Theoretical Computer Science 13 (1981) 151–174

THE CONGRUENCE OF TWO PROGRAMMING LANGUAGE DEFINITIONS

Joseph E. STOY

Oxford University Computing Laboratory, Programming Research Group, Oxford, United Kingdom

Communicated by M. Nivat
Received January 1978
Revised December 1979

1. Introduction

In recent joint work with Jack Dennis, which has so far appeared only as M.I.T. course notes [2], we have described several methods of formally defining the semantics of a programming language, using as an example a simple language designed for the purpose, called PL. The question naturally arose whether our various definitions were *congruent*, in the sense that they defined the same language. In particular, one definition gave the *denotational semantics*, in the style of Scott and Strachey [12], and another defined the language using an *interpreter*, along lines similar to the use of VDL [6]. In this paper we present a proof of the congruence of these two definitions.

We first give the two definitions, with a bare minimum of explanation: the reader is referred to works like [2, 13, 16] for more leisurely introductions to the methods. It will be seen that the interpreter manages the sequencing of its operations by means of structures called *continuations*, while the denotational semantics has nothing corresponding to these. A denotational semantics based on the use of continuations (as described by Strachey and Wadsworth [14]) would have a structure much closer to the interpreter's, and the proof of their congruence would be simplified. We therefore find it convenient to split up our proof into two stages. In the first stage we introduce a new version of denotational semantics, using continuations, and prove that it is congruent with the original version. This proof is a structural induction to show that pairs of corresponding values arising in the two definitions satisfy appropriate predicates. The existence of these predicates will itself require proof: we shall consider this problem briefly, but defer a fuller discussion until a similar question arises again in the second stage.

Secondly, we prove the congruence of the continuation semantics with the interpretive definition. The interpreter is defined as an iteration, and so it is natural to

use fixed point induction [7] to analyse its properties; but with this technique we can establish only that the value given for an expression by the interpreter is an *approximation* of that specified by the denotational semantics. We can also try structural induction, just as we did in the first stage; but here too it is necessary to demonstrate the existence of the various predicates we use in the proof. In fact we shall find we are unable to show the existence of the predicates which assert exact congruence between the definitions: we must be content with predicates asserting that the denotational value is an approximation of the interpretive one. The reasons why we must attack the problem from both sides in this way will be discussed further when we embark on the second stage; the final phase of the proof combines the two results into a satisfactory statement of congruence.

This proof provides an introduction to the techniques developed by Milne [8, 9], and used by him and Strachey [9] to prove the correctness of an implementation of a much more complicated language. Similar work has been carried out by Reynolds [11], who showed the congruence of versions of denotational semantics with and without continuations for a form of the λ-calculus. Our first stage closely resembles this proof, though our expression of it is a little different: Reynolds's use of "directed complete relations" corresponds to our use of "inclusive predicates".

2. The denotational semantics of PL

Parts of the definition in this section and the interpreter to be defined in the next have already appeared in [1]. We make one or two small changes in convention, and the careful reader will also be able to discern one or two changes which attest to the debugging effects of the construction of a congruence proof.

The definition is given in Appendix 1. For the sake of brevity we omit all references to the data structuring facilities of PL, and the actual definitions of $\mathscr{B}$, $\mathscr{C}$, and $\mathscr{W}$.

2.1. *Notational conventions*

Syntactic objects are denoted by Greek capitals ($E, \Gamma, \ldots$) and are elements of domains denoted by short names (**Exp**, **Com**, ...). Corresponding lower case Greek letters ($\varepsilon, \gamma, \ldots$) denote the appropriate semantic values, which come from domains denoted by bold capitals ($\boldsymbol{E}, \boldsymbol{C}, \ldots$); script letters ($\mathscr{E}, \mathscr{C}, \ldots$) are used for the 'valuation' functions which map syntactic objects to the values they denote.

$\boldsymbol{D}_1 + \boldsymbol{D}_2$ denotes the *separated* sum of domains $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$.

Each domain includes an *error element*, denoted by ?. $\bot \sqsubseteq ? \sqsubseteq \top$, of course, but ? is incomparable with all other elements. If an element is projected into a subdomain where it does not belong, it is mapped to ?; conversely, the ? element of each subdomain is mapped to ? in the sum. We shall refer to $\bot$, ? and $\top$ as *improper* elements of a domain; the predicate *Proper x* will be *true* provided x is neither $\bot$, ? nor $\top$.

For any domain D and $a, b \in D$, $a \equiv b$ is *true* if a is the same element as b, and *false* otherwise. We reserve '=' to denote a *continuous* equality predicate; thus, for example, if D is a flat lattice we might define $a = b$ to be $\perp$ if either a or b is $\perp$, $\top$ if neither is $\perp$ but either a or b is $\top$, and equal to $a \equiv b$ otherwise.

PL is simple enough for its semantics to be given solely in terms of environments, whereas in more complicated languages the notion of a store mapping is also required. In such languages commands normally specify store transformations rather than environment transformations, and some workers therefore prefer not to regard the PL domain **Com** as commands, but rather as *definitions*. Unfortunately, however, if $\rho \in U$ is to do duty for both environment and state, U cannot be simply $[\mathbf{Ide} \to E]$, as usual. We need, for example, to be able to distinguish between the error environment, arising from some invalid computation, and the environment which, though itself valid, associates all identifiers to ?. (This latter environment, $\lambda I.?$, is called the *arid* environment). U must therefore include an extra set of improper elements. The definitions of the operations $\rho[\![I]\!]$ and $\rho[\varepsilon/I]$ must be extended to cover these extra values: if ρ is $\perp_U$, $?_U$ or $\top_U$, then $\rho[\![I]\!]$ is $\perp_E$, $?_E$ or $\top_E$ and $\rho[\varepsilon/I] \equiv \rho$ for all I and ε.

3. The PL interpreter

This interpreter is defined only for a subset of PL, called the *kernel* language. It is shown in [1] that any PL program may be transformed into an equivalent one in this kernel language, of which the syntax is as follows:

$$E ::= I \mid \mathbf{I}(I) \mid \mathbf{proc}(I) : E \mid \mathbf{rec}\, I(I) : E \mid \Gamma\ \mathbf{res}\ E \mid$$
$$B \mid OI \mid I \Omega I,$$
$$\Gamma ::= I := E \mid \mathbf{while}\ I\ \mathbf{do}\ \Gamma \mid$$
$$\Gamma; \Gamma \mid \mathbf{if}\ I\ \mathbf{then}\ \Gamma\ \mathbf{else}\ \Gamma.$$

The interpreter operates on states. Informally, its action may be given by

$$\mathbf{until}\ \mathit{Terminal}(\sigma)\ \mathbf{do}\ \sigma := \mathit{Step}(\sigma), \tag{3.1}$$

where $\sigma \in \mathbf{State}$; more formally we may say

$$\mathit{Interpret} = \mathit{fix}(\lambda\phi.\lambda\sigma.\mathit{Terminal}(\sigma) \to \sigma, \phi(\mathit{Step}(\sigma))). \tag{3.2}$$

The syntax of states is as follows, where components are denoted by the same Greek letters as are used for the corresponding entities in denotational semantics (some of these will not be introduced until the next section):

$$\sigma ::= \mathbf{eval}\ E\ \mathbf{in}\ \rho; \kappa \mid$$
$$\mathbf{perform}\ \Gamma\ \mathbf{in}\ \rho; \theta \mid$$
$$\mathbf{assign}\ \varepsilon\ \mathbf{to}\ I\ \mathbf{in}\ \rho; \theta \mid$$
$$\mathbf{done}\ \varepsilon \mid \mathbf{error}.$$

Here, κ and θ denote the 'continuations' of expression and command evaluations; their syntax is as follows.

$$\kappa ::= \textbf{done}\ \langle\rangle \mid \textbf{assign}\ \langle\rangle\ \textbf{to}\ l\ \textbf{in}\ \rho;\ \theta,$$

$$\theta ::= \textbf{perform}\ \Gamma\ \textbf{in}\ \langle\rangle;\ \theta \mid \textbf{eval}\ E\ \textbf{in}\ \langle\rangle;\ \kappa.$$

Note that $\langle\rangle$ represents missing components, so that

$$\textit{Append 'val'}: \varepsilon\ \textit{to}\ \kappa \quad \text{and} \quad \textit{Append 'env'}: \rho\ \textit{to}\ \theta$$

both produce complete states. This (very straightforward) machinery is expounded further in [2]. Similarly, values denoted by ρ are data structures, with operations defined on them as follows:

- *Has* (ρ, l) tests whether ρ has a component with selector l,
- *Select* (ρ, l) gives the component of ρ with selector l,
- *Append* $l: \varepsilon$ *to* ρ gives a new structure containing ε with selector l (replacing any component with selector l in ρ), and all other components are as in ρ.

We shall not bother to give the axioms for these operations: they are given in [2].

It remains to give the definitions of *Terminal* and *Step*. In these definitions, which are given in Appendix 2, an expression such as $\sigma = \ulcorner \textbf{eval}\ E\ \textbf{in}\ \rho;\ \kappa \urcorner$ tests whether σ is of the specified syntactic form, and also introduces names for the various components, which may be used in any arm of a conditional expression invoked by satisfying the test.

In (A2.8) the function *Rep* maps basic constants in **Bas** to forms $\ulcorner \textbf{boolean}\ \beta \urcorner$, $\ulcorner \textbf{integer}\ \nu \urcorner$ etc.: we omit the full definition of *Rep*. Likewise, we omit further details about *Check*1 and *Check*2 ((A2.9) and (A2.10)) which check the validity of the operands for monadic and dyadic operators, and about *Oper*1 and *Oper*2, which actually perform the operations.

4. Continuation semantics for PL

In this section we introduce the intermediate semantics for PL, which is a denotational semantics using continuations. In this scheme the valuations for expressions and commands are given an extra parameter, the *continuation*, which embodies the dynamic effect of the rest of the program: so it maps the result of the construct (the value of an expression, or the environment left by a command) into an element of a domain of *final answers*, $\boldsymbol{A}$. As it happens, $\boldsymbol{A}$ is here the same as $\boldsymbol{E}$, but this is not necessary. The domain of function values is now also different, in that such a value is not given a continuation until it is applied (cf. a return link in an implementation). The semantics is therefore as given in Appendix 3. Note the convention that braces $\{\}$ surround the continuation argument of a valuation function. Note, too, that we now no longer require the extra improper elements in $\boldsymbol{U}$, needed in Section 2. This is because improper commands now give rise at once to improper values in $\boldsymbol{A}$, and their effects do not have to propagate by means of the environment produced.

5. Congruence relations between the two denotational definitions

We next define the predicates which a pair of values, one from each of the two denotational definitions, must satisfy if they are to be regarded as congruent. From now on we shall be making extensive use of a notational convention due to Robert Milne, and known as the *diacritical convention*. When relating values belonging to the two definitions, we use acute accents to decorate variables denoting values from one of the definitions, and grave accents for the other. Thus $\acute{e}$ might denote an expressed value in the non-continuation semantics, and $\grave{e}$ an expressed value in the continuation version. We often wish to consider a pair of values such as $\langle \acute{e}, \grave{e} \rangle$, and it is convenient to abbreviate this further; so we write $\hat{e}$ to stand for $\langle \acute{e}, \grave{e} \rangle$. At each stage we use acute accents for the version of semantics we consider more normative; so at present the semantics of Section 2 will have acutes, and that of Section 4 graves.

We first consider values in the two $\boldsymbol{E}$ domains. The function elements of these two domains have different functionality, so for congruence we cannot simply demand equality. We spell our requirements out as follows:

$$\begin{aligned}
&e\hat{\varepsilon} \Leftrightarrow (\acute{\varepsilon} = \bot \vee \grave{\varepsilon} = \bot) \to (\acute{\varepsilon} = \bot \wedge \grave{\varepsilon} = \bot),\\
&\quad (\acute{\varepsilon} = ? \vee \grave{\varepsilon} = ?) \to (\acute{\varepsilon} = ? \wedge \grave{\varepsilon} = ?),\\
&\quad (\acute{\varepsilon} = \top \vee \grave{\varepsilon} = \top) \to (\acute{\varepsilon} = \top \wedge \grave{\varepsilon} = \top),\\
&\quad (\acute{\varepsilon} \in \boldsymbol{B}) \to (\grave{\varepsilon} \in \boldsymbol{B} \wedge (\acute{\varepsilon} \mid \boldsymbol{B}) = (\grave{\varepsilon} \mid \boldsymbol{B})),\\
&\quad (\acute{\varepsilon} \in \boldsymbol{F}) \to (\grave{\varepsilon} \in \grave{\boldsymbol{F}} \wedge f\langle (\acute{\varepsilon} \mid \acute{\boldsymbol{F}}), (\grave{\varepsilon} \mid \grave{\boldsymbol{F}}) \rangle),\\
&\quad \mathit{false}
\end{aligned} \tag{5.1}$$

Note here that if $x \in \boldsymbol{D}_1 + \cdots$, then $x \in \boldsymbol{D}_1$ is *true* if x is in the subdomain $\boldsymbol{D}_1$, *false* if x is in some other subdomain, and $\bot$, ? or $\top$ if x is $\bot$, ? or $\top$.

f, the congruence predicate for functions, is satisfied if congruent results are obtained for congruent arguments. So

$$f\hat{\phi} \Leftrightarrow \bigwedge\{s\langle \acute{\phi}\acute{e}, \grave{\phi}\grave{e} \rangle \mid e\hat{e}\}. \tag{5.2}$$

The predicate s used in 5.2 acts on pairs in which the second element requires a continuation; so it also applies to pairs such as $\langle \acute{\mathscr{E}}[\![E]\!]\acute{\rho}, \grave{\mathscr{E}}[\![E]\!]\grave{\rho} \rangle$. It is defined as follows.

$$\begin{aligned}
&s\hat{\psi} \Leftrightarrow (\acute{\psi} = \bot \vee \grave{\psi} = \bot) \to (\acute{\psi} = \bot \wedge \grave{\psi} = \bot),\\
&\quad (\acute{\psi} = \top \vee \grave{\psi} = \top) \to (\acute{\psi} = \top \wedge \grave{\psi} = \top),\\
&\quad (\acute{\psi} = ? \vee \grave{\psi} = ?) \to (\acute{\psi} = ? \wedge \grave{\psi} = ?),\\
&\quad (\exists \grave{e} : \mathit{Proper}\ \grave{e} \wedge \grave{\psi} = \lambda \kappa .\ \kappa\grave{e} \wedge e\langle \acute{\psi}, \grave{e} \rangle)
\end{aligned} \tag{5.3}$$

A pair of environments is congruent if each identifier denotes congruent values. So

$$u\hat{\rho} \Leftrightarrow \bigwedge\{i\langle \acute{\rho}[\![I]\!], \grave{\rho}[\![I]\!] \rangle \mid I \in \mathbf{Ide}\}. \tag{5.4}$$

We need, finally, a relation for pairs of the form $\langle \mathscr{E}[\![\Gamma]\!]\rho, \grave{\mathscr{E}}[\![\Gamma]\!]\grave{\rho}\rangle$.
If $\hat{\gamma}$ is such a pair, then $\acute{\gamma} \in \boldsymbol{U}$ and $\grave{\gamma} \in [\boldsymbol{G} \to \boldsymbol{A}]$, where $\boldsymbol{G} = [\boldsymbol{U} \to \boldsymbol{A}]$. We define

$$\begin{aligned} c\hat{\gamma} \Leftrightarrow\ & (\acute{\gamma} \equiv \bot \vee \grave{\gamma} \equiv \bot) \to (\acute{\gamma} \equiv \bot \wedge \grave{\gamma} \equiv \bot), \\ & (\acute{\gamma} \equiv \top \vee \grave{\gamma} \equiv \top) \to (\acute{\gamma} \equiv \top \wedge \grave{\gamma} \equiv \top), \\ & (\acute{\gamma} \equiv ? \vee \grave{\gamma} \equiv ?) \to (\acute{\gamma} \equiv ? \wedge \grave{\gamma} \equiv ?), \\ & \exists \grave{\rho}: \grave{\gamma} \equiv \lambda\theta\,.\,\theta\grave{\rho} \wedge u\langle\acute{\gamma}, \grave{\rho}\rangle). \end{aligned} \tag{5.5}$$

These predicates have definitions which involve circularity and they can be rewritten as a fixed point equation. However, the function of which this set of predicates is the fixed point is not monotonic, and so the existence of such a fixed point cannot be inferred by the usual appeal to the result of Tarski [15]. We shall see later that this difficulty is sometimes resolved by regarding the proposed predicates as mapping their arguments into the domain $\{true, untrue\}$, where $true \sqsubseteq untrue$; but that does not help in this case.

In order to be satisfied that these predicates do indeed exist, we consider their arguments, which are drawn from pairs of domains which are themselves circularly defined. The construction of solutions to these recursive domain equations involves sequences of 'approximation' domains, of which the required domains are, in some well-defined sense, the limits. We may define predicates on each of these approximation domains: this involves no circularity, but instead uses the predicates on earlier domains. Then we may define predicates for some element of the limit domain by applying the appropriate predicates to the element's images in all the approximation domains, and taking the conjunction of the results. So, for example, if $\hat{\varepsilon} \in [\acute{\boldsymbol{E}} \times \grave{\boldsymbol{E}}]$ we define

$$e\hat{\varepsilon} \Leftrightarrow \bigwedge_{n=0}^{\infty} \{e_n\hat{\varepsilon}_n\}, \tag{5.6}$$

where $\hat{\varepsilon}_n$ is the image of $\hat{\varepsilon}$ in the approximation domain $[\acute{\boldsymbol{E}}_n \times \grave{\boldsymbol{E}}_n]$. It remains to be proved that the predicates so defined satisfy the required equations: this is true in the present case.

We shall consider this technique in slightly more detail later, in the second phase of our proof.

6. Congruence of the two definitions

Definition 6.1. A function f is *strict* if $f\bot \equiv \bot$, $f? \equiv ?$ and $f\top \equiv \top$. (We adopt this definition for brevity's sake; some workers prefer to call such a property 'double strictness', and say that f is strict simply if $f\bot \equiv \bot$.)

Lemma 6.2. *For the semantics of Appendix* 1, $\mathscr{E}[\![E]\!]$ *and* $\mathscr{C}[\![\Gamma]\!]$ *are strict in* ρ *for all* $E \in$ **Exp**, $\Gamma \in$ **Cmd**.

Proof. This is our first example of *structural induction*: we prove the result for all expressions and commands, assuming that it holds for any subexpressions and subcommands. The proof is by cases on the possible syntactic forms of the constructs, using the clauses of Appendix 1. All the cases are straightforward: for clause (A1.17) we use the fact that $fix\,F = F\,(fix\,F)$.

Lemma 6.3. $Proper\rho \wedge u\hat{\rho} \wedge e\hat{e} \Rightarrow Proper(\rho[\acute{e}/I]) \wedge u\langle\rho[\acute{e}/I], \rho[\grave{e}/I]\rangle$.

Proof. Recalling our remarks in Section 2 about the nature of U, we see that $\rho[\acute{e}/I]$ is proper if ρ is. For $\langle\rho[\acute{e}/I], \rho[\grave{e}/I]\rangle$ by (5.4) we must show

$$\wedge\{e\langle(\rho[\acute{e}/I])[\![I_1]\!], (\rho[\grave{e}/I])[\![I_1]\!]\rangle | I \in \mathbf{Ide}\}.$$

Since ρ is proper, if $I = I_1$ this is $e\hat{e}$, which is given; if $I \neq I_1$ it is

$$\wedge\{e\langle\rho[\![I_1]\!], \rho[\![I_1]\!]\rangle | I_1 \neq I\},$$

which follows from $u\hat{\rho}$.

Our main aim in this section is to prove the congruence of the two values specified for a PL expression (or command) when evaluated in congruent environments. The result is proved (and holds) only for proper environments in the non-continuation semantics. By making this restriction we do not exclude programs for which improper environments arise in the course of the calculation; but we do insist that the program *as a whole* is evaluated in a proper environment (such as $\lambda I.?$), which is not an unreasonable constraint. Our result is then:

Theorem 6.4.

$$\wedge\{s\langle\mathscr{E}[\![E]\!]\rho, \grave{\mathscr{E}}[\![E]\!]\grave{\rho}\rangle | Proper\rho \wedge u\hat{\rho}\} \quad \textit{for all } E \in \mathbf{Exp}$$

and

$$\wedge\{c\langle\mathscr{C}[\![\Gamma]\!]\rho, \grave{\mathscr{C}}[\![\Gamma]\!]\grave{\rho}\rangle | Proper\rho \wedge u\hat{\rho}\} \quad \textit{for all } \Gamma \in \mathbf{Com}.$$

Proof. The proof is by structural induction, and is again by cases of the possible syntactic forms of expressions and commands.

Corollary 6.5

$$\wedge\{e(\mathscr{E}[\![E]\!](\lambda I\,.\,?), \grave{\mathscr{E}}[\![E]\!](\lambda I\,.\,?)\{\lambda\varepsilon.\,\varepsilon\})\}.$$

(We check from (5.1) and (5.3) that if $s\hat{\psi}$ then $e\langle\psi, \grave{\psi}\{\lambda\varepsilon\,.\,\varepsilon\}\rangle$, and from (5.4) that $u\langle\lambda I\,.\,?, \lambda I\,.\,?\rangle$.)

7. Phase two – Strategy

We have now validated the continuation semantics of Section 4, and can turn to the second phase of our investigation, in which we prove the congruence of that semantics with the interpreter. From now on we regard the denotational semantics as the more normative, so we shall use acute accents for entities involved in that definition, and grave accents for those arising in the interpretive scheme.

The interpreter is a continuous function on trees, with a fairly conventional fixed point definition: as usual for such definitions, we shall (in Section 8) employ fixed point induction for its analysis. In essence what we shall be doing in this technique is assuming that the interpreter is 'correct' for programs which require, say, n steps of the interpreter for their evaluation, and proving that the interpreter is also 'correct' for programs which require $n+1$ steps. In effect we shall be doing induction on the duration of the execution.

On the other hand, when we analyse the denotational semantics (in Section 9 and 10), we shall use structural induction: we shall prove that the definitions 'agree' for a program assuming that they do so for its syntactic subcomponents. That is to say, our induction will be based on the *size* of a program rather than its duration. More important than this, however, is that the structural induction will be based, like the last one, on a family of predicates defined circularly. As before, in order to verify the existence of these predicates we shall have to perform induction on the sequences of domains arising in the construction of the various domains of the semantic definition. For example, our domain $\boldsymbol{E}$, which (since $\boldsymbol{K} = [\boldsymbol{E} \to \boldsymbol{E}]$) satisfies

$$\boldsymbol{E} = \boldsymbol{B} + [\boldsymbol{E} \to [[\boldsymbol{E} \to \boldsymbol{E}] \to \boldsymbol{E}]],$$

is constructed as the limit of a sequence $\boldsymbol{E}_0, \boldsymbol{E}_1, \ldots, \boldsymbol{E}_n, \ldots$, where $\boldsymbol{E}_{n+1} = \boldsymbol{B} + [\boldsymbol{E}_n \to [[\boldsymbol{E}_n \to \boldsymbol{E}_n] \to \boldsymbol{E}_n]]$ and $\boldsymbol{E}_0 = \boldsymbol{B} + \{\perp_{\boldsymbol{E}}\}$ (with the extra element standing for all the functions). In this sequence the first domain includes only the basic values, the next includes functions on those basic values too, and in general each domain includes the functions on values in its predecessor in the sequence. Thus we have now introduced three possible kinds of induction on programs – based on their size, their duration, and the richness of their value domains – and there is no correlation whatever between them.

One effect of this is that in the fixed point induction all we shall be able to prove at a typical point in the sequence (where the result of a program is undefined if it does not terminate within a particular number of steps), and therefore all we can prove for the limit, is that the interpreter defines a function which is *weaker* (in the sense of the partial ordering of the appropriate lattice) than that defined by the denotational semantics. On the other hand, when we construct the family of predicates for the structural induction, we shall be concerned with a sequence of approximation domains with limited complexity of type structure, and at a typical point in the sequence some functions will not yet be included, and so the predicate will be unable to express any information about them. Under these circumstances, the

only predicates we can construct for the limit domain assert that the denotational semantics defines a *weaker* function than the interpreter. Only when we have separately proved each definition weaker than the other will we be able (in Section 11) to tie both proofs together and show the definitions precisely congruent.

This is in contrast to the situation in Section 5 and 6. There the two definitions being related were much more alike, and the value domains on each side were so similar that it was possible to define a suitable predicate, expressing complete congruence, on each pair of approximation domains involved in their construction. There was no need to be content with proving an inequality, and so no need to work separately from each side. The present extra complications arise in general when one tries to relate an infinitary abstract object, such as the function associated with some PL procedure by the denotational definition, with its finitary representation, such as the tree which represents the procedure in the interpreter.

This strategy was derived by Milne, who used it in his thesis [8] to investigate mode declarations in Algol 68. He has since used it [9], in a proof which is similar in outline to the present exercise but enormously more complicated, to prove the correctness of a translation into what is in effect a machine code of a big high level language. Gordon [3] has used somewhat different but related techniques for proving the correctness of a LISP interpreter.

8. Analysis of the interpreter by fixed point induction

The denotational semantics is in terms of abstract objects, while the interpreter manipulates representations. So, in order to compare values drawn from each of these schemes, we first define a family of *derepresentation* functions, which map representations into the values they represent.

Expressible Values

$$
\begin{aligned}
E \equiv \lambda \grave{\varepsilon}\ .\ & (\grave{\varepsilon} = \ulcorner \textbf{integer}\ \nu \urcorner) \vee \\
& (\grave{\varepsilon} = \ulcorner \textbf{boolean}\ \beta \urcorner) \vee \\
& (\grave{\varepsilon} = \ulcorner \textbf{string}\ \sigma \urcorner) \rightarrow B(\grave{\varepsilon})\ \textbf{in}\ \acute{E}, \\
& (\grave{\varepsilon} = \ulcorner \textbf{function}\ (I) : E\ \textbf{in}\ \grave{\rho} \urcorner) \rightarrow \lambda\alpha.\mathscr{E}[\![E]\!]((U(\grave{\rho}))[\alpha/I])\ \textbf{in}\ \acute{E}, \\
& (\grave{\varepsilon} = \ulcorner \textbf{recfun}\ I_1(I_2) : E\ \textbf{in}\ \grave{\rho} \urcorner) \rightarrow \\
& \qquad (fix\{\lambda\phi\lambda\alpha\ .\ \mathscr{E}[\![E]\!]((U(\grave{\rho}))[\phi\ \textbf{in}\ E/I_1][\alpha/I_2])\}\ \textbf{in}\ \acute{E}, \qquad (8.1) \\
& ?
\end{aligned}
$$

(Note that we are here using a convention about local names similar to that introduced in Section 3.) This definition treats the representations of values in $\acute{E}$.

Basic Values (8.2)

Since the basic values and their representations are both elements of countable flat lattices we may take for granted the existence of a function B and we shall not bother to define it further: it is the inverse of the function *Rep* introduced in (A2.8).

Environments

Next we have the derepresentation function for environments which are represented by structures.

$$U \equiv \lambda\dot{\rho}\lambda I \,.\, Has(\dot{\rho}, I) \to E(Select(\dot{\rho}, I)), ?. \tag{8.3}$$

Continuations

The next two functions concern continuations.

$$\begin{aligned} K = \lambda\dot{\kappa} \,.\, \dot{\kappa} = \ulcorner \textbf{done}\ \langle\rangle \urcorner \to \lambda\varepsilon \,.\, \varepsilon, \\ \dot{\kappa} = \ulcorner \textbf{assign}\ \langle\rangle\ \textbf{to}\ I\ \textbf{in}\ \dot{\rho}; \dot{\theta} \urcorner \to \lambda\varepsilon \,.\, T(\dot{\theta})(U(\dot{\rho})[\varepsilon/I]), \\ ?, \end{aligned} \tag{8.4}$$

$$\begin{aligned} T \equiv \lambda\dot{\theta} \,.\, \dot{\theta} = \ulcorner \textbf{perform}\ \Gamma\ \textbf{in}\ \langle\rangle; \dot{\theta}_1 \urcorner \to \lambda\dot{\rho} \,.\, \mathscr{E}[\![\Gamma]\!]\dot{\rho}\{T(\dot{\theta}_1)\}, \\ \dot{\theta} = \ulcorner \textbf{evaluate}\ E\ \textbf{in}\ \langle\rangle; \dot{\kappa} \urcorner \to \lambda\dot{\rho} \,.\, \mathscr{E}[\![E]\!]\dot{\rho}\{K(\dot{\kappa})\}, \\ ?. \end{aligned} \tag{8.5}$$

Answers

Finally we give a function for possible answers from the interpreter.

$$\begin{aligned} A = \lambda\sigma \,.\, \sigma = \ulcorner \textbf{error} \urcorner \to ?, \\ \sigma = \ulcorner \textbf{done}\ \dot{\varepsilon} \urcorner \to E(\dot{\varepsilon}), \\ ?. \end{aligned} \tag{8.6}$$

It may be verified that the only results possible from the interpreter are $\perp$, $\ulcorner$**error**$\urcorner$ or $\ulcorner$**done** $\dot{\varepsilon}\urcorner$ for some $\dot{\varepsilon}$. (The proof is similar to that for Hoare's [4] **while**-loop axiom, given in [5]: we verify that either $\sigma \equiv \perp$ or $Terminal(\sigma)$ is *true* for the final state σ.) So since by (8.6) $A(\perp) \equiv \perp$, this definition introduces no surprises.

We are assuming that identifiers $I \in$ **Ide** and their representations are identical: if required we could easily add a derepresentation function I for them.

Note that all these derepresentation functions are continuous, so that although they are recursively defined no special pains need be taken to show their existence. Their inverses, however, mapping from abstract objects to their representations, are not monotonic, or even (in general) single valued.

Lemma 8.7. $U(Append\ I : \dot{\varepsilon}\ to\ \dot{\rho}) \equiv (U\dot{\rho})[E\dot{\varepsilon}/I]$.

Proof. We check that each side gives the same result when applied to an identifier I_1, for all $I_1 \in$ **Ide**.

The main theorem of this section is the following:

Theorem 8.8. *For all states of the appropriate form, with proper components,*

(a) $A(Interpret(\ulcorner \textbf{done } \grave{\varepsilon} \urcorner)) \sqsubseteq E\grave{\varepsilon}$,

(b) $A(Intepret(\ulcorner \textbf{error} \urcorner)) \sqsubseteq ?$,

(c) $A(Interpret(\ulcorner \textbf{eval } E \textbf{ in } \grave{\rho}; \grave{\kappa} \urcorner)) \sqsubseteq \mathscr{E}[\![E]\!](U\grave{\rho})\{K\grave{\kappa}\}$,

(d) $A(Interpret(\ulcorner \textbf{perform } \Gamma \textbf{ in } \grave{\rho}; \grave{\theta} \urcorner)) \sqsubseteq \mathscr{C}[\![\Gamma]\!](U\grave{\rho})\{T\grave{\theta}\}$,

(e) $A(Interpret(\ulcorner \textbf{assign } \grave{\varepsilon} \textbf{ to } I \textbf{ in } \grave{\rho}; \grave{\theta} \urcorner)) \sqsubseteq (T\grave{\theta})((U\grave{\rho})[E\grave{\varepsilon}/I])$.

Proof. We use fixed point induction, remembering that

$$Interpret = fix(H),$$

where

$$H\phi = \lambda\sigma \,.\, Terminal\sigma \to \sigma, \phi(Step\sigma).$$

We must therefore show:

(1) The results (a) to (e) hold when *Interpret* is replaced by $\bot$. This is immediately obvious.

(2) Results (a) to (e) hold when *Interpret* is replaced by $H\phi$, assuming that they hold when *Interpret* is ϕ.

For results (a) and (b) the state σ is such that $Terminal(\sigma)$ holds; so $(H\phi)\sigma = \sigma$ for all ϕ, and therefore by (8.6) the required inequalities hold. For results (c) and (d) we consider by cases possible values of E and Γ respectively. For case (e), noting that the left-hand side becomes

$$A(\phi(Append\ 'env' : (Append\ I : \grave{\alpha}\ to\ \grave{\rho})\ to\ \grave{\theta})),$$

we consider both possible values of $\grave{\theta}$, namely

$$\ulcorner \textbf{perform } \Gamma_1 \textbf{ in } \langle\rangle; \grave{\theta}_1 \urcorner$$

and

$$\ulcorner \textbf{eval } E_1 \langle\rangle; \grave{\kappa}_1 \urcorner.$$

In both cases the result follows by Lemma 8.7 and the inductive hypothesis.

Corollary 8.9. *For all* $E \in$ **Exp**,

$$A(Interpret(\ulcorner \textbf{eval } E \textbf{ in nil}; \ulcorner \textbf{done } \langle\rangle \urcorner\urcorner)) \sqsubseteq \mathscr{E}[\![E]\!](\lambda I \,.\, ?)\{\lambda\varepsilon \,.\, \varepsilon\}.$$

9. Predicates for the structural induction

Having completed the first arm of the congruence proof we now embark on the second, which involves an analysis of the denotational definition by structural induction. For this purpose we define a family of predicates, similar to those used in Section 5.

First we relate values in $\acute{E}$ with their counterparts in the interpretive system:

$$\begin{aligned} &e\hat{e} \Leftrightarrow IsBasic\grave{e} \rightarrow \acute{e} \in \acute{B} \wedge (\acute{e}|\acute{B}) = B\grave{e}, \\ &\quad IsFunction\acute{e} \rightarrow \acute{e} \in \acute{F} \wedge f\hat{e}, \\ &\quad false \end{aligned} \tag{9.1}$$

where

$$IsBasic\grave{e} \Leftrightarrow (\grave{e} = \ulcorner \textbf{integer } \nu \urcorner) \vee (\grave{e} = \ulcorner \textbf{boolean } \beta \urcorner) \vee (\grave{e} = \ulcorner \textbf{string } \sigma \urcorner) \tag{9.2}$$

and

$$IsFunction\grave{e} \Leftrightarrow (\grave{e} = \ulcorner \textbf{function } (I) : E \textbf{ in } \rho \urcorner) \vee (\grave{e} = \ulcorner \textbf{recfun } I_1(I_2) : E \textbf{ in } \rho \urcorner). \tag{9.3}$$

Next we relate answers: a compares a value $\acute{e}$ arising from an expression in denotational semantics with a final state $\grave{\sigma}$ arising from the interpreter:

$$\begin{aligned} a\langle \acute{e}, \grave{\sigma} \rangle \Leftrightarrow &\grave{\sigma} = \bot \rightarrow \acute{e} = \bot, \\ &\grave{\sigma} = \ulcorner \textbf{error} \urcorner \rightarrow \acute{e} \sqsubseteq ?, \\ &\grave{\sigma} = \ulcorner \textbf{done } \grave{\varepsilon} \urcorner \rightarrow (IsBasic\grave{\varepsilon} \rightarrow \acute{e} \sqsubseteq B\grave{\varepsilon}, \\ &\qquad\qquad IsFunction\grave{\varepsilon} \rightarrow f\hat{e}, \\ &\qquad\qquad false), \\ &false. \end{aligned} \tag{9.4}$$

Note the inequalities here, which we shall be discussing a little later.

When comparing functions, all we can demand is that congruent answers are obtained whenever they are applied to congruent values. More precisely, if a pair of expression values $\hat{e}$ correspond to functions, they are congruent if the result of applying $\acute{e}$ to any argument $\acute{\alpha}$ with any continuation $\acute{\kappa}$ agrees with the result of interpreting an application expression $\ulcorner I_1(I_2) \urcorner$ with a continuation $\grave{\kappa}$ congruent with $\acute{\kappa}$, in an environment in which I_1 denotes $\grave{e}$ and I_2 denotes a value $\grave{\alpha}$ congruent with $\acute{\alpha}$. So we have:

$$\begin{aligned} f\hat{e} \Leftrightarrow \bigwedge \{ &\alpha\langle (\acute{e}|\acute{F})\acute{\alpha}\acute{\kappa}, Interpret(\ulcorner \textbf{eval} \ulcorner I_1(I_2) \urcorner \textbf{ in } \grave{\rho}\,;\, \grave{\kappa} \urcorner) \rangle | \\ &\qquad k\hat{\kappa} \wedge e\hat{\alpha} \wedge contains(\grave{\rho}, \grave{e}, \grave{\alpha}) \}, \end{aligned} \tag{9.5}$$

where

$$contains(\dot\rho, \dot\varepsilon, \dot\alpha) \Leftrightarrow Has(\dot\rho, I_1) \wedge Select(\dot\rho, I_1) \equiv \dot\varepsilon$$
$$\wedge\, Has(\dot\rho, I_2) \wedge Select(\dot\rho, I_2) \equiv \dot\alpha. \tag{9.6}$$

The next two definitions are about continuations.

$$k\hat\kappa \Leftrightarrow \bigwedge\{a\langle\acute\kappa\acute\varepsilon, Interpret(Append'val':\dot\varepsilon\ to\ \dot\kappa)\rangle \mid e\hat\varepsilon\}, \tag{9.7}$$

$$t\hat\theta \Leftrightarrow \bigwedge\{a\langle\acute\theta\acute\rho, Interpret(Append'env':\dot\rho\ to\ \dot\theta)\rangle \mid u\hat\rho\}. \tag{9.8}$$

Finally, a pair of environments is congruent if the pair of values denoted by any identifier is congruent.

$$u\hat\rho \Leftrightarrow \bigwedge\{Has(\dot\rho, I) \rightarrow e\langle\dot\rho[\![I]\!], Select(\dot\rho, I)\rangle, \dot\rho[\![I]\!] \equiv ? \mid I \in \mathbf{Ide}\}. \tag{9.9}$$

9.2. *The existence of these predicates*

The reader will have noticed that if only the two inequalities in (9.4) were equalities instead, we would have defined conditions for precise congruence between the two semantic definitions; we would have avoided the need to prove the opposite inequality, and Section 8 and 11 could have been omitted. We claim, however, that to eliminate these inequalities would invalidate our technique for constructing the required predicates. Note that we are not thereby claiming that no such predicates exist (that is still an open question), but merely that we do not know how to demonstrate their existence, and therefore that we have no right to use them in our argument.

The reason for this difficulty has already been outlined above (Section 7), namely that the existence proof is based on an induction on the series of approximation domains $\boldsymbol{E}_n$ whose limit is the reflexive domain $\boldsymbol{E}$. For each n, some of the functions in $\boldsymbol{E}$ have to be represented by approximations in $\boldsymbol{E}_n$, so the series of predicates e_n cannot lead in the limit to an e asserting precise congruence for elements of $\boldsymbol{E}$. Formally, it is the basis of the induction which fails. In $\boldsymbol{E}_0$ there is but one element, $\perp_{\boldsymbol{E}}$, doing duty for all the functions; so e_0 would have to assert that the interpreter's behaviour for *every* function is congruent to $\perp_{\boldsymbol{E}}$, which is plainly ridiculous. To achieve the weaker predicates we have actually specified, however, all we require is that e_0 asserts that $\perp_{\boldsymbol{E}}$ *approximates* every function modelled by the interpreter, which is plainly obvious.

The approximate predicates themselves are defined in Section 11 below; the details of the proof for a slightly simpler case may be seen in [13].

Notice that our 'approximate' predicates have been comparing elements from the 'approximate' denotational domains with the 'perfect' values on the interpreter side. If we wished we could use approximations to these other values too. But this would be no help, since the approximate values manipulated by the interpreter are trees of limited depth, and the approximate values on the other side have limited complexity of functionality. The lack of correlation between these concepts means that, though matters would be much more confusing, they would fail in a similar way.

10. Analysis of the denotational semantics by structural induction

Lemma 10.1. *If* $u\hat{\rho}$ *and* $e\hat{\varepsilon}$, *then* $u\langle\dot{\rho}[\dot{\varepsilon}/l],$ *Append* l : $\dot{e}$ *to* $\dot{\rho}\rangle$.

Proof. Check that the requirements of (9.9) are satisfied for all $l' \in$ Ide, because $e\hat{\varepsilon}$ (for $l' = l$) and $u\hat{\rho}$ (for $l' \neq l$).

The main result of this section, which leads to the opposite inequality to Corollary 8.9, is as follows.

Theorem 10.2.

$$\bigwedge\{a\langle\mathscr{E}[\![E]\!]\dot{\rho}\dot{\kappa}, \mathit{Interpret}\ulcorner\textbf{eval } E \textbf{ in } \dot{\rho}; \dot{\kappa}\urcorner\rangle \mid u\hat{\rho} \wedge k\hat{\kappa}\}$$

for all $E \in$ **Exp**, *and*

$$\bigwedge\{a\langle\mathscr{C}[\![\Gamma]\!]\dot{\rho}\dot{\theta}, \mathit{Interpret}\ulcorner\textbf{perform } \Gamma \textbf{ in } \dot{\rho}; \dot{\theta}\urcorner\rangle \mid u\hat{\rho} \wedge t\hat{\theta}\}$$

for all $\Gamma \in$ **Com**.

Proof. As usual, the proof is a structural induction, by analysis of the various cases of possible forms for E and Γ.

11. The relationship between these results

Strategy

To combine Theorem 10.2 with Corollary 6.5 and 8.9, we wish to show that it implies the following particular result:

$$a\langle\mathscr{E}[\![E]\!](\lambda l\,.\,?)\{\lambda\varepsilon\,.\,\varepsilon\}, \mathit{Interpret}(\ulcorner\textbf{eval } E \textbf{ in nil};\ulcorner\textbf{done}\langle\,\rangle\urcorner\urcorner)\rangle.$$

To do this we must show that

$$u\langle(\lambda l\,.\,?), \textbf{nil}\rangle$$

which is immediately obvious from (9.9), and also that

$$k\langle\lambda\varepsilon\,.\,\varepsilon, \ulcorner\textbf{done}\,\langle\,\rangle\urcorner\rangle$$

which is more difficult. Indeed, we demonstrate this only as a corollary of the following much more general result.

Theorem 11.1. (a) $\bigwedge\{e\langle E\dot{\varepsilon}, \dot{\varepsilon}\rangle \mid \mathit{Proper}\dot{\varepsilon}\}$,
(b) $\bigwedge\{u\langle U\hat{\rho}, \dot{\rho}\rangle \mid \mathit{Proper}\dot{\rho}\}$,
(c) $\bigwedge\{t\langle T\hat{\theta}, \dot{\theta}\rangle \mid \mathit{Proper}\dot{\theta}\}$,
(d) $\bigwedge\{k\langle K\dot{\kappa}, \dot{\kappa}\rangle \mid \mathit{Proper}\dot{\kappa}\}$.

Proof. The proof is by induction. The proof of Theorem 3.8 was essentially by induction on the number of execution steps performed by the interpreter, and the proof of Theorem 10.2 by structural induction on the structure of the program. For the present proof we use the third of the possible schemes for induction discussed in Section 7: we consider the sequence of approximations to the predicates e, u, t and k. So to prove (a), for example, we show that for all n,

$$\bigwedge\{e_n\langle E\grave{\varepsilon}, \grave{\varepsilon}\rangle \mid Proper\grave{e}\}$$

for then, since $e\hat{e} \equiv \bigwedge_n e_n\hat{e}$, the required result follows.

We first give the definitions of these approximations of e, u, t, k and a, and the domains on which they operate.

$$\boldsymbol{E}_n = \boldsymbol{B} + \boldsymbol{F}_n, \tag{11.2}$$

$$\boldsymbol{F}_0 = \{\bot\}, \qquad \boldsymbol{F}_{n+1} = [\boldsymbol{E}_n \to [\boldsymbol{K}_n \to \boldsymbol{E}_n]], \tag{11.3}$$

$$\boldsymbol{K}_n = [\boldsymbol{E}_n \to \boldsymbol{E}_n], \tag{11.4}$$

$$\begin{aligned} a_0\langle \acute{\varepsilon}, \grave{\sigma}\rangle \Leftrightarrow\ & \grave{\sigma} \equiv \bot \to \acute{\varepsilon}_0 \equiv \bot, \\ & \grave{\sigma} \equiv \ulcorner \textbf{error} \urcorner \to \acute{\varepsilon}_0 \sqsubseteq ?, \\ & \grave{\sigma} \equiv \ulcorner \textbf{done}\ \grave{e} \urcorner \to (IsBasic\grave{e} \to (\acute{\varepsilon}_0 \mid \boldsymbol{B}) \sqsubseteq \mathcal{B}\grave{e}, \\ & \qquad IsFunction\grave{e} \to f_0\hat{\varepsilon}, \\ & \qquad false), \\ & false, \end{aligned} \tag{11.5}$$

$$\begin{aligned} a_n\langle \acute{\varepsilon}, \grave{\sigma}\rangle \Leftrightarrow\ & \grave{\sigma} \equiv \bot \to \acute{\varepsilon}_n \equiv \bot, \\ & \grave{\sigma} \equiv \ulcorner \textbf{error} \urcorner \to \acute{\varepsilon}_n \sqsubseteq ?, \\ & \grave{\sigma} \equiv \ulcorner \textbf{done}\ \grave{e} \urcorner \to (IsBasic\grave{e} \to (\acute{\varepsilon}_n \mid \boldsymbol{B}) \sqsubseteq B\grave{e}, \\ & \qquad IsFunction\grave{e} \to f_n\hat{\varepsilon}, \\ & \qquad false), \\ & false, \end{aligned} \tag{11.6}$$

$$\begin{aligned} e_n\hat{e} \Leftrightarrow\ & IsBasic\grave{e} \to \acute{\varepsilon} \in \boldsymbol{B} \wedge (\acute{\varepsilon} \mid \boldsymbol{B}) \equiv B\grave{e}, \\ & IsFunction\grave{e} \to \acute{\varepsilon}_n \in \boldsymbol{F}_n \wedge f_n\hat{\varepsilon}, \\ & false, \end{aligned} \tag{11.7}$$

$$f_0\hat{\varepsilon} \Leftrightarrow true,$$

$$\begin{aligned} f_{n+1}\hat{\varepsilon} \Leftrightarrow \bigwedge\{ & a_n\langle (\acute{\varepsilon} \mid \boldsymbol{F})\grave{\alpha}\acute{\kappa},\ Interpret(\ulcorner \textbf{eval}\ \ulcorner I_1(I_2) \urcorner \textbf{in}\ \grave{e}\ ,\ \grave{\kappa} \urcorner)\rangle \mid \\ & k_n\hat{\kappa} \wedge e_n\hat{\alpha} \wedge contains(\grave{\rho}, \grave{e}, \grave{\alpha})\}, \end{aligned} \tag{11.8}$$

$$k_n\hat{\kappa} \Leftrightarrow \bigwedge\{a_n\langle \acute{\kappa}\acute{\varepsilon},\ \mathit{Interpret}(\mathit{Append'val'}:\grave{\varepsilon}\ \mathit{to}\ \grave{\kappa})\rangle \mid e_n\hat{\varepsilon}\}, \tag{11.9}$$

$$t_n\hat{\theta} \Leftrightarrow \bigwedge\{a_n\langle \acute{\alpha}\acute{\rho},\ \mathit{Interpret}(\mathit{Append'env}':\grave{\rho}\ \mathit{to}\ \grave{\theta})\rangle \mid u_n\hat{\rho}\}, \tag{11.10}$$

$$u_n\hat{\rho} \Leftrightarrow \bigwedge\{(\mathit{Has}\ (\grave{\alpha}, l) \rightarrow e_n\langle \acute{\rho}[\![l]\!], \mathit{Select}(\grave{\rho}, l)\rangle,\ \acute{\rho}[\![l]\!] = ?) \mid l \in \mathbf{Ide}\}. \tag{11.11}$$

Remark 11.12. The results which we have proved for the limit predicates hold also in versions involving their approximations. Thus, for example, Theorem 10.2 implies that for all $E \in \mathbf{Exp}$

$$\bigwedge\{a_n\langle \mathscr{E}[\![E]\!]\acute{\rho}\acute{\kappa}, \mathit{Interpret}(\ulcorner \mathbf{eval}\ E\ \mathbf{in}\ \grave{\rho};\ \grave{\kappa}\urcorner)\rangle \mid u_n\hat{\rho} \wedge k_n\hat{\kappa}\}.$$

To see that this is so, we note first that if $k_n\hat{\kappa}$, for example, then also $k_m\langle \acute{\kappa}_{(n)}, \grave{\kappa}\rangle$ for all m, where $\acute{\kappa}_{(n)}$ is the element of $\acute{\mathbf{K}}_n$ corresponding to $\acute{\kappa}$ in $\acute{\mathbf{K}}$, and hence $k\langle \acute{\kappa}_{(n)}, \grave{\kappa}\rangle$; that, by definition of a, if $a\hat{\beta}$, then also $a_n\hat{\beta}$ for all n; and finally that

$$a_n\langle \mathscr{E}[\![E]\!]\acute{\rho}_{(n)}\acute{\kappa}_{(n)}, \grave{\sigma}\rangle = a_n\langle \mathscr{E}[\![E]\!]\acute{\rho}\acute{\kappa}, \grave{\sigma}\rangle.$$

Proof of Theorem 11.1. We prove that for all n

(a) $\bigwedge\{e_n\langle E\grave{\varepsilon}, \grave{\varepsilon}\rangle \mid \mathit{Proper}\grave{\varepsilon}\}$,
(b) $\bigwedge\{u_n\langle U\grave{\rho}, \grave{\rho}\rangle \mid \mathit{Proper}\grave{\rho}\}$,
(c) $\bigwedge\{t_n\langle T\grave{\theta}, \grave{\theta}\rangle \mid \mathit{Proper}\grave{\theta}\}$,
(d) $\bigwedge\{k_n\langle K\grave{\kappa}, \grave{\kappa}\rangle \mid \mathit{Proper}\grave{\kappa}\}$.

The proof is by induction on n. The arguments for the basis and for the inductive step are similar: for the basis in case (a) we use the fact that $f_0\hat{\varepsilon} \Leftrightarrow \mathit{true}$, where in the other argument we use the inductive hypothesis. T and K are defined mutually recursively, so for cases (c) and (d) it is also necessary to use fixed point induction. Otherwise, the proof consists of considering the possible forms of $\grave{\varepsilon}$, $\grave{\theta}$ and $\grave{\kappa}$. The subcase $\grave{\kappa} = \ulcorner \mathbf{done}\ \langle\rangle \urcorner$ is the only one which uses the definition of a. This is only to be expected, as it is only when the continuation $\ulcorner \mathbf{done}\ \langle\rangle \urcorner$ is reached that actual answers are produced by the interpreter; so only here must we check that such answers meet our requirements.

Corollary 11.13.

$$a\langle \mathscr{E}[\![E]\!](\lambda l\ .\ ?)\{\lambda\varepsilon\ .\ \varepsilon\}, \mathit{Interpret}(\ulcorner \mathbf{eval}\ E\ \mathbf{in\ nil};\ \ulcorner \mathbf{done}\ \langle\rangle \urcorner\urcorner)\rangle.$$

Now without further complication we can show the following result.

Theorem 11.14. *For any* $E \in \mathbf{Exp}$, let

$$\alpha_1 = \mathscr{E}[\![E]\!](\lambda l\ .\ ?)\{\lambda\varepsilon\ .\ \varepsilon\},$$

$$\alpha_2 = A(\mathit{Interpret}(\ulcorner \mathbf{eval}\ E\ \mathbf{in\ nil};\ \ulcorner \mathbf{done}\ \langle\rangle \urcorner\urcorner)).$$

Then

$$(\alpha_1 \mathbf{E} F \wedge \alpha_2 \mathbf{E} F) \vee (\alpha_1 \equiv \alpha_2).$$

Proof. The result follows immediately from Corollary 8.9 and 11.13, with their associated definitions (8.6) and (9.4).

The usefulness of Theorem 11.14

For most purposes Theorem 11.14 is a sufficient statement of the congruence between the denotational semantics and the interpreter. Moreover, Corollary 6.5 allows a similar congruence to be established with the original (non-continuation) semantics. These results may readily be extended for situations where the initial environments are non-empty (for example, to accommodate library functions).

For expressions whose values are functions, however, Theorem 11.14 obliges us to accept from the interpreter *any* representation of a function, without worrying about whether it represents the right one. This corresponds with our normal practice: when a computer spews out at us its own representation of a function we rarely subject the binary code to careful mathematical analysis – indeed, in many cases what function is represented depends on the contents of the computer store, so such analysis may be impossible. In the present simpler case, however, functions are represented by closures, and we *do* have a function, E, to tell us what function such a closure represents. So it is reasonable to ask whether it represents the right one.

Two functions are equal if equal results are obtained when each is applied to any possible argument. However, we shall not prove here that any function is *equal* to the correct one, as we shall be confining our attention to argument values which can be represented in a form suitable for the interpreter (that is to say, arguments of the form $E\grave{\varepsilon}$ for some $\grave{\varepsilon}$). Instead of equality, therefore, we have another equivalence relation: two functions are equivalent if they give equivalent results when each is applied to the same argument drawn from the class of acceptable arguments. This notion will be formalised below, where we define the equivalence by means of a quasi-ordering. Equality, of course, implies equivalence; whether the reverse is true depends on the structure of the domains involved.

Definition 11.15. For $\alpha_1, \alpha_2 \in \acute{E}$,

$$\alpha_1 \lesssim \alpha_2 \Leftrightarrow (\alpha_2 \in \acute{F}) = \mathit{true} \rightarrow \bigwedge\{(\alpha_1|\acute{F})(E\grave{\beta})\{\lambda\varepsilon\,.\,\varepsilon\} \lesssim (\alpha_2|\acute{F})(E\grave{\beta})\{\lambda\varepsilon\,.\,\varepsilon\} \mid \grave{\beta} \in \grave{E}\},$$
$$\alpha_1 \sqsubseteq \alpha_2.$$

This definition, like many previous ones, is a circular definition of an inclusive predicate. A predicate satisfying this definition would be a fixed point of the function ψ where

$$\psi = \lambda\phi\,.\,\lambda\langle\alpha_1,\alpha_2\rangle\,.\,(\alpha_2 \in \acute{F} \wedge \alpha_1 \sqsubseteq \alpha_2) \vee$$
$$(\alpha_2 \in \acute{F} \wedge \bigwedge\{\phi\langle(\alpha_1|\acute{F})(E\grave{\beta})\{\lambda\varepsilon\,.\,\varepsilon\}, (\alpha_2|\acute{F})(E\grave{\beta})\{\lambda\varepsilon\,.\,\varepsilon\}\rangle \mid \grave{\beta} \in \grave{E}\}). \qquad (11.16)$$

Remembering that we regard inclusive predicates as mapping their operands into the domain {*true*, *untrue*} where *true* $\sqsubseteq$ *untrue*, we may easily see that ψ is monotonic; so, since inclusive predicates form a complete lattice, such a fixed point certainly exists. This is in contrast to earlier predicates (such as those of Section 5) which were defined as the fixed points of non-monotonic functions, and which therefore required a more elaborate existence proof. The essential difference is that the earlier definitions used a recursive invocation to qualify the set of permissible arguments to a function application (and hence to *reduce* a universe of quantification), whereas here all $\dot{e}$ are allowed.

Since ψ may also be shown to be continuous we may investigate appropriate properties of its minimal fixed point (*fix* ψ) by fixed point induction, remembering that the minimal element in the lattice of inclusive predicates is $(\lambda\langle\alpha_1, \alpha_2\rangle \,.\, \mathit{true})$.

Examples of this are given by the proofs of the following lemmas.

Lemma 11.17. *For* $\alpha_1, \alpha_2 \in \boldsymbol{E}$, $\alpha_1 \sqsubseteq \alpha_2 \Rightarrow \alpha_1 \preccurlyeq \alpha_2$.

Corollary 11.18.

$$A(\mathit{Interpret}(\ulcorner\textbf{eval } E \textbf{ in nil};\ulcorner\textbf{done}\ \langle\,\rangle\urcorner\urcorner)) \preccurlyeq \mathscr{E}[\![E]\!](\lambda l \,.\, ?)\{\lambda\varepsilon \,.\, \varepsilon\}.$$

Lemma 11.19. $\preccurlyeq$ *is transitive*; *that is to say, for all x, y and z,*

$$x \preccurlyeq y \wedge y \preccurlyeq z \Rightarrow x \preccurlyeq z.$$

Lemma 11.20. $a\langle\acute{\alpha}, \grave{\sigma}\rangle \Rightarrow \acute{\alpha} \preccurlyeq A\grave{\sigma}$.

Proof. We use fixed point induction to prove a more powerful result (actually the conjunction of Lemma 11.17, 11.19 and 11.20). Specifically, we prove $P(\mathit{fix}\psi)$, where

$$\begin{aligned} P \equiv (\lambda\phi \,.\, &(\forall\alpha_1, \alpha_2 : \alpha_1 \sqsubseteq \alpha_2 \Rightarrow \phi\langle\alpha_1, \alpha_2\rangle) \wedge \\ &(\forall\alpha_1, \alpha_2, \alpha_3 : \phi\langle\alpha_1, \alpha_2\rangle \wedge \phi\langle\alpha_2, \alpha_3\rangle \Rightarrow \phi\langle\alpha_1, \alpha_3\rangle) \wedge \\ &(\forall\acute{\alpha}, \grave{\sigma} : a\langle\acute{\alpha}, \grave{\sigma}\rangle \Rightarrow \phi\langle\acute{\alpha}, A\grave{\sigma}\rangle)). \end{aligned}$$

Corollary 11.21.

$$\mathscr{E}[\![E]\!](\lambda l \,.\, ?)\{\lambda\varepsilon \,.\, \varepsilon\} \preccurlyeq A(\mathit{Interpret}(\ulcorner\textbf{eval } E \textbf{ in nil};\ulcorner\textbf{done}\ \langle\,\rangle\urcorner\urcorner)).$$

Definition 11.22. $\alpha_1 \approx \alpha_2 \Leftrightarrow \alpha_1 \preccurlyeq \alpha_2 \wedge \alpha_2 \preccurlyeq \alpha_1$.

Notice that this defines an equivalence relation.

Corollary 11.23.

$$\begin{aligned} \alpha_1 \approx \alpha_2 \Leftrightarrow \alpha_1 \mathbf{E} \acute{F} \rightarrow \alpha_2 \mathbf{E} \acute{F} \wedge \bigwedge\{(\alpha_1|\acute{F})(E\grave{\beta})\{\lambda\varepsilon \,.\, \varepsilon\} \approx (\alpha_2|\acute{F})(E\grave{\beta})\{\lambda\varepsilon \,.\, \varepsilon\} \\ |\,\grave{\beta} \in \grave{E}\}, \end{aligned}$$

$\alpha_1 \equiv \alpha_2$.

Theorem 11.24.

$$\mathscr{E}[\![E]\!](\lambda l\,.\,?)\{\lambda\varepsilon\,.\,\varepsilon\} \approx A(\mathit{Interpret}(\ulcorner\textbf{eval } E \textbf{ in nil}; \ulcorner\textbf{done}\ \langle\rangle\urcorner\urcorner)).$$

Proof. Immediate, from Corollary 11.18 and 11.21.

12. Conclusion

Theorem 11.24 is the extension of Theorem 11.14 that we sought. It says that the denotational semantics of Section 4 and the interpretive semantics are congruent in the following way: unless the answer is a function they give identical values; if the answer is a function they give equivalent values, in the sense that equivalent results are obtained when the values are applied to any representable argument. Corollary 6.5 stated a similar congruence between the two forms of denotational semantics. Again the definitions specify identical results unless the answer is a function; if the answer is a function the two values given cannot be equal (indeed, they are members of different domains), but they are again equivalent, this time in the sense of Definition 11.12). Together, these results imply the congruence of the two original definitions: for any expression, identical values are given if the answer is not a function, and equivalent values if it is.

The proof of this congruence has been long. Some of the length is unavoidable: for example, tedious case-analyses (omitted from the present text) are an essential comparison of the small print of the two definitions. We can expect, however, that increased use of mechanical aids, such as those developed by Milner and others [10], will relieve this situation. Other factors affecting the length of the proof include the need (in Sections 8 and 10) to cover similar ground in two different ways, and also the elaborate existence proofs required for the various predicates (here, too, our exposition omits many of the details). This is an area where workers in the field develop a 'feel' for what is likely to be true: such a feeling is, of course, no substitute for actually carrying out the checks, but we may reasonably hope for some more mathematics to simplify the work.

Appendix 1. Standard semantics for PL

Syntactic domains

$B \in$ **Bas** (Basic constants)

$I \in$ **Ide** (Identifiers)

$O \in$ **Mon** (Monadic operators)

$\Omega \in$ **Dya** (Dyadic operators)

$E \in$ **Exp** (Expressions)

$\Gamma \in$ **Com** (Commands)

Syntax

$$E ::= I \mid E(E) \mid \textbf{proc}(I) : E \mid \textbf{rec } I(I) : E \mid \Gamma \textbf{ res } E \mid$$
$$B \mid OE \mid E \Omega E \mid \textbf{if } E \textbf{ then } E \textbf{ else } E \mid \textbf{let } I = E \textbf{ in } E \mid$$
$$\textbf{iterate } I \textbf{ to } E \textbf{ from } E \textbf{ while } E$$

$$\Gamma ::= I := E \mid \textbf{while } E \textbf{ do } \Gamma \mid$$
$$\Gamma ; \Gamma \mid \textbf{if } E \textbf{ then } \Gamma \textbf{ else } \Gamma \mid ( )$$

A complete program in PL is an expression.

Auxiliary definitions

For $b \in \mathbf{T}$ (the domain of truth values), and $x, y \in \mathbf{D}$,

$$b \to x, y \equiv \begin{cases} x & \text{if } b = \mathit{true} \\ y & \text{if } b = \mathit{false} \\ \bot_D, \top_D, ?_D & \text{if } b = \bot_T, \top_T, ?_T \end{cases} \quad \text{(A1.1)}$$

For $x, y \in \mathbf{D}$ and b in some domain including $\mathbf{T}$,

$$\mathit{cond}\langle x, y\rangle b \equiv (b \mid \mathbf{T}) = \mathit{true} \to x,$$
$$(b \mid \mathbf{T}) = \mathit{false} \to y,$$
$$?_D, \quad \text{(A1.2)}$$

$$\mathit{strict}\, f x \equiv \begin{cases} \bot, \top, ? & \text{if } x = \bot, \top, ? \\ f(x) & \text{otherwise,} \end{cases} \quad \text{(A1.3)}$$

$$\mathit{fix}\, f \equiv \bigsqcup_{n=0}^{\infty} f^n(\bot). \quad \text{(A1.4)}$$

Note that $\mathit{fix}\, f$ is the minimal fixed point of f; so $\mathit{fix}\, f = f(\mathit{fix}\, f)$, and if $a = f(a)$, then $a \sqsubseteq \mathit{fix}\, f$.

Semantic domains

$\beta \in \mathbf{B}$	(Basic values, including *true* and *false*)
$\phi \in \mathbf{F} = [\mathbf{E} \to \mathbf{E}]$	(Function values)
$\varepsilon \in \mathbf{E} = \mathbf{B} + \mathbf{F}$	(Expressed values)
$\rho \in \mathbf{U} = [\mathbf{Ide} \to \mathbf{E}] + \{?\}$	(Environments)
$\theta \in \mathbf{C} = [\mathbf{U} \to \mathbf{U}]$	(Commands)

Semantic Valuations

$$\mathcal{B} : [\mathbf{Bas} \to \mathbf{B}]$$
$$\mathcal{O} : [\mathbf{Mon} \to [\mathbf{E} \to \mathbf{E}]]$$
$$\mathcal{W} : [\mathbf{Dya} \to [[\mathbf{E} \times \mathbf{E}] \to \mathbf{E}]]$$
$$\mathcal{E} : [\mathbf{Exp} \to [\mathbf{U} \to \mathbf{E}]]$$

$$\mathcal{E}[\![I]\!]\rho = \rho[\![I]\!] \quad \text{(A1.5)}$$
$$\mathcal{E}[\![E_0(E_1)]\!]\rho = \mathit{strict}(\mathcal{E}[\![E_0]\!]\rho \mid [\mathbf{E} \to \mathbf{E}])(\mathcal{E}[\![E_1]\!]\rho) \quad \text{(A1.6)}$$
$$\mathcal{E}[\![\textbf{proc}(I) : E]\!]\rho = \mathit{strict}(\lambda\rho \,.\, \lambda\varepsilon \,.\, \mathcal{E}[\![E]\!](\rho[\varepsilon / I])\ \textbf{in}\ \mathbf{E}) \quad \text{(A1.7)}$$

$\mathscr{E}[\![\textbf{rec}\ I_0(I_1):E]\!] \equiv$
$\quad strict(\lambda\rho\ .\ fix[\lambda\phi\ .\ \lambda\varepsilon\ .\ \mathscr{E}[\![E]\!](\rho[\phi\ \textbf{in}\ E/I_0][\varepsilon/I_1])]\ \textbf{in}\ E)$ (A1.8)

$\mathscr{E}[\![\Gamma\ \textbf{res}\ E]\!]\rho \equiv \mathscr{E}[\![E]\!](\mathscr{C}[\![\Gamma]\!]\rho)$ (A1.9)

$\mathscr{E}[\![B]\!] \equiv strict(\lambda\rho\ .\ \mathscr{B}[\![B]\!]\ \textbf{in}\ E)$ (A1.10)

$\mathscr{E}[\![OE]\!]\rho \equiv \mathscr{O}[\![O]\!](\mathscr{E}[\![E]\!]\rho)$ (A1.11)

$\mathscr{E}[\![E_0 \Omega E_1]\!]\rho \equiv \mathscr{W}[\![\Omega]\!]\langle\mathscr{E}[\![E_0]\!]\rho, \mathscr{E}[\![E_1]\!]\rho\rangle$ (A1.12)

$\mathscr{E}[\![\textbf{if}\ E_0\ \textbf{then}\ E_1\ \textbf{else}\ E_2]\!]\rho \equiv$
$\quad cond\langle\mathscr{E}[\![E_1]\!]\rho, \mathscr{E}[\![E_2]\!]\rho\rangle(\mathscr{E}[\![E_0]\!]\rho)$ (A1.13)

$\mathscr{E}[\![\textbf{let}\ I = E_0\ \textbf{in}\ E_1]\!]\rho \equiv$
$\quad strict(\lambda\varepsilon\ .\ \mathscr{E}[\![E_1]\!](\rho[\varepsilon/I]))(\mathscr{E}[\![E_0]\!]\rho)$ (A1.14)

$\mathscr{E}[\![\textbf{iterate}\ I\ \textbf{to}\ E_0\ \textbf{from}\ E_1\ \textbf{while}\ E_2]\!]\rho \equiv$
$\quad strict\{fix(\lambda\phi\ .\ \lambda\varepsilon\ .\ cond\langle\phi(\mathscr{E}[\![E_0]\!](\rho[\varepsilon/I])), \varepsilon\rangle$
$\qquad (\mathscr{E}[\![E_2]\!](\rho[\varepsilon/I])))\}(\mathscr{E}[\![E_1]\!]\rho)$ (A1.15)

$\mathscr{C}: [\textbf{Cmd} \to [U \to U]]$
$\mathscr{C}[\![I := E]\!]\rho \equiv strict(\lambda\varepsilon\ .\ \rho[\varepsilon/I])(\mathscr{E}[\![E]\!]\rho)$ (A1.16)

$\mathscr{C}[\![\textbf{while}\ E\ \textbf{do}\ \Gamma]\!] \equiv$
$\quad fix(\lambda\theta\ .\ \lambda\rho\ .\ cond\langle\theta(\mathscr{C}[\![\Gamma]\!]\rho), \rho\rangle(\mathscr{E}[\![E]\!]\rho))$ (A1.17)

$\mathscr{C}[\![\Gamma_0; \Gamma_1]\!]\rho \equiv \mathscr{C}[\![\Gamma_1]\!](\mathscr{C}[\![\Gamma_0]\!]\rho)$ (A1.18)

$\mathscr{C}[\![\textbf{if}\ E\ \textbf{then}\ \Gamma_0\ \textbf{else}\ \Gamma_1]\!]\rho \equiv$
$\quad cond\langle\mathscr{C}[\![\Gamma_0]\!]\rho, \mathscr{C}[\![\Gamma_1]\!]\rho\rangle(\mathscr{E}[\![E]\!]\rho)$ (A1.19)

$\mathscr{C}[\![()]\!]\rho \equiv \rho$ (A1.20)

Appendix 2. The PL interpreter

$Terminal(\sigma) \equiv (\sigma \equiv \ulcorner\textbf{done}\urcorner) \vee (\sigma \equiv \ulcorner\textbf{error}\urcorner)$ (A2.1)

$Step(\sigma) \equiv$ (A2.2)
$\sigma \equiv \ulcorner\textbf{eval}\ E\ \textbf{in}\ \rho; \kappa\urcorner \to$
$\quad E \equiv \ulcorner I\urcorner \to$
$\qquad (Has(\rho, I) \to Append\ 'val' : Select(\rho, I)\ to\ \kappa,$
$\qquad \ulcorner\textbf{error}\urcorner),$ (A2.3)
$\quad E \equiv \ulcorner I_0(I_1)\urcorner \to$
$\qquad Has(\rho, I_0) \to Has(\rho, I_1) \to$
$\qquad\quad Select(\rho, I_0) \equiv \ulcorner\textbf{function}(I_2) : E_2\ \textbf{in}\ \rho_2\urcorner \to$
$\qquad\qquad \ulcorner\textbf{eval}\ E_2\ \textbf{in}\ (Append\ I_2 : Select(\rho, I_1)\ to\ \rho_2); \kappa\urcorner,$
$\qquad\quad Select(\rho, I_0) \equiv \ulcorner\textbf{recfun}\ I_2(I_3) : E_2\ \textbf{in}\ \rho_2\urcorner \to$
$\qquad\qquad \ulcorner\textbf{eval}\ E_2\ \textbf{in}\ (Append\ I_3 : Select\ (\rho, I_1)\ to$
$\qquad\qquad\qquad (Append\ I_2 : Select(\rho, I_0)\ to\ \rho_2)); \kappa\urcorner,$
$\qquad\quad \ulcorner\textbf{error}\urcorner,$
$\qquad \ulcorner\textbf{error}\urcorner, \ulcorner\textbf{error}\urcorner,$ (A2.4)
$\quad E \equiv \ulcorner\textbf{proc}(I) : E_0\urcorner \to$
$\qquad Append\ 'val' : \ulcorner\textbf{function}\ (I) : E_0\ \textbf{in}\ \rho\urcorner\ to\ \kappa,$ (A2.5)

$E \equiv$ ⌜**rec** $I_0(I_1) : E_0$⌝ →
Append 'val' : ⌜**rec fun** $I_0(I_1) : E_0$ **in** ρ⌝ *to* κ, (A2.6)

$E \equiv$ ⌜Γ **res** E_0⌝ →
⌜**perform** Γ **in** ρ; ⌜**eval** E_0 **in** ⟨⟩; κ⌝⌝, (A2.7)

$E \equiv$ ⌜B⌝ →
Append 'val' : Rep(B) *to* κ, (A2.8)

$E \equiv$ ⌜OI⌝ →
Has(ρ, I) →
*Check*1(O, *Select*(ρ, I)) →
*Append 'val' : Oper*1(O, *Select*(ρ, I)) *to* κ,
⌜**error**⌝, ⌜**error**⌝, (A2.9)

$E \equiv$ ⌜$I_1 \Omega I_2$⌝ →
Has(ρ, I_1) → *Has*(ρ, I_2) →
*Check*2(Ω, *Select*(ρ, I_1), *Select*(ρ, I_2)) →
*Append 'val' : Oper*2(Ω, *Select*(ρ, I_1), *Select*(ρ, I_2))
to κ,
⌜**error**⌝, ⌜**error**⌝, ⌜**error**⌝, (A2.10)

⌜**error**⌝,

$\sigma \equiv$ ⌜**perform** Γ **in** ρ; θ⌝ →

$\Gamma \equiv$ ⌜$I := E$⌝ →
⌜**eval** E **in** ρ; ⌜**assign** ⟨⟩ **to** I **in** ρ; θ⌝⌝, (A2.11)

$\Gamma \equiv$ ⌜**while** I **do** Γ_0⌝ →
Has(ρ, I) → *Select*(ρ, I) ≡ ⌜**boolean** B⌝ →
$B \equiv$ ⌜**true**⌝ → ⌜**perform** Γ_0 **in** ρ;
⌜**perform** ⌜**while** I **do** Γ⌝ **in** ⟨⟩; θ⌝⌝,
Append 'env' : ρ *to* θ,
⌜**error**⌝, ⌜**error**⌝, (A2.12)

$\Gamma \equiv$ ⌜Γ_0; Γ_1⌝ →
⌜**perform** Γ_0 **in** ρ; ⌜**perform** Γ_1 **in** ⟨⟩; θ⌝⌝, (A2.13)

$\Gamma \equiv$ ⌜**if** I **then** Γ_0 **else** Γ_1⌝ →
Has(ρ, I) → *Select*(ρ, I) ≡ ⌜**boolean** B⌝ →
$B \equiv$ ⌜**true**⌝ → ⌜**perform** Γ_0 **in** ρ; θ⌝,
⌜**perform** Γ_1 **in** ρ; θ⌝,
⌜**error**⌝, ⌜**error**⌝, (A2.14)

$\Gamma \equiv$ ⌜( )⌝ →
Append 'env' : ρ *to* θ, (A2.15)

$\ulcorner$**error**$\urcorner$,

$\sigma \equiv \ulcorner$**assign** ε **to** l **in** $\rho; \theta\urcorner \rightarrow$
Append 'env':(*Append* $l:\varepsilon$ *to* ρ) *to* θ, (A2.16)

$\ulcorner$**error**$\urcorner$

Appendix 3. Continuation semantics for PL

Semantic Domains

$\beta \in \boldsymbol{B}$	(Basic values)
$\phi \in \boldsymbol{F} = [\boldsymbol{E} \rightarrow [\boldsymbol{K} \rightarrow \boldsymbol{A}]]$	(Function values)
$\varepsilon \in \boldsymbol{E} = \boldsymbol{B} + \boldsymbol{F}$	(Expressed values)
$\rho \in \boldsymbol{U} = [\mathbf{Ide} \rightarrow \boldsymbol{E}]$	(Environments)
$\theta \in \boldsymbol{G} = [\boldsymbol{U} \rightarrow \boldsymbol{A}]$	(Command continuations)
$\kappa \in \boldsymbol{K} = [\boldsymbol{E} \rightarrow \boldsymbol{A}]$	(Expression continuations)
$\boldsymbol{A} = \boldsymbol{E}$	(Answers)

Semantic Valuations

$\mathscr{B}:[\mathbf{Bas} \rightarrow \boldsymbol{B}]$
$\mathscr{O}:[\mathbf{Mon} \rightarrow [\boldsymbol{E} \rightarrow \boldsymbol{E}]]$
$\mathscr{W}:[\mathbf{Dya} \rightarrow [[\boldsymbol{E} \times \boldsymbol{E}] \rightarrow \boldsymbol{E}]]$
$\mathscr{E}:[\mathbf{Exp} \rightarrow [\boldsymbol{U} \rightarrow [\boldsymbol{K} \rightarrow \boldsymbol{A}]]]$

$\mathscr{E}[\![I]\!]\rho\kappa \equiv (strict\kappa)(\rho[\![I]\!])$ (A3.1)
$\mathscr{E}[\![E_0(E_1)]\!]\rho\kappa \equiv \mathscr{E}[\![E_1]\!]\rho\{\lambda\varepsilon_1 \,.\, \mathscr{E}[\![E_0]\!]\rho\{\lambda\varepsilon_0 \,.\, (\varepsilon_0|\boldsymbol{F})\varepsilon_1\kappa\}\}$ (A3.2)
$\mathscr{E}[\![\mathbf{proc}(I):E]\!]\rho\kappa \equiv \kappa(\lambda\varepsilon \,.\, \mathscr{E}[\![E]\!](\rho[\varepsilon/I])\ \mathbf{in}\ \boldsymbol{E})$ (A3.3)
$\mathscr{E}[\![\mathbf{rec}\ I_0(I_1):E]\!]\rho\kappa \equiv$
$\quad \kappa(fix(\lambda\phi\lambda\varepsilon \,.\, \mathscr{E}[\![E]\!](\rho[\phi\ \mathbf{in}\ \boldsymbol{E}/I_0][\varepsilon/I_1]))\ \mathbf{in}\ \boldsymbol{E})$ (A3.4)
$\mathscr{E}[\![\Gamma\ \mathbf{res}\ E]\!]\rho\kappa \equiv \mathscr{C}[\![\Gamma]\!]\rho\{\lambda\rho' \,.\, \mathscr{E}[\![E]\!]\rho'\kappa\}$ (A3.5)
$\mathscr{E}[\![OE]\!]\rho\kappa \equiv \mathscr{E}[\![E]\!]\rho\{\lambda\varepsilon \,.\, (strict\ \kappa)((\mathscr{O}[\![O]\!])\varepsilon)\}$ (A3.6)
$\mathscr{E}[\![E_0\Omega E_1]\!]\rho\kappa \equiv$
$\quad \mathscr{E}[\![E_0]\!]\rho\{\lambda\varepsilon_0 \cdot \mathscr{E}[\![E_1]\!]\rho\{\lambda\varepsilon_1 \,.\, (strict\kappa)((\mathscr{W}[\![\Omega]\!])(\varepsilon_0, \varepsilon_1))\}\}$ (A3.7)
$\mathscr{E}[\![\mathbf{if}\ E_0\ \mathbf{then}\ E_1\ \mathbf{else}\ E_2]\!]\rho\kappa \equiv \mathscr{E}[\![E_0]\!]\rho\{cond\langle\mathscr{E}[\![E_1]\!]\rho\kappa, \mathscr{E}[\![E_2]\!]\rho\kappa\rangle\}$ (A3.8)
$\mathscr{E}[\![\mathbf{let}\ I = E_0\ \mathbf{in}\ E_1]\!]\rho\kappa \equiv \mathscr{E}[\![E_0]\!]\rho\{\lambda\varepsilon \,.\, \mathscr{E}[\![E_1]\!](\rho[\varepsilon/I])\kappa\}$ (A3.9)
$\mathscr{E}[\![\mathbf{iterate}\ I\ \mathbf{to}\ E_0\ \mathbf{from}\ E_1\ \mathbf{while}\ E_2]\!]\rho\kappa \equiv$
$\quad \mathscr{E}[\![E_1]\!]\rho\{fix(\lambda\kappa'\lambda\varepsilon \cdot \mathscr{E}[\![E_2]\!](\rho[\varepsilon/I])$
$\quad \{cond\langle\mathscr{E}[\![E_0]\!](\rho[\varepsilon/I])\kappa', \kappa\varepsilon\rangle\})\}$ (A3.10)

$\mathscr{C}:[\mathbf{Com} \rightarrow [\boldsymbol{U} \rightarrow [\boldsymbol{G} \rightarrow \boldsymbol{A}]]]$
$\mathscr{C}[\![I:\!=E]\!]\rho\theta \equiv \mathscr{E}[\![E]\!]\rho\{\lambda\varepsilon \,.\, \theta(\rho[\varepsilon/I])\}$ (A3.11)
$\mathscr{C}[\![\mathbf{while}\ E\ \mathbf{do}\ \Gamma]\!]\rho\theta \equiv$
$\quad [fix(\lambda\theta'\lambda\rho' \,.\, \mathscr{E}[\![E]\!]\rho'\{cond\langle\mathscr{C}[\![\Gamma]\!]\rho'\theta', \theta\rho'\rangle\})]\rho$ (A3.12)
$\mathscr{C}[\![\Gamma_0; \Gamma_1]\!]\rho\theta \equiv \mathscr{C}[\![\Gamma_0]\!]\rho\{\mathscr{C}[\![\Gamma_1]\!]\rho\theta\}$ (A3.13)
$\mathscr{C}[\![\mathbf{if}\ E\ \mathbf{then}\ \Gamma_0\ \mathbf{else}\ \Gamma_1]\!]\rho\theta \equiv \mathscr{E}[\![E]\!]\rho\{cond\langle\mathscr{C}[\![\Gamma_0]\!]\rho\theta, \mathscr{C}[\![\Gamma_1]\!]\rho\theta\rangle\}$ (A3.14)
$\mathscr{C}[\![()]\!]\rho\theta \equiv \theta\rho$ (A3.15)

References

[1] J.B. Dennis, On storage management for advanced programming languages. MIT Computation Structures Group Memo 109-1 (1974).
[2] J.B. Dennis, Semantic theory for computer systems (Course Notes for 6.841), Massachusetts Institute of Technology (1976).
[3] M.J.C. Gordon, Models of pure lisp. Experimental Programming Report 31, Department of Machine Intelligence, University of Edinburgh (1973).
[4] C.A.R. Hoare, An axiomatic basis for computer programming, *Comm. ACM* **12** (1969) 576.
[5] G.F. Ligler, Surface properties of programming language constructs, in: G. Huet and G. Kahn Eds., *Actes du Colloque Construction, Amélioration et Vérification du Programmes* (Institut de Recherches d'Informatique et d'Automatique, Rocquencourt, France, 1975) 299.
[6] P. Lucas and K. Walk, On the formal description of PL/I, *Annual Review in Automatic Programming* 6 (Pergamon, 1969) 3.
[7] Z. Manna and J. Vuillemin, Fixpoint approach to the theory of computation, *Comm. ACM* **15** (1972) 528. (Note: 'computational induction' is the name used in this reference for what we have called 'fixed point induction'.)
[8] R.E. Milne, The formal semantics of computer languages and their implementations, Ph.D. thesis, Cambridge University (1974–75).
[9] R.E. Milne and C. Strachey, *A Theory of Programming Language Semantics* (Chapman and Hall, London, and Wiley, New York, 1976).
[10] A.J.R.G. Milner, Implementation and applications of Scott's logic for computable functions, *SIGPLAN Notices* **7** (1972) 1.
[11] J.C. Reynolds, On the relation between direct and continuation semantics, *Proc. 2nd Colloquium on Automata, Languages and Programming* (Saarbrucken, 1974).
[12] D.S. Scott and C. Strachey, Toward a mathematical semantics for computer languages, *Proc. Symposium on Computers and Automata*, Microwave Research Institute Symposium Series **21** (Polytechnic Institute of Brooklyn, 1971).
[13] J.E. Stoy, *Denotational Semantics: The Scott–Strachey Approach to Programming Language Theory* (MIT Press, Cambridge, MA, 1977).
[14] C. Strachey and C.P. Wadsworth, Continuations: a mathematical semantics for handling full jumps, Technical Monograph PRG-11 (Oxford University Computing Laboratory, Programming Research Group (1974)).
[15] A. Tarski, A lattice-theoretical fixpoint theorem and its application, *Pacific J. Math.* **5** (1955) 285.
[16] R.D. Tennent, The denotational semantics of programming languages, *Comm. ACM* **19** (1976) 437.