Reasoning About Procedures as Parameters

S. M. German*
Harvard University

E. M. Clarke, Jr.
Carnegie-Mellon University

J. Y. Halpern
IBM Research, San Jose

1. Introduction

In [2] it was shown that for sufficiently complex Algol-like languages there cannot be a Hoare axiom system which is sound and relatively complete in the sense of Cook [4]. The incompleteness exists whenever a programming language contains (or can simulate) the following combination of features: (i) procedures with procedures passed as parameters, (ii) recursion, (iii) use of non-local variables, (iv) static scoping, and (v) local procedure declarations. Moreover, if any one of the features (i), (ii), (iv), or (v) is dropped from Algol, a sound and relatively complete axiomatization can be obtained for the resulting languages (called L2, L3, L5, and L6 in [2]). It has long been conjectured that the same is true for the language L4 which results when feature (iii), use of non-local variables, is dropped.

The languages L2, L3, L5, and L6 are relatively easy to axiomatize, since they all have the *finite range* property. Informally, this property is that for each program, there is a bound on the number of distinct procedure environments, or associations between procedure names and bodies, that can be reached. However, L4 does not have the finite range property. Intuition suggests that some new reasoning methods are needed for such programs. This intuition is supported by [9], where a precise characterization is given for the class of Hoare axiom systems based on copy rules, and it is shown that none of these axiom systems can deal adequately with infinite range.

The main new results in this paper are an axiom system for reasoning about programs with infinite range and a new technique for constructing relative completeness proofs for languages with procedure parameters. We also present a new way of formalizing the semantics of programs with free procedure names. Many of the techniques introduced in this paper are of general use beyond the immediate problem of the language L4. In the course of the relative completeness proof, we develop results of independent interest concerning the existence in general programming languages of *interpreter programs*; i.e., fixed programs capable of simulating any program in the language.

---

*Current Address: GTE Laboratories, Inc., 40 Sylvan Road, Waltham, Ma 02254

This research was supported by NSF Grant MCS-82-16706.

For a brief preview of our approach to reasoning about programs with infinite range, let us consider a small example of a formula in our logic. We retain the idea of using partial correctness assertions $\{U\}S\{V\}$, where U and V are first order, for specifying and reasoning about statements. To specify a procedure p with a procedure parameter r, we construct more complicated formulas containing partial correctness assertions, to describe how the semantics of r affects the semantics of p(r). For instance, let p be the procedure

proc p(x:r); **begin** r(x); r(x) **end**

which calls the formal procedure r twice on the variable parameter x. For an arithmetic domain, p satisfies the formula

$$\forall r,v(\{y=y_0\}\ r(y)\{y=y_0\cdot v\}\ \rightarrow\ \{x=x_0\}\ p(x:r)\ \{x=x_0\cdot v^2\})$$

Intuitively, this formula says that for all procedures r and domain values v, if the call r(y) multiplies y by v, then for the same procedure r and value v, the call p(x:r) multiplies x by v^2.

At this point, one might wonder whether this approach is sufficient to specify all procedures. Indeed, the essence of the relative completeness proof for our axiom system is that in L4, the necessary facts about procedures can always be expressed by an appropriate formula of our logic.

A different approach to axiomatizing procedures as parameters, based on the use of higher order logic in the assertion language, has been developed in [10, 5]. In both of these papers, the axiom system is assumed to include as axioms all of the formulae valid in a certain higher order theory related to the interpretation. In contrast, our axiom system includes as axioms only the first order theory of the interpretation. Also, in [10, 5], the notion of expressiveness used in establishing relative completeness takes a more general form, involving higher order formulas, while we use the familiar notion of expressiveness as in [4]. It has been conjectured that the two notions of expressiveness are equivalent; this problem is under study [6].

2. Programming Language

A statement has one of the forms:

$$\langle\text{statement}\rangle ::= x := e \mid S_1; S_2 \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \mid S_1 \textbf{ or } S_2$$
$$\mid \textbf{begin var } x; S \textbf{ end} \mid \textbf{begin } E; S \textbf{ end} \mid p(\bar{x}:\bar{r})$$

The statement S_1 **or** S_2 makes a nondeterministic choice and executes one of the statements. In **begin** E; S **end**, E is a *procedure environment*; i.e., a set of *procedure declarations*. We sometimes abbreviate this as E | S. In $p(\bar{x}:\bar{r})$, $\bar{x}$ is a list of variable identifiers and $\bar{r}$ is a list of procedure identifiers. We often abbreviate **begin** E; S **end** to E | S.

A set of procedure declarations has the form

$$\textbf{proc } p_1(\bar{x}_1;\bar{r}_1);\ B_1$$
$$\vdots$$
$$\textbf{proc } p_m(\bar{x}_m;\bar{r}_m);\ B_m$$

and introduces possibly mutually recursive declarations of $p_1 \ldots p_m$. The p_i are called *declared procedure names*; the r_i are *formal procedure names*. B_i is the body of procedure p_i.

An occurrence of an identifier in a statement may be either *free* or *bound* in the usual sense. Note that we allow free procedure identifiers to appear in statements. A *program* A is a statement with no free procedures.

A declaration **proc** $p(\bar{x};\bar{r})$; B is said to *have no global variables* if all the free variables of B are in $\bar{x}$. An environment (statement, program) has no global variables if all its declarations have no free variables. Note that such an environment (statement, program) *may* have free procedures.

We are primarily concerned with programs which have no global variables. For historical reasons [2] this language is often called L4. In L4, the only variables that can be accessed or changed by a procedure call are the actual variable parameters in the call. This property will help us to get a sound and relatively complete axiom system for L4.

3. Semantics of statements

Let I be a given first order interpretation. A program *state* is a mapping from the set of program variables to dom(I). The meaning of a statement is a binary relation on states. Given procedure environment E, we associate with each statement A its meaning in I and E via the function $\mathcal{M}_{I,E}$. We define $\mathcal{M}_{I,E}$ first for statements without procedure calls or procedure declarations by induction on structure of the statement:

$$\mathcal{M}_{I,E}(\textbf{error}) = \varnothing$$

$$\mathcal{M}_{I,E}(x:=e) = \{(s,s[I(e)/x]) \mid s \text{ is a state}\}$$

$$\mathcal{M}_{I,E}(A_1;A_2) = \{(s,s') \mid \exists\, t\, ((s,t) \in \mathcal{M}_{I,E}(A_1) \text{ and } (t,s') \in \mathcal{M}_{I,E}(A_2))\}$$

$$\mathcal{M}_{I,E}(A_1 \textbf{ or } A_2) = \mathcal{M}_{I,E}(A_1) \cup \mathcal{M}_{I,E}(A_2)$$

$$\mathcal{M}_{I,E}(\textbf{if b then } A_1 \textbf{ else } A_2) = \{(s,s') \in \mathcal{M}_{I,E}(A_1) \mid I,s \models b\} \cup \{(s,s') \in \mathcal{M}_{I,E}(A_2) \mid I,s \models \neg\, b\}$$

$$\mathcal{M}_{I,E}(\textbf{begin var x; A end}) = \{(s,s') \mid \exists\, (u,u') \in \mathcal{M}_{I,E}(A),\ u = s[I(\underline{a}/x)],\ s' = u'[s(x)/x]\}$$

(where $\underline{a}$ is a fixed constant)

We give meaning to statements with procedure declarations and procedure calls by first converting them to statements without procedure declarations and calls, by using an auxiliary function Approx^k_E. Informally, Approx^k_E

gives the k^{th} approximation to the fixed-point meaning of a recursively defined procedure in procedure environment E. We define $Approx_E^k$ by induction on k and the structure of statements:

1. $Approx_E^k$ (**error**) = **error**

2. $Approx_E^k$ (x:=e) = x:=e

3. $Approx_E^k$ (Λ_1;Λ_2) = $Approx_E^k$(Λ_1);$Approx_E^k$(Λ_2)

4. $Approx_E^k$ (Λ_1 **or** Λ_2) = $Approx_E^k$(Λ_1) **or** $Approx_E^k$(Λ_2)

5. $Approx_E^k$ (**if** b **then** Λ_1 **else** Λ_2) = **if** b **then** $Approx_E^k$ (Λ_1) **else** $Approx_E^k$(Λ_2)

6. $Approx_E^k$ (**begin var** x; Λ **end**) = **begin var** x; $Approx_E^k$(Λ) **end**

   renaming the bound variable x if it appears free in E (see below).

7. $Approx_E^k$ (E' | A) = $Approx_{E \cup E'}^k$ (A)

   renaming bound variables in E' if necessary (see below).

8. $Approx_E^k$ (p ($\bar{x}$:$\bar{q}$)) =

   error if k=0 and p is declared in E

   $Approx_E^{k-1}$ ([$\bar{x}$ / $\bar{x}$', $\bar{q}$ / $\bar{q}$']B)if k > 0 and the declaration **proc** p($\bar{x}$':$\bar{q}$'); B $\in$ E

   E_k | p($\bar{x}$:$\bar{q}$) otherwise, where E_k is defined below.

If E consists of the declarations **proc** $p_i(\bar{x}_i:\bar{r}_i)$; B_i, i=1, ..., n, then E_K consists of the declarations **proc** $p_i(\bar{x}_i:\bar{r}_i)$; $Approx_E^k(p_i(\bar{x}_i:\bar{r}_i))$. Note $Approx_E^k(p_i(\bar{x}_i:\bar{r}_i)) = Approx_E^{k-1}(B_i)$ if k > 0, so this inductive definition is indeed well defined.

In clause 6 if the bound variable x appears free in E, then we have to rename the x to some fresh variable x' to avoid capturing the free variable in E. Thus we would get

begin var x'; $Approx_E^{k-1}$([x'/x]A) **end**

Similarly, in clause 7, if some procedure identifier declared in E' already appears in E, we have to rename the identifiers in E' (and all their bound occurrences in A) to avoid naming conflicts.

Note that if A is a program, then $Approx_E^k(A) = Approx_{\emptyset}^k(A)$ (i.e., $Approx_E^k(A)$ is independent of E), and $Approx_{\emptyset}^k(A)$ does not contain any procedure declarations or procedure calls.

Given a procedure environment E and statement A, let E^A = E $\cup$ {p($\bar{x}$:$\bar{r}$); **error** | p appears

free in (E | A)}. Note $E^{\Lambda}|\Lambda$ is a program, since it has no free procedure identifiers. To complete our semantics, we define, for any statement Λ,

$$\mathcal{M}_{I,E}(\Lambda) = \cup_k \text{Approx}^k_{\emptyset}(E^{\Lambda}|\Lambda)$$

We next define two statements Λ_1 and Λ_2 to be equivalent, written $\Lambda_1 \equiv \Lambda_2$, if $\mathcal{M}_{I,E}(\Lambda_1) = \mathcal{M}_{I,E}(\Lambda_2)$ for all interpretations I and procedure environments E. Similarly, we write $\Lambda_1 \leq \Lambda_2$ if $\mathcal{M}_{I,E}(\Lambda_1) \subseteq \mathcal{M}_{I,E}(\Lambda_2)$ for all interpretations I and procedure environments E.

The following lemma will be used throughout the paper.

Lemma 1:

(a)If $p(\bar{x}:\bar{r})$; $B \in E$, then

$$E \mid p(\bar{y}:\bar{s}) \equiv E \mid [\bar{x}/\bar{y}, \bar{r}/\bar{s}]B$$

(b) $E \mid \Lambda_1;\Lambda_2 \equiv (E \mid \Lambda_1);(E \mid \Lambda_2)$

(c) $E \mid \Lambda_1 \text{ or } \Lambda_2 \equiv E \mid \Lambda_1$ **or** $E \mid \Lambda_2$

(d) $E \mid$ **if** b **then** Λ_1 **else** $\Lambda_2 \equiv$ **if** b **then** $E \mid \Lambda_1$ **else** $E \mid \Lambda_2$

(e) If x does not appear free in Λ, then

$E|$ **begin var** x; A **end** $\equiv$ **begin var** x; $E|A$ **end**

(f) If E_1, E_2 do not contain distinct declarations for the same procedure identifier, then

$E_1|(E_2|A) \equiv (E_1 \cup E_2)|A$

(g) If none of the procedures declared in E appear free in A, then

$E|A \equiv A$

(h) If A and A' are identical up to renaming of bound varaibles,

$A \equiv A'$

From Lemma 1, we immediately get the following:

Corollary: Every statement is equivalent to one in a normal form, where $E|A$ occurs only if A is a procedure call.

4. Syntax and Semantics of Formulas

To define the set of formulas used in our axiom system, we begin by fixing a first order type Σ which determines the finite set of constant, predicate, and function symbols that can appear in programs and first-order formulas. We permit three distinct kinds of variables: ordinary variables (x), environment variables (v), and procedure variables (r). The syntactic distinction between ordinary and environment variables is that ordinary variables, like the variables in most Hoare axiom systems, may appear in both programs and first-order formulas; environment variables are a new class of variables which may appear only in first-order formulas. Procedure variables may appear only in programs, subject to these restrictions on the use of variables, a formula has the form

$$\langle \text{formula} \rangle ::= U \mid \{U\}\ S\ \{V\} \mid \{H_1,\ldots,H_n\} \mid (H_1 \rightarrow H_2) \mid \forall v H \mid \forall r H$$

where U and V are first order, S is any statement, H and $H_1,\ldots,H_n$ are formulas, v is an environment variable and r is a procedure variable. Arbitrary nesting of $(H_1 \rightarrow H_2)$, $\forall vH$, and $\forall rH$, is permitted.

In order to give meaning to formulas we need an interpretation I, which gives meaning to the symbols in Σ in the usual way, an environment valuation σ which assigns an element of dom(I) to each environment variable, and a procedure environment E.

$I, E, \sigma \models U$ iff for all s, $I, s \models U$ (where $I, s \models U$ is defined in the usual way)

$I, E, \sigma \models \{U\}\,\Lambda\,\{V\}$ iff for all s,s': $I, s \models U$ and $(s,s') \in \mathcal{M}_{b_{I,E}}(A)$ implies $I, s' \models V$.

$I, E, \sigma \models \{H_1,\ldots,H_n\}$ iff $I, E, \sigma \models H_i$, $i = 1,\ldots,n$.

$I, E, \sigma \models H_1 \rightarrow H_2$ iff $I, E, \sigma \models H_1$ implies $I, E, \sigma \models H_2$.

$I, E, \sigma \models \forall v\, H$ iff for all $d \in$ dom(I): $I,E,\sigma[d / v] \models H$.

$I, E, \sigma \models \forall r\, H$ iff for all procedure declarations **proc** $r'(\vec{x};\vec{q})$; B $I,E \cup \{\textbf{proc}\ r'(\vec{x};\vec{q});\ B\} \models H\,[r'/r]$.

where r' is a fresh variable which does not appear in E and has the same type as r.

Finally, we define $I \models H$ iff for all E, σ: $I,E,\sigma \models H$.

Note that the meaning of a free environment variable in a formula is the same wherever it appears. In contrast, the meaning of a program variable is "local" to each partial correctness assertion in which it appears, since it is effectively universally quantified. For example, consider the following two formulas

(1) {True} y : = y {x = 3} → {True} y : = y {False}

(2) {True} y : = y {v = 3} → {True} y : = y {False}

where x and y are ordinary variables and v is an environment variable. Formula 1 is valid, because the antecedent {True} y : = y {x = 3} is false: it is not the case that for all initial values of x and y, y : = y sets x to 3. Formula 2 is *not* valid (in any interpretations with more than one domain element), because v is quantified over the whole formula. For $\sigma(v) = 3$, the antecedent is true but the consequent is false, giving a

counterexample to (2).

5. Axiom System

Consider the following collection of axiom schemes and rules of inference.

Axiom schemes

AX 1. {True} error {False}

AX 2. {U [e / x]} x : = e {U}

AX 3. {{U} Λ_1 {V}, {V} Λ_2 {W}} → {U} Λ_1;Λ_2 {W}

AX 4. {{U ∧ b} Λ_1 {V}, {U ∧ ¬ b} Λ_2 {V}} → {U} if b then Λ_1 else Λ_2 {V}

AX 5. {{U} Λ_1 {V}, {U} Λ_2 {V}} → {U} Λ_1 or Λ_2 {V}

AX 6. {U} Λ[x' / x] {V} → {U} begin var x; Λ end {V}, where x' does not appear in U,V, or A.

AX 7. {U_1 ⊃ U, {U} Λ {V}, V ⊃ V_1} → {U_1} Λ {V_1}

AX 8. {U} Λ {V} → {∃ y U} Λ {∃ yV} if y is an ordinary variable not free in A.

AX 9. {U} Λ {V} → {U ∧ Q} Λ {V ∧ Q} if no variable free in Q is also free in A.

AX 10. {U} Λ {V} → {U} Λ' {V} provided Λ ≡ Λ' via the rules of Lemma 1.

AX 11a. ∀ vH → H[v' / v]

AX 11b. ∀ rH → H[r' / r] where v is an environment variable, and r is a procedure variable.

AX 12. {U} Λ {V} → {U π} Λ π {V π} where π is an injective mapping on the set of ordinary variables.

AX 13. H → H & C

provided C is a first order formula whose only free variables are environment variables and H & C is defined. We define H & C by induction. For cases 3-6 below, H & C is defined on the left side of the equivalence if all of the formulas on the right side are defined.

1. H & C is not defined if H is a first order formula.
2. {U} Λ {V} & C ≡ {U ∧ C} Λ {V ∧ C}.
3. {H_1, ..., H_n} & C ≡ {H_1 &C, ..., H_n & C}.
4. (H_1 → H_2) & C ≡ H_1 & C → H_2 & C.
5. (∀ v H) & C ≡ ∀ v' (H[v'/v] & C) where v' is not free in H, C.
6. (∀ r H) & C ≡ ∀ r (H & C).

AX 14. $\{H_1, \ldots, H_n\} \rightarrow H_i \quad 1 \leq i \leq n$

AX 15a. $(H_1 \rightarrow (H_2 \rightarrow H_3)) \rightarrow (H_1 \cup H_2 \rightarrow H_3)$

AX 15b. $((H_1 \cup H_2) \rightarrow H_3) \rightarrow (H_1 \rightarrow (H_2 \rightarrow H_3))$

AX 16a. $H \rightarrow (\varnothing \rightarrow H)$

AX 16b. $(\varnothing \rightarrow H) \rightarrow H$

AX 17. $\{H_1 \rightarrow H_2, H_3 \rightarrow H_4\} \rightarrow \{H_1 \cup H_3 \rightarrow H_2 \cup H_4\}$

AX 18. $\{H_1 \rightarrow H_2, H_2 \rightarrow H_3\} \rightarrow \{H_1 \rightarrow H_3\}$

Rules of Inference

R1. $$\frac{H_1, (H_1 \rightarrow H_2)}{H_2}$$

R2. $$\frac{H_1 \rightarrow H_2}{E \mid H_1 \rightarrow E \mid H_2}$$

where $E \mid H$ is the result of replacing every p.c.a. $\{U\} A \{V\}$ in H by $\{U\} E \mid A \{V\}$, subject to the usual conditions about renaming bound variables to avoid capture of free variables in E.

R3. $$\frac{H \rightarrow \{U\} A \{V\}}{H \rightarrow \{\exists v\, U\} A \{\exists v\, V\}}$$

R4. $$\frac{H \rightarrow H_1, \ldots, H \rightarrow H_n}{H \rightarrow \{H_1, \ldots, H_n\}}$$

R5. $$\frac{H \rightarrow H'}{\{H \rightarrow \forall v\, H', H \rightarrow \forall r\, H'\}}$$

provided v and r are not free in H.

R6. Suppose E consists of the declarations **proc** $p_i(\bar{x}_i : \bar{r}_i)$; B_i, $i = 1, \ldots, n$,

and $p_1, \ldots, p_n$ do not appear free in $H_1\ H_1, \ldots, H_n$.

$$\frac{H \to (\{\forall \bar{r}_i\, \bar{v}_i (H_i \to \{U_i\}\ p_i(\bar{x}_i : \bar{r}_i)\ \{V_i\}),\ i = 1, \ldots, n\} \to \{\forall \bar{r}_i\, \bar{v}_i (H_i \to \{U_i\}\ B_i\ \{V_i\}), i = 1, \ldots, n\})}{H \to \{\forall \bar{r}_i\, \bar{v}_i (H_i \to \{U_i\}\ E \mid p_i(\bar{x}_i : \bar{r}_i)\{V_i\}),\ i = 1, \ldots, n\}}$$

Roughly speaking R6, the recursion rule, says that whenever we can infer something about a call $p_i(\bar{x}_i : \bar{r}_i)$ from some hypotheses H_i, we can infer the same thing about the associated body B_i (again from the hypothesis H_i) then from the hypothesis H_i we can draw the same conclusion about the declared call $E \mid p_i(\bar{x}_i : \bar{r}_i)$.

Several of the rules, such as R3, R4, R5, and R6, involve a formula H which appears in both the antecedent and consequent of the rule. In all of these rules, the role of H is to allow the rule to be applied relative to some chosen set of assumptions. Rule R4, for instance, could have been stated in a less general form as

R4'
$$\frac{H_1, \ldots, H_n}{\{H_1, \ldots, H_n\}}$$

which says that if all of the H_i are valid formulas, then $\{H_1, \ldots, H_n\}$ is valid. However, it is sometimes necessary to make more general deductions of the form: if each of the H_i is a valid consequence of H, then so is $\{H_1, \ldots, H_n\}$.

6. Soundness of R3

In this section we show that the axiom schemes and rules of inference presented in the previous section are sound; i.e., if $\mathrm{Th}(I) \vdash H$ then $I \models H$ for any interpretation I and formula H (where $\mathrm{Th}(I)$ is the set of all first-order formulas valid in I). We will concentrate on proving the soundness of the recursion rule R6 here, leaving the soundness of the rest of the system to the full paper. We must show that whenever the antecedent of R6 is valid, then the conclusion is also valid. So suppose that

$$(0) I \models H \to (\{\forall \bar{r}_i \bar{v}_i (H_i \to \{U_i\}\ p_i(\bar{x}_i : \bar{r}_i)\ \{V_i\}), i = 1, \ldots, n\} \to \{\forall \bar{r}_i \bar{v}_i (H_i \to \{U_i\} B_i \{V_i\}), i = 1, \ldots, n\})$$

We want to show that for all environments F and valuations σ that

$$(1)\quad I, F, \sigma \models H \to (\{\forall \bar{r}_i \bar{v}_i (H_i \to \{U_i\} p_i(\bar{x}_i : \bar{r}_i)\{V_i\}), i = 1, \ldots, n\}$$

So suppose

$$(2)\quad I, F, \sigma \models H$$

(otherwise the result is immediate). Thus we must show

(3) $I,F,\sigma \models \{\forall \bar{r}_i,\bar{v}_i(H_i \rightarrow \{U_i\}\ p_i(\bar{x}_i:\bar{r}_i)\ \{V_i\}), i=1, \ldots, n\}$

We can suppose without loss of generality that $p_1, \ldots, p_n$ do not appear in F (otherwise we could just rename these bound variables). Let $F_m = F \cup \{\textbf{proc}\ p_i(\bar{x}_i:\bar{r}_i);\ \Lambda\text{pprox}^m_{\emptyset}(E|\,p_i\ (\bar{x}_i:\bar{r}_i)), i=1, \ldots, n\}$. We will show by induction on m that for all m

(4) $I,F_m,\sigma \models \{\forall \bar{r}_i,\bar{v}_i(H_i \rightarrow \{U_i\}\ p_i(\bar{x}_i:\bar{r}_i)\ \{V_i\}), i=1, \ldots, n\}$

By a straightforward argument we can show that no matter how the procedures in $\bar{r}_i$ are declared in F we have

$$\mathcal{M}_{I,F}(E|\,p_i(\bar{x}_i,\bar{r}_i)) = \cup_m \mathcal{M}_{I,F_m}(p_i(\bar{x}_i:\bar{r}_i))$$

Thus (4) suffices to prove (3). Proving (4) for $m = 0$ is trivial, since in F_0, we have **proc** $p_i(\bar{x}_i:\bar{r}_i)$; **error**. Assume (4) holds for $m = N - 1$. We now show it holds for $m = N$. It clearly suffices to show, for all choices of F and σ that

(5) $I,F_N,\sigma \models \{H_i \rightarrow \{U_i\}\ p_i(\bar{x}_i:\bar{r}_i)\{V_i\}, i=1, \ldots, n\}$

Without loss of generality, we can assume

(6) $I,F_N,\sigma \models \{H_1, \ldots, H_n\}$

Under this assumption, we must show

(7) $I,F_N,\sigma \models U_i\{p_i(x_i:r_i)\}V_i$

Using our inductive hypothesis (4) for $m = N\text{-}1$, the validity of (0), assumption (2), and the fact that free $(H) \cap \{p_1, \ldots, p_n\} = \emptyset$, we get

(8) $I,F_{N-1},\sigma \models \{\forall\ \bar{r}_i,\bar{v}_i(H_i \rightarrow \{U_i\}\ B_i\ \{V_i\}), i = 1, \ldots, n\}$

From (6) and the fact that free $(H_1, \ldots, H_n) \cap \{p_1, \ldots, p_n\} = \emptyset$, we get

(9) $I,F_{N-1},\sigma \models \{H_1, \ldots, H_n\}$

Using (8) and (9), we can conclude

(10) $I,F_{N-1},\sigma \models \{U_i\}\ B_i\ \{V_i\}$

We will now show

(11) $\mathcal{M}_{I,F_{N-1}}(B_i) \supseteq \mathcal{M}_{I,F_N}(p_i(x_i:r_i))$

(7) follows immediately from (10) and (11), so the proof of (11) will complete the inductive step of our proof. The

proof of (11) follows from the following chain of containments:

$$\begin{aligned}
&\mathcal{M}_{I,F_N}(p_i(\bar{x}_i:\bar{r}_i)) \\
&\quad = \mathcal{M}_{I,F_N}(\Lambda\text{pprox}^N_{\varnothing}(E \mid p_i(\bar{x}_i:\bar{r}_i))) \\
&\quad = \mathcal{M}_{I,F_{N-1}}(\Lambda\text{pprox}^N_{\varnothing}(E \mid p_i(\bar{x}_i:\bar{r}_i))) \text{ (since } F_{N-1}, F_N \text{ only differ on procedures declared in } E) \\
&\quad = \mathcal{M}_{I,F_{N-1}}(\Lambda\text{pprox}^{N-1}_{\varnothing}(E \mid B_i)) \text{ (by definition of Approx)} \\
&\quad \subseteq \mathcal{M}_{I,F_{N-1}}(B_i)
\end{aligned}$$

The last containment follows by induction on N followed by a subinduction on the structure of B_i. The only difficulty occurs if B_i is of the form $p_k(\bar{y}:\bar{q})$ where p_k is declared in E. Note that

$$\mathcal{M}_{I,F_{N-1}}(p_k(\bar{y}:\bar{q})) = \mathcal{M}_{I,F_{N-1}}([\bar{y}/\bar{x}_k,\bar{q}/\bar{r}_k])(\Lambda\text{pprox}^{N-1}_{\varnothing}(E \mid p_k(\bar{x}_k:\bar{r}_k)))$$

(since in F_{N-1}, we have the declaration proc $p_k(\bar{x}_i:\bar{r}_i)$), $(\Lambda\text{pprox}^{N-1}_{\varnothing}(E \mid p_k(\bar{x}_k:\bar{r}_k)))$

Thus we must show

$$\mathcal{M}_{I,F_{N-1}}(\Lambda\text{pprox}^{N-1}_{\varnothing}(E \mid p_k(\bar{y},\bar{q}))) \subseteq \mathcal{M}_{I,F_{N-1}}([\bar{y}/\bar{x}_k,\bar{q}/\bar{r}_k])(\Lambda\text{pprox}^{N-1}_{\varnothing}(E \mid p_k(\bar{x}_k:\bar{r}_k)))$$

This last inequality follows from the more general

(12) $\mathcal{M}_{I,F_{N-1}}(\Lambda\text{pprox}^{N-1}_{\varnothing}(E \mid S(\bar{q},\bar{y}))) \subseteq \mathcal{M}_{I,F_{N-1}}([\bar{y}/\bar{x}_k,\bar{q}/\bar{r}_k])(\Lambda\text{pprox}^{N-1}_{\varnothing}(E \mid S(\bar{x}_k:\bar{r}_k)))$

(12) is proved by induction on N and a subinduction on the structure of S. We leave details to the reader. □

7. Relative Completeness

In this section, we outline a proof of the relative completeness of the axiom system. The proof uses some interesting new ideas to deal with statements having free procedure names. The following discussion, however, is intended only to give an informal overview of the completeness proof. A more precise account appears in the final version of the paper and in [7].

First we need the following definitions. An interpretation I is *Herbrand definable* if every element of dom(I) is represented by a term involving only the constant and function symbols of I. In a fixed interpretation I, the *strongest postcondition* of a program A with respect to a (first order) precondition U, SP(A,U), is the set of all final states A can reach when started in a state satisfying U:

$$SP(A,U) = \{s' \mid \exists\, s\,(I,s \models U \wedge (s,s') \in \mathcal{M}_I(A))\}.$$

An interpretation I is *expressive* for a programming language L if for every program $A \in L$ and precondition U, SP(A,U) can be expressed by a first order formula using only the symbols of I.

Our main result is

Relative Completeness: Let I be Herbrand definable and expressive for $L4$, and let $A \in L4$. Then $I \models \{U\}\ A\ \{V\}$ implies $Th(I) \vdash \{U\}\ A\ \{V\}$; i.e., if a partial correctness assertion is true in an interpretation I, then it can be proved in our system using the first order theory of I as axioms.

In contrast, the relative completeness results obtained in [10, 5] depend on a more general notion of expressiveness with respect to preconditions in a higher order logic and require that a certain higher order theory of the interpretation be added as axioms.

The completeness proof uses the fact that in a language which does not permit non-local use of variables, a procedure call $p(\bar{x}:\bar{r})$ does not depend on any variables other than the ones in $\bar{x}$. Without this restriction, $p(\bar{x}:\bar{r})$ could depend on variables global to the body of p, global to procedures free in the body of p, or global to any of the procedures in $\bar{r}$.

One of the central ideas of the proof is that the act of passing a procedure parameter may be regarded as passing an input-output relation on a set of variables: in the call $p(:r)$, where r has type $r(\bar{x})$, r is a relation on $\bar{x}$. When r has higher type $r(\bar{x} : \bar{q})$, r is still an input-output relation on $\bar{x}$, but one which depends on the relations corresponding to its procedure parameters in $\bar{q}$. We wish to show that these relations can be represented by formulas in our logic. Returning to the example formula mentioned in the introduction,

$$\forall\, r,v\ (\{y=y_0\}\ r\ (y)\ \{y=y_0 \cdot v\}\ \rightarrow\ \{x=x_0\}\ p\ (x:r)\ \{x=x_0 \cdot v^2\})$$

observe how the environment variable v, appearing in the postconditions of the calls $r(y)$ and $p(x:r)$, is used to express the relationship between the semantics of $r(y)$ and $p(x:r)$. The formula states that if $r(y)$ multiplies y by v, then $p(x:r)$ multiplies x by v^2. In order to prove relative completeness, we must show that the logic can express all of the necessary relations of this sort. We will return to this problem and make it more precise later.

Another problem related to expressiveness is the question of when we can assume that the strongest postcondition of a statement is expressible in the first order assertion language. Roughly speaking, most relative completeness proofs proceed by showing the following is provable in the axiom system:

$$(2) \qquad \vdash \{U\}\ A\ \{SP(A,U)\}$$

for any statement A and precondition U. From $\vdash \{U\}\ A\ \{SP(A,U)\}$ and rule of consequence one can prove that if $I \models \{U\}\ A\ \{V\}$, then $Th(I) \vdash \{U\}\ A\ \{V\}$ for it must be the case that if $\models U\ A\ V$, then $I \models SP(A,U) \supset V$. This chain of reasoning depends on the assumption of expressiveness, which was used implicitly in writing (2).

However, the usual notion of expressiveness is that $SP(A,U)$ can be expressed for *any program A*. By definition, a program does not have free procedure names; hence expressiveness does not immediately guarantee that one can express $SP(A,U)$ for an arbitrary statement A which may have free procedure names. Thus, our relative completeness proof cannot proceed directly by proving a lemma of the same form as (2). Roughly, if A is a

statement with free procedures, we will be able to show $\vdash H \rightarrow \{U\}\ A\ \{SP(A,U)\}$, where H is a suitably chosen set of hypothesis about the free procedures in A, and A is a program (with no free procedures), which in some sense simulates A. We proceed by using some of the properties of the Herbrand definable interpretations.

Lemma 2 [8]. If I is an interpretation which is Herbrand definable and the programming language is L4 (or more generally, any "acceptable programming language with recursion" in the sense of [3]) then either

1. I is finite or
2. there are programs in the language which simulate arithmetic in dom (I).

One can use this fact about Herbrand definable domains to prove the existence of *interpreter programs*. Roughly speaking, an interpreter program receives as inputs a number of ordinary variables containing an encoding of an arbitrary relation to be computed, and a number of other variables to which the relation is to be applied. The interpreter then modifies the second set of variables according to the relation. Using interpreter programs, we can transform any L4 program into a program without procedures passed as parameters by adding additional ordinary variables to pass values which encode the procedures. Specifically, one can show that for any statement A in L4, there is another statement A^* having the following properties. In place of each formal procedure name r free in A, A^* has a new group of free ordinary variables, r^*. The r^* variables are distinct from all other variables. If A is a statement whose only free procedure names are the formals $r_1, \ldots, r_n$, then the relational semantics of A in an environment where r_i is bound to B_i is the same as the semantics of A^* provided r_i^* is initially set to the encoding of the relation corresponding to procedure r_i. For a program A, $\mathcal{M}(A) \equiv \mathcal{M}(A^*)$.

As it happens, there is a way to construct A^* so that if A has no non-local use of variables, then neither does A^*. This means that if A is in L4 and the only procedures free in A are formals, then A^* is a program of L4. Consequently, if I is expressive then $SP(A^*,U)$ is expressible in I for such A.

Using A^*, we can carry out the relative completeness proof without the expressiveness problems of formula (2). For each statement A whose free procedures are the formals $r_1, \ldots, r_n$ and declared procedures $p_1, \ldots, p_k$, we can show that the following is provable in the axiom system

$$(3) \qquad \vdash \{R_1, \ldots, R_n, P_1, \ldots, P_k\} \rightarrow \{U\}\ A\ \{SP(E \mid A)^*\}$$

where E is any environment such that $(E \mid A)^*$ is a program, and $R_1, \ldots, R_n$ and $P_1, \ldots, P_k$ are formulas of the logic which describe $r_1, \ldots, r_n$ and $p_1, \ldots, p_k$, respectively. Intuitively, R_i has the form $R_i(r_i, r_i^*)$, and says that the semantics of r_i is a subset of the relation encoded by r_i^*. For r_i of type $r_i(\bar{x})$, R_i is just $\{\bar{x} = \overline{x0}\}r_i(\bar{x})\ \{SP(r_i(\bar{x})^*, \bar{x} = \overline{x0}\}$. For higher types, a more complex formula is defined by induction; e. g., for $r_j(\bar{x}: r_i)$ where r_i has type $r_i(\bar{x})$, R_j is

$$\forall r_i\, r_i^*(R_i(r_i, r_i^*) \rightarrow \{\bar{x} = \overline{x0}\}\ r_j\ (\bar{x}: r_i)\ \{SP(r_j(\bar{x}: r_i)^*, \bar{x} = \overline{x0})\})$$

Similarly, P_i is a formula which says that the semantics of p_i is a subset of the semantics determined by the

environment E and the relations for $r_1, \ldots, r_n$ encoded by $r_1^*, \ldots, r_n^*$. The formulas R_i and P_i give the general representation in our logic of the meaning of procedures, as alluded to earlier.

In the full paper, we show that (3) is provable, by induction on the structure of statements. For a program A, (3) gives

$$\vdash \{U\}\ A\ \{SP(A^*,U)\}$$

from which the desired result follows because

$$I \models SP(A^*,U) \equiv SP(A,U).$$

Hence, if $I \models \{U\}\ A\ \{V\}$, then $Th(I) \vdash \{U\}\ A\ \{V\}$.

8. Conclusion

We have presented a sound and relatively complete axiom system for the language L4. Such an axiom system has been sought by a number of other researchers since the appearance of [CL79]. But because of the infinite range problem, no completely satisfactory axiomatization has been previously given.

In order to deal with infinite range, we introduce a class of generalized partial correctness assertions, which permit implication between partial correctness assertions, universal quantification over procedure names, and universal quantification over environment variables. These assertions enable us to relate the semantics of a procedure with the semantics of procedures passed to it as parameters. By using these assertions we are able to provide a new principle for reasoning about procedures with procedure parameters; this principle is incorporated in our recursion rule.

Many of the techniques introduced in this paper appear to have application beyond L4. We believe that the ideas used in our recursion rule may be helpful with other languages which have infinite range [1]. Moreover, the way that we have structured the inductive argument in the relative completeness proof is new and may also be useful in this respect. Finally, in the course of the relative completeness proof we have derived some new results of independent interest about the power of acceptable programming languages and the existence of expressive interpretations.

9. Acknowledgment

We want to thank Magdalena Muller for her infinite patience in typing this document.

10. References

1. de Bakker, J. W., Klop, J. W., Meyer, J.-J. Ch. Correctness of programs with function procedures. Tech. Rept. IW 170/81, Mathematisch Centrum, Amsterdam, 1981.

2. Clarke, E. M. "Programming language constructs for which it is impossible to obtain good Hoare-like axioms." *JACM 26* (1979), 129-147.

3. Clarke, E. M., Jr., German, S., and Halpern, J. Y. "Effective axiomatization of Hoare logics." *JACM 30* (1983), 612-636.

4. Cook, S. A. "Soundness and completeness of an axiom system for program verification." *SIAM J. Comput. 7* (1978), 70-90.

5. Damm, W. and Josko, B. A sound and relatively complete Hoare-logic for a language with higher type procedures. Tech. Rept. Bericht No. 77, Lehrstuhl fur Informatik II, RWTH Aachen, April, 1982.

6. Damm, W. and Josko, B. personal communication.

7. German, S. Relative completeness proofs for languages with infinite range.

8. German, S. and Halpern, J. On the power of acceptable programming languages with recursion.

9. Olderog, E.-R. "Sound and complete Hoare-like calculi based on copy rules." *Acta Informatica 16* (1981), 161-197.

10. Olderog, E.-R. "Hoare-style proof and formal computations." *Jahrestagugn, IFB 50 GI-11* (1981), 65-71.