothesis)
$= L$, say.

If N_2 is not a value then, $L = N{:}K$. Otherwise,
$L = (\lambda\beta\Psi(M_1)\ \beta K)\ \Psi(N_2)$
$\to N{:}K$.

4. $M = M_1\ M_2$, $M_1 \underset{V}{\to} N_1$ and $N = (N_1\ M_2)$
$M{:}K = M_1{:}(\lambda\alpha\bar{M}_2(\lambda\beta\alpha\beta K))$
$\overset{+}{\to} N_1{:}(\lambda\alpha\bar{M}_2(\lambda\beta\alpha\beta K))$
$= L$, say.

If N_1 is not a value, $L = N{:}K$. Otherwise,
$L \overset{+}{\to} \bar{M}_2(\lambda\beta\Psi(N_1)\ K) = L'$, say.

If M_2 is not a value, $L = N{:}K$. Otherwise,
$L' \overset{+}{\to} \Psi(N_1)\ \Psi(N_2)\ K = N{:}K$,

concluding the proof.

One can see that, in any call-by-value language, if $M \overset{+}{\underset{V}{\to}} N$, for any term N and M is not a value and M is closed, then M is in the set Sticks_V defined inductively by:

1. If $\text{Constapply}\ (a, b)$ is not defined, $(ab) \in \text{Sticks}_V$.
2. For any term N, $(a\ (\lambda x N)) \in \text{Sticks}_V$.
3. If $N \in \text{Sticks}_V$, then $((\lambda x M)\ N) \in \text{Sticks}_V$, for any term M.
4. If $M \in \text{Sticks}_V$, then $(MN) \in \text{Sticks}_V$, for any term N.

Similarly, in any call-by-name language, if $M \overset{+}{\underset{N}{\to}} N$ for any term N, M is not a value and M is closed then M is in the set Sticks_N defined inductively by:

1. If $\text{Constapply}\ (a, b)$ is not defined $(ab) \in \text{Sticks}_N$.

2. For any term N, $(a\,(\lambda xN)) \in \mathrm{Sticks}_N$.
3. If $M \in \mathrm{Sticks}_N$, $(MN) \in \mathrm{Sticks}_N$, for any term N.
4. If $N \in \mathrm{Sticks}_N$, then $aN \in \mathrm{Sticks}_N$.

Clearly, if the two languages have the same constants, variables and Constapply then $\mathrm{Sticks}_N \subseteq \mathrm{Sticks}_V$.

Lemma 4. If $M \in \mathrm{Sticks}_V$ then $M{:}K \in \mathrm{Sticks}_N^{\mathcal{L}'} \subseteq \mathrm{Sticks}_V^{\mathcal{L}''}$.

Proof. By induction on the size of M and cases according to the definition of Sticks_V.

1. $(ab){:}K = abK \in \mathrm{Sticks}_N$($\mathrm{Constapply}_\gamma(a, b)$ not defined).
2. Like case 1.
3. $M_2 \in \mathrm{Sticks}_V^{\mathcal{L}}$, $(\lambda xM_1)\,M_2{:}K = M_2{:}(\lambda\beta\Psi(M_1)\,\beta K) \in \mathrm{Sticks}_N$ (by induction hypothesis).
4. $M_1 \in \mathrm{Sticks}_V^{\mathcal{L}}$, $(M_1, M_2){:}K = M_1{:}(\lambda\alpha M_2(\lambda\beta\alpha\beta K)) \in \mathrm{Sticks}_N$ (by induction hypothesis)

Proof of Theorems 1 and 2

1. $\mathrm{Eval}_V^{\mathcal{L}}(M)$ is defined and is N, say. By Theorem 4.4, $M \overset{*}{\underset{V}{\to}} N$. So by Lemmas 2, 3

$$\begin{aligned}\bar{M}\,(\lambda xx) &\overset{+}{\to} \bar{M}{:}(\lambda xx)\\ &\overset{*}{\to} \bar{N}{:}(\lambda xx)\\ &\to \Psi(N) \text{ (as } N \text{ is a value)}\\ &= \Psi(\mathrm{Eval}_V^{\mathcal{L}}(M)).\end{aligned}$$

Therefore, in this case

$$\begin{aligned}\mathrm{Eval}_N(\bar{M}\,(\lambda xx)) &= \mathrm{Eval}_V^{\mathcal{L}''}(\bar{M}\,(\lambda xx))\\ &= \Psi(\mathrm{Eval}_V(M)),\end{aligned}$$

by Theorems 4.2 and 5.2.

2. $\mathrm{Eval}_V^{\mathcal{L}}(M)$ is not defined. By Theorem 4.4, either $M \overset{*}{\underset{V}{\to}} N \in \mathrm{Sticks}_V^{\mathcal{L}}$ or else there is an infinite series $M_1, M_2, \ldots$, such that

$$M = M_1 \underset{V}{\to} M_2 \underset{V}{\to} M_2 \to \ldots$$

In the first case

$$\begin{aligned}\bar{M}\,(\lambda xx) &\overset{+}{\to} \bar{M}{:}(\lambda xx)\\ &\overset{*}{\to} \bar{N}{:}(\lambda xx)\\ &\in \mathrm{Sticks}_N \text{ (by Lemma 4).}\end{aligned}$$

Then neither $\mathrm{Eval}_N(\bar{M}\,(\lambda xx))$ nor $\mathrm{Eval}_V^{\mathcal{L}}(\bar{M}\,(\lambda xx))$ are defined by Theorems 4.4 and 5.2.

In the second case, we have

$$\bar{M}\,(\lambda xx) \overset{+}{\to} \bar{M}{:}(\lambda xx) = \bar{M}_1{:}(\lambda xx) \overset{+}{\to} \bar{M}_2{:}(\lambda xx) \overset{+}{\to} \ldots,$$

with the same result, concluding the proof.

Proof of Theorem 3. We omit the straightforward proof that $\lambda_V^{\mathcal{L}} \vdash M \geqslant N$ implies. $\lambda_V^{\mathcal{L}''} \vdash \bar{M} \geqslant \bar{N}$, from which it follows immediately by the Church-Rosser theorem that if $\lambda_V^{\mathcal{L}} \vdash M = N$ then $\lambda_V^{\mathcal{L}''} \vdash \bar{M} = \bar{N}$. It is then clear that $\lambda_N \vdash \bar{M} = \bar{N}$.

Conversely suppose $\lambda_N \vdash \bar{M} = \bar{N}$. Note that $\bar{M}$ and $\bar{N}$ have this property:

If $L_1 \ldots L_n$ is a subterm and L_1 is a value so are $L_2 \ldots L_n$. Any call-by-name redex in a term with this property is also a call-by-value one, and if L has the property and $\lambda_N \vdash L \geqslant L'$, then L' also has the property. It follows from these remarks that $\lambda_V^{\mathcal{L}'} \vdash \bar{M} = \bar{N}$.

Finally, if $M = ((\lambda xxx)(\lambda xxx))\, y$ and $N = (\lambda xxy)((\lambda xxx)(\lambda xxx))$ then $\lambda_V^{\mathcal{L}''} \vdash \bar{M} = \bar{N}$ but $\lambda_V^{\mathcal{L}} \nvdash M = N$. Note that $M \approx_V N$.

Having completed our treatment of the simulation of call-by-value by call-by-name, we pass next to the simulation of call-by-name by call-by-value. So suppose we are given a call-by-name language $\mathcal{L}$, with some Constapply_N. Consider that call-by-value language $\mathcal{L}'$ whose constants are those of $\mathcal{L}$, whose variables are those of $\mathcal{L}$, plus $\varkappa$ and α, whose variable-list is that of $\mathcal{L}$ and whose Constapply_V will be given later via a function $M \mapsto \underline{M}$ from $\mathcal{L}$ terms to $\mathcal{L}'$ terms which also helps to specify the simulation map. It is convenient to let $\mathcal{L}''$ be the language which, apart from its being call-by-name, is identical to $\mathcal{L}'$. We will use the $\mathcal{L}$'s as superscripts or prefixes as before. We will only consider the case where the constants of $\mathcal{L}$ are divided into basic and functional classes. This allows a simple simulation map.

The map $M \mapsto \underline{M}$ is defined recursively by:

$$\begin{aligned}
&\underline{x} = x \\
&\underline{a} = \lambda\varkappa(\varkappa(\lambda\alpha\alpha(\alpha I))) && (a \in \mathcal{F}\text{-constants}) \\
&\underline{b} = \lambda\varkappa(\varkappa b) && (b \in \mathcal{B}\text{-constants}) \\
&\underline{(\lambda xM)} = \lambda\varkappa\varkappa(\lambda x\underline{M}) \\
&\underline{(MN)} = \lambda\varkappa\underline{M}(\lambda\alpha\alpha\underline{N}\varkappa).
\end{aligned}$$

Here, and below, I is the term (λxx).

Then, $\text{Constapply}_V(a, b) =_{\text{def}} \underline{\text{Constapply}_N(a, b)}$, and it is also convenient to define a map Φ from $\mathcal{L}$-values to $\mathcal{L}'$-terms by:

$$\Phi(x) = (xI);\ \Phi(a) = \lambda\alpha a(\alpha I);\ \Phi(b) = b;\ \Phi(\lambda xM) = \lambda x\underline{M}.$$

We intend to prove these three theorems:

Theorem 4. (Indifference). $\text{Eval}_V(\underline{M}I) = \text{Eval}_N^{\mathcal{L}''}(\underline{M}I)$, *for any term* M.

Theorem 5. (Simulation). $\Phi(\text{Eval}_N^{\mathcal{L}}(M)) = \text{Eval}_V(\underline{M}I)$.

Theorem 6. (Translation). $\lambda_N^{\mathcal{L}} \vdash M = N$ *iff* $\lambda_V^{\mathcal{L}'} \vdash \underline{M} = \underline{N}$ *iff* $\lambda_V^{\mathcal{L}'} \vdash \underline{M}I = \underline{N}I$ *iff* $\lambda_N^{\mathcal{L}''} \vdash \underline{M} = \underline{N}$ *iff* $\lambda_N^{\mathcal{L}''} \vdash \underline{M}I = \underline{N}I$.

As before, it follows that Eval_V gives all the functions Eval_N does; but once again the simulation is not perfect:

Corollary 3. 1. *For any terms M, N, if $\underline{M} \approx_V \underline{N}$ then $M \approx_N N$.*

2. *For any closed terms M, N, if $\underline{M}I \approx_V \underline{N}I$ then $M \approx_N N$.*

3. *Neither of the converses of 1 or 2 hold.*

Proof. This is similar to that of Corollary 2. As a counterexample one can take $M = \lambda xxx$ and $N = \lambda xx(\lambda yxy)$.

We omit the proofs of the first few lemmas, as they are quite similar to the proofs of the corresponding lemmas for the previous simulation.

Lemma 5. $[\underline{M}/x]\underline{N} = [\underline{M}/\underline{x}]\underline{N}$. $(x \notin \{\varkappa, \alpha\})$.

Now the infix operator $:$, in $\mathcal{L}$-Closed Terms $\times$ $\mathcal{L}'$-Terms $\to$ $\mathcal{L}'$-Terms, is defined by:

$$a{:}K = K(\lambda\alpha c(\alpha I))$$
$$b{:}K = Kb$$
$$(\lambda xM){:}K = K(\lambda x\underline{M})$$
$$(MN){:}K = M{:}(\lambda\alpha\alpha\underline{N}K) \quad (M \text{ not a value})$$
$$(aN){:}K = a\,\Phi(N)\,K \quad (N \text{ a value})$$
$$(aN){:}K = a(\mathrm{N}{:}I)\,K \quad (N \text{ not a value})$$
$$(bN){:}K = b\underline{N}K$$
$$(\lambda xM)N{:}K = (\lambda x\underline{M})\underline{N}K.$$

Lemma 6. *If K is a value then $\underline{M}K \xrightarrow{+} M{:}K$ (M closed).*

Lemma 7. *If $M \underset{N}{\to} N$ then $M{:}K \xrightarrow{+} N{:}K$ if M is not of the form $b\,L_1 \ldots L_n$ (M closed).*

Lemma 8. *If $M \in \mathrm{Sticks}_N^{\mathcal{L}}$ then $M{:}K \in \mathrm{Sticks}_N^{\mathcal{L}''} \subseteq \mathrm{Sticks}_V^{\mathcal{L}'}$.*

Theorems 4 and 5 now follow just as Theorems 1 and 2 did.

The proof of Theorem 6 is, as it were, a non-deterministic analogue of the proof of Theorems 4 and 5.

Consider a term M and the corresponding terms $\underline{M}$ and $\underline{M}I$. Once again reductions on $\underline{M}$ and $\underline{M}I$ can be either administrative or proper, but since we are considering $\geqslant$ rather than $\to$ we need relations rather that a function like $:$. So $M \sim M'$ will mean that M' can be obtained from $\underline{M}$ by administrative reductions and $M \mathrel{\wr} M'$ means that M' can be obtained from $\underline{M}I$ by administrative reductions. Lemma 9 below corresponds to Lemmas 1 and 5 above. Lemma 11 corresponds to Lemmas 2, 3, 5 and 7 above, and is expected if we have the right definitions of $\sim$ and $\mathrel{\wr}$. It is then

straightforward to show that if $\lambda_V^{\mathcal{L}''} \vdash MI = NI$ then MI and NI reduce to terms in the relation $\mathrel{\mathring{\sim}}$ to alphabetically equivalent terms and one then uses the rather technical Lemma 10. In this way one shows that $\lambda_V^{\mathcal{L}''} \vdash MI = NI$ implies $\lambda_N^{\mathcal{L}} \vdash M = N$ and the rest of Theorem 6 is straightforward. It should be remarked that the analogue of Lemma 10, for the simulation in the other direction, fails.

We set

$$(M; N) =_{\text{def}} \begin{cases} [N/x]\, M_1 & (\text{if } M \text{ is } (\lambda x M_1)) \\ (MN) & (\text{otherwise}) \end{cases}$$

$(M|N)$ will be used, ambiguously, to abbreviate (MN) and $(M; N)$. The relations $\sim$ and $\mathrel{\mathring{\sim}}$ are now defined by means of a formal system whose formulae are of the form $M \sim M'$ or $M \mathrel{\mathring{\sim}} M'$, where M is an $\mathcal{L}$-term and M' is an $\mathcal{L}'$-term.

I. $x \sim x$

III1. $a \sim \lambda\varkappa\varkappa\,(\lambda\alpha\alpha\,(\alpha I))$ 2. $a \mathrel{\mathring{\sim}} (\lambda\alpha a\,(\alpha I))$

III1. $b \sim \lambda\varkappa\varkappa b$ 2. $b \mathrel{\mathring{\sim}} b$

IV1. $\dfrac{M \sim M'}{(\lambda x M) \sim (\lambda\varkappa\,(\lambda x M'))}$ 2. $\dfrac{M \sim M'}{(\lambda x M) \mathrel{\mathring{\sim}} (\lambda x M')}$

V. $\dfrac{M \sim M' N \sim N'}{(MN) \sim \lambda\varkappa\,(M'|(\lambda\alpha\alpha N'\varkappa))}$

VI1. $\dfrac{N \sim N'}{(aN) \sim \lambda\varkappa\,((\lambda\alpha a\,(\alpha I))\, N' K}$ 2. $\dfrac{N \sim N'}{(aN) \sim \lambda\varkappa a N'\varkappa}$

VII. $\dfrac{N \sim N'}{(bN) \sim \lambda\varkappa\,(bN'\varkappa)}$

VIII. $\dfrac{N \sim N' M \sim M'}{(\lambda x M)\, N \sim \lambda\varkappa\,(\lambda x M')\, N'\varkappa}$

IX. $\dfrac{M \sim M'}{M \mathrel{\mathring{\sim}} (M'|I)}$.

X. $\dfrac{M \sim M'}{M \sim \lambda\varkappa M'\varkappa}$

We will also use $M \sim M'$ $(M \mathrel{\mathring{\sim}} M')$ to mean that the formula $M \sim M'$ $(M \mathrel{\mathring{\sim}} M')$ is provable by the above rules. Notice that if $M \sim M'$ or $M \mathrel{\mathring{\sim}} M'$ then $FV(M) = FV(M')$. If $M \sim M' = \lambda\varkappa M''$, then $\varkappa$ occurs in position 1 of M'' iff M is a value, other than a variable; occurs in position 2 of M'' iff $M \sim M'$ follows from an application of one of rules VI, VII, VIII or X.

Lemma 9. 1. *If $M \sim M'$ and $N \sim N'$ then for some L,*
$[N/x]\, M \sim L =_\alpha [N'/x]\, M'$.

2. *If $M \mathrel{\mathring{\sim}} M'$ and $N \sim N'$ then for some L,*
$[N/x]\, M \mathrel{\mathring{\sim}} L =_\alpha [N'/x]\, M'$. $(x \notin \{\varkappa, \alpha\})$.

We omit the proof, which is a simple induction on the size of M.

Lemma 10. 1. *If $M \sim M'$ and $N \sim N' = {}_{\alpha}M'$ then $M = {}_{\alpha}N$.*
2. *If $M \lesssim M'$ and $N \lesssim N' = {}_{\alpha}M'$ then $M = {}_{\alpha}N$.*

Proof. We define sets $\mathcal{K}_n$, $\mathcal{I}_n (n \geqslant 0)$ by:

$$\mathcal{K}_0 = \{x\}; \mathcal{K}_{n+1} = \{\lambda\alpha\alpha L' K_n | L \sim L' \text{ for some } L \text{ and } K_n \in \mathcal{K}_n\} (n \geqslant 0).$$
$$\mathcal{I}_n = \mathcal{K}_n \cup \{[I/x] K_n | K_n \in \mathcal{K}_n\} (n \geqslant 0).$$

We use K_n, K_n' etc. to vary over $\mathcal{K}_n$ and I_n etc. to vary over $\mathcal{I}_n$. Then $K_n = {}_{\alpha}K_{n'}$ implies $n = n'$ and $[I_n/x] K_m$ is in $\mathcal{I}_{m+n}$.

In the following, V is some constant, or has the form $\lambda\alpha a (\alpha I)$ or $\lambda x L_1'$ where $L_1 \sim L_1'$ for some L_1, and L' will be some term such that $L \sim L'$ for some L.

A term is of *type A* if it has the form λxxV.

A term is of *type B* if it has the form $\lambda xVL'x$.

A term is of *type C* if it has the form $\lambda xL'K_m$, with $m > 0$.

A term is of *type D* if it has the form $\lambda xVL'K_m$, with $m > 0$.

A term is of *type E* if it has the form $\lambda xK_m V$, with $m > 0$.

It is not hard to show that if $M \sim M'$ then M' is of type A iff the last rule used in the proof of $M \sim M'$ was one of III1, III1 or IV1; of type B iff it was one of VI, VII or VIII; and of one of types C, D or E iff it was V.

Let $(a)_n$ be the statement that if $M \sim M'$, $N \sim N'$ have proofs of total size (= no. of steps) n then if $M' = {}_{\alpha}N'$, $M = {}_{\alpha}N$.

Let $(b)_n$ be the statement that if $M \lesssim M'$, $N \lesssim N'$ have proofs of total size n then if $M' = {}_{\alpha}N'$, $M = {}_{\alpha}N$.

Let $(c)_n$ be the statement that if $M \sim M'$, $N \sim N'$ have proofs of combined size n then if $M'|I_n = N'|I_n'$,for some $I_n, I_n' \in \mathcal{I}_n$ then $M = {}_{x}N$ and $I_n = {}_{\alpha}I_n'$.

We will prove that if $(c)_n$ then $(a)_n$, that if $(a)_m$ and $(c)_m$ for all $m < n$ then $(b)_n$, and if $(a)_m$ and $(b)_m$ and $(c)_m$ for all $m < n$ then $(c)_n$. The Lemma is then immediate.

(i) Suppose $(c)_n$ and that $M \sim M'$, $N \sim N'$ have proofs of total size n and $M' = {}_{\alpha}N'$. Then $(c)_n$ applies with $M'|I_n = (M'x)$ and $(N'|I_n') = (N'x)$.

(ii) Suppose $(a)_m$ and $(c)_m$ for all $m < n$, $M \lesssim M'$, and $N \lesssim N'$ have proofs of total size n, and $M' = {}_{\alpha}N'$. We divide the proof into cases according to the last rule used in the proof of $M \lesssim M'$.

II2. Here $M = a$ and $M' = \lambda\alpha a (\alpha I)$ for some a. The last rule used in the proof of $N \lesssim N'$ clearly cannot be III1 or IV2. If it is IX then we must have $N \sim N''$ and $N' = (N''; I)$ where the last rule in the proof of $N \sim N''$ was V as N' is an abstraction. A comparison of the positions in M' and N'' of α shows that this is impossible. Lastly, if it is II2 the result is immediate.

III2, IV2. This is similar but IV2 uses $(a)_{m-2}$.

IX. By symmetry, we need only consider the case where the last rule used to prove $N \lesssim N'$ is also IX. Then, for some M'', N'' $M \sim M''$ and $N \sim N''$ have proofs of total size $n-2$, $M' = (M''|I)$ and $N' = (N''|I)$. By $(c)_{m-2}$ it follows that $M'' = {}_{\alpha}N''$.

(iii) Suppose $(a)_m$ and $(b)_m$ and $(c)_m$ for all $m < n$, $M \sim M'$, $N \sim N'$ have proofs of combined size n and $M'|I_n = {}_\alpha N'|I'_n$ for some I_n, I'_n in some $\mathcal{I}_n$.

First we assume that neither $M \sim M'$ nor $N \sim N'$ follows from X.

Suppose, $M'|I_n = M'I_n$ and $N'|I'_n = N'; I_n$. If $N \sim N'$ follows from I, the result is immediate. If N' is of type A, then $M'I_n = {}_\alpha I'_n V$, with a V as above. This is a contradiction as $I_n = {}_\alpha V$ is impossible. The same objection rules out type E.

If N' is of type B, then $M'I_n = {}_\alpha VL'I'_n$ which is impossible as M' is a value. This also rules out type D. Lastly, if N' is of type C then $M'I_n = {}_\alpha L'[I'_n/x]\, K_m = L'I'_{n+m}$ for some I'_{n+m} and then $I_n = {}_\alpha I'_{n+m}$ a contradiction, as $m > 0$.

The case $M'|I_n = M'; I_n$ and $N'|I_n = N'I'_n$ is similar.

Suppose $M'|I_n = M'; I_n$ and $N'|I_n = N'; I_n$. If either $M \sim M'$ or $N \sim N'$ follows from I the result is immediate; otherwise, the proof splits into cases according to the type of M'.

A. Here $M' = \lambda x x V$ for some V, as above, and $M \sim M'$ follows by one of III1, IIII or IV1 and $M'I_n = I_n V$.

If N' is of type A we see that $N \sim N'$ follows from whichever of III1, IIII or IV1, $M \sim M'$ does and the result is immediate, possibly using $(a)_{n-2}$.

If N' is of type B we have $I_n V = {}_\alpha V'L'I'_n$ for suitable V', L', contradicting the fact that I_n is a value.

If N' is of type C we have $I_n V = {}_\alpha L'I_{m+n}$ for some I_{m+n} which is impossible.

If N' is of type D, the argument is the same as for type B.

If N' is of type E, we get $I_n V = {}_\alpha I'_{n+m} V'$ for some I'_{n+m}, V' which is a contradiction as $m > 0$.

B. Here $M' = \lambda x V L' x$ and $M'; I_n = VL'I_n$. The proof divides according to the type of N'.

A. By symmetry from the case AB above.

B. Here $N \sim N'$ follows from whichever of VI, VII, VIII proves $M \sim M'$ and the result is immediate using either $(a)_{n-2}$ or $(b)_{n-2}$.

C. Here $VL'I_n = {}_\alpha L'I'_{m+n}$ for suitable L', I'_{m+n}, a clear contradiction.

D. Here $VL'I_n = {}_\alpha V'L'I'_{m+n}$, for suitable V', L', I'_{m+n}, a contradiction as $m > 0$.

E. Here $VL'I_n = {}_\alpha I'_{m+n} V'$, for suitable I'_{m+n}, a contradiction.

C, D, E. If $N \sim N'$ does not follows from V, the result follows by symmetry. Otherwise, $M = (M_1\, M_2)$, and the last step in the proof of $M \sim M'$ has the form:

$$\frac{M_1 \sim M'_1 \quad M_2 \sim M'_2}{(M_1 M_2) \sim \lambda x (M'_1 |\, \lambda \alpha x\, M'_2 x)}$$

and similarly for N.

Then $M; I_n = M'_1|(\lambda\alpha\, \alpha M'_2\, I_n)$ and $N; I'_n = N'_1|(\lambda\alpha\alpha N'_2\, I'_n)$ and so by hypothesis. $M'_1 = {}_\alpha N'_1$ and $\lambda\alpha\alpha M'_2\, I_n = {}_\alpha \lambda\alpha\alpha N'_2\, I'_n$. Then by hypothesis, $M_i = {}_\alpha N_i$, so $M = {}_\alpha N$, and $I_n = I'_n$ concluding this part of the proof.

Finally, suppose $M'|I_n = M'I_n$ and $N'|I_n = N'I'_n$. Then $I_n = {}_\alpha I'_n$ is immediate and $M' = {}_\alpha N'$. So $M'; x = {}_\alpha N'; x$ and by the previous part of the proof, $M = {}_\alpha N$.

It only remains to consider the possibility that one of $M \sim M'$ or $N \sim N'$ follows

from X. Suppose, w.l.o.g. that $M \sim M'$ does and so $M' = \lambda\varkappa M'_1\ \varkappa$ where $M \sim M'_1$. If $M'|I_n = M'; I_n$ the result follows by the induction hypothesis. Otherwise, if $N'|I'_n = (N'I'_n)$ then $I_n = {}_\alpha I_{n'}$ and $M' = {}_\alpha N'$. Therefore $M'_1\ \varkappa = M'; \varkappa = {}_\alpha N'; \varkappa$ and so, by induction hypothesis, $M = {}_\alpha N$. So the remaining case is $M'|I_n = M'I_n$ and $N'|I'_n = = N'; I'_n$. Inspection of the various types rules them all out leaving only the possibility that $N \sim N'$ follows by X which is handled by the induction hypothesis, concluding the proof.

Lemma 11. *Suppose $M \sim M'$ and N' is obtained from M' by contracting one β- or δ-redex. Then for some L either $M \sim L = {}_\alpha N'$ or else $N \sim L = {}_\alpha N'$, where N is obtained from M by contracting one β-redex, or δ-redex. The same statement holds with $\sim$ replaced by $\underset{\sim}{\circ}$.*

We omit the proof which is a straightforward induction on the number of steps in the proof of $M \sim M'$, or $M \underset{\sim}{\circ} M'$.

Proof of Theorem 6. It is straightforward to prove that $\lambda_N^{\mathcal{L}} \vdash M = N$ implies $\lambda_V^{\mathcal{L}'} \vdash \underline{M} = \underline{N}$ from which it follows immediately that $\lambda_V^{\mathcal{L}'} \vdash \underline{M}I = \underline{N}I$ and $\lambda_N^{\mathcal{L}''} \vdash \vdash \underline{M} = \underline{N}$ and then that $\lambda_N^{\mathcal{L}''} \vdash \underline{M}I = \underline{N}I$.

Conversely suppose $\lambda_N^{\mathcal{L}''} \vdash \underline{M}I = \underline{N}I$. Then there is a term Z such that $\lambda_N^{\mathcal{L}''} \vdash \vdash \underline{M}I \geqslant Z$ and $\lambda_N^{\mathcal{L}''} \vdash \underline{N}I \geqslant Z$. As $M \underset{\sim}{\circ} \underline{M}I$, by Lemma 11 there is a term M_1 such that $\lambda_N^{\mathcal{L}} \vdash M \geqslant M_1$ and $M_1 \underset{\sim}{\circ} M'_1 = {}_\alpha Z$ for some term M'_1. Similarly, there are terms M_2 and M'_2 such that $\lambda_N^{\mathcal{L}'} \vdash N \geqslant M_2$ and $M_2 \underset{\sim}{\circ} M'_2 = {}_\alpha Z$. By Lemma 10, $M_1 = {}_\alpha M_2$. Therefore $\lambda_N^{\mathcal{L}} \vdash M = N$ concluding the proof.

A rather more complex simulation of call-by-name by call-by-value works in the general case where the constants are not divided up into basic and functional ones. It results from the following mapping $M \mapsto \underset{\sim}{M}$

$$\underset{\sim}{a} = \lambda\varkappa\varkappa\,(\lambda\pi\pi K a)$$
$$\underset{\sim}{x} = x$$
$$\underset{\sim}{\lambda x M} = \lambda\varkappa\varkappa\,(\lambda\pi\pi\,(KI)\,(\lambda x \underset{\sim}{M}))$$
$$\underset{\sim}{M}\underset{\sim}{N} = \lambda\varkappa \underset{\sim}{M}\,(\lambda\alpha\,(\alpha\,(KI))\,((\varkappa K)\,(\lambda\delta\,(\underset{\sim}{N}I)\,(KI))\,(\lambda\delta\underset{\sim}{N})\,I)\,\varkappa))$$

where $K = \lambda x\lambda y x$ and $I = \lambda x x$.

We have only checked the analogues of Theorems 4 and 5.

Acknowledgments

This work was carried out during a visit to Syracuse University, for which I would like to thank Professor J. A. Robinson, and then on a visit to Stanford University, for which I would like to thank Professor J. McCarthy, and finished at Edinburgh

University, with the aid of an S.R.C. grant under the direction of Dr. R. M. Burstall. The following people also provided help, both direct and indirect: M. Gordon, R. Milner, L. Morris, J. Reynolds, J. Schwarz, L. Stephenson and C. Wadsworth. I would also like to thank E. Kerse for doing the typing.

References

[1] H. P. Barendregt, Some extensional term models for combinatory logics and λ-calculi Ph. D. thesis (University of Utrecht, 1971).
[2] H. B. Curry, and R. Feys, Combinatory logic, Vol. 1, North Holland, Amsterdam (1958).
[3] M. J. Fischer, Lambda calculus schemata, Proc. of an ACM Conference on Proving Assertions about Programs, Las Cruces (1972).
[4] N. D. Goodman, A simplification of combinatory logic, Journal of Symbolic Logic, Vol. 37, No. 2 (1972).
[5] P. J. Landin, The mechanical evaluation of expressions, Computer Journal, Vol. 6, No. 4 (1964).
[6] P. J. Landin, A correspondence between ALGOL 60 and Church's lambda notation, Comm. ACM., Vol. 8, Nos. 2 and 3 (1965).
[7] P. J. Landin, A lambda-calculus approach, Advances in Programming and Non-Numerical Computation (Pergamon Press, 1966).
[8] C. McGowan, The correctness of a modified SECD machine, Second ACM symposium on theory of computing (1970).
[9] J. H. Morris, Lambda-calculus models of programming languages, MAC-TR-57, Project MAC, MIT, Cambridge, Mass. (1968).
[10] L. Morris, The next 700 programming language descriptions (Unpublished, 1971).
[11] J. C. Reynolds, GEDANKEN — A simple typeless language based on the principle of completeness and the reference concept, Comm. of the ACM., Vol. 13, No. 5 (1970).
[12] J. C. Reynolds, Definitional interpreters for higher-order programming languages, Presented at ACM National Meeting, Boston (1972).
[13] J. C. Reynolds, Personal communication (1972).
[14] D. Scott, and C. Strachey, Toward a mathematical semantics for computer languages, Proceedings of the Symposia on Computers and Automata, Microwave Research Institute, Symposia Series Vol. 21 (1971) (Polytechnic Institute of Brooklyn, to appear).
[15] C. Strachey, and C. P. Wadsworth, Continuations — a mathematical semantics for handling full jumps, Technical Monograph PRG-11, Oxford, (1974).
[16] J. Vuillemin, Implementations of recursion in a simple programming language, Proc 5th Annual ACM symposium on computing (1973).
[17] C. P. Wadsworth, Semantics and pragmatics of the lambda-calculus. Ph. D. thesis (University of Oxford, 1971).
[18] P. Wegner, Programming Languages, Information Structures and Machine Organisation, (McGraw-Hill, New York, 1968).
[19] J. M. Wozencraft and A. Evans, Notes on programming linguistics, MIT, Cambridge, Mass. (1971).