Theoretical Computer Science 5 (1977) 223–255.
© North-Holland Publishing Company

LCF CONSIDERED AS A PROGRAMMING LANGUAGE

G.D. PLOTKIN

Department of Artificial Intelligence, University of Edinburgh, Hope Park Square, Meadow Lane, Edinburgh EH8 9NW, Scotland

Communicated by Robin Milner
Received July 1975

Abstract. The paper studies connections between denotational and operational semantics for a simple programming language based on LCF. It begins with the connection between the behaviour of a program and its denotation. It turns out that a program denotes $\perp$ in any of several possible semantics iff it does not terminate. From this it follows that if two terms have the same denotation in one of these semantics, they have the same behaviour in all contexts. The converse fails for all the semantics. If, however, the language is extended to allow certain parallel facilities, behavioural equivalence does coincide with denotational equivalence in one of the semantics considered, which may therefore be called "fully abstract". Next a connection is given which actually determines the semantics up to isomorphism from the behaviour alone. Conversely, by allowing further parallel facilities, every r.e. element of the fully abstract semantics becomes definable, thus characterising the programming language, up to interdefinability, from the set of r.e. elements of the domains of the semantics.

1. Introduction

We present here a study of some connections between the operational and denotational semantics of a simple programming language based on LCF [3, 5]. While this language is itself rather far from the commonly used languages, we do hope that the kind of connections studied will be illuminating in the study of these languages too.

The first connection is the relation between the behaviour of a program and the nature of its denotation. For us a program will be a term of ground type of an LCF whose extra-logical constants are arithmetical. (Other possibilities for these constants are considered later.) Its behaviour is given by an evaluation function which specifies whether or not it terminates and its value when it does. It will be shown that a program terminates iff it does not denote $\perp$. Its value, if it terminates, is that specified by its denotation.

This kind of theorem has been proved in other cases. For example, systems of recursion equations have been considered in [10], pure LISP in [1], and MAL, a language of greater expressive power than ALGOL 60, in [4].

However more than one denotational semantics can be connected to a given operational semantics in this way. The second connection allows rather more restrictive requirements. Two terms (more generally, two members of the same syntactic category in the specification of the language) are denotationally equivalent iff in any environment they denote the same thing. It is natural to specify that two terms are operationally equivalent if either can be removed from a program and replaced by the other without altering the behaviour of the program. These equivalences are induced by similarly defined quasi-orderings. We find that semantic equivalence (quasi-ordering) implies operational equivalence (quasi-ordering) but not conversely. The reason is that the denotational semantics allows functions which require parallel facilities to realise while the programming language is deterministic. However, we can now discriminate between different denotational semantics by considering their closeness to the operational semantics according to the various equivalence relations, or quasi-orderings.

Complete identity of all the relations is obtained by adding limited parallel facilities to the programming language. The kind of parallelism considered does not allow inconsistent (different) results in parallel computations. It remains an open problem to find a denotational semantics whose quasi-ordering corresponds exactly to the deterministic operational one.

Although it is probable that more than one denotational semantics can give the same quasi-ordering as the operational one, nevertheless the denotational semantics which we give for the language with some parallel facilities can be characterised, up to isomorphism, as the least such denotational semantics, in a sense to be described.

Lastly we might consider how the operational semantics is determined by the denotational one. It is clear that the first connection considered answers this question since, if the connection is to hold, the evaluation function is completely specified. A more interesting question along the same lines is whether keeping the domains of the denotational semantics fixed we ought to add anything to the programming language. The class of recursively enumerable elements of the domains involved (see below) form a natural class to consider. It turns out that while all the closed terms do define r.e. elements not all the r.e. elements are definable. However, adding one more parallel facility keeps all the nice properties so far established and also allows all, and only, the r.e. elements to be defined. Of course this is all really recursion theory, and indeed continues some work of Scott [7] who needed a discontinuous search operator for his definability result. But it also indicates how one might use denotational semantics to find facilities which may have been unintentionally left out by the language designer.

Of course instead of inventing the programming language, we could have interpreted these results directly in terms of LCF. However, we feel that they exemplify a programme for the investigation of programming languages which, in fact, we learned from Wadsworth [11, 12]. In particular, we regard Theorems 3.1 and 4.3 as analogous for PCF of corresponding theorems of his for the λ-calculus. Theorem 3.1 corresponds to that fact that in the standard Scott model of the

$\lambda\beta\eta$-calculus a closed term denotes $\perp$ iff it has no head normal form. Head normal forms are a kind of weak normal form which capture our intuition about non-termination for the λ-calculus better than the usual normal forms. Theorem 4.3(2) corresponds to the fact that in that model two terms have the same denotation iff embedding one in a context gives a term with a head normal form iff embedding the other in the same context also does; Theorem 4.3(1) corresponds to a similar fact. However, apart from the use of the $Y^{(n)}$'s below, the proof methods are not directly analogous.

2. The programming language, PCF

PCF is a programming language for computable functions, based on LCF, Scott's logic of computable functions [3]. The syntax of PCF is that of LCF, except that only terms are considered and some of the terms are singled out as programs.

The set of *types* is the least set containing ι, o and containing $(\sigma \to \tau)$ whenever it contains σ and τ. The Greek letters σ and τ range over types, $(\sigma_1, \ldots, \sigma_n, \tau)$ abbreviates $(\sigma_1 \to (\sigma_2 \to \cdots (\sigma_n \to \tau) \cdots))$ $(n \geq 0)$ and ι and o are the *ground* types (of individuals and truthvalues respectively). The *level* of a type is defined by: $\text{level}(\iota) = \text{level}(o) = 0$ and $\text{level}(\sigma \to \tau) = 1 + \max(\text{level}(\sigma), \text{level}(\tau))$.

Starting with a collection $\mathscr{L}$ of constants, each having a fixed type, and denumerably many variables α_i^σ $(i \geq 0)$ of each type, the $\mathscr{L}$*-terms* are given by the rules:

(1) Every variable α_i^σ is an $\mathscr{L}$-term of type σ.

(2) Every constant of type σ is an $\mathscr{L}$-term of type σ.

(3) If M and N are $\mathscr{L}$-terms of types $(\sigma \to \tau)$ and σ respectively then (MN) is an $\mathscr{L}$-term of type τ.

(4) If M is an $\mathscr{L}$-term of type τ then $(\lambda\alpha_i^\sigma M)$ is one of type $(\sigma \to \tau)$ $(i \geq 0)$.

When $\mathscr{L}$ is understood from the context it need not be used as a prefix. The letters M and N, and, occasionally, other capital letters, possibly with suffices to indicate type, will range over $\mathscr{L}$-terms, c over $\mathscr{L}$ and α over the variables, x, y, z will sometimes be meta-variables of type ι and p and q will sometimes be meta-variables of type o.

$FV(M)$, the set of free variables in M is defined by:

$$FV(\alpha_i^\sigma) = \{\alpha_i^\sigma\}; \qquad FV(\iota) = \emptyset; \qquad FV((MN)) = FV(M) \cup FV(N);$$

$$FV((\lambda\alpha_i^\sigma M)) = FV(M) \setminus \{\alpha_i^\sigma\}.$$

The term M is *closed* if $FV(M) = \emptyset$ and *open* otherwise.

Terms of the form (MN) are called *combinations* and sometimes the brackets are dropped, when they are understood as associating to the left; terms of the form $(\lambda\alpha M)$ are *abstractions*. *Contexts* are terms with one or more "holes" in them written as $C[\cdot, \ldots, \cdot]$. They can be filled with terms of the appropriate type to give a term: $C[M_1, \ldots, M_n]$. We omit a formal definition.

$[M_\sigma/\alpha_i^\sigma]N$ is the result of substituting the term M_σ for all free occurrences of α_i^σ in N, making appropriate changes in the bound variables of N so that no free variables of M_σ become bound. We omit a formal definition and the derivation of elementary properties.

The *programs* are the closed $\mathscr{L}$-terms of ground type. The idea is that the ground types are the datatypes and programs produce data (via the operational semantics). The other terms are significant only as subterms of programs.

All languages, $\mathscr{L}$, considered include $\mathscr{L}_0$, the set of *standard* constants. These, together with their types, are:

$$tt : o,$$

$$ff : o,$$

$$\supset_\iota : (o, \iota, \iota, \iota),$$

$$\supset_o : (o, o, o, o),$$

$$Y_\sigma : ((\sigma \to \sigma) \to \sigma) \text{ (one for each } \sigma).$$

Generally we will be interested in a language $\mathscr{L}_A$ for arithmetic which also has:

$$k_n : \iota \text{ (one for each integer } n \geq 0),$$

$$(+1) : (\iota \to \iota),$$

$$(-1) : (\iota \to \iota),$$

$$Z : (\iota \to o).$$

The operational semantics is given by a partial function, *Eval*$_\mathscr{L}$, which gives constants from programs. Eval$_\mathscr{L}$ is defined by means of an *immediate reduction relation*, $\to_\mathscr{L}$ between terms by:

$$\text{Eval}_\mathscr{L}(M) = c \text{ iff } M \overset{*}{\underset{\mathscr{L}}{\to}} c, \text{ for any program } M \text{ and constant } c.$$

Here $\to_\mathscr{L}^*$ is the transitive reflexive closure of $\to_\mathscr{L}$. For this definition to make sense it is necessary that $M \to_\mathscr{L}^* c$ and $M \to_\mathscr{L}^* c'$ implies that c and c' are identical. Notice that for constants, c, of ground type Eval$_\mathscr{L}(c) = c$.

The definition of $\to_\mathscr{L}$ depends on $\mathscr{L}$, and is given by some rules. These will always include:

(I1) $\quad \supset_\sigma tt M_\sigma N_\sigma \to M_\sigma, \quad \supset_\sigma ff M_\sigma N_\sigma \to N_\sigma \quad (\sigma \text{ ground}),$

(I2) $\quad Y_\sigma M \underset{\mathscr{L}}{\to} M(Y_\sigma M),$

(I3) $\quad ((\lambda \alpha M)N) \underset{\mathscr{L}}{\to} [N/\alpha]M,$

(II1) $$\frac{M \underset{\mathscr{L}}{\rightarrow} M'}{(MN) \underset{\mathscr{L}}{\rightarrow} (M'N)},$$

(II2) $$\frac{M_o \underset{\mathscr{L}}{\rightarrow} M_o'}{(\supset_\sigma M_o) \underset{\mathscr{L}}{\rightarrow} (\supset_\sigma M_o')}.$$

If $\mathscr{L} = \mathscr{L}_A$ we also add:

(I4) $(+1)k_m \underset{\mathscr{L}_A}{\rightarrow} k_{m+1} \quad (m \geqslant 0)$,

(I5) $(-1)k_{m+1} \underset{\mathscr{L}_A}{\rightarrow} k_m \quad (m \geqslant 0)$,

(I6) $Zk_0 \underset{\mathscr{L}_A}{\rightarrow} tt \quad Zk_{m+1} \underset{\mathscr{L}_A}{\rightarrow} ff$,

(II3) $$\frac{N \underset{\mathscr{L}_A}{\rightarrow} N'}{(MN) \underset{\mathscr{L}_A}{\rightarrow} (MN')} \quad \text{(if } M \text{ is } +1, -1 \text{ or } Z\text{)}.$$

The relation $\rightarrow_{\mathscr{L}_A}$ is actually a partial function, which is undefined on constants and so $\mathrm{Eval}_{\mathscr{L}_A}$ is well-defined. Generally $\mathrm{Eval}_{\mathscr{L}_A}$ is abbreviated to Eval_A and similar abbreviations will be made elsewhere. This style of operational semantics is different from the SECD style, although in fact an equivalent SECD-type semantics could be given (cf. [6]). Here operational semantics is part of the provability relation, e.g. if $M \rightarrow_{\mathscr{L}_A} N$ then $M = N$ is provable in LCF togther with the axioms for arithmetic [5]. Notice that, because of (I3), (II1) function calls are by name rather than value.

Turning now to the denotational semantics, we briefly recall some definitions and facts. A fuller treatment containing proofs and other definitions can be found in [3] which we are following with a few changes.

A subset X of a set D partially ordered by $\sqsubseteq$ is *directed* iff it is non-empty and every pair of elements of X have an upper bound in X.

A partial order $(D, \sqsubseteq)$ is a *complete partial order* (cpo) iff it has a minimum element $\perp_D$ (or $\perp$ if that causes no confusion) and every directed subset, X, of D has a least upper bound, $\bigsqcup_D X$ (or $\bigsqcup X$ if that causes no confusion). (In [3] Milner takes closure under lub's of denumerable chains rather than directed sets, but, as he remarks, all of that paper goes through with closure under directed sets.)

A function $f: D \rightarrow E$ from a cpo D to another E is *continuous* iff $f(\bigsqcup_D X) = \bigsqcup_E \{f(x): x \in X\}$ for all directed subsets of D. Then the set $[D \rightarrow E]$ of all continuous functions in $(D \rightarrow E)$ is itself a cpo under the induced pointwise ordering.

At the cost of some artificiality we could use complete lattices instead of cpo's. Example 3.5 shows what goes wrong if one uses them in the natural way. (A complete lattice is a po in which every subset has a lub; if D, E are complete lattices (and hence cpo's), so is $[D \to E]$.)

A *collection of domains for PCF* is a family $\{D_\sigma\}$ of cpo s, one for each type, such that $D_{\sigma\to\tau} = [D_\sigma \to D_\tau]$. It is *standard* if D_o is the truthvalue cpo $\mathbf{T} = \{\bot, tt, ff\}$ ordered as in Fig. 1.

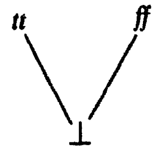

Fig. 1.

An *interpretation* of a language $\mathscr{L}$ is a collection, $\{D_\sigma\}$ of domains for PCF together with a mapping

$$\mathscr{A} : \mathscr{L} \to \bigcup \{D_\sigma\}$$

which is type-respecting; that is if c is a constant of type σ, then $\mathscr{A}[\![c]\!]$ is in D_σ. Notice the use of decorated square brackets when passing from syntactic entities to denotational ones.

Such an interpretation is *standard* if $\{D_\sigma\}$ is, $\mathscr{L} \supseteq \mathscr{L}_0$ and:

$$\mathscr{A}[\![tt]\!] = tt,$$

$$\mathscr{A}[\![ff]\!] = ff,$$

$$\mathscr{A}[\![\supset_\sigma]\!](p)(x)(y) = \begin{cases} x & (\text{if } p = tt) \\ y & (\text{if } p = ff) \\ \bot & (\text{if } p = \bot) \end{cases} \quad (p \in D_o,\ x, y \in D_\sigma \text{ and } \sigma \text{ ground}),$$

$$\mathscr{A}[\![Y_\sigma]\!](f) = \bigsqcup_{n\geq 0} {}_{D_\sigma} f^n(\bot) \qquad (f \in D_{(\sigma\to\sigma)}).$$

Each $\mathscr{A}[\![c]\!]$ is in D_σ, for constants c in $\mathscr{L}_0$ of type σ.

The *standard collection of domains for arithmetic* is the standard collection with $D_\iota = \mathbf{N} = \{\bot, 0, 1, \ldots\}$ ordered as in Fig. 2.

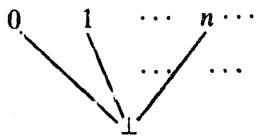

Fig. 2.

The *standard interpretation of the arithmetic language*, $\mathscr{L}_A$ is the standard collection, $\{D_\sigma\}$, of domains for arithmetic together with the standard interpretation,

$$\mathscr{A}_A : \mathscr{L}_A \to \bigcup \{D_\sigma\}$$

such that:

$$\mathscr{A}_A [\![k_n]\!] = n \qquad (n \geq 0),$$

$$\mathscr{A}_A [\![(+1)]\!](x) = \begin{cases} x+1 & (x \geq 0) \\ \perp & (x = \perp) \end{cases} \qquad (x \in D_\iota),$$

$$\mathscr{A}_A [\![(-1)]\!](x) = \begin{cases} x-1 & (x \geq 1) \\ \perp & (x = \perp, 0) \end{cases} \qquad (x \in D_\iota),$$

$$\mathscr{A}_A [\![Z]\!](x) = \begin{cases} tt & (x = 0) \\ ff & (x > 0) \\ \perp & (x = \perp) \end{cases} \qquad (x \in D_\iota).$$

Given an interpretation $\langle \{D_\sigma\}, \mathscr{A} \rangle$ of $\mathscr{L}$ we obtain a denotational semantics $\hat{\mathscr{A}}$ for $\mathscr{L}$.

First the set, Env, of *environments* is the set of type-respecting functions from the set of variables to $\bigcup \{D_\sigma\}$. It is ranged over by ρ. If $x \in D_\sigma$, then $\rho[x/\alpha_i]$ is that environment ρ' such that:

$$\rho' [\![\alpha]\!] = \begin{cases} d & (\alpha = \alpha_i^\sigma), \\ \rho [\![\alpha]\!] & (\alpha \neq \alpha_i^\sigma). \end{cases}$$

The denotational semantics $\hat{\mathscr{A}} : \text{Terms} \to (\text{Env} \to \bigcup \{D_\sigma\})$ is defined by:

$$\hat{\mathscr{A}} [\![\alpha_i^\sigma]\!](\rho) = \rho [\![\alpha_i^\sigma]\!],$$

$$\hat{\mathscr{A}} [\![c]\!](\rho) = \mathscr{A} [\![c]\!],$$

$$\hat{\mathscr{A}} [\![(MN)]\!](\rho) = \hat{\mathscr{A}} [\![M]\!](\rho)(\hat{\mathscr{A}} [\![N]\!](\rho)),$$

$$\hat{\mathscr{A}} [\![(\lambda \alpha_i^\sigma M)]\!](\rho)(x) = \hat{\mathscr{A}} [\![M]\!](\rho[x/\alpha_i^\sigma]). \quad (x \in D_\sigma).$$

This is a good definition and if $\rho \in \text{Env}$, $\hat{\mathscr{A}} [\![M]\!](\rho) \in D_\sigma$. Further if $\mathscr{A}$ is a standard interpretation all the axioms and rules of LCF are satisfied and $\mathscr{A}_A$ also satisfies the axioms for arithmetic. In particular, if $M \to_{\mathscr{L}_A} N$ then $\hat{\mathscr{A}}_A [\![M]\!](\rho) = \mathscr{A}_A [\![N]\!](\rho)$ for all ρ.

Here and later we confuse $\mathscr{A}$ with $\langle \{D_\sigma\}, \mathscr{A} \rangle$ and even $\hat{\mathscr{A}}$.

The following three assertions are made without proof.

If $\rho[\![\alpha]\!] = \rho'[\![\alpha]\!]$ for all α in $FV(M)$, then $\hat{\mathscr{A}}[\![M]\!](\rho) = \hat{\mathscr{A}}[\![M]\!](\rho')$. For terms M and N, and any ρ, $\hat{\mathscr{A}}[\![[M_\sigma/\alpha_i^\sigma]N]\!](\rho) = \hat{\mathscr{A}}[\![N]\!](\rho[\hat{\mathscr{A}}[\![M]\!](\rho)/\alpha_i^\sigma])$. If $\hat{\mathscr{A}}[\![M]\!](\rho) = \hat{\mathscr{A}}[\![N]\!](\rho)$ for all ρ then for any context $C[\;]$ for which M and N have the appropriate type, $\hat{\mathscr{A}}[\![C[M]]\!](\rho) = \hat{\mathscr{A}}[\![C[N]]\!](\rho)$ for all ρ.

Sometimes we will say that M *defines* (or *denotes*) f in the environment ρ, if $\hat{\mathscr{A}}[\![M]\!](\rho) = f$. The reference to ρ is omitted if M is closed. The *undefined environment*, $\perp$, sends α_i^σ to $\perp_{D_\sigma}$.

3. Termination of PCF programs

The behaviour of a program, M, is determined by whether it terminates and its value when it does; that is whether $\text{Eval}_A(M)$ exists and which constant it is, if it does. From remarks made above, if $\text{Eval}_A(M) = c$, then $\hat{\mathscr{A}}_A[\![M]\!](\perp) = \mathscr{A}[\![c]\!]$. So here the behaviour of a terminating program determines its denotation. Our aim is to fill the gap and show that nonterminating programs denote $\perp$, demonstrating:

Theorem 3.1. *For any $\mathscr{L}_A$-program M and constant c,* $\text{Eval}_A(M) = c$ *iff* $\hat{\mathscr{A}}_A[\![M]\!](\perp) = \mathscr{A}_A[\![c]\!]$.

It is important here that a ground constant cannot denote $\perp$.

Theorem 3.1 could be proved directly by first establishing it for terms not containing any Y_σ and then using Lemma 3.2 below. However, there is a more flexible method, borrowed from proof theory [9], which allows easy extensions to languages other than $\mathscr{L}_A$, as will be seen. In proof theory the disadvantage of the method is that it uses strong principles of proof and does not (directly) yield information about certain proof-theoretic ordinals, as do methods involving proof by induction on these ordinals. These do not constitute disadvantages for us, since we have no interest in using weak methods, and since no use is known for the extra information provided by the other methods, which are also more difficult to carry out [2].

The method is simply a proof of the required property by structural induction on terms, which requires a suitable induction hypothesis at higher types. Predicates, Comp_σ, are defined by induction on types by:

(1) If M_σ is a program then M_σ has property Comp_σ iff $\hat{\mathscr{A}}_A[\![M]\!](\perp) = \mathscr{A}_A[\![c]\!]$ implies $\text{Eval}_A(M) = c$.

(2) If $M_{(\sigma\to\tau)}$ is a closed term it has property $\text{Comp}_{(\sigma\to\tau)}$ iff whenever N_σ is a closed term with property Comp_σ, $(M_{(\sigma\to\tau)}N_\sigma)$ has property Comp_τ.

(3) If M_σ is an open term with free variables $\alpha_1, \ldots, \alpha_n$ of types $\sigma_1, \ldots, \sigma_n$ then it has property Comp_σ iff $[N_1/\alpha_1]\cdots[N_n/\alpha_n]M_\sigma$ has property Comp_σ whenever $N_1, \ldots, N_n$ are closed terms having properties $\text{Comp}_{\sigma_1}, \ldots, \text{Comp}_{\sigma_n}$ respectively.

A term M_σ is *computable* iff it has property Comp_σ. Clearly if $M_{(\sigma\to\tau)}$ and N_σ are closed computable terms, so is $(M_{(\sigma\to\tau)}N_\sigma)$ and also a term M_σ, where $\sigma = (\sigma_1,\ldots,\sigma_n,\sigma')$ is computable iff $\underset{\sim}{M}_\sigma N_1,\ldots,N_n$ is computable whenever $N_1,\ldots,N_n$ are closed computable terms of types $\sigma_1,\ldots,\sigma_n$ and $\underset{\sim}{M}_\sigma$ is a closed instantiation of M_σ by computable terms. The latter assertion is proved by using clause 3 of the definition if M_σ is open, and then clause 2 n times.

The only difficulty in proving all terms computable is caused by the recursion operators, Y_σ. These can be approximated by certain terms $Y_\sigma^{(n)}$, $(n \geqslant 0)$. Define terms Ω_σ by $\Omega_\iota = Y_\iota(\lambda\alpha_0^\iota\alpha_0^\iota)$, $\Omega_o = Y_o(\lambda\alpha_0^o\alpha_0^o)$ and $\Omega_{(\sigma\to\tau)} = \lambda\alpha_0^\sigma\Omega_\tau$. Then the terms $Y_\sigma^{(n)}$ are defined by putting $Y_\sigma^{(0)} = \Omega_{((\sigma\to\sigma)\to\sigma)}$ and $Y_\sigma^{(n+1)} = (\lambda\alpha_0^{(\sigma\to\sigma)}(\alpha_0^{(\sigma\to\sigma)}(Y_\sigma^{(n)}\alpha_0^{(\sigma\to\sigma)})))$.

Then $\hat{\mathscr{A}}[\![Y_\sigma]\!](\perp) = \sqcup\{\hat{\mathscr{A}}[\![Y_\sigma^{(n)}]\!]: n \geqslant 0\}$ expresses the LHS as the union of a denumerable chain, for any standard interpretation $\mathscr{A}$.

We also need some syntactic information. Let $\leqslant$ be the least relation between terms such that

(1) $\Omega_\sigma \leqslant M_\sigma$ and $Y_\sigma^{(n)} \leqslant Y_\sigma$, for all σ and $n \geqslant 0$.

(2) $M_\sigma \leqslant M_\sigma$.

(3) If $M_{(\sigma\to\tau)} \leqslant M'_{(\sigma\to\tau)}$ and $N_\sigma \leqslant N'_\sigma$ then $(\lambda\alpha N_\sigma) \leqslant (\lambda\alpha N'_\sigma)$ and $(M_{(\sigma\to\tau)}N_\sigma) \leqslant (M'_{(\sigma\to\tau)}N'_\sigma)$.

Lemma 3.2. *If $M \leqslant N$ and $M \to_{\mathrm{A}} M'$ then either $M' \leqslant N$ or else for some N', $N \to_{\mathrm{A}} N'$ and $M' \leqslant N'$.*

Proof. By structural induction on M and cases according to why $M \to_{\mathrm{A}} M'$. □

Lemma 3.3. *Every term is computable.*

Proof. (1) Every variable α is computable since any closed instantiation, $\underset{\sim}{\alpha}$, of it by a computable term is computable.

(2) Every constant other than the Y_σ's is computable. This is clear for constants of ground type. Out of $(+1)$, (-1), Z, $\supset_\iota$, $\supset_o$ we only consider (-1) as an example. It is enough to show $(-1)M_\iota$ computable when M_ι is a closed computable term. Suppose $\hat{\mathscr{A}}_{\mathrm{A}}[\![(-1)M]\!](\perp) = \mathscr{A}_{\mathrm{A}}[\![c]\!]$. Then $c = k_m$ for some m and so $\hat{\mathscr{A}}_{\mathrm{A}}[\![M]\!](\perp) = (m+1)$. Therefore as M is computable, $M \to_{\mathrm{A}}^* k_{(m+1)}$ and so $(-1)M \to_{\mathrm{A}}^* k_m = c$.

(3) If $M_{(\sigma\to\tau)}$ and N_σ are computable, so is $(M_{(\sigma\to\tau)}N_\sigma)$. If $(M_{(\sigma\to\tau)}N_\sigma)$ is closed so are $M_{(\sigma\to\tau)}$ and N_σ and its computability follows from clause (2). If it is open, any closed instantiation L of it by computable terms has the form $(\underset{\sim}{M}_{(\sigma\to\tau)}\underset{\sim}{N}_\sigma)$ where $\underset{\sim}{M}_{(\sigma\to\tau)}$ and $\underset{\sim}{N}_\sigma$ are such instantiations of $M_{(\sigma\to\tau)}$ and N_σ and are therefore themselves computable which in turn implies the computability of L and hence of $(M_{(\sigma\to\tau)}N_\sigma)$.

(4) If M_τ is computable so is $(\lambda\alpha^\sigma M_\tau)$. It is enough to show that the ground term $LN_1\cdots N_n$ is computable when $N_1,\ldots,N_n$ are closed computable terms and L is a

closed instantiation of $(\lambda\alpha^\sigma M_\tau)$ by computable terms. Here L must have the form $(\lambda\alpha^\sigma \underset{\sim}{M}_\tau)$ where $\underset{\sim}{M}_\tau$ is an instantiation of all the free variables of M_τ, except α^σ, by closed computable terms. If $\hat{\mathscr{A}}_A[\![LN_1\cdots N_n]\!](\perp)=\mathscr{A}_A[\![c]\!]$, then we have

$$\hat{\mathscr{A}}_A[\![[N_1/\alpha]\underset{\sim}{M}_\tau N_2\cdots N_n]\!](\perp)=\hat{\mathscr{A}}_A[\![LN_1\cdots N_n]\!](\perp)=\mathscr{A}_A[\![c]\!].$$

But $[N_1/\alpha]\underset{\sim}{M}_\tau$ is computable and so too therefore is $[N_1/\alpha]\underset{\sim}{M}_\tau N_2\cdots N_n$. Therefore $LN_1\cdots N_n\rightarrow_A[N_1/\alpha]\underset{\sim}{M}_\tau N_2\cdots N_n\rightarrow_A^* c$, as required.

(5) Each Y_σ is computable. It is enough to prove $Y_\sigma N_1\cdots N_n$ computable when $N_1\cdots N_n$ are closed computable terms and $Y_\sigma N_1\cdots N_n$ is ground. Suppose $\hat{\mathscr{A}}_A[\![YN_1\cdots N_k]\!](\perp)=\mathscr{A}_A[\![c]\!]$. Since $\hat{\mathscr{A}}_A[\![Y_\sigma]\!](\perp)=\bigsqcup_{n\geq 0}\hat{\mathscr{A}}_A[\![Y_\sigma^{(n)}]\!](\perp)$, $\hat{\mathscr{A}}_A[\![Y^{(n)}N_1\cdots N_k]\!](\perp)=\mathscr{A}_A[\![c]\!]$ for some n. Since $\hat{\mathscr{A}}_A[\![\Omega_\sigma]\!](\perp)=\perp$ for σ ground, Ω_σ is computable for σ ground. From this and (1), (3) and (4) proved above it follows that every Ω_σ and $Y_\sigma^{(n)}$ is computable. Therefore $Y^{(n)}N_1\cdots N_k\rightarrow_A^* c$ and so by Lemma 3.2, $Y_\sigma N_1\cdots N_n\rightarrow_A^* c$, concluding the proof. □

Theorem 3.1 follows at once from previous remarks and Lemma 3.3.

The denotational semantics $\hat{\mathscr{A}}_A$ determines the operational semantics Eval_A, if we require Theorem 3.1 to be true, for at most one partial function Eval_A from programs to constants can satisfy the statement of the theorem. Of course, Eval_A can be defined in many ways, some of which, no doubt, give faster algorithms than the one provided by the definition from $\rightarrow_A$ (cf. [10, 11]). In the last section it will be shown how starting with the collection of domains $\{D_\sigma\}$ above one can determine, to an extent, what $\mathscr{L}_A$ and $\mathscr{A}_A$ and hence Eval_A ought to be.

In the other direction, the mere requirement that Theorem 3.1 holds by no means determines $\mathscr{A}_A$ and $\hat{\mathscr{A}}_A$. Some examples follow.

Example 3.4. Take the standard collection of domains $\{D_\sigma\}$ generated from $D_\iota=\{\perp,0,1,\ldots,\infty\}$ ordered as in Fig. 3.

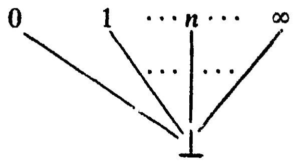

Fig. 3.

The interpretation $\mathscr{A}_1$, is determined by the conditions for $\mathscr{A}_A$ together with: $\mathscr{A}_1[\![(+1)]\!](\infty)=\mathscr{A}_1[\![(-1)]\!](\infty)=\infty$, $\mathscr{A}_1[\![Z]\!](\infty)=f\!f$. Although $\hat{\mathscr{A}}_1$ is not a model of the axioms for arithmetic, it is quite straightforward to modify the previous proof of properties of $\hat{\mathscr{A}}_A$ to show that if $\mathrm{Eval}_A(M)=c$ then $\hat{\mathscr{A}}_1[\![M]\!](\perp)=\mathscr{A}_1[\![c]\!]$. The proof of the converse goes through word for word as above. However, it does not follow that if a program does not terminate that it denotes $\perp$, since it might denote ∞. However this can also be ruled out by amending the first clause in the definition of Comp_σ to:

(1′) If M_σ is a closed term of ground type then M_σ has property Comp_σ iff either $\hat{\mathscr{A}}_1[\![M_\sigma]\!](\bot) = \mathscr{A}_1[\![c]\!]$ and $\mathrm{Eval}_A(M_\sigma) = c$ or else $\hat{\mathscr{A}}_1[\![M_\sigma]\!](\bot) = \bot$.

Then only a slight change in part 2 of the proof of the computability lemma is required.

Example 3.5. Take the collection of domains generated from $D_0 = \mathbf{T}^+$ and $D_\iota = \mathbf{N}^+$ where $\mathbf{T}^+ = \{\bot, T, tt, ff\}$ and $\mathbf{N}^+ = \{\bot, T, 0, 1, \ldots\}$ are ordered as in Fig. 4:

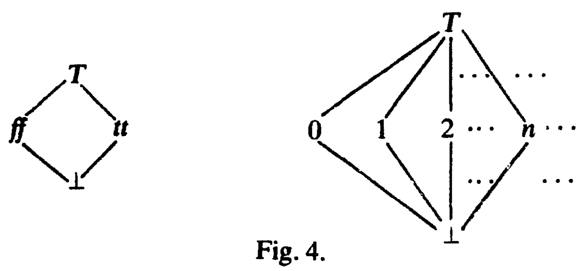

Fig. 4.

The interpretation $\mathscr{A}_2$ is determined by the conditions for $\mathscr{A}_A$ together with:

$$\mathscr{A}_2[\![\supset_\sigma]\!](T)(x)(y) = T \quad (x, y \in D_\sigma),$$

$$\mathscr{A}_2[\![+1]\!](T) = \mathscr{A}_2[\![-1]\!](T) = T,$$

$$\mathscr{A}_2[\![Z]\!](T) = T.$$

The correspondence between the behaviour of a program and its denotation according to $\mathscr{A}_2$ is similar to that of Example 3.4 and is left to the reader. This example would be a natural denotational semantics to use if only complete lattices and not other cpo's were considered. In the next section it will be shown how the standard semantics corresponds more closely, in a certain sense, to Eval_A then $\mathscr{A}_2$.

4. Equivalence of PCF terms

Since terms are only of interest insofar as they are part of programs, we can regard two terms as operationally equivalent if they can be freely substituted for each other in a program without affecting its behaviour. Therefore given $\mathrm{Eval}_{\mathscr{L}}$ we define operational equivalence, $\approx_{\mathscr{L}}$ by:

$M_\sigma \approx_{\mathscr{L}} N_\sigma$ iff whenever $C[M_\sigma]$ and $C[N_\sigma]$ are programs either both of $\mathrm{Eval}_{\mathscr{L}}(C[M_\sigma])$ and $\mathrm{Eval}_{\mathscr{L}}(C[N_\sigma])$ are undefined or else both are defined and equal.

It requires a little proof to show that $\approx_{\mathscr{L}}$ is an equivalence relation. Along the same lines an operational quasi-order, $\sqsubseteq_{\mathscr{L}}$ can be defined by:

$M_\sigma \sqsubseteq_{\mathscr{L}} N_\sigma$ iff whenever $C[M_\sigma]$ and $C[N_\sigma]$ are programs then if $\mathrm{Eval}_{\mathscr{L}}(C[M_\sigma]) = c$ so does $\mathrm{Eval}_{\mathscr{L}}(C[N_\sigma])$.

Clearly, $M_\sigma \approx_{\mathscr{L}} N_\sigma$ iff $M_\sigma \sqsubseteq_{\mathscr{L}} N_\sigma$ and $N_\sigma \sqsubseteq_{\mathscr{L}} M_\sigma$.

It is natural to compare $\sqsubseteq_{\mathscr{L}}$ and $\approx_{\mathscr{L}}$ with the relations $\sqsubseteq_{\mathscr{A}}$, $\equiv_{\mathscr{A}}$ between terms defined by:

$$M_\sigma \sqsubseteq_{\mathscr{A}} N_\sigma \quad \text{iff } \hat{\mathscr{A}}[\![M_\sigma]\!](\rho) \sqsubseteq \hat{\mathscr{A}}[\![N_\sigma]\!](\rho) \text{ for all } \rho,$$

$$M_\sigma \equiv_{\mathscr{A}} N_\sigma \quad \text{iff } M_\sigma \sqsubseteq_{\mathscr{A}} N_\sigma \quad \text{and} \quad N_\sigma \sqsubseteq_{\mathscr{A}} M_\sigma.$$

This concern is the second connection between operational and denotational semantics. Hopefully $\underset{\sim}{\sqsubset}_{\mathscr{L}}$ and $\sqsubseteq_{\mathscr{A}}$ will coincide and so will $\approx_{\mathscr{L}}$ and $\equiv_{\mathscr{A}}$. In the former case, we say that $\mathscr{A}$ is *fully abstract, relative to Eval*$_{\mathscr{L}}$.

Under fairly general circumstances the denotational relations are included in the corresponding operational ones.

Theorem 4.1. *Suppose that* $\mathscr{A}$ *is a model of* Eval$_{\mathscr{L}}$ *in the sense that if* Eval$_{\mathscr{L}}(M) = c$ *then* $\hat{\mathscr{A}}[\![M]\!](\perp) = \mathscr{A}[\![c]\!]$ *and suppose too that if* $\hat{\mathscr{A}}[\![M]\!](\perp) \sqsupseteq \mathscr{A}[\![c]\!]$ *for a program* M *then* $\hat{\mathscr{A}}[\![M]\!](\perp) = \mathscr{A}[\![c]\!]$. *Then the following are equivalent*:

(1) *For any program* M, $\hat{\mathscr{A}}[\![M]\!](\perp) = \mathscr{A}[\![c]\!]$ *implies* Eval$_{\mathscr{L}}(M) = c$.

(2) *For any terms* M_σ, N_σ, $M_\sigma \sqsubseteq_{\mathscr{A}} N_\sigma$ *implies* $M_\sigma \underset{\sim}{\sqsubset}_{\mathscr{L}} N_\sigma$.

(3) *For any terms* M_σ, N_σ, $M_\sigma \equiv_{\mathscr{A}} N_\sigma$ *implies* $M_\sigma \approx_{\mathscr{L}} N_\sigma$.

Proof. (1) $\Longrightarrow$ (2). Suppose $C[M_\sigma]$ and $C[N_\sigma]$ are programs and Eval$_{\mathscr{L}}(C[M_\sigma]) = c$. Then $\hat{\mathscr{A}}[\![C[N_\sigma]]\!](\perp) \sqsupseteq \hat{\mathscr{A}}[\![C[M_\sigma]]\!](\perp) = \mathscr{A}[\![c]\!]$. Therefore from hypothesis and (1), Eval$_{\mathscr{L}}(C[N_\sigma]) = c$.

(2) $\Longrightarrow$ (3). Trivial.

(3) $\Longrightarrow$ (1). If $\hat{\mathscr{A}}[\![M]\!](\perp) = \mathscr{A}[\![c]\!]$ then $M \equiv_{\mathscr{A}} c$. Therefore $M \approx_{\mathscr{L}} c$. As Eval$_{\mathscr{L}}(c) = c$, it follows that Eval$_{\mathscr{L}}(M) = c$.

From the previous section we know that the hypotheses of this theorem and (1) are satisfied by all of $\mathscr{A}_{\mathrm{A}}$, $\mathscr{A}_1$, $\mathscr{A}_2$. Therefore if $M_\sigma \sqsubseteq_{\mathrm{A}} N_\sigma$, $M_\sigma \sqsubseteq_1 N_\sigma$ or $M_\sigma \sqsubseteq_2 N_\sigma$ then $M_\sigma \underset{\sim}{\sqsubset}_{\mathrm{A}} N_\sigma$ for any terms M_σ and N_σ and similarly for $\equiv_{\mathrm{A}}$ and $\approx_{\mathrm{A}}$.

Unfortunately the converses do not hold. For a counter-example, define M_i $(i = 0, 1)$ by:

$$M_i = \lambda\alpha \supset_\iota (\alpha tt\Omega_o)(\supset_\iota (\alpha\Omega_o tt)(\supset_\iota (\alpha ff ff)(\Omega_\iota)(k_i))(\Omega_\iota))(\Omega_\iota)$$

where α has type (o, o, o). The terms M_i are, perhaps, more comprehensible if written diagrammatically as in Fig. 5.

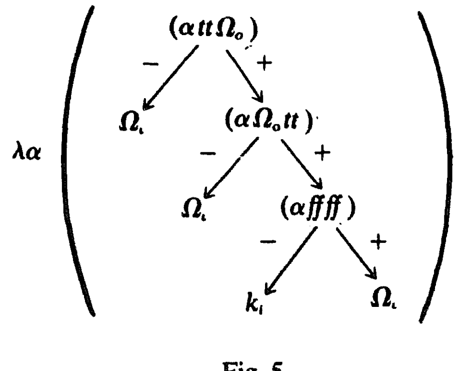

Fig. 5.

We are going to show that $M_0 \approx_A M_1$ but neither $M_0 \sqsubseteq M_1$ nor $M_1 \sqsubseteq M_0$ when $\sqsubseteq$ is any one of $\sqsubseteq_A$, $\sqsubseteq_1$ or $\sqsubseteq_2$.

For $\sqsubseteq_A$, $\sqsubseteq_1$ define V in $D_{(o,o,o)}$ by Table 1.

Table 1

V	$\perp$	*ff*	*tt*
$\perp$	$\perp$	$\perp$	*tt*
ff	$\perp$	*ff*	*tt*
tt	*tt*	*tt*	*tt*

Then $\hat{\mathscr{A}}_A [\![M_i]\!](\perp)(V) = k_i$ and similarly for $\hat{\mathscr{A}}_1$.

The same thing works for $\sqsubseteq_2$ if we define V^+ by Table 2.

Table 2

V^+	$\perp$	*ff*	*tt*	T
$\perp$	$\perp$	$\perp$	*tt*	*tt*
ff	$\perp$	*ff*	*tt*	T
tt	*tt*	*tt*	*tt*	*tt*
T	*tt*	T	*tt*	T

Therefore M_0 and M_1 are incomparable by any of $\sqsubseteq_A$, $\sqsubseteq_1$ or $\sqsubseteq_2$. On the other hand $M_0 \approx_A M_1$. To see this it is necessary to gather some information about $\rightarrow_A$.

The *active* subprogram in a program M, if any, is defined by:

(1) If M has one of the forms $((\lambda \alpha M_1) \cdots)$, $(Y \cdots)$, $((+1)c)$, $((-1)c)$, (Zc) or $(\supset_o c \cdots)$ then M is the active program in M.

(2) If M has one of the forms $((\pm 1)M_1)(ZM_1)$ or $(\supset_\sigma M_1 \cdots)$ where M_1 is not a constant, the active program in M, if any, is the one in M_1, if any.

Notice that if a program has no active program it is a constant and that if a program terminates and has an active program then the active program also terminates.

Lemma 4.2. (Activity lemma). *Suppose $C[M_1, \ldots, M_m]$ is a terminating program with value c, containing closed terms $M_1, \ldots, M_m$. Then either $C[M'_1, \ldots, M'_m]$ terminates with value c for all closed terms $M'_1, \ldots, M'_m$ of appropriate type or else there is a context $D[, \ldots,]$ and an integer i, $1 \leq i \leq m$, and integers $d_1, \ldots, d_k$ such that for all closed terms $M'_1, \ldots, M'_m$ of the appropriate types, $C[M'_1, \ldots, M'_m] \rightarrow^*_A D[M'_{d_1}, \ldots, M'_{d_k}]$ and the active program in $D[M'_{d_1}, \ldots, M'_{d_k}]$ exists and either is the active program in a term of the form $(M'_i \cdots)$, or else has one of the forms $((\pm 1)M'_i)$, (ZM'_i) or $(\supset_\sigma M'_i \cdots)$.*

Proof. We omit the proof which is a straightforward induction on n, where $C[M_1, \ldots, M_m] = N_0 \rightarrow_A \cdots \rightarrow_A N_n = c$, for appropriate $N_0, \ldots, N_n$. □

Now suppose $C[M_0]$ and $C[M_1]$ are programs, that $C[M_0]$ terminates and $C[M_1]$ if it terminates does so with a different value. By the activity lemma applied to $C[M_0]$, $C[M_0] \to_A^* M$ and the active program in M exists and terminates and since M_0 has type $((o, o, o), o)$ it has the form $(M_0 L)$. From the definition of M_0 it follows that the programs $L\,tt\,\Omega_o$, $L\Omega_o tt$ and $Lf\!f f\!f$ terminate with values tt, tt and $f\!f$ respectively. Applying the activity lemma to $C'[tt, \Omega_o] = Ltt\Omega_o$, since Ω_o does not terminate either $\mathrm{Eval}_A(LM_0'N_0') = tt$ for all M_0', N_0' or else for all $M_o'N_o'$, $LM_0''N_0' \to_A^*$ some program M whose active program is the active program in M_0' or else has the form $(\supset_\sigma M_0' \cdots)$ and terminates if $LM_0'N_0'$ does. The first possibility is ruled out because $\mathrm{Eval}_A(Lf\!f f\!f) = f\!f$ and the second because $L\Omega_\sigma tt$ terminates but Ω_o does not. This contradiction proves that $M_0 \sqsubseteq_A M_1$; $M_1 \sqsubseteq_A M_0$ is proved in the same way.

If we require that denotational and operational semantics provide the same equivalence relations and the appropriate analogue of Theorem 4.1 holds then we must therefore reject $\mathscr{A}_A$, $\mathscr{A}_1$ and $\mathscr{A}_2$. If we want to choose between them according to the closeness of $\equiv$ to $\approx_A$, it will be seen later that $\mathscr{A}_A$ is preferable to $\mathscr{A}_1$ and $\mathscr{A}_2$ and they are incomparable.

It should be noted that examples like the M_i can be given in ALGOL 60 if the parameters of type o and ι are called by name; other examples at higher type can be given if these are to be called by value.

One practical consequence of such mismatches is an unpleasant incompleteness phenomenon in program proving systems. If we wish to prove an $\approx_{\mathscr{L}}$ relation, but our proof system is based on axioms about $\equiv_{\mathscr{A}}$, we might not be able to prove equivalences, such as $M_0 = M_1$ which on other operational grounds can be fairly easily seen to hold. Further, proofs of $\not\approx_{\mathscr{L}}$ relations may not even be valid.

The basic difficulty is that the collections of domains considered allow such "parallel" functions as V or V^+ whereas $\to_A$ provides a deterministic operational semantics as is shown by the activity lemma: if a "subprocedure" is called first the corresponding one is also called first in corresponding programs.

One way to close the gap would be to define a "smaller" collection of domains containing only functions capable of deterministic realisation and starting with $D_\iota = \mathbf{N}$ and $D_o = \mathbf{T}$. Vuillemin in [10] defined a notion of sequential function which seems appropriate for types of the form $(\sigma_1, \ldots, \sigma_n)$ with the σ_i's ground, but not at higher types since everything in $D_{(\iota\to\iota)\to\iota}$ is sequential. We leave this as an open problem.

It will prove more fruitful to extend $\mathscr{L}_A$ by adding some parallel facilities and also extend $\mathscr{A}_A$, $\mathscr{A}_1$ amd $\mathscr{A}_2$. First it is convenient to change notation slightly by replacing the subscripts A, 1, 2 by DA, $D1$, $D2$ respectively.

The new language $\mathscr{L}_{PA}$ consists of $\mathscr{L}_{DA}$ together with two parallel conditionals:

$$:\supset_\sigma : (0, \sigma, \sigma, \sigma) \qquad (\sigma \text{ ground}).$$

The operational semantics is given via $\rightarrow_{\mathscr{L}_{PA}}$ which is specified by the rules for $\rightarrow_{\mathscr{L}_{DA}}$ together with:

(I7) $\quad :\supset_\sigma Mcc \rightarrow_{PA} c, \quad :\supset_\sigma ttMN \rightarrow_{PA} M, \quad :\supset_\sigma ffMN \rightarrow_{PA} N$

(I13) $\quad \dfrac{M \rightarrow_{PA} M'}{:\supset_\sigma M \rightarrow_{PA} :\supset_\sigma M'}, \quad \dfrac{N \rightarrow_{PA} N'}{:\supset_\sigma MN \rightarrow_{PA} :\supset_\sigma MN'}, \quad \dfrac{L \rightarrow_{PA} L'}{:\supset_\sigma MNL \rightarrow_{PA} :\supset_\sigma MNL'}$

Clearly $\rightarrow_{PA}$ is non-deterministic. It is not hard to show that if $M_1 \rightarrow_{PA} M_i$ $(i = 2, 3)$ then there is an M_4 such that for each i either $M_i = M_4$ or else $M_i \rightarrow_{PA} M_4$. Therefore $\rightarrow^*_{PA}$ has the Church–Rosser property and $\mathrm{Eval}_{\mathscr{L}_{PA}}$ is well-defined.

$\mathscr{A}_{PA} : \mathscr{L}_{PA} \rightarrow \bigcup \{D_\sigma\}$ is the extension of $\mathscr{A}_A$ such that:

$$\mathscr{A}_{PA}[\![:\supset_o]\!](p)(x)(y) = \begin{cases} x & (p = tt) \\ y & (p = ff) \\ x & (p = \bot \text{ and } x = y) \\ \bot & (p = \bot \text{ and } x \neq y) \end{cases} \quad (p \in D_o, x, y \in D_\iota).$$

Note that $\hat{\mathscr{A}}_{PA}$ is an extension of $\hat{\mathscr{A}}_{DA}$. The deterministic conditional $\supset_\sigma$ can be simulated by:

$$\lambda p \lambda \alpha_1^\sigma \lambda \alpha_2^\sigma (:\supset_\sigma p\, (:\supset_\sigma p \alpha_1^\sigma \Omega_\sigma)(:\supset_\sigma p \Omega_\sigma \alpha_2^\sigma))$$

which has the same denotation as $\supset_\sigma$.

Also V is defined by $\lambda p \lambda q\, (:\supset_\sigma pttq)$.

The analogue of Theorem 3.1 holds, the proof in one direction being easy and in the other direction requires only a slight addition to part 2 of the proof of the computability lemma:

To show that $:\supset_\sigma$ is computable consider $:\supset_\sigma L_o M_\sigma N_\sigma$ where L_o, M_σ and N_σ are computable and suppose $\hat{\mathscr{A}}_{PA}[\![:\supset_o L_o M_\sigma N_\sigma]\!](\bot) = \mathscr{A}_{PA}[\![c]\!]$. Either $\hat{\mathscr{A}}_{PA}[\![L_o]\!](\bot) = tt$ and $\hat{\mathscr{A}}_{PA}[\![M_\sigma]\!](\bot) = \mathscr{A}_{PA}[\![c]\!]$ or else $\hat{\mathscr{A}}_{PA}[\![L_o]\!](\bot) = ff$ and $\hat{\mathscr{A}}_{PA}[\![N_\sigma]\!](\bot) = \mathscr{A}_{PA}[\![c]\!]$ or else $\hat{\mathscr{A}}_{PA}[\![M_\sigma]\!](\bot) = \hat{\mathscr{A}}_{PA}[\![N_\sigma]\!](\bot) = \mathscr{A}_{PA}[\![c]\!]$.

In the first case $:\supset_\sigma L_o M_\sigma N_\sigma \rightarrow^*_{PA} :\supset_\sigma tt M_\sigma N_\sigma \rightarrow_{PA} M_\sigma \rightarrow^*_{PA} c$, and similarly in the second case. In the third case $:\supset_\sigma L_o M_\sigma N_\sigma \rightarrow^*_{PA} \supset_\sigma L_o c N_\sigma \rightarrow^*_{PA} \supset_\sigma Lcc \rightarrow_{PA} c$, which concludes the proof that: $\supset_\sigma$ is computable.

The definitions of $\mathscr{A}_{P1}$ and $\mathscr{A}_{P2}$ and their properties are now fairly clear and are left to the reader.

We now have an example of a pair of semantics which provide the same partial orders and equivalence relations:

Theorem 4.3. (1) *For any $\mathscr{L}_{PA}$-terms M_σ and N_σ, $M \sqsubseteq_{\mathscr{A}_{PA}} N$ iff $M \underset{\sim}{\sqsubset}_{PA} N$.*

(2) *For any $\mathscr{L}_{PA}$-terms M_σ and N_σ, $M \equiv_{\mathscr{A}_{PA}} N$ iff $M \approx_{PA} N$.*

The proof requires some information on the collection of domains $\{D_\sigma\}$ in $\mathscr{A}_{PA}$, and the elements in these domains definable by $\mathscr{L}_{PA}$-terms.

An element d of a cpo $\langle D, \sqsubseteq \rangle$ is *finite* iff whenever $d \sqsubseteq \bigsqcup_D X$ for a directed X, $d \sqsubseteq$ some e in X.

A cpo$\langle D, \sqsubseteq\rangle$ is *algebraic* if, for each x, the set $\{d \sqsubseteq x : d \text{ finite}\}$ is directed and its lub is x.

If d, e are finite members of D and E, then $(d \Rrightarrow e)$ is the member of $[D \to E]$ defined by:

$$(d \Rrightarrow e)(x) = \begin{cases} e & (x \sqsupseteq d), \\ \bot & (x \not\sqsupseteq d). \end{cases}$$

A cpo D is *consistently complete* iff whenever x and y in D have an upper bound they have a least upper bound, $x \sqcup y$.

If D and E are consistently complete cpo's and $d_1, \ldots, d_n$ are finite elements of D and $e_1, \ldots, e_n$ of E it is not hard to see that the set $\{d_i \Rrightarrow e_i\}$ has a least upper bound in $[D \to E]$ iff whenever a subset of $\{d_i\}$ has a lub so does the corresponding subset of $\{e_i\}$, and then the lub is given by

$$\bigsqcup\{d_i \Rrightarrow e_i\}(x) = \bigsqcup\{e_i : x \sqsupseteq d_i\}$$

Lemma 4.4 (Scott). (1) **T** *and* **N** *are consistently complete algebraic* cpo's.

(2) *If D and E are consistently complete algebraic* cpo's *so is* $[D \to E]$. *Its finite elements are all* lub's *of finite sets of elements of the form* $(d \Rrightarrow e)$, *with* d, e *finite members of D and E respectively.*

Proof. (1) Obvious.

(2) Suppose f, g in $[D \to E]$ have an upper bound h, then for any x, $h(x)$ is an upper bound of $f(x)$ and $g(x)$ and so as E is consistently complete we can define a function $k : D \to E$ by

$$k(x) = f(x) \sqcup g(x) \quad (x \in D).$$

Clearly k is the lub of f and g in $[D \to E]$. So $[D \to E]$ is consistently complete.

Now suppose $(d \Rrightarrow e) \sqsubseteq \bigsqcup F$ where F is a directed subset of $[D \to E]$. Then $e = (d \Rrightarrow e)(d) \sqsubseteq (\bigsqcup F)(d) = \bigsqcup_{f \in F} f(d)$. As e is finite, $e \sqsubseteq f(d)$ for some $f \in F$. Then $(d \Rrightarrow e) \sqsubseteq f$. Therefore $(d \Rrightarrow e)$ is finite and it follows that any lub of a finite set of elements of the form $(d \Rrightarrow e)$ is also finite.

Take $f \in [D \to E]$ and consider the set $F = \{\bigsqcup F' : F' \text{ is a finite set of elements of } [D \to E] \text{ of the form } (d \Rrightarrow e) \text{ and } \sqsubseteq f\}$. It will be shown that $f = \bigsqcup F$. Clearly if $f' \in F$ then $f \sqsupseteq f'$, so $\bigsqcup F$ exists and $f \sqsupseteq \bigsqcup F$. To show that $f \sqsubseteq (\bigsqcup F)$, take $x \in D$ and a finite element $e \sqsubseteq f(x)$.

Then

$$\begin{aligned} e \sqsubseteq f(x) &= f(\bigsqcup\{d \in D : \quad d \text{ finite and } \sqsubseteq x\}) \\ &= \bigsqcup\{f(d) : \quad d \text{ finite and } \sqsubseteq x\}. \end{aligned}$$

Therefore for some finite $d \sqsubseteq x$, $e \sqsubseteq f(d)$. Therefore $(d \Rrightarrow e) \sqsubseteq f$ and as $(d \Rrightarrow e) \in F$, $(\bigsqcup F)(x) \sqsupseteq d$. As d was arbitrary, $(\bigsqcup F)(x) \sqsupseteq f(x)$.

Now if f is finite then as F is directed, f is the lub of a finite set of elements of the form $(d \Rrightarrow e)$, as required.

Therefore if instead f is an arbitrary member of $[D \to E]$, F is the set of all finite members of $[D \to E]$, $\sqsubseteq f$. As $f = \sqcup F$, $[D \to E]$ is algebraic, concluding the proof of the lemma. $\square$

So each D_σ is a consistently complete algebraic cpo. From Lemma 4.4 and the fact that $d \Rrightarrow (e \sqcup e') = (d \Rrightarrow e) \sqcup (d \Rrightarrow e')$, one side existing iff the other does, for d a finite member of a consistently complete cpo, D and e, e' of another, E, the finite members of D_σ are lub's of finite subsets F of D_σ such that:

(1) Each element of F has the form $(e_1 \Rrightarrow \cdots \Rrightarrow (e_n \Rrightarrow d) \cdots)$ where $e_1, \ldots, e_n$ are finite elements of $D_{\sigma_1}, \ldots, D_{\sigma_n}$ and $d \in D_\tau$ is not $\perp$.

(2) If $(e_1 \Rrightarrow \cdots \Rrightarrow (e_n \Rrightarrow d) \cdots)$ and $(e'_1 \Rrightarrow \cdots \Rrightarrow (e'_n \Rrightarrow d') \cdots)$ are in F and $e_1 \sqcup e'_1, \ldots, e_n \sqcup e'_n$ exist then $d = d'$.

Here $\sigma = (\sigma_1, \ldots, \sigma_n, \tau)$ with τ ground.

Also a set F satisfying (1) has a lub iff it satisfies (2).

Lemma 4.5. *Every finite element of each D_σ is definable by an $\mathcal{L}_{PA}$-term.*

Proof. The proof is by induction on types and shows that if e, f are finite elements in D_σ then $e, e \Rrightarrow tt$ and, if it exists, $(e \Rrightarrow tt) \sqcup (f \Rrightarrow ff)$ are definable by $\mathcal{L}_{PA}$-terms.

$\sigma = o$: $\perp$, tt and ff are defined by Ω_o, tt, and ff.
$\perp \Rrightarrow tt$, $tt \Rrightarrow tt$ and $ff \Rrightarrow tt$ are defined by $\lambda p tt$, $\lambda p(\supset_o p tt \Omega_o)$ and $\lambda p(\supset_o p \Omega_o ff)$, respectively.
$(tt \Rrightarrow tt) \sqcup (ff \Rrightarrow ff)$ and $(ff \Rrightarrow tt) \sqcup (tt \Rrightarrow ff)$ are defined by λpp and $\lambda p(\supset_o p ff tt)$.

$\sigma = \iota$: $\perp$ and n are defined by Ω_ι and k_n.
$\perp \Rrightarrow tt$ and $n \Rrightarrow tt$ are defined by $\lambda x tt$ and $\lambda x(\supset_o Z((-1)^n x) tt \Omega_o)$; $(k_m \Rrightarrow tt) \sqcup (k_n \Rrightarrow ff)$ and $(k_n \Rrightarrow tt) \sqcup (k_m \Rrightarrow ff)$, where $m < n$, are defined by $\lambda x(\supset_o (Z((-1)^m x)) tt (\supset_o (Z((-1)^n x)) ff \Omega_o))$ and $\lambda x(\supset_o (Z((-1)^m x)) ff (\supset_o (Z((-1)^n x)) tt \Omega_o))$.

$\sigma = (\sigma_1, \ldots, \sigma_n, \tau)$ with τ ground: Suppose e and f are lub's of finite sets F, F' as explained above. To show that $e, e \Rrightarrow tt$ are definable we use induction on the size of F and then show $(e \Rrightarrow tt) \sqcup (f \Rrightarrow ff)$ definable, if it exists.

For e, if $F = \emptyset$, e is defined by Ω_σ.

Suppose $F \neq \emptyset$. If there are $e_1 \Rrightarrow \cdots \Rrightarrow e_n \Rrightarrow d$ and $e'_1 \Rrightarrow \cdots \Rrightarrow e'_n \Rrightarrow d'$ in F such that for some i, $e_i \sqcup e'_i$ does not exist then $(e_i \Rrightarrow tt) \sqcup (e'_i \Rrightarrow ff)$ exists and is definable, say by M_1. Also $\sqcup(F \setminus \{e_1 \Rrightarrow \cdots \Rrightarrow d\})$ and $\sqcup(F \setminus \{e'_1 \Rrightarrow \cdots \Rrightarrow d'\})$ are definable by, say, F_1 and F_2. Then $\sqcup F$ itself is definable by:

$$\lambda \alpha_1^{\sigma_1} \cdots \lambda \alpha_n^{\sigma_n} (:\supset_\tau (M_1 \alpha_i^{\sigma_i})(F_2 \alpha_1^{\sigma_1} \cdots \alpha_n^{\sigma_n})(F_1 \alpha_1^{\sigma_1} \cdots \alpha_n^{\sigma_n})).$$

Otherwise if $e_1 \Rightarrow \cdots \Rightarrow d$ and $e'_1 \Rightarrow \cdots \Rightarrow d'$ are in F then $d = d'$, and all the $e_i \sqcup e'_i$'s exist. Take $e_1 \Rightarrow \cdots \Rightarrow e_n \Rightarrow d$ in F and let $E_1, \ldots, E_n$, D, F_1 define $(e_1 \Rightarrow tt), \ldots, (e_n \Rightarrow tt)$, d and $\sqcup F \setminus \{e_1 \Rightarrow \cdots \Rightarrow d\}$. Then $\sqcup F$ is defined by

$$\lambda\alpha_1^{\sigma_1}\cdots\lambda\alpha_n^{\sigma_n}(:\supset_\tau((E_1\alpha_1^{\sigma_1})\text{AND}\cdots\text{AND}(E_n\alpha_n^{\sigma_n}))D(F_1\alpha_1^{\sigma_1}\cdots\alpha_n^{\sigma_n})).$$

Here AND is the term $\lambda p\lambda q(\supset_o p(\supset_o q\, tt\, ff)ff)$ and is used as an infix.

For $e \Rightarrow tt$, if $F = \emptyset$, e is defined by $\lambda\alpha^\sigma tt$.

Suppose $F \neq \emptyset$ and take $e_1 \Rightarrow \cdots \Rightarrow e_n \Rightarrow d$ in F and let $E_1, \ldots, E_n, D, F_1$ define $e_1, \ldots, e_n$, $d \Rightarrow tt$, $(\sqcup F \setminus \{e_1 \Rightarrow \cdots \Rightarrow d\}) \Rightarrow tt$ respectively. Then $e \Rightarrow tt$ is defined by:

$$\lambda\alpha^\sigma(\supset_o(D(\alpha^\sigma E_1\cdots E_n))(F_1\alpha^\sigma)\Omega_o).$$

If $(e \Rightarrow tt)\sqcup(f \Rightarrow ff)$ exists, $e \sqcup f$ does not and so there are $e_1 \Rightarrow \cdots \Rightarrow e_n \Rightarrow d$ in F and $e'_1 \Rightarrow \cdots \Rightarrow e'_n \Rightarrow d'$ in F' such that $e'_1 \sqcup e'_1, \ldots, e_n \sqcup e'_n$ exist but $d \neq d'$. Let $E_1, \ldots, E_n, F_1, F'_1$, D define $(e_1 \sqcup e'_1), \ldots, (e_n \sqcup e'_n), e \Rightarrow tt, f \Rightarrow tt, (d \Rightarrow tt)\sqcup(d' \Rightarrow ff)$ respectively.

Then $(e \Rightarrow tt)\sqcup(f \Rightarrow ff)$ is defined by:

$$\lambda\alpha^\sigma(\supset_o(D(\alpha^\sigma E_1\cdots E_n))(F_1\alpha^\sigma)(\text{NEG}(F'_1\alpha^\sigma)))$$

where NEG is the term $\lambda p(\supset_o p\, ff\, tt)$.

This concludes the proof of the lemma. □

It is now possible to prove Theorem 4.3. First suppose M_σ and N_σ are $\mathscr{L}_{PA}$-terms such that $M_\alpha \sqsubseteq_{PA} N_\sigma$ but $M_\sigma \not\sqsubseteq_{\mathscr{A}_{PA}} N_\sigma$. Suppose too that M_σ and N_σ are closed and define f and g, respectively, in D_σ where $\sigma = (\sigma_1, \ldots, \sigma_n, \tau)$ with τ ground. Then $f \not\sqsubseteq g$. Therefore there are $x_1, \ldots, x_n$ in $D_{\sigma_1}, \ldots, D_{\sigma_n}$ such that $f(x_1)\cdots(x_n) \neq \bot$ and $f(x_1)\cdots(x_n) \neq g(x_1)\cdots(x_n)$. Since $D_{\sigma_1}\cdots D_{\sigma_n}$ are algebraic and every element of D_τ is finite we can assume that $x_1, \ldots, x_n$ are finite and so, by Lemma 4.5 can be defined by closed $\mathscr{L}_{PA}$-terms $X_1, \ldots, X_n$ say. Then, by the analogue of Theorem 3.1, $M_\sigma X_1\cdots X_n$ terminates in a constant c and, if $N_\sigma X_1, \ldots, X_n$ terminates it does so in a different constant. This contradicts $M_\sigma \sqsubseteq_{PA} N_\sigma$ and therefore for closed M_σ, N_σ, if $M_\sigma \sqsubseteq_{PA} N$ then $M_\sigma \sqsubseteq_{\mathscr{A}_{PA}} N_\sigma$. For open M_σ, N_σ with free variables $\{\alpha_1, \ldots, \alpha_n\}$, we have $M_\sigma \sqsubseteq_{PA} N_\sigma$ implies $\lambda\alpha_1\cdots\lambda\alpha_n M \sqsubseteq_{PA} \lambda\alpha_1\cdots\lambda\alpha_n N$ implies $\lambda\alpha_1\cdots\lambda\alpha_n M \sqsubseteq_{\mathscr{A}_{PA}} \lambda\alpha_1\cdots\lambda\alpha_n N$ implies $M_\sigma \sqsubseteq_{\mathscr{A}_{PA}} N_\sigma$. That $M_\sigma \sqsubseteq_{\mathscr{A}_{PA}} N_\sigma$ implies $M_\sigma \sqsubseteq_{PA} N_\sigma$ is already known from the remarks on $\mathscr{A}_{PA}$ above, and the rest of Theorem 4.3 is immediate.

Now we shall see that the analogue of theorem 4.3 does not hold for either $\mathscr{A}_{P1}$ or $\mathscr{A}_{P2}$ and so we prefer the denotational semantics for $\mathscr{L}_{PA}$ given by $\mathscr{A}_{PA}$ to either of the $\mathscr{A}_{Pi}$, if we want a good correspondence with Eval_{PA}.

For $\mathscr{A}_{P1}$ consider the term $N = Y_{(\iota\to\iota)}(\lambda\alpha\lambda x(\supset_\iota(Zx)k_0(\alpha((-1)x))))$. Then $N \simeq_{PA} (\lambda xx)$ as $N \equiv_{PA} (\lambda xx)$. But since $\mathscr{A}_{P1}[\![N]\!](\bot)(\infty) = \bot$, $N \not\equiv_{P1} (\lambda xx)$. Note too that $N \equiv_{P2} (\lambda xx)$.

For $\mathscr{A}_{P2}$ consider the terms N_i and M_i $(i = 0, 1)$ defined by: $N_o = \lambda p\ (\supset_o ptt\Omega_o)$, which defines $(tt \Rightarrow tt) \sqcup (T \Rightarrow T)$ and $N_1 = \lambda p(\supset_o p\Omega_o ff)$ which defines $(ff \Rightarrow ff) \sqcup (T \Rightarrow T)$ in the collection of domains $\{D_\sigma\}$ given by $\mathscr{A}_{P2}$.

Let

$$M_0 = (\lambda\alpha\ (\supset_o (\alpha N_0)(\supset_o (\alpha N_1)\Omega_o tt)\Omega_o)),$$

$$M_1 = (\lambda\alpha\ (\supset_o (\alpha N_o)(\supset_o (\alpha N_1)\Omega_o ff)\Omega_o)),$$

where α has type $(o \to o) \to o$. Let $h = ((tt \Rightarrow tt) \Rightarrow tt) \sqcup ((ff \Rightarrow ff) \Rightarrow ff)$ which exists as $D'_{((o,o),o)}$ is a complete lattice.

Then $\hat{\mathscr{A}}_{P2}[\![M_0]\!](\bot)(h) = tt \neq ff = \hat{\mathscr{A}}_{P1}[\![M_1]\!](\bot)(h)$. Therefore $M_0 \not\equiv_{P2} M_1$.

Switching to the collection, $\{D_\sigma\}$ of domains for $\mathscr{A}_{PA}$, N_0 and N_1 define $(tt \Rightarrow tt)$ and $(ff \Rightarrow ff)$ respectively. Suppose h is $D_{((o,o),o)}$. If $\hat{\mathscr{A}}_{PA}[\![M_o]\!](\bot)(h) \neq \bot$ then $h(tt \Rightarrow tt) = tt$ and $h(ff \Rightarrow ff) = ff$, which is a contradiction as then $h((tt \Rightarrow tt) \sqcup (ff \Rightarrow ff)) \sqsupseteq tt, ff$. Therefore $M_0 \equiv_{PA} \Omega_\sigma$, for the appropriate σ and similarly $M_1 \equiv_{PA} \Omega_\sigma$. Therefore $M_0 \not\equiv_{P2} M_1$ but $M_0 \approx_{PA} M_1$. Note too that $M_0 \equiv_{P1} M_1$.

Returning to $\mathscr{L}_{DA}$ for a moment we can now verify an earlier claim. For any two $\mathscr{L}_{DA}$ terms M_σ, N_σ we have $M \sqsubseteq_{Di} N$ implies $M \sqsubseteq_{Pi} N$ implies $M \sqsubseteq_{PA} N$ implies $M \sqsubseteq_{PA} N$ implies $M \sqsubseteq_{DA} N$ for $i = 1, 2$ and by the notes above since the various counterexamples do not use $a : \supset_\sigma$, neither of $\sqsubseteq_{D1}$, $\sqsubseteq_{D2}$ are included in the other.

We now consider to what extent interpretations of $\mathscr{L}_{PA}$ are restricted by requiring that they be fully abstract relative to Eval_{PA}. The answer depends on the class of interpretations considered. For example, we shall see that any fully abstract standard interpretation of $\mathscr{L}_{PA}$ is isomorphic to $\hat{\mathscr{A}}_{PA}$, but there are fully abstract interpretations of a more general kind which are not isomorphic to $\hat{\mathscr{A}}_{PA}$. However $\hat{\mathscr{A}}_{PA}$ will stand out as a kind of weak initial element in the category formed from the fully abstract general interpretations.

First of all we choose a convenient notion of general interpretation. A *general collection of domains for PCF* is a family $\{D_\sigma\}$ of cpo's, one for each type and a family $\{Ap^\sigma_\tau\}$ of continuous maps, one for each pair of types such that:

$$Ap^\sigma_\tau : [D_{\sigma\to\tau} \to [D_\sigma \to D_\tau]].$$

When convenient we shall drop the Ap^σ_τ's and write $f(x)$ instead of $Ap^\sigma_\tau(f)(x)$.

Such a collection is *pointwise ordered* iff for any types σ, τ:

$$\forall f, g \in D_{\sigma\to\tau},\ f \sqsubseteq g \quad \text{iff } (\forall x \in D_\sigma, fx \sqsubseteq gx).$$

Clearly any collection of domains for PCF provides a pointwise ordered general collection, if we take the Ap^σ_τ's to be ordinary application.

A *general interpretation* of $\mathscr{L}$ is a general collection of domains $\langle\{D_\sigma\}, \{Ap^\sigma_\tau\}\rangle$ for PCF together with a type-respecting map,

$$\hat{\mathscr{A}} : \text{Terms} \to (\text{Env} \to \bigcup \{D_\sigma\}),$$

(where Env is the set of type-respecting maps from the set of variables to $\bigcup\{D_\sigma\}$) such that:

(1) $\hat{\mathscr{A}}[\![\alpha]\!](\rho)=\rho[\![\alpha]\!]$,
(2) $\hat{\mathscr{A}}[\![(M_{\sigma\to\tau}N_\sigma)]\!](\rho)=\hat{\mathscr{A}}[\![M_{\sigma\to\tau}]\!](\rho)(\hat{\mathscr{A}}[\![N_\sigma]\!](\rho))$,
(3) if M is closed $\hat{\mathscr{A}}[\![M]\!](\rho)$ is independent of ρ.

Sometimes $\hat{\mathscr{A}}$ will be confused with the whole general interpretation $\langle\hat{\mathscr{A}},\{D_\sigma\},\{Ap^\sigma_\tau\}\rangle$. Clearly any interpretation provides a general interpretation. If a general interpretation is a model of β-conversion, then we have:

(3′) If ρ and ρ' have the same values on $FV(M_\sigma)$ then $\hat{\mathscr{A}}[\![M]\!](\rho)=\hat{\mathscr{A}}[\![M]\!](\rho')$.

For,

$$\begin{aligned}\hat{\mathscr{A}}[\![M]\!](\rho)&=\hat{\mathscr{A}}[\![(\lambda\alpha_1\cdots\lambda\alpha_nM)\alpha_1\cdots\alpha_n]\!](\rho)\qquad\text{(by }\beta\text{-conversion)}\\&=\hat{\mathscr{A}}[\![(\lambda\alpha_1\cdots\lambda\alpha_nM)]\!](\rho)(\hat{\mathscr{A}}[\![\alpha_1]\!](\rho))\cdots(\hat{\mathscr{A}}[\![\alpha_n]\!](\rho))\qquad\text{(by (2))}\\&=\hat{\mathscr{A}}[\![(\lambda\alpha_1\cdots\lambda\alpha_n)M]\!](\rho')(\rho'[\![\alpha_1]\!])\cdots(\rho'[\![\alpha_n]\!])\\&\qquad\qquad\text{(by (1), (3) and assumption)}\\&=\hat{\mathscr{A}}[\![M]\!](\rho')\qquad\text{(by (1), (2) and }\beta\text{-conversion).}\end{aligned}$$

A general interpretation, $\hat{\mathscr{A}}$ of $\mathscr{L}$ is *fully abstract relative to Eval*$_{\mathscr{L}}$ iff for all terms M_σ, N_σ:

$$M_\sigma\sqsubseteq_{\mathscr{L}}N_\sigma\quad\text{iff }(\forall\rho\in\text{Env}\,\hat{\mathscr{A}}[\![M]\!](\rho)\sqsubseteq\hat{\mathscr{A}}[\![N]\!](\rho)).$$

The condition on the right is written as: $M\sqsubseteq_{\hat{\mathscr{A}}}N$; $M\equiv_{\hat{\mathscr{A}}}N$ has the obvious meaning.

With the evident definition of product one can show that the class of fully abstract general interpretations of $\mathscr{L}$ is closed under the product operation. Therefore, for example, $\hat{\mathscr{A}}_{PA}\times\hat{\mathscr{A}}_{PA}$ is fully abstract, but not isomorphic to $\hat{\mathscr{A}}_{PA}$.

To compare general interpretations, we introduce a convenient notion of morphism. Suppose $\langle\hat{\mathscr{A}},\{D_\sigma\},\{Ap^\sigma_\tau\}\rangle$ and $\langle\hat{\beta},\{E_\sigma\},\{Ap'^\sigma_\tau\}\rangle$ are general interpretations of $\mathscr{L}$.

A *morphism* $\Phi:\hat{\mathscr{A}}\to\hat{\mathscr{B}}$ is a collection of continuous maps, $\Phi_\sigma:D_\sigma\to E_\sigma$ such that:

(1) $\Phi_\tau(Ap^\sigma_\tau(f)(x))=Ap'^\sigma_\tau(\Phi_{\sigma\to\tau}(f))(\Phi_\sigma(x))$ (for any types σ and τ; f in $D_{\sigma\to\tau}$ and x in D_σ),
(2) $\Phi_\sigma(\hat{\mathscr{A}}[\![M_\sigma]\!](\bot))\sqsubseteq\hat{\mathscr{B}}[\![M_\sigma]\!](\bot)$($M_\sigma$ a closed term of type σ),
(3) $\Phi_\sigma(\hat{\mathscr{A}}[\![M_\sigma]\!](\bot))=\hat{\mathscr{B}}[\![M_\sigma]\!](\bot)$($M_\sigma$ a program of type σ).

Here $\bot$ is the environment such that $\bot(\alpha^\sigma)=\bot_{D_\sigma}$. If $\hat{\mathscr{A}}$ and $\hat{\mathscr{B}}$ are models of β-conversion, one can prove a stronger version of (2):

(2′) $\Phi_\sigma(\hat{\mathscr{A}}[\![M_\sigma]\!](\rho))\sqsubseteq\hat{\mathscr{B}}[\![M_\sigma]\!]((\bigcup\Phi)\circ\rho)$ (M_σ any term of type σ).

The identity morphism on $\hat{\mathscr{A}}$ is $\text{Id}_{\hat{\mathscr{A}}}=\{\text{Id}_{D_\sigma}\}$ where $\text{Id}_{D_\sigma}(x)=x$ for x in D_σ. Composition of morphisms is defined by $\Phi\circ\Psi=\{\Phi_\sigma\circ\Psi_\sigma\}$. Thus we have a category of general interpretations of a given language $\mathscr{L}$.

If $\Phi : \hat{\mathscr{A}} \to \hat{\mathscr{B}}$ is an isomorphism, then $\sqsubseteq$ can be replaced by $=$ in condition (2) and, consequently, in condition (2′).

Theorem 4.6. (1) *If $\hat{\mathscr{B}}$ is a general interpretation, which is fully abstract relative to* $\mathrm{Eval}_{\mathrm{PA}}$, *then there is a monomorphism $\Phi : \hat{\mathscr{A}}_{\mathrm{PA}} \to \hat{\mathscr{B}}$.*

(2) *If $\hat{\mathscr{A}}$ is a general interpretation, which is fully abstract relative to* $\mathrm{Eval}_{\mathrm{PA}}$, *and it has the property ascribed to $\hat{\mathscr{A}}_{\mathrm{PA}}$ in* 1 *then there is a unique isomorphism $\Phi : \hat{\mathscr{A}}_{\mathrm{PA}} \to \hat{\mathscr{A}}$.*

Proof. (1) We define $\Phi = \{\Phi_\sigma\}$ by

$$\Phi_\sigma(x) = \bigsqcup\{\hat{\mathscr{B}}[\![M]\!](\bot) \colon M \text{ is closed and } \hat{\mathscr{A}}_{\mathrm{PA}}[\![M]\!](\bot) \text{ is finite and } \sqsubseteq x\} \qquad (x \in D_\sigma).$$

First we check that the set on the RHS is directed. Suppose $\hat{\mathscr{B}}[\![M_i]\!](\bot)$ is in the set for $i = 1, 2$. Then each $\hat{\mathscr{A}}_{\mathrm{PA}}[\![M_i]\!](\bot) \sqsubseteq x$ and so by Lemma 4.4 there is a finite $\hat{\mathscr{A}}_{\mathrm{PA}}[\![M]\!](\bot)$, where M is closed, such that $\hat{\mathscr{A}}_{\mathrm{PA}}[\![M_i]\!](\bot) \sqsubseteq \hat{\mathscr{A}}_{\mathrm{PA}}[\![M]\!](\bot) \sqsubseteq x$ for $i = 1, 2$. Then as $\hat{\mathscr{A}}_{\mathrm{PA}}$ is fully abstract $M_i \sqsubseteq_{\mathrm{PA}} M$ and so $\hat{\mathscr{B}}[\![M_i]\!](\bot) \sqsubseteq \hat{\mathscr{B}}[\![M]\!](\bot)$ for $i = 1, 2$ as required, for $\hat{\mathscr{B}}$ is fully abstract too.

Next we establish condition (2) for Φ to be a morphism. This follows at once from the definition of Φ_σ and the fact that if M, M' are closed terms such that $\hat{\mathscr{A}}_{\mathrm{PA}}[\![M']\!](\bot) \sqsubseteq \hat{\mathscr{A}}_{\mathrm{PA}}[\![M]\!](\bot)$ then $\hat{\mathscr{B}}[\![M]\!](\bot) \sqsubseteq \hat{\mathscr{B}}[\![M']\!](\bot)$, as $\hat{\mathscr{A}}_{\mathrm{PA}}$ and $\hat{\mathscr{B}}$ are fully abstract.

Now, if M_σ defines a finite element of D_σ, we can see that

$$\Phi_\sigma(\hat{\mathscr{A}}_{\mathrm{PA}}[\![M_\sigma]\!](\bot)) = \hat{\mathscr{B}}[\![M_\sigma]\!](\bot).$$

This follows from condition (2) and the definition of Φ_σ. It establishes condition (3) for Φ to be a morphism and shows that for x in D_σ, $\Phi_\sigma(x) = \bigsqcup\{\Phi_\sigma(d) \colon d \sqsubseteq x$ and finite$\}$. This last remark implies that Φ_σ is continuous.

For condition (1) we calculate,

$$\begin{aligned}
\Phi_\tau(f(x)) &= \bigsqcup\{\Phi_\tau(f'(x')) \colon f', x' \text{ finite and } \sqsubseteq f,\, x \text{ respectively}\} \quad (\Phi_\tau \text{ is continuous})\\
&= \bigsqcup\{\Phi_\tau(\hat{\mathscr{A}}_{\mathrm{PA}}[\![F]\!](\bot)(\hat{\mathscr{A}}_{\mathrm{PA}}[\![X]\!](\bot))) \colon F \text{ and } X \text{ are } \mathscr{L}_{\mathrm{PA}}\text{-terms}\\
&\qquad\qquad \text{defining finite elements } \sqsubseteq f, x \text{ respectively}\}\\
&= \bigsqcup\{\Phi_\tau(\hat{\mathscr{A}}_{\mathrm{PA}}[\![FX]\!](\bot)) \colon \cdots\}\\
&= \bigsqcup\{\hat{\mathscr{B}}[\![FX]\!](\bot) \colon \cdots\} \quad (\text{each } FX \text{ defines a finite element})\\
&= \bigsqcup\{Ap^\sigma_\tau(\hat{\mathscr{B}}[\![F]\!](\bot))(\hat{\mathscr{B}}[\![X]\!](\bot)) \colon \cdots\}\\
&= Ap^\sigma_\tau(\bigsqcup\{\hat{\mathscr{B}}[\![F]\!](\bot) \colon F \text{ defines a finite element } \sqsubseteq f\})\\
&\qquad (\bigsqcup\{\hat{\mathscr{B}}[\![X]\!](\bot) \colon X \text{ defines a finite element } \sqsubseteq x\})\\
&\qquad\qquad (Ap^\sigma_\tau \text{ is continuous})\\
&= Ap^\sigma_\tau(\Phi_{\sigma\to\tau}(f))(\Phi_\sigma(x)).
\end{aligned}$$

Therefore Φ is indeed a morphism. To show that it is a monomorphism it is enough to show that each Φ_σ is 1-1. This is clear for σ ground since then all elements D_σ are definable and we can use condition (3). Suppose the Φ_{σ_i} are 1-1 for $i = 1, n$. We show that Φ_σ is also 1-1 for $\sigma = (\sigma_1, \ldots, \sigma_n, \tau)$ with τ ground. Suppose $\Phi_\sigma(f) = \Phi_\sigma(g)$ for $f, g \in D_\sigma$. Take $x_1, \ldots, x_n$ in $D_{\sigma_1}, \ldots, D_{\sigma_n}$ respectively.

Then

$$\begin{aligned}\Phi_\sigma(fx_1 \cdots x_n) &= \Phi_\sigma(f)\Phi_{\sigma_1}(x_1) \cdots \Phi_{\sigma_n}(\sigma_n) \quad \text{(by condition (1))}\\ &= \Phi_\tau(gx_1 \cdots x_n)\end{aligned}$$

and so as Φ_τ is 1-1, $fx_1 \cdots x_n = gx_1 \cdots x_n$. Therefore Φ_σ is 1-1 and by induction Φ is a monomorphism.

(2) Let $\hat{\mathscr{A}}$ be such an interpretation. Let $\Phi : \hat{\mathscr{A}}_{PA} \to \hat{\mathscr{A}}$ be the monomorphism given in part 1 and let $\Phi' : \hat{\mathscr{A}} \to \hat{\mathscr{A}}_{PA}$ be any monomorphism — and one exists, by hypothesis. A straightforward induction on σ using conditions (1) and (3) shows that $\hat{\mathscr{A}}_{PA}$ is rigid in the sense that the only morphism $\Psi : \hat{\mathscr{A}}_{PA} \to \hat{\mathscr{A}}_{PA}$ is the identity. Therefore $\Phi' \circ \Phi$ is the identity, and as Φ' is a monomorphism so is $\Phi \circ \Phi'$.

This concludes the proof. □

Presumably the Φ_σ's defined in part 1 of the proof are not unique in general, so that we cannot expect $\mathscr{A}_{PA}$ to be the initial object in the evident category. At any rate the theorem does characterise $\mathscr{A}_{PA}$ as a kind ot weak initial object.

Theorem 4.7. *Let $\langle \hat{\mathscr{A}}, \{E_\sigma\}, \{Ap_\tau^\sigma\}\rangle$ be a pointwise ordered general interpretation of $\mathscr{L}_{PA}$, which is fully abstract relative to* Eval_{PA}. *Suppose too that $\hat{\mathscr{A}}[\![Y_{(\iota,\iota)}]\!](\bot)(f) = \bigsqcup_{n \geq 0} f^n(\bot)$ (f in $E_{(\iota,\iota)}$) and that there are elements T and F in $E_{(o,o)}$ such that:*

$$T(x) = \begin{cases} \hat{\mathscr{A}}[\![tt]\!](\bot) & (\textit{if } x \not\sqsubseteq \hat{\mathscr{A}}[\![ff]\!](\bot)),\\ \bot & (\textit{otherwise}), \end{cases}$$

$$F(x) = \begin{cases} \hat{\mathscr{A}}[\![ff]\!](\bot) & (\textit{if } x \not\sqsubseteq \hat{\mathscr{A}}[\![tt]\!](\bot)),\\ \bot & (\textit{otherwise}). \end{cases}$$

Then $\hat{\mathscr{A}}$ is isomorphic to $\hat{\mathscr{A}}_{PA}$.

Proof. In this proof, we confuse closed $\mathscr{L}_{PA}$ terms with their denotations and will continue to drop Ap_τ^σ's when convenient.

We begin by showing that E_o and D_o are isomorphic. Notice that $\alpha_0^{(o,o)} p \sqsubseteq_{\hat{\mathscr{A}}} (:\supset p)(\alpha_0^{(o,o)} tt)(\alpha_0^{(o,o)} ff)$ since $\sqsubseteq_{\hat{\mathscr{A}}_{PA}}$ holds between the terms and $\hat{\mathscr{A}}$ and $\hat{\mathscr{A}}_{PA}$ are fully abstract.

It follows that $fd \sqsubseteq {:\supset_o} d(f(tt))(f(ff))$ for any f in $E_{(o,o)}$ and d in E_o by choosing an appropriate ρ and using conditions (1) and (2) on general interpretations.

Similarly, $fd \sqsupseteq \supset_o d(f(tt))(f(ff))$, for any f in $E_{(o,o)}$ and d in E_o. These facts will be used repeatedly.

We now show that every element, d, of D_o is either $\sqsubseteq tt$ or else is $\sqsubseteq ff$. Suppose otherwise and that d in D_o is $\not\sqsubseteq tt$ and $\not\sqsubseteq ff$. Then $e = (: \supset_o (d)(tt)(ff))$ is $\sqsupseteq tt$ and $\sqsupseteq ff$.

For on the one hand,

$$\begin{aligned} :\supset_o (d)(tt)(ff) \sqsupseteq {:}\supset_o (d)(tt)(\perp) &= {:}\supset_o (d)(T(tt))(T(ff)) \\ &\sqsupseteq T(d) \quad \text{(by the above remarks)} \\ &= tt \quad (\text{as } d \not\sqsubseteq ff), \end{aligned}$$

and on the other,

$$\begin{aligned} :\supset_o (d)(tt)(ff) \sqsupseteq {:}\supset_o (d)(\perp)(ff) &= {:}\supset_o (d)(F(tt))(F(ff)) \\ &\sqsupseteq F(d) \quad \text{(by the above remarks)} \\ &= ff \quad (\text{as } d \not\sqsubseteq tt). \end{aligned}$$

Now Ω_o denotes $\perp$ as $\Omega_o \sqsubseteq_{\mathscr{A}} p$ and further $\Omega_o \equiv_{\mathscr{A}} \supset_o p(\supset_o p\Omega_o tt)\Omega_o$. Therefore,

$$\begin{aligned} \perp &= \supset_o (e)(\supset_o (e)(\perp)(tt))(\perp) \\ &\sqsupseteq \supset_o (tt)(\supset_o (ff))(\perp)(tt)h(\perp) \quad (\text{as } e \sqsupseteq tt, ff) \\ &= tt \quad (\text{as } \supset_o ffpq \equiv_{\mathscr{A}} q \text{ and } \supset_o ttpq \equiv_{\mathscr{A}} p). \end{aligned}$$

But this contradicts the fact that $tt \not\sqsubseteq_{\mathscr{A}} \Omega_o$.

Therefore we have indeed proved that every element d of D_o is either $\sqsubseteq tt$ or ff.

Now we show that every element, d, of E_o is $\perp$, tt or ff. Let

$$J = \hat{\mathscr{A}}[\![\lambda p : \supset_o p(\alpha_0^{(o,o)} p)(\alpha_1^{(o,o)} p)]\!](\perp[T/\alpha_0^{(o,o)}][F/\alpha_1^{(o,o)}]).$$

Clearly, $J(x) = {:}\supset_o (x)(T(x))(F(x))$ for any x in E_o as β-conversion is valid in $\hat{\mathscr{A}}$. We will show that $J(x) = x$ (x in E_o). We have:

$$\begin{aligned} x &= {:}\supset_o (x)(tt)(ff) \quad (\text{as } p \equiv_{\mathscr{A}} (:\supset_o p\, tt ff)) \\ &= {:}\supset_o (x)(J(tt))(J(ff)) \\ &\sqsupseteq J(x) \\ &\sqsupseteq \supset_o (x)(J(tt))(J(ff)) \\ &= \supset_o (x)(tt)(ff) \\ &= x \quad (\text{as } p \equiv_{\mathscr{A}} (\supset_o p\, tt ff))(x \in D_o). \end{aligned}$$

Now let d be in E_o. As $d \sqsubseteq tt$ or $d \sqsubseteq ff$ there are three cases.

(a) $d \sqsubseteq tt$ and $d \sqsubseteq ff$. Then $d = J(d)$

$$= :\supset_o(d)(\bot)(\bot)$$
$$= \bot \text{ (as: } \supset_o pqq \equiv_{\mathscr{A}} q\text{)}.$$

(b) $d \sqsubseteq tt$ and $d \not\sqsubseteq ff$. Then $d = J(d)$

$$= :\supset_o(d)(tt)(\bot) = :\supset_o(d)(T(tt))(T(ff))$$
$$\sqsupseteq T(d)$$
$$= tt.$$

(c) $d \not\sqsubseteq tt$ and $d \sqsubseteq ff$. Then $d = J(d) = :\supset_o(d)(\bot)(ff)$

$$= :\supset_o(d)(F(tt))(F(ff))$$
$$\sqsupseteq F(d)$$
$$= ff.$$

So $E_o = \{\bot, tt, ff\}$. Further $\bot \sqsubseteq tt, ff$ as $\Omega_o \sqsubseteq_{\mathscr{A}} tt, ff$, $tt \not\sqsubseteq$ either $\bot$ or ff as $tt \not\sqsubseteq_{\mathscr{A}}$ either Ω_o or ff, and similarly ff is $\not\sqsubseteq$ either $\bot$ or tt. Therefore E_o is isomorphic to D_o. Let $\Psi_o : E_o \to \mathbf{T}$ be the isomorphism defined by:

$$\Psi_o(e) = \begin{cases} tt & (e = \hat{\mathscr{A}}[\![tt]\!](\bot)), \\ ff & (e = \hat{\mathscr{A}}[\![ff]\!](\bot)), \\ \bot & (e = \hat{\mathscr{A}}[\![\Omega_o]\!](\bot)). \end{cases}$$

Next we show that E_ι is isomorphic to D_ι. It is clear that the elements $\hat{\mathscr{A}}[\![k_n]\!](\bot)$ are all incomparable and $\sqsupseteq \hat{\mathscr{A}}[\![\Omega_\iota]\!](\bot)$ which is $\bot$. Now, $x \equiv_{\mathscr{A}} Y_{(\iota,\iota)} Mx$ where,

$$M = \lambda \alpha_0^{(\iota,\iota)}(\supset_\iota (Zx) k_o((\alpha_0^{(\iota,\iota)}(x-1)) + 1)).$$

As $Y_{(\iota,\iota)}(f) = \bigsqcup_{n \geqslant 0} f^n(\bot)$, it follows that, if d is in E_ι, then $d = \bigsqcup_{n \geqslant 0}(M^n(\bot)(d))$ (by the continuity of Ap_ι^ι).

We show by induction on n that for all d in D_ι, $M^n(\bot)(d)$ is in $\{\hat{\mathscr{A}}[\![k_n]\!](\bot) : n \geqslant 0\} \cup \{\bot\}$. It will then follow immediately that E_ι is isomorphic to D_ι and we may define an isomorphism $\Psi_\iota : E_\iota \to D_\iota$ by:

$$\Psi_\iota(e) = \begin{cases} k_n & (e = \hat{\mathscr{A}}[\![k_n]\!](\bot)), \\ \bot & (\text{otherwise}). \end{cases}$$

Now for $n = 0$, clearly $M^0(\bot)(d) = \bot$.

For $n+1$, $M^{n+1}(\bot)(d) = (\supset_\iota (Zd)(k_0)(M^n(\bot)(d-1) + 1))$.

As Zd is in E_o it is either $\bot$, tt or ff. If it is $\bot$ or tt then the LHS is $\bot$ or k_0 as $\supset_\iota \Omega_o xy \equiv_{\mathscr{A}} \Omega_o$ and $\supset_\iota ttxy \equiv_{\mathscr{A}} x$. Otherwise it is ff and the LHS is $M^n(\bot)(d-1) + 1$

as $\supset_\iota ffxy \equiv_{\mathscr{A}} y$. Now $M^n(\perp)(d-1)$ is either $\perp$ or some k_n. In the first case the LHS is $\perp$ as $\Omega_\iota + 1 \equiv_{\mathscr{A}} \Omega_\iota$. In the second it is k_{n+1} as $k_n + 1 =_{\mathscr{A}} k_{n+1}$. This completes the inductive proof.

Let $\Phi : \hat{\mathscr{A}}_{PA} \rightarrow \mathscr{A}$ be the monomorphism constructed in the proof of Theorem 4.6. We will construct an inverse morphism $\Psi : \mathscr{A} \rightarrow \hat{\mathscr{A}}_{PA}$. The Ψ_σ's are defined by induction. First Ψ_o and Ψ_ι are as defined above; $\Psi_{\sigma\rightarrow\tau} : E_{(\sigma\rightarrow\tau)} \rightarrow D_{(\sigma\rightarrow\tau)}$ is defined by:

$$\Psi_{\sigma\rightarrow\tau}(f)(d) = \Psi_\tau(Ap_\tau^\sigma(f)(\Phi_\sigma(d)))(f \in E_{\sigma\rightarrow\tau}, x \in D_\sigma).$$

It is clearly continuous.

We now prove by induction that Φ_τ and Ψ_τ are inverses for all τ. This is clear when τ is ground. For $\sigma \rightarrow \tau$, f in $E_{\sigma\rightarrow\tau}$ and e in E_σ, we calculate:

$$\begin{aligned} Ap_\tau^\sigma(\Phi_{\sigma\rightarrow\tau} \circ \Psi_{\sigma\rightarrow\tau}(f))(e) &= Ap_\tau^\sigma(\Phi_{\sigma\rightarrow\tau} \circ \Psi_{\sigma\rightarrow\tau}(f))(\Phi_\sigma \circ \Psi_\sigma(e)) \\ &\qquad \text{(by induction hypothesis)} \\ &= \Phi_\tau((\Psi_{\sigma\rightarrow\tau}(f))(\Psi_\sigma(e))) \\ &= \Phi_\tau(\Psi_\tau(Ap_\tau^\sigma(f)(\Phi_\sigma(\Psi_\sigma(e))))) \\ &\qquad \text{(by definition of } \Psi_{\sigma\rightarrow\tau}) \\ &= Ap_\tau^\sigma(f)(e) \quad \text{(by induction hypothesis).} \end{aligned}$$

As $\hat{\mathscr{A}}$ is pointwise ordered, it follows that $\Phi_{\sigma\rightarrow\tau} \circ \Psi_{\sigma\rightarrow\tau} = \mathrm{Id}_{E_{(\sigma\rightarrow\tau)}}$. Further,

$$\begin{aligned} \Phi_{\sigma\rightarrow\tau} \circ (\Psi_{\sigma\rightarrow\tau} \circ \Phi_{\sigma\rightarrow\tau}) &= (\Phi_{\sigma\rightarrow\tau} \circ \Psi_{\sigma\rightarrow\tau}) \circ \Phi_{\sigma\rightarrow\tau} \\ &= \mathrm{Id}_{E_{(\sigma\rightarrow\tau)}} \circ \Phi_{\sigma\rightarrow\tau} \\ &= \Phi_{\sigma\rightarrow\tau} \circ \mathrm{Id}_{D_{(\sigma\rightarrow\tau)}}. \end{aligned}$$

Therefore as $\Phi_{\sigma\rightarrow\tau}$ is 1-1, $\Psi_{\sigma\rightarrow\tau} \circ \Phi_{\sigma\rightarrow\tau} = \mathrm{Id}_{D_{(\sigma\rightarrow\tau)}}$ which concludes the induction.

We can now establish condition (1) that Ψ be a morphism. For $f \in E_{\sigma\rightarrow\tau}$ and x in E_σ, we have:

$$\begin{aligned} \Psi_{\sigma\rightarrow\tau}(f)(\Psi_\sigma(x)) &= \Psi_\tau(Ap_\tau^\sigma(f)(\Phi_\sigma \circ \Psi_\sigma(x))) \quad \text{(by definition of } \Psi_{\sigma\rightarrow\tau}) \\ &= \Psi_\tau(Ap_\tau^\sigma(f)(x)) \quad \text{(as } \Phi_\sigma \circ \Psi_\sigma = \mathrm{Id}_{E_\sigma}). \end{aligned}$$

For conditions (2) and (3) it is enough to prove that

$$(*) \qquad \Phi_\sigma(\hat{\mathscr{A}}[\![M_\sigma]\!])(\perp) = \hat{\mathscr{B}}[\![M_\sigma]\!](\perp),$$

(M_σ a closed term of type σ) as it then follows that

$$\Psi_\sigma(\hat{\mathscr{B}}[\![M_\sigma]\!])(\perp) = \Psi_\sigma \circ \Phi_\sigma(\hat{\mathscr{A}}[\![M_\sigma]\!])(\perp) = \hat{\mathscr{A}}[\![M_\sigma]\!](\perp).$$

When σ is of ground type $(*)$ holds by virtue of condition (3) that Φ be a morphism. For $\sigma = (\sigma_1, \ldots, \sigma_n, \tau)$, with τ ground, let $e_1, \ldots, e_n$ be finite elements of

$E_{\sigma_1}, \ldots, E_{\sigma_n}$. Then there are closed terms $M_1, \ldots, M_n$ of types $\sigma_1, \ldots, \sigma_n$ defining the finite elements $\Psi_{\sigma_1}(e_1), \ldots, \Psi_{\sigma_1}(e_n)$. Then,

$$\begin{aligned}
&\Phi_\sigma(\hat{\mathscr{A}}[\![M_\sigma]\!](\bot))(e_1)\cdots(e_n) = \\
&\quad = \Phi_\sigma(\hat{\mathscr{A}}[\![M_\sigma]\!](\bot))(\Phi_{\sigma_1}(\hat{\mathscr{A}}[\![M_{\sigma_1}]\!](\bot)))\cdots(\Phi_{\sigma_n}(\hat{\mathscr{A}}[\![M_{\sigma_n}]\!](\bot))) \\
&\quad = \Phi_\sigma(\hat{\mathscr{A}}[\![M_\sigma M_{\sigma_1}\cdots M_{\sigma_n}]\!](\bot)) \\
&\quad = \hat{\mathscr{B}}[\![M_\sigma M_{\sigma_1}\cdots M_{\sigma_n}]\!](\bot) \\
&\quad = \hat{\mathscr{B}}[\![M_\sigma]\!](\bot)(\hat{\mathscr{B}}[\![M_{\sigma_1}]\!](\bot))\cdots(\hat{\mathscr{B}}[\![M_{\sigma_n}]\!](\bot)) \\
&\quad = \hat{\mathscr{B}}[\![M_\sigma]\!](\bot)(\Phi_{\sigma_1}(\hat{\mathscr{A}}[\![M_{\sigma_1}]\!](\bot)))\cdots(\Phi_{\sigma_n}(\hat{\mathscr{A}}[\![M_{\sigma_n}]\!](\bot))) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by a known property of } \Phi) \\
&\quad = \hat{\mathscr{B}}[\![M_\sigma]\!](\bot)(e_1)\cdots(e_n).
\end{aligned}$$

Therefore as $\hat{\mathscr{A}}$ is pointwise ordered and E_σ is algebraic, (*) holds. Therefore Ψ is a morphism. Now, as Φ_σ and Ψ_σ are inverses for all σ, so are Φ and Ψ. Both are therefore isomorphisms, concluding the proof. □

A careful examination of the proof shows that we do not need to use all the power of full abstraction. One only actually needs a recursive subset of the relations $\{M \sqsubseteq_{\hat{\mathscr{A}}} N : M \underset{\sim}{\sqsubset}_{PA} N\}$.

The theorem applies to any fully abstract standard interpretation and indeed to any fully abstract interpretation in which $Y_{(\iota,\iota)}$ is given the standard denotation. It also applies to any fully abstract pointwise ordered general interpretation whose ground domains are isomorphic to the corresponding ground domains of $\hat{\mathscr{A}}_{PA}$, although it is quicker to note that in the above proof the assumptions about $Y_{(\iota,\iota)}$, T and F are only used to establish the ground domain isomorphisms. We conjecture that there is a fully abstract interpretation in which $Y_{(\iota,\iota)}$ is not given the standard denotation.

5. Computability

It is now time to face a question raised in Section 3. Suppose we are given the collection, $\{D_\sigma\}$ of domains for $\mathscr{L}_{PA}$ provided by $\mathscr{A}_{PA}$. Is $\mathscr{L}_{PA}$ the correct language in the sense that all computable elements are definable by $\mathscr{L}_{PA}$-terms and, conversely, all $\mathscr{L}_{PA}$-terms define computable elements? It seems natural to take as computable elements the lub's of recursively enumerable sets of finite elements. To make this idea precise we show first how to code the finite elements by integers.

$\sigma = o$: Let 0 code $e_0^o = \bot$, 1 code $e_1^o = tt$ and $2, 3, \ldots$ code $e_2^o, e_3^o, \ldots$ where $e_n^o = ff$ if $n \geq 2$. Thus $e_0^o, e_1^o, \ldots$ enumerates the finite elements of D_o.

$\sigma = \iota$: Let 0 code $e_0^\iota = \bot$ and $(n+1)$ code $e_{n+1}^\iota = n$. Thus $e_0^\iota, e_1^\iota, \ldots$ enumerates the finite elements of D_ι.

$\sigma = (\sigma_1, \ldots, \sigma_n, \tau)$ with τ ground and $n > 0$: Let $\langle \cdot, \ldots, \cdot \rangle$ be any coding function of $(n+1)$-tuples of integers, each integer m being the code of $\langle (m)_1, \ldots, (m)_n, (m)_{n+1} \rangle$.

Then $m = \sum_{i \geq 0} 2^{m_i}$ codes $e_m^\sigma = \sqcup F$ where, if

$$F' = \left\{ \left(e_{(m_i)_1}^{\sigma_1} \Rightarrow \cdots \Rightarrow (e_{(m_i)_n}^{\sigma_n} \Rightarrow e_{(m_i)_{(n+1)}}^{\tau}) \cdots \right) : i \geq 0 \right\}$$

then either F' satisfies the conditions mentioned after Lemma 4.4, and $F = F'$ or else if F' does not satisfy these conditions $F = \emptyset$. In the second case we say that m is trivial as a σ-code. Then $e_0^\sigma, e_1^\sigma, \ldots$ enumerates the finite elements of D_σ, $e_0^\sigma = \bot$.

The relations $e_m^\sigma \sqsubseteq e_n^\sigma$, $e_n^\sigma \sqcup e_m^\sigma$ exists and the functions $e_n^{\sigma\to\tau}(e_m^\sigma) = e_k^\tau$ and $e_n^\sigma \bar{\sqcup} e_m^\sigma = e_k^\sigma$, where $e_n^\sigma \bar{\sqcup} e_m^\sigma = e_n^\sigma \sqcup e_m^\sigma$ if it exists and e_0^σ otherwise, are all primitive recursive in their indices.

So we say that $x \in D_\sigma$ is *computable* iff $\{n : e_n^\sigma \sqsubseteq x\}$ is r.e.

An equivalent definition is that x is the lub of a chain $e_{n_0}^\sigma \sqsubseteq e_{n_1}^\sigma \sqsubseteq \cdots$ where the sequence $n_0, n_1, \ldots$ is primitive recursive.

It turns out that every $\mathcal{L}_{PA}$-definable element is computable, but, perhaps surprisingly, the converse fails. One way to see that every $\mathcal{L}_{PA}$-term defines a computable element is first to notice by the usual combinatory methods that, to within $\equiv_{PA}$, all the terms are combinations of the terms $(+1)$, (-1), k_0, *tt*, *ff*, Z, $\supset_\sigma$, $:\supset_\sigma$, Y_{σ_1}, $S_{\sigma_1,\sigma_2,\sigma_3}$ and K_{σ_1,σ_2} (all σ_1, σ_2 and σ_3 and ground σ) where:

$$K_{\sigma_1,\sigma_2} = \lambda\alpha_1^{\sigma_1}\lambda\alpha_2^{\sigma_2}\alpha_1^{\sigma_1},$$

$$S_{\sigma_1,\sigma_2,\sigma_3} = \lambda\alpha_3^{(\sigma_1,\sigma_2,\sigma_3)}\lambda\alpha_2^{(\sigma_1,\sigma_2)}\lambda\alpha_1^{\sigma_1}((\alpha_3^{(\sigma_1,\sigma_2,\sigma_3)}\alpha_1^{\sigma_1})(\alpha_2^{(\sigma_1,\sigma_2)}\alpha_1^{\sigma_1})).$$

Secondly all these terms define computable elements and the class of computable elements is closed under application. To see this for $K_{\sigma_1\sigma_2}$, $S_{\sigma_1\sigma_2\sigma_3}$ and Y_{σ_1}, one proves, without much difficulty that:
K_{σ_1,σ_2} defines

$$\sqcup\{e_{n_1}^{\sigma_1} \Rightarrow e_{n_2}^{\sigma_2} \Rightarrow e_{n_1}^{\sigma_1} : n_1, n_2 \geq 0\},$$

$S_{\sigma_1,\sigma_2,\sigma_3}$ defines

$$\sqcup\{(e_{n_1}^{\sigma_1} \Rightarrow e_{n_2}^{\sigma_2} \Rightarrow e_{n_3}^{\sigma_3}) \Rightarrow (e_{n_1}^{\sigma_1} \Rightarrow e_{n_2}^{\sigma_2}) \Rightarrow e_{n_1}^{\sigma_1} \Rightarrow e_{n_3}^{\sigma_3} : n_1, n_2, n_3 \geq 0\},$$

and Y_{σ_1} defines

$$\sqcup\{((e_0^{\sigma_1} \Rightarrow e_{n_1}^{\sigma_1}) \sqcup \cdots \sqcup (e_{n_k}^{\sigma_1} \Rightarrow e_{n_{k+1}}^{\sigma_1})) \Rightarrow e_{n_{k+1}}^{\sigma_1} :$$
$$\text{if it exists, for } k \geq 1,\ n_1, \ldots, n_{k+1} \geq 0\}.$$

The other terms are easy and closure under application follows from:

$$f(x) = \sqcup\{e : (d \Rightarrow e) \sqsubseteq f \text{ and } d \sqsubseteq x\} \quad (f \in D_{\sigma\to\tau}, x \in D_\sigma).$$

One computable element which is not $\mathcal{L}_{PA}$-definable is $\exists: D_{((\iota,o),o)}$ defined by:

$$\exists(f)=\begin{cases} ff & (f(\bot)=ff),\\ tt & (f(n)=tt \text{ for some } n\geq 0),\\ \bot & (\text{otherwise}).\end{cases}$$

It can be regarded as the best continuous approximation to the existential quantifier.

Let F be a term defining $\bot_\iota \Longrightarrow ff$ and let T_n be a term defining $n \Longrightarrow tt$ for each $n\geq 0$. We prove that whenever E_σ is a term of ground type with $FV(E)=\{\alpha^{(\iota\to o)}\}$ such that $[F/\alpha^{(\iota\to o)}]E$ and $[T_n/\alpha^{(\iota\to o)}]E$ denote different things neither of which is $\bot_\sigma$, for infinitely many $n\geq 0$ then $[F/\alpha^{(\iota\to o)}]E$ cannot terminate in less than k steps for any k (that is if $[F/\alpha^{(\iota\to o)}]E \xrightarrow{k'} c$ then $k'>k$). Since a definition, M, of $\exists$ would provide such a term, namely $M\alpha^{(\iota\to o)}$, we would then find that MF cannot terminate, and yet denotes ff which contradicts the analogue for PA of Theorem 3.1 showing that $\exists$ is not $\mathcal{L}_{PA}$-definable.

The proof proceeds by induction on k and cases on the form of E_σ. We can assume w.l.o.g. that $\supset_D$ does not occur in E_σ. For $k=0$ since $[F/\alpha^{(\iota\to o)}]E_\sigma$ is not a constant the assertion is obvious.

For $k>0$ the first case is when E_σ has the form:

$$(\lambda\alpha E_1)E_2\cdots$$

Here $E\to_{PA}E'$ iff $E'=[E_2/\alpha]E_1\cdots$. Since $E'\equiv_{PA}E$ we can apply the induction hypothesis to E'.

The case where E has the form $YE_1\cdots$ is similar.

The case where E is a constant cannot occur.

In the case where E has the form $((+1)E_1)$ or $((-1)E_1)$ or (ZE_1) apply the induction hypothesis to E_1.

The case $E=(\alpha^{(\iota\to o)}E_1)$ leads to a contradiction, for then there are $n,n'\geq 0$, with $n\neq n'$ such that neither of $[T_n/\alpha^{(\iota\to o)}]E$ nor $[T_{n'}/\alpha^{(\iota\to o)}]E$ define $\bot_\sigma$. Then $[T_n/\alpha^{(\iota\to o)}]E_1$ and $[T_{n'}/\alpha^{(\iota\to o)}]E_1$ must define n and n' respectively. So if T defines $(n \Longrightarrow tt)\sqcup(n' \Longrightarrow tt)$, $[T/\alpha^{(\iota\to o)}]E_1$ defines something $\sqsupseteq n,n'$, which is a contradiction.

Finally we consider the case where E has the form: $\supset E_1E_2E_3$.

Suppose $[F/\alpha^{(\iota\to o)}]E$ terminates in k steps. There are three subcases.

(1) $[F/\alpha^{(\iota\to o)}]E_1$ terminates in k_1 steps with value tt and $[F/\alpha^{(\iota\to o)}]E_2$ terminates in k_2 steps, and $k>(k_1+k_2)$. By induction assumption, for all but finitely many $n\geq 0$, $[T_n/\alpha]E_1$ defines $\bot_o$ and $[T_n/\alpha]E_2$ defines $\bot_\sigma$. Then for all but finitely many $n\geq 0$, $[T_n/\alpha^{(\iota\to o)}]E$ defines $\bot_\sigma$. This contradiction finishes this subcase.

(2) As in case (1) but with $[F/\alpha^{(\iota\to o)}]E_1$ terminating with value ff and $[F/\alpha^{(\iota\to o)}]E_3$ terminating.

(3) $[F/\alpha^{(\iota\to o)}]E_i$ terminates in k_i steps with value c, for $i = 1,2$ where $k_1 + k_2 < k$. Here for almost all $n \geq 0$, and $i = 1,2$ $[T_n/\alpha^{(\iota\to o)}]E_i$ defines $\perp_\sigma$, which concludes the proof.

It is now natural to add a constant $\exists$ of type $((\iota, o), o)$ to $\mathcal{L}_{PA}$ obtaining $\mathcal{L}_{PA+\exists}$ and adding a clause in the definition of $\mathcal{A}_{PA}$, obtaining the definition of $\mathcal{A}_{PA+\exists}$. The definition of $\text{Eval}_{PA+\exists}$ and the examination of its properties is postponed. From the point of view of computability no more constants need be introduced.

Theorem 5.1. (Definability theorem.) *An element of D_σ is computable iff it is definable from $\mathcal{L}_{PA+\exists}$.*

First we may regard a primitive recursive function f of n arguments as a member $\hat{f}$ of D_σ where $\sigma = (\iota, \ldots, \iota, \iota)$ by:

$$\hat{f}(x_1)\cdots(x_n) = \begin{cases} \perp & \text{(if some } x_i = \perp\text{)}, \\ f(x_1, \ldots, x_n) & \text{(otherwise)}. \end{cases}$$

Under this identification it is straightforward to show that every primitive recursive function is definable by an $\mathcal{L}_{DA}$-term.

Similarly if a primitive recursive predicate P of n arguments is regarded as a member of D_σ with $\sigma = (\iota, \ldots, \iota, o)$ in the corresponding way, it can be defined by an $\mathcal{L}_{DA}$-term.

The *apartness* relation $\#^\sigma$ on D_σ is defined by:

$x \#^\sigma y$ iff x and y have no upper bond.

In general, $x \#^\sigma y$ iff there are finite elements $d, e \sqsubseteq x, y$ respectively such that $d \#^\sigma e$. A criterion for the existence of an upper bound of d and e has been given earlier.

For $\sigma = (\sigma_1 \to \sigma_2)$, $x \#^\sigma y$ iff for some finite e in D_{σ_1}, $x(e) \#^{\sigma_2} y(e)$.

From this one can show that for $x, y, z \in D_\sigma$, if $y \sqcup z$ exists then $x \#^\sigma (y \sqcup z)$ iff $x \#^\sigma y$ or $x \#^\sigma z$.

A computable approximation to $x \#^\sigma e_n^\sigma$ is provided by $\#_n^\sigma : D_\sigma \to D_o$ defined by:

$$\#_n^\sigma(x) = \begin{cases} ff & (x \sqsupseteq e_n), \\ tt & (x \#^\sigma e_n^\sigma), \\ \perp & \text{(otherwise)}. \end{cases}$$

That $\#_n^\sigma$ is continuous follows from previous remarks.

Lemma 5.2. *There are $\mathcal{L}_{PA+\exists}$ terms $\#^\sigma : D_{(\iota,\sigma,o)}$ such that $(\#^\sigma k_n)$ defines $\#_n^\sigma$. If σ is ground $\#^\sigma$ is an $\mathcal{L}_{PA}$-term.*

Proof. The proof is like that of Lemma 4.5. It is an induction on σ, but the subcases are, as it were, handled primitive recursively inside the $\#^{\sigma}$'s rather than in the proof.

In fact the terms $\#^{\sigma}_{n}$ and also closed terms $EN^{\sigma}: (\iota, \iota, o)$ are defined by induction on σ. Each $(EN^{\sigma}k_n)$ defines a function f, such that:

$$f(\bot) \sqsupseteq e^{\sigma}_{n},$$

$f(0), f(1), \ldots$ is a list of the finite elements of D_{σ} which are $\sqsupseteq e^{\sigma}_{n}$.

$\sigma = o$: Take $\#^{o} = \lambda x \lambda p$ $(if_o\ Zx\ then\ ff\ else\ (if\ Z((-1)x)\ then\ \text{NEG}\ p\ else\ p))$ *and* $EN^{o} = \lambda x \lambda y$ $(if_o\ Zx\ then\ Z((-1)y)\ else\ Z((-1)x))$.

Here $(if_{\sigma} \cdots\ then\ \text{---}\ else\ \sim\sim\sim)$ abbreviates $(\supset_{\sigma}(\cdots)(\text{---})(\sim\sim\sim))$;
NEG is the term defined in the proof of Lemma 4.5.

$\sigma = \iota$: Take $\#^{\iota} = \lambda x \lambda y$ $(if_o\ Zx\ then\ ff\ else\ \text{NEG}(\text{EQ}((-1)x)y))$, and $EN^{\iota} = \lambda x \lambda y$ $(if_{\iota}\ Zx\ then\ (-1)y\ else\ (-1)x)$.

Here EQ is an $\mathcal{L}_{\text{DA}}$-term defining the (p.r.) equality predicate, under the above identification.

$\sigma = (\sigma_1, \ldots, \sigma_n, \tau)$ with τ ground and $n > 0$: First define the context OR[ ] by:

$$\text{OR}[\ ] = Y_{(\iota \to o)}(\lambda \alpha^{(\iota \to o)} \lambda x (if_o\ Zx\ then\ ff\ else\ V([\ ]x)(\alpha^{(\iota \to o)}((-1)x)))).$$

Here V is the term defining parallel or given above.

If $F: (\iota \to o)$ is an open term then if all of $(Fk_1) \cdots (Fk_m)$ define ff in an environment ρ then $(\text{OR}[F]k_m)$ defines ff in ρ and if one of them defines tt in ρ, $(\text{OR}[F]k_m)$ defines tt in ρ, $(m \geq 1)$.

Also $(\text{OR}[F]k_0)$ defines ff.

Then take

$$\#^{\sigma} = \lambda x \lambda \alpha^{\sigma} (\text{OR}[\lambda z \exists (\lambda y\ \#^{\sigma^1}(\text{SECOND}\ xz)$$
$$(\alpha^{\sigma}(\text{EN}^{\sigma_1}(\text{FIRST}\ xz)y)))](\text{SIZE}\ x)).$$

Here $\sigma^1 = (\sigma_2, \ldots, \sigma_n, \tau)$ and FIRST, SECOND and SIZE define primitive recursive functions f, g and h such that if m is a trivial σ-code or the set F associated with e^{σ}_{m} by m is empty then $h(m) = 0$; otherwise $h(m)$ is the size of F and $F = \{e^{\sigma_1}_{f(m,i)} \Longrightarrow e^{\sigma^1}_{g(m,i)} : i = 1, h(m)\}$.

It follows from the facts that $e^{\sigma}_{m} \#^{\sigma} d$ iff $(e^{\sigma_1}_{f(m,i)} \Longrightarrow e^{\sigma^1}_{g(m,i)}) \#^{\sigma} d$ for some i between 1 and $h(m)$ iff there is a finite $e \sqsupseteq e^{\sigma_1}_{f(m,i)}$ such that $e^{\sigma^1}_{g(m,i)} \#^{\sigma^1} d(e)$, for some i between 1 and $h(m)$ $(d \in D_{\sigma}; h(m) \geq 1)$, that this is a correct definition of $\#^{\sigma}$.

Next set

$$K = \lambda \alpha^{\sigma_1}_{1} \cdots \lambda \alpha^{\sigma_n}_{n} \lambda y \lambda \alpha^{\tau}_{(n+1)} (Y_{(\iota \to \tau)}(\lambda \alpha^{(\iota \to \tau)}_{(n+2)} \lambda z\ (if_{\tau}\ Zz\ then\ \alpha^{\tau}_{(n+1)}$$
$$else\ (:\supset_{\tau}((\#^{\sigma_1}(\text{COMP}\ yk_1 z)\alpha^{\sigma_1}_{1}) \vee \cdots \vee$$
$$(\#^{\sigma_n}(\text{COMP}\ yk_n z)\alpha^{\sigma_n}_{n}))(\alpha^{(\iota \to \tau)}_{(n+2)}((-1)z))(\text{EN}^{\tau}k_o\ (\text{COMP}\ yk_{n+1}z))))(\text{SIZE}\ z)).$$

COMP: $D_{(\iota,\iota,\iota,\iota)}$ defines a primitive recursive function j such that if $h(m) \neq 0$ then the set F associated with e^{σ}_{m} by m is $\{e^{\sigma_1}_{j(m,1,l)} \Rightarrow \cdots \Rightarrow e^{\sigma_n}_{j(m,n,l)} \Rightarrow e^{\tau}_{j(m,n+1,l)}$: $1 \leqslant l \leqslant h(m)\}$.

Notice that if, for $x_1, \ldots, x_n$ in $D_{\sigma_1}, \ldots, D_{\sigma_n}$, $e^{\sigma}_{m}x_1 \cdots x_n = d' \neq \bot$ and $d' \neq e^{\tau}_{j(m,n+1,l)}$ then for some i, with $1 \leqslant i \leqslant n$, $e^{\sigma_i}_{j(m,i,l)} \#^{\sigma_i} x_i$ ($h(m) > 0$ and $1 \leqslant l \leqslant h(m)$).

K defines k: $D_{(\sigma_1,\ldots,\sigma_n,\iota,\tau,\tau)}$ and if the set F associated with m is $\emptyset$ then $kx_1, \ldots, x_n m\, d = d$ and if $e^{\sigma}_{m}x_1 \cdots x_n = d' = \bot$ then $kx_1 \cdots x_n m\, d = d'$ and if for all l, with $1 \leqslant l \leqslant h(m)$ there is an i with $1 \leqslant i \leqslant n$ such that $x^{\sigma_i}_{i} \# e^{\sigma_i}_{j(m,i,l)}$ then $kx_1 \cdots x_n m\, d = d$ ($x_i \in D_{\sigma_i}$, $d \in D_{\tau}$).

Finally, if $e^{\sigma}_{m}x_1 \cdots x_n = \bot$, and whenever $e^{\sigma_i}_{j(m,i,l)} \#^{\sigma_i} x_i$ is false for all i between 1 and n, $e^{\tau}_{j(m,n+1,l)} = d$ then $kx_1 \cdots x_n m\, d = d$ and otherwise $kx_1 \cdots x_n m\, d = \bot$.

Now we can take

$$\mathrm{EN}^{\sigma} = \lambda x \lambda y \lambda \alpha^{\sigma_1}_{1} \cdots \lambda \alpha^{\sigma_n}_{n}(K\alpha^{\sigma_1}_{1} \cdots \alpha^{\sigma_n}_{n} x (K\alpha^{\sigma_1}_{1} \cdots \alpha^{\sigma_n}_{n} y \Omega_{\tau})). \quad \square$$

It is now possible to prove the definability theorem. Suppose $x \in D_{\sigma}$ is computable. If σ is ground it is certainly definable, and if it is $\bot$ it is defined by Ω_{σ}. Otherwise there are primitive recursive functions $f_1, \ldots, f_n, g$ such that

$$x = \bigsqcup \{e^{\sigma_1}_{f_1(m)} \Rightarrow \cdots \Rightarrow e^{\sigma_n}_{f_n(m)} \Rightarrow e^{\tau}_{g(m)}: m \geqslant 0\},$$

where $\sigma = (\sigma_1, \ldots, \sigma_n, \tau)$ with τ ground, and $g(m) > 0$ for all $m \geqslant 0$. Let $F_1, \ldots, F_n, G$ define $f_1, \ldots, f_n, g$ respectively.

Then x can be defined by the term:

$$\lambda \alpha^{\sigma_1}_{1} \cdots \lambda \alpha^{\sigma_n}_{n}((Y_{(\iota \to \tau)}(\lambda \alpha^{(\iota \to \tau)}_{(n+1)} \lambda x (:\supset_{\tau} ((\#^{\sigma_1}(F_1 x)\alpha^{\sigma_1}_{1}) \vee \cdots \vee$$

$$(\#^{\sigma_n}(F_n x)\alpha^{\sigma_n}_{n}))(\alpha^{(\iota \to \tau)}_{(n+1)}((+1)x))(\mathrm{EN}^{\tau} k_0\, Gx))))k_0).$$

It is interesting to notice that only recursion operators of level 3 have been used. Notice too that $\exists$ is only needed to define elements of level $\geqslant 2$, confirming the impression that only systems of McCarthy-recursion equations perhaps with some parallel facilities, are needed to define functions of level 1.

It is not difficult to define enumeration functions $\{\ \}^{\sigma}: D_{\iota \to \sigma}$ such that if n codes an r.e. set of finite elements with lub, f then $\{n\}^{\sigma} = f$. Indeed this follows from the definability theorem. There are computable functions $a^{\sigma}_{\tau}: D_{(\iota,\iota,\iota)}$ such that $\{n\}^{(\sigma \to \tau)}(\{m\}^{\sigma}) = \{a^{\sigma}_{\tau}(n)(m)\}^{\tau}$ ($m, n \in D_{\iota} \neq \bot$), and one then obtains a version of Kleene's second recursion theorem: if $f: D_{\iota} \to D_{\sigma}$ is computable then $f(n) = \{n\}^{\sigma}$ for some $n \in D_{\iota}$. All of this is taken from [7]. We resist the temptation to go any further into recursion theory.

The definability theorem is a completeness result. If $\mathscr{L}_{PA+\exists}$ is extended by adding other constants and then $\mathscr{A}_{PA+\exists}$ is extended to give them computable values, they can already be defined by $\mathscr{L}_{PA+\exists}$-terms. So there can be no new parallel facilities of the type exemplified by $\exists$ or $:\supset_{\sigma}$, and $\mathscr{L}_{PA+\exists}$ is determined as the maximal language of computable functions of $\{D_{\sigma}\}$, to within interdefinability.

Having satisfied ourselves as to the definability credentials of $\mathscr{L}_{PA+\exists}$ we give it an operational semantics which is provided by the rules for the reduction relation of $\mathscr{L}_{PA}$, extended to all $\mathscr{L}_{PA+\exists}$-terms together with the following:

(I8) $\dfrac{F\Omega_\iota \to^*_\exists ff}{\exists F \to_\exists ff}, \dfrac{Fk_m \to^*_\exists tt}{\exists F \to_\exists tt}$ $(m \geq 0)$,

(III1) $M_o \to^*_\exists M_o$,

(III2) $\dfrac{M_o \to_\exists M'_o, M'_o \to^*_\exists M''_o}{M_o \to^*_\exists M''_o}$.

Here $\to_\exists$ abbreviates $\to_{\mathscr{L}_{PA+\exists}}$.

Clearly $\to^*_\exists$ as defined is the transitive reflexive closure of $\to_\exists$ for terms of type o. It can be shown (by a simultaneous induction) that $\to_\exists$ is consistent in the sense that $\exists F \to_\exists ff$ and $\exists F \to_\exists tt$ cannot both hold and that if $M_1 \to_\exists M_i$ $(i = 2, 3)$ then for some term $M_4 M_i \to^*_\exists M_4$ $(i = 2, 3)$. Then the Church–Rosser theorem holds for $\to^*_\exists$ and we can define the evaluation function, $\text{Eval}_{PA+\exists}$ by:

$$\text{Eval}_{PA+\exists}(M) = c \text{ iff } M \to^* c, \quad \text{for any program } M.$$

The computability lemma goes through as before, with addition of the easy proof of the computability of $\exists$. Thus we find that $\text{Eval}_{PA+\exists}(M) = c$ iff $\hat{\mathscr{A}}_{PA+\exists}[\![M]\!](\bot) = \mathscr{A}_{PA+\exists}[\![c]\!]$, since the other half is routine. The proof of Theorem 4.3 clearly extends and we have $M_\sigma \underset{\sim}{\sqsubseteq}_{PA+\exists} N_\sigma$ iff $M_\sigma \sqsubseteq_{PA+\exists} N_\sigma$ and also $M_\sigma \approx_{PA+\exists} N_\sigma$ iff $M_\sigma \equiv_{PA+\exists} N_\sigma$. Finally the analogue of Theorems 4.6 and 4.7 hold, the proofs being similar and of course the definability theorem holds too. With all this, we have at last an example of an operational and denotational semantics which fit together harmoniously.

As regards the problem of generalising these results to other domains than the integers, it seems feasible to carry them over to other discrete cpo's, that is, cpo's with no increasing chains of length > 2, considering in some way effectively given domains and base functions on them for the computability considerations. A more challenging problem would be to give a good definition of "concrete datatype" and take D to be an arbitrary one. Among the concrete datatypes should be the denumerable discrete domains, and the cpo of all finite and infinite sequences of integers with $\sqsubseteq$ taken as the initial subsequence ordering, and also, perhaps, the cpo of real intervals described in [8]. Roughly, the concrete datatypes should be effectively given cpo's for which a good notation exists so that notations for approximations to elements can be printed out, bit by bit.

Acknowledgements

This work has been carried out with the aid of an S.R.C. grant under the direction of Dr. R.M. Burstall.

References

[1] M. Gordon, Evaluation and denotation of pure LISP programs: a worked example in semantics, Ph.D. Thesis, University of Edinburgh, Edinburgh (1973).

[2] W.A. Howard, Assignment of ordinals to terms for primitive recursive functionals of finite type, in: A. Kino, J. Myhill, R. E. Vesley, eds., *Intuitionism and Proof Theory* (North-Holland, Amsterdam, 1970).

[3] R. Milner, Models of LCF, Memo. AIM-186, Stanford Artificial Intelligence Laboratory, Stanford University, Stanford, CA (1973).

[4] R. Milne, The formal semantics of computer languages and their implementations, Ph.D. Thesis, University of Cambridge, Cambridge (1974).

[5] M. Newey, Axioms and theorems for integers, lists and finite sets in LCF, Memo AIM-184, Computer Science Department, Stanford University, Stanford, CA (1973).

[6] G. D. Plotkin, Call-by-name, call-by-value and the λ-calculus, *Theoret. Comput. Sci* 1 (1975).

[7] D. Scott, A theory of computable functions of higher type, unpublished seminar notes, University of Oxford, Oxford (1969).

[8] D. Scott, Lattice theory, data types and semantics, in: P. Rustin, ed., *Formal Semantics of Programming Languages* (Prentice-Hall, Englewood Cliff, NJ, 1970).

[9] A.S. Troelstra, ed., *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, Lecture Notes in Math. **344** (Springer, Berlin, 1973).

[10] J. Vuillemin, Correct and optimal implementations of recursion in a simple programming language, *Proc. Fifth ACM Symposium on Theory of Computing* (1973) 224–239.

[11] C.P. Wadsworth Semantics and pragmatics of the lambda-calculus, University of Oxford, Oxford (1971).

[12] C.P. Wadsworth, The relationships between lambda-expressions and their denotations in Scott's models for the lambda-calculus (to appear).