A GOOD HOARE AXIOM SYSTEM FOR AN ALGOL-LIKE LANGUAGE

Joseph Y. Halpern
IBM Research Laboratory
San Jose, California 95193

Abstract: Clarke has shown that it is impossible to obtain a relatively complete axiomatization of a block-structured programming language if it has features such as static scope, recursive procedure calls with procedure parameters, and global variables, provided that we take first-order logic as the underlying assertion language [Cl]. We show that if we take a more powerful assertion language, and hence a more powerful notion of expressiveness, such a complete axiomatization is possible. The crucial point is that we need to be able to express weakest preconditions of commands with free procedure parameters. The axioms presented here are natural and reflect the syntax of the programming language. Such an axiom system provides a tool for understanding how to reason about languages with powerful control features.

1. INTRODUCTION

In a paper entitled "Programming languages for which it is impossible to obtain good Hoare axiom systems" [Cl], Clarke showed that it is impossible to obtain a "good" Hoare axiom system for a block-structured programming language with the following features:

(i) procedure names as parameters of procedure calls,
(ii) recursion,
(iii) static scope,
(iv) global variables, and
(v) internal procedures.

In this paper, we give a "good" Hoare axiom system for PROG 83, a large subset of PROG, the ALGOL-like language introduced and studied in [THM1,THM2]. PROG 83 allows nondeterminism and sharing of variables (aliasing) and has all five features mentioned above.

To explain this paradox, we must look a little more carefully at the meaning of "good". Of course, we expect a good Hoare axiom system to be sound (everything which is provable should be true) and complete (every partial correctness assertion which is true should be provable). It should also be "natural"; i.e., the axioms should in some sense reflect the syntax of the programming language.

From the Gödel incompleteness theorem, we know that for sufficiently rich interpretations, such as arithmetic, it is impossible to obtain a sound and complete axiomatization for first-order formulas, let alone for partial correctness assertions involving first-order formulas. In order to talk about the completeness of a Hoare axiom system independent of the underlying interpretation, Cook [Co] proposed the notion of *relative completeness*: under the assumption that for a given interpretation I and assertion language $\mathcal{L}$, we are given an oracle for the formulas of $\mathcal{L}$ which are true in I, and that $\mathcal{L}$ is *expressive* for I, then every true partial correctness assertion is provable. *Expressiveness* means that for every program g in the programming language and formula P in $\mathcal{L}$, there is a formula in $\mathcal{L}$ equivalent to the *weakest precondition* (cf. [Di]) of g with respect to P.

Taking $\mathcal{L}$ to be first-order logic, Cook gave a relatively complete axiomatization of a subset of ALGOL with a while-statement and nonrecursive programs. Gorelick [Go] extended Cook's work to include recursive procedures. On the other hand, Clarke [Cl] showed that if we take a language with the five features mentioned above, no relative completeness proof is possible *provided we take first-order logic as the underlying assertion language.*

The work of papers such as [GCH,THM,DJ] shows that in order to reason about programs in languages with rich control structures, it is also necessary to be able to reason

about commands with possibly free procedure identifiers. In order to facilitate reasoning about such commands, we will require that our assertion language be powerful enough to express the weakest precondition of g with respect to P even if g has some occurrences of free procedure identifiers. Thus, our assertion language is a higher order one, in much the same spirit as that of [DJ]. We present a natural axiomatization of PROG 83 which is sound and, under our stronger expressiveness hypothesis, relatively complete.

The axiom system presented here draws heavily on those of [GCH] and [THM1]. From [GCH] we get the style of the axiom system, which is presented in terms of (possibly nested and universally quantified) sequents of partial correctness assertions, and the recursion rule. The completeness result presented here seems to be convincing evidence that this rule is indeed the "right" way to reason about recursion. Following [THM1], we distinguish locations and their contents, allowing us to deal with aliasing in a clean way. The assignment axiom is taken directly from [THM1], as well as techniques for reasoning about invariance.

This last point deserves some further discussion. In papers such as [Cl, Go, Ol], one sees a variant of the following axiom:

P{g}P, provided the free variables of P are disjoint from those of g.

Intuitively, this is sound because a program can only affect the values of its free variables. Since the truth of P only depends on the values of its free variables, if P is true before g is run, then it will still be true afterwards. However, the statement, "a program g can only affect the values of its free variables," no longer holds if we allow free procedure identifiers with global variables. For example, if q is the parameterless procedure x:=3 and cont(x) denotes the value stored at location x, then

$$cont(x)=2\{q\}cont(x)=2$$

is clearly not valid.

In order to deal with this problem, we introduce a new class of assertions called *covering assertions*. In our notation, the partial correctness assertion above becomes:

$$cov(q,\{y_1, ..., y_k\}) \rightarrow (x\neq y_1 \wedge ... \wedge x\neq y_k \wedge cont(x)=2)\{q\}cont(x)=2,$$

i.e., if q is covered by the locations $y_1,...,y_k$ (roughly speaking, if q "reads" and "writes" at most these locations) and x is distinct from $y_1,...,y_k$, then if the contents of location x is 2 before we run q, then it will be 2 afterwards. Covering assertions provide a general technique for dealing with global variables; moreover, no special axioms are required for them.

The rest of this paper is organized as follows. In Section 2, we give the syntax and semantics of PROG 83, while in Section 3, we give the syntax and semantics of the assertion language, covering assertions, and partial correctness formulas. In Section 4, we present the axiom system, and prove its soundness and completeness. We conclude in Section 5 with a discussion of further applications of these results.

2. SYNTAX AND SEMANTICS OF PROG 83

To illustrate our axiom system, we use the programming language PROG 83, a subset of the language PROG described in greater detail in [THM1,THM2]. PROG 83 is a fully-typed, block-structured programming language, with a number of non-trivial features including nondeterminism, shared variables, nondeterminism, and procedure parameters nested to arbitrary depth. We have omitted a number of features found in PROG, including lambda abstraction and higher order declarations. Moreover, procedures in PROG 83 can only take identifiers as parameters, rather than arbitrary expressions of the right type. Although it is straightforward to give such features semantics in the framework developed in [THM1, THM2], axiomatizing them seems a bit more complicated. We hope to axiomatize more features of PROG in future work.

The primitive types of PROG 83 are int, loc, prog, intexp, and locexp. A *store* is a mapping from locations to their values. The types int and loc are intended to be the domains of storable values and the locations in which these values are stored. The domain prog is that of program meanings: nondeterministic mappings from stores to sets of stores. Elements of type intexp and locexp are expressions which evaluate to values and locations respectively in a given store, i.e. functions from stores to int (resp. loc) (in ALGOL jargon, these are "thunks"). We call loc and int *basic types*. For ease of exposition, the only tests we allow are equality tests between expressions of basic type. *Procedure types* are defined inductively to be of the form $\alpha_1 \rightarrow \cdots \rightarrow \alpha_k \rightarrow$ prog, where $\alpha_1,...,\alpha_k$ are either of procedure type or of basic type. As in ALGOL 68, starting with variables of basic type, we can form more complicated location and integer expressions. We use the notation x^α to indicate that variable x has type α.

We briefly review the syntax of PROG 83, and refer the reader to [THM1,THM2] for more details.

Integer expressions:

IntE::= a (where a is a constant symbol) | x^{int} | cont(LocE) | f(IntE$_1$, ..., IntE$_k$) (where f has type int^k→int) | if BoolE then IntE$_1$ else IntE$_2$ fi.

Location expressions:

LocE::= x^{loc} | if BoolE then LocE$_1$ else LocE$_2$ fi.

Boolean expressions:

BoolE::=IntE$_1$=IntE$_2$ | LocE$_1$=LocE$_2$.

Procedure expressions:

ProcE$^\alpha$::= p^α | ProcE$_1^{\beta\to\alpha}$ProcE$_2^\beta$ | ProcE$^{\gamma\to\alpha}x^\gamma$ (where α, β are procedure types, γ is a basic type).

ProcE=$\cup_\alpha$PE$^\alpha$.

Calls:

Call::= ProcE^{prog} (a procedure expression of type prog).

Commands:

Com::= Call | diverge | LocE:=IntE | Com$_1$;Com$_2$ | Com$_1$ or Com$_2$ | BBl | PBl | if BoolE then Com$_1$ else Com$_2$ fi.

Basic Blocks:

BBl::= let x^{int}⇐IntE in Cmd tel | new x^{loc} in Cmd wen | let x^{loc}⇐LocE in Cmd tel.

Procedure Blocks:

PBl::= proc PDecl do Cmd end.

Procedure Declarations:

PDecl::=$p_1x_{11}...x_{1n_1}$⇐Com$_1$, ..., $p_mx_{m1}...x_{mn_m}$⇐Com$_m$, where $p_i,x_{i1},...,x_{in_i}$ are distinct for i = 1,...,m, p_i has procedure type α_i, and x_{ij} has type α_{ij}, where $\alpha_i = \alpha_{i1}\to\alpha_{i2}\to\cdots\to\alpha_{in_i}\to$prog, i = 1,...,m. We say p_i is *declared* in this finite system of *mutual procedure declarations* with *formal parameters* $x_{i1},...,x_{in_i}$ and *declaration body* Com$_i$.

Notation: We often use the letter E (possibly primed or subscripted) to represent a finite system of mutual procedure declarations. Procedure blocks of the form proc E do Com end will usually be abbreviated as E|Com. For readability, we will usually write $p(x_1,...,x_k)$ rather than $px_1...x_k$ in procedure calls and on the left side of the ⇐ in procedure declarations.

Note that PROG 83 allows recursion in procedure declarations, procedures of arbitrarily high finite type, unrestricted procedure nesting, and arbitrarily complex calls. Explicit sharing is possible by use of the declaration x^{loc} ⇐ LocE in a basic block.

Commands in PROG 83 may have free procedure identifiers. A *program* is a command without free procedure identifiers.

We give semantics to expressions of PROG 83 by mapping them into elements of an *algebraic store model* D. An algebraic store model D consist of a collection of partial orders $\{D_\alpha\}$, one for each type α, such that each element of $D_{\alpha\to\beta}$ is a monotonic function from $D_\alpha\to D_\beta$. There is a least element $\perp_\alpha \in D_\alpha$, and for each function $f \in D_{\alpha\to\alpha}$, the sequence $\perp_\alpha$, $f(\perp_\alpha)$, $f(f(\perp_\alpha))$, ..., has a least upper bound in D_α. Finally, for each $d\in D_\alpha$, there is a finite set of locations which *covers* d. The properties of store models, the method of assigning semantics, and the precise definition of the covering relation is given in detail in [THM2]. We give a brief sketch here.

We have a meaning function $\mathcal{M}$ such that for each constant c of type α, $\mathcal{M}(c) \in D_\alpha$. An *environment* e gives meaning to the variables: if x is a variable of type α, $e(x)\in D_\alpha$. To every PROG 83 expression v of type α there corresponds an element $\mathcal{M}(v)e\in D_\alpha$ which is the meaning of v in environment e.

With each element of D_α, we can define what it means for a set of locations $L \subset D_{loc}$ to *cover* it. Some important properties of the covering relation include:

1. If $p \in D_{prog}$, then p is covered by L iff (cf. [MM]):
   (a) for any store s, for all $s'\in ps$, s and s' agree off L; i.e., for all $h\notin L$, $s'(h)=s(h)$. Thus, p does not change the contents of locations not in L.
   (b) if s and s' agree on L, then ps and ps' agree on L.
2. If $h \in D_{loc}$, then h is covered by L iff $h \in L$.
3. If $d \in D_{int}$, then d is covered by L for all sets L.
4. If p is covered by L and q is covered by L', then p(q) is covered by L∪L'.

Crucial use is made of the covering relation when defining the semantics of the new declaration. Roughly speaking, to run new x in g wen in environment e and store s, we proceed as follows. We first find a set L which covers g (or, more accurately, $\mathcal{M}(g)e$), choose a location h not in L, run g in environment e' and store s' which are identical to e

and s except that x is set to h with its contents initialized to a_0, and then reset the contents of h after the computation of g has ended. In this way we maintain the stack discipline and use a truly "new" location for x. Again, we refer the reader to [THM2] for more details.

As a consequence of the semantics, we get the following proposition, which shows that every command is equivalent to one in which the procedure declarations are "pushed in" so that they only occur in front of procedure calls. This proposition will be useful in our axiomatization, by enabling us to restrict attention to commands in this special form.

Proposition 1 (cf. [THM1,GCH]): The following equivalences hold:

(a) $E|(g_1;g_2) \equiv (E|g_1);(E|g_2)$,

(b) $E|(g_1 \text{ or } g_2) \equiv (E|g_1) \text{ or } (E|g_2)$,

(c) E|(if BoolE then g_1 else g_2 fi) ≡
if BoolE then $E|g_1$ else $E|g_2$ fi,

(d) If x does not appear free in E, then
E|(new x in g wen) ≡
new x in E|g wen,

(e) If x is of basic type, BasE is a basic expression of the same type, and x does not appear free in E, then
E|(let x ⇜ BasE in g tel) ≡
let x ⇜ BasE in E|g tel,

(f) If E_1 and E_2 do not contain distinct declaration for the same procedure identifier, then
$E_1|(E_2|g) \equiv ((E_1 \cup E_2)|g)$,

(g) If none of the procedures declared in E appears free in g, then
$E|g \equiv g$,

(h) If g and g′ are identical up to renaming of bound variables, then
$g \equiv g'$.

From Proposition 1 we immediately get the following

Corollary: Every command g is equivalent to a command g′ in a normal form, where the subcommand E|h occurs in g′ only if h is a procedure call.

3. THE ASSERTION LANGUAGE AND PARTIAL CORRECTNESS FORMULAS

To permit as much generality as possible, we do not describe the assertion language $\mathscr{L}$ in detail here, but state some abstract properties it must satisfy:

(a) $\mathscr{L}$ is many-sorted. Among its sorts are int and loc. To every integer (respectively, location) expression BasE in PROG 83, there is a corresponding term BasE^t in $\mathscr{L}$ with the same meaning (we shall omit the superscript t when it is clear from context). The assertions $x=\perp_{int}$ and $y=\perp_{loc}$ are definable in $\mathscr{L}$.

(b) Formulas in $\mathscr{L}$ are closed under the first-order connectives $\neg$, $\wedge$, and $\forall$, which are defined in the usual way. Truth for formulas of $\mathscr{L}$ is defined relative to a model D, an environment e, and a store s. For a formula P of $\mathscr{L}$, we write $D,e,s \models P$ if P is true with respect to D,e,s and $D,e \models P$ iff for all stores s, $D,e,s \models P$.

(c) A formula has a certain set of free variables, and the truth of a formula depends only on the values given to its free variables. More precisely, if the free variables of P are among $x_1,\ldots,x_m$, and $e(x_i)=e'(x_i)$, $i=1,\ldots,m$, then
$D,e,s \models P$ iff $D,e',s \models P$

(d) If P is a formula in $\mathscr{L}$, IntE is an integer expression, and LocE is a location expression, then we can effectively find a formula [LocE ← IntE]P such that
$D,e,s \models [\text{LocE} \leftarrow \text{IntE}]P$ iff $D,e,s[\text{LocE}/\text{IntE}] \models P$,
where s[LocE/IntE] is identical to s except that the value it assigns to (the meaning in D,e,s of) location expression LocE is (the meaning in D,e,s of) integer expression IntE.

The first-order language defined in [THM] has all the properties of (a)-(d) above.

The model D is said to be *strongly expressive* if for all commands g (even those with free procedure parameters), and formulas $P \in \mathscr{L}$ there is a formula $WP(g,P) \in \mathscr{L}$, called the *weakest precondition* of P with respect to g, such that $D,e,s \models WP(g,P)$ iff for all $s' \in \mathscr{M}(g)es$ we have $D,e,s' \models P$.

Remark: Note that *Dynamic Logic* (cf. [Ha]) is an example of an assertion language for which all models are strongly expressive in this sense, since, by definition, the Dynamic Logic formula [g]P is the weakest precondition of g with respect to P. In Dynamic Logic, we can find weakest preconditions effectively, but we do *not* require this for our results.

A *partial correctness assertion* is a triple P{g}Q, where P and Q are formulas of $\mathscr{L}$ and g is a command. We define

D,e,s $\models$ P{g}Q iff

(D,e,s $\models$ P implies $\forall s' \in \mathscr{M}(g)es$, D,e,s' $\models$ Q),

and

D,e $\models$ P{g}Q iff $\forall s(D,e,s \models P\{g\}Q)$.

(Note our definition of D,e $\models$ P{g} is slightly different from that of [THM1] in that with our formalism we do not need the notion of *matching* environments; i.e. environments which agree in the meanings that they give to all procedure identifiers.)

In order to deal with global variables we use *covering assertions*, which have the form cov(p,X), where p is a variable of procedure type or of basic type, and $X=\{x_1,...,x_k\}$ is a finite set of location variables. For covering assertions we define:

D,e $\models$ cov(p,X) iff e(p) is covered by $\{e(x_1),...,e(x_k)\}$.

From the properties of the covering relation it follows that if y has type int, then cov(y,X)≡true, while if z has type loc, then $cov(z,\{x_1,...,x_k\}) \equiv z=x_1 \vee ... \vee z=x_k$.

The set of *partial correctness formulas* is defined inductively as follows (cf. [GCH]):

(a) If H is a formula of $\mathscr{L}$, or a covering assertion, or a partial correctness assertion, then H is a partial correctness formula.

(b) If $H_1,...,H_n$ are partial correctness formulas, then so is $\{H_1,...,H_n\}$.

(c) If H_1, H_2 are partial correctness formulas, then so is $H_1 \rightarrow H_2$.

(d) If H is a partial correctness formula and x^α is a variable (of any type α), then $\forall x^\alpha H$ is a partial correctness formula.

We define the truth of a partial correctness formula relative to D,e. We have already done this for formulas of $\mathscr{L}$, covering assertions and partial correctness assertions. For the other types of partial correctness formulas, we define:

D,e $\models \{H_1,...,H_n\}$ iff D,e $\models H_i$, i=1,...,n,

D,e $\models H_1 \rightarrow H_2$ iff (D,e $\models H_1$ implies D,e $\models H_2$),

D,e $\models \forall x^\alpha H$ iff D,e[d/x] $\models$ H for all $d \in D_\alpha$.

A formula H is *valid* in D, written D $\models$ H, iff for all environments e, D,e $\models$ H.

If g is a command with free identifiers $p_1,...,p_k$ and X is a finite set of location identifiers, we take cov(g,X) to be an abbreviation for the partial correctness formula:

$\{cov(p_1,X),...,cov(p_k,X)\}$.

Note that if g has no free procedure identifiers and all its free location identifiers are contained in X then cov(g,X) is vacuously true.

Before we can give our axiomatization of PROG 83, we shall need several more definitions.

If E is a set of procedure declarations, we will say that the variable q is *reachable from p via E* iff q=p or, inductively, if p is declared in E with declaration body Com and q is reachable from r via E for some free variable r in Com. For example, if E consists of the procedure declarations

p ⇐ if w=z then q else p(r)

q ⇐ x:=y,

then p, q, r, w, z, x, and y are reachable from p via E, while q, x, and y are reachable from q via E, and only r is reachable from r via E. If X is some finite set of location variables, we define cov(E|p,X) to be an abbreviation for

{cov(q,X) | q is reachable from p via E and q is not declared in E}

Thus cov(E|p,X) holds exactly if p is covered by X when it is bound to its declaration body in E.

If H is a partial correctness formula such that no procedure declared in E occurs free in any of the subformulas of H which are formulas of $\mathscr{L}$, then we define E|H to be an abbreviation for the formula which results when we replace each subformula of H of the form cov(p,X) by cov(E|p,X), and replace each partial correctness formula of the form P{g}Q by P{E|g}Q (subject to the usual provisos about renaming any bound variables in H so that they have names distinct from any variables free in E, and renaming the declared variables in E so that they have names distinct from the free variables in H, in order to avoid capture of free variables). E|H is undefined if some $\mathscr{L}$-subformula of H has a free occurrence of a variable declared in E.

Roughly speaking, E|H is the result of binding the meanings of free variables in H which are declared in E to their declarations in E. Note that we can find E|H effectively given E and H. If the assertion language $\mathscr{L}$ were Dynamic Logic, we could also effectively transform any formula P in $\mathscr{L}$ to a formula E|P, again binding procedures to their declarations in E, but for arbitrary $\mathscr{L}$ this in general cannot

be done. For this reason we have not allowed procedures variables declared in E to appear free in $\mathscr{L}$-subformulas of H.

If P is a formula of $\mathscr{L}$, we define [LocE:=IntE]P to be an abbreviation for

$$[LocE \leftarrow IntE]P \vee LocE = \perp_{loc} \vee IntE = \perp_{int}.$$

As shown in [THM1], [LocE:=IntE]P is the weakest precondition of P with respect to the assignment LocE:=IntE.

Finally, let w^{int} be a variable which does not appear in P, and let x^{loc} be any location variable. Then we define

$$Inv(P,x) =_{def} (P \equiv \forall w([x \leftarrow w]P)).$$

Inv(P,x) says that P is invariant under changes to the contents of x. If X is a finite set of location variables, let

$$Inv(P,X) =_{def} \wedge_{x \in X} Inv(P,x).$$

Inv(P,X) holds exactly if P is invariant under changes to the contents of all the location variables in X.

4. A SOUND AND RELATIVELY COMPLETE AXIOMATIZATION OF PROG 83

Consider the following collection of axioms and rules of inference for partial correctness formulas. Axioms 1-7 are variants of the corresponding axioms in [THM1], expressed in our formalism, and successfully capture the semantics of all the constructs in PROG 83 other that procedure declarations. Rule 8 is the recursion rule of [GCH], reformulated here to allow us to deal with global variables. Axiom 9 allows us to reduce consideration to commands in normal form when doing the completeness proof. Axioms and rules 10-14 are again quite standard and are used to prove arbitrary valid partial correctness formulas once we have the "most general" partial correctness formula. This technique goes back to [Go], and has also been used in many other papers, (eg. [THM1,Cl,Ol,GCH,...]). The remaining rules and axioms simply allow us to manipulate partial correctness formulas.

1. **Assignment axiom:**

   $$([LocE:=IntE]P)\{LocE:=IntE\}P.$$

2. **Axiom of divergence:**

   $$true\{diverge\}false.$$

3. **Sequencing axiom:**

   $$\{P\{g\}Q,\ Q\{g'\}R\} \rightarrow P\{g;g'\}R.$$

4. **Choice axiom:**

   $$\{P\{g\}Q,\ P\{g'\}Q\} \rightarrow P\{g \text{ or } g'\}Q.$$

5. **Conditional axiom:**

   $$\{(P \wedge BoolE)\{g_1\}Q,\ (P \wedge \neg BoolE)\{g_2\}Q\} \rightarrow P\{\text{if BoolE then } g_1 \text{ else } g_2 \text{ fi}\}Q.$$

6. **Axiom of let declarations:** If x is a variable of basic type, BasE is an expression of the same type, and y a variable of the same type which is not free in P, Q, BasE, or g, then

   $$(P \wedge y = BasE \wedge y \neq \perp)\{[y/x]g\}Q \rightarrow P\{\text{let } x \Leftarrow BasE \text{ in } g \text{ tel}\}Q.$$

7. **Axiom of new declaration:** Let y^{loc} and z^{int} be variables not free in P, Q, or g, and let X be a finite set of location variables. Recall that a_0 is the constant used to initialize the new location in a new declaration. Then

   $$\{\wedge_{x \in X} x \neq y,\ cov(g,X),\ ([y:=z]P \wedge cont(y)=a_0)\{[y/x]g\}[y:=z]Q\} \rightarrow P\{\text{new x in g wen}\}Q.$$

8. **Recursion rule:** Suppose E is a set of procedure declarations of the form

   $$p_1(r_{11},\ldots,r_{1k_1}) \Leftarrow body_1, \ldots, p_n(r_{n1},\ldots,r_{nk_n}) \Leftarrow body_n.$$

   Suppose $p_1,\ldots,p_n$ do not appear free in H_i, P_i, or Q_i, i = 1,...,n, and only appear free in subformulas of H which are covering formulas. Then

   $$\frac{H \rightarrow [\{\forall r_{i1},\ldots,r_{ik_i}(H_i \rightarrow P_i\{p_i(r_{i1},\ldots,r_{ik_i})\}Q_i),\ i = 1,\ldots,n\} \rightarrow \{\forall r_{i1},\ldots,r_{ik_i}(H_i \rightarrow P_i\{body_i\}Q_i),\ i = 1,\ldots,n\}]}{E|H \rightarrow [\forall r_{i1},\ldots,r_{ik_i}(H_i \rightarrow P_i|E|p_i(r_{i1},\ldots,r_{ik_i})\}Q_i),\ i = 1,\ldots,n\}]}.$$

   Suppose we are given some set of procedure declarations E. Intuitively, the recursion rule says that if, whenever we can prove from some hypothesis H_i something about each of the calls p_i in E for all possible values of their parameters, we can also prove the same thing about the corresponding $body_i$ of p_i (for all possible values of the parameters, and from the same hypothesis H_i), then we can conclude the partial correctness assertion holds of p_i when it is declared in the environment E (again for all values of the parameters and under hypothesis H_i).

9. **Normal form axiom:** If g and g′ are provably equivalent using the equivalences of Proposition 1, then

   $$P\{g\}Q \rightarrow P\{g'\}Q.$$

10. **Axiom of consequence:**

    $$\{P' \supset P,\ P\{g\}Q,\ Q \supset Q'\} \rightarrow P'\{g\}Q'.$$

11. **Axiom of conjunction:**

    $$\{P\{g\}Q,\ P'\{g\}Q'\} \rightarrow (P \wedge P')\{g\}(Q \wedge Q').$$

12. **Invariance axiom:** Let X be a finite set of location variables. Then

$$\{cov(g,X), Inv(P,X)\} \rightarrow P\{g\}P.$$

Intuitively, the invariance axiom says that if g is covered by X and P is a formula whose truth is independent of the contents of these locations, then if P is true before we run g then P will be true afterwards.

13. **Substitution rule:** Let σ be any mapping on variables which respects types (i.e., $\sigma(x)$ has the same type as x). Let $H\sigma$ be the result of replacing x by $\sigma(x)$ wherever x occurs in H. Then

$$\frac{H}{H\sigma}.$$

Note we allow arbitrary substitutions here, not just injections. This will enable us to deal with sharing.

14. **Rule of quantification introduction:** If x does not appear free in g or H, then

$$\frac{H \rightarrow P\{g\}Q}{H \rightarrow \exists xP\{g\}\exists xQ}.$$

Note that the H appears here as a hypothesis in both the antecedent and the conclusion of this rule. This gives us a more powerful rule than the corresponding rule without the H, since we can apply the rule relative to the hypothesis H. A similar phenomenon occurs in the recursion rule (rule 8 above) and in rules 15 and 22 below.

15. **Rule of universal quantification:** If x does not appear free in H_1, then

$$\frac{H_1 \rightarrow H_2}{H_1 \rightarrow \forall xH_2}.$$

16. **Rule of declaration binding:** If none of the procedures declared in E appear free in $\mathscr{L}$-subformulas of H, then

$$\frac{H}{E|H}.$$

17. **Implication axiom:** If P and Q are formulas of $\mathscr{L}$, then

$$(P \supset Q) \rightarrow (P \rightarrow Q).$$

18. **Instantiation axiom:**

$$\forall rH \rightarrow H.$$

19. **Axiom of transitivity:**

$$\{H_1 \rightarrow H_2, H_2 \rightarrow H_3\} \rightarrow (H_1 \rightarrow H_3).$$

20. **Modus ponens:**

$$\frac{H_1, H_1 \rightarrow H_2}{H_2}.$$

21. **Axiom of trivial implication:**

$$\{H_1, ..., H_n\} \rightarrow H_i, \ i = 1, ... n.$$

22. **Set formation rule:** For all $n \geq 0$

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

Note that $H \rightarrow \phi$ is a special case of this rule (taking n=0).

23. **Empty set introduction axiom:**

(a) $H \rightarrow (\phi \rightarrow H)$,

(b) $(\phi \rightarrow H) \rightarrow H$.

24. **Currying axiom:**

$$(\{H_1, H_2\} \rightarrow H_3) \rightarrow (H_1 \rightarrow (H_2 \rightarrow H_3)).$$

Theorem 1: The axiom system presented above is *sound*; i.e. for any model D and any environment e, $D,e \models H$ for every axiom scheme H above, and for every rule of inference with antecedents $H_1,...,H_k$, and conclusion H, if $D,e \models H_i$, $i=1,...,k$, then $D,e \models H$.

Proof: The soundness of axioms and rules 1-7 and 9-13 follows from the soundness of the corresponding axioms and rules of [THM1]. We refer the reader there for details. The recursion rule is just a reformulation of the well-known *fixed-point induction rule*, and unlike the recursion rule of [THM1] or [Ol], its soundness follows immediately from the denotational (fixed-point) semantics, and does not require copy-rule semantics (and thus a tedious proof of the equivalence of fixed-point and copy-rule semantics; cf. [THM1,THM2]). The soundness of the remaining axioms and rules is almost an immediate consequence of the definitions. We leave details to the full paper. □

Let $Th^*(D)$ consist of all the partial correctness formulas of the form $cov(g,X) \rightarrow P$ which are valid in D, where P is a formula of $\mathscr{L}$.

We can now state our relative completeness theorem.

Theorem 2: If D is strongly expressive, then the axiom system above is complete relative to $Th^*(D)$ for partial correctness assertions involving programs; i.e., if g is a program (with no free procedure identifiers) and $D \models P\{g\}Q$, then $Th^*(D) \vdash P\{g\}Q$.

In order to prove the completeness theorem, we need to develop an analogue for the "most general partial correctness

formula" of [Go, Ol]. Let X be the set of location variables $\{x_1,...,x_k\}$, and Y be the set of integer variables $\{y_1,...,y_k\}$. Let U(X,Y) be the formula

$$cont(x_1)=y_1 \wedge ... \wedge cont(x_k)=y_k.$$

For each variable of basic or procedure type p and set of procedure declarations E, we define the *most general partial correctness formula of p with respect to U(X,Y) and E*, written mg(p,U(X,Y),E), by induction on the type of p as follows:

(a) if p is of basic type, then $mg(p,U(X,Y),E)) =_{def} cov(p,X)$

(b) if p has type $\alpha_1 \to ... \to \alpha_m \to prog$, $m \geq 0$, let $r_1,...,r_m$ be variables of type $\alpha_1,...,\alpha_m$, respectively. Then

$$mg(p,U(X,Y),E) =_{def}$$
$$\forall r_1,...,r_m[\{mg(r_1,U(X,Y),E),...,mg(r_m,U(x,y),E)\}$$
$$\to (\{cov(p,X),cov(E|p,X)\}$$
$$\to WP(E|p(r_1,...,r_m),U(X,Y))\{p(r_1,...,r_m)\}U(X,Y))].$$

Roughly speaking, mg(p,U(X,y),E) says that p "acts right" provided that that it is covered by the locations in X and all of its parameters (if any) act right (where "acting right" in the case of procedure identifiers means that

$$WP(E|p(r_1,...,r_m),U(X,Y))\{p(r_1,...,r_m)\}U(X,Y)$$

is true).

If g is a command with free identifiers $p_1,...,p_n$, we take mg(g,U(X,Y),E) to be an abbreviation for the partial correctness formula

$$\{mg(p_1,U(X,Y),E),...,mg(p_n,U(X,Y),E)\}.$$

The core of the relative completeness proof consists of the following two lemmas.

Lemma 1: If $cov(h,X) \to P\{h\}Q$ is valid, then

$$Th^*(D) \vdash \{cov(h,X),cov(g,X),WP(h,U(X,Y))\{g\}U(X,Y)\}$$
$$\to P\{g\}Q.$$

Lemma 2: If $X = \{x_1,...,x_k\}$ is a set of location variables, and $Y = \{y_1,...,y_k\}$ is a set of integer variables which do not appear in g, then

$$\vdash mg(g,U(X,Y)),E) \to [\{cov(g,X),cov(E|g,X)\}$$
$$\to \forall y_1,...,y_k(WP(E|g,U(X,Y))\{g\}U(X,Y))].$$

Lemma 1 is just a variant of the observation that any valid partial correctness assertion about g can be obtained from the most general partial correctness assertion about g (cf. [Go,Cl,Ol]). Lemma 2, which is proved by a straightforward induction on the structure of commands, intuitively says that once we know the most general partial correctness formulas for all the free variables in g then we can prove the most general partial correctness formula for g. We leave details of the proofs of Lemmas 1 and 2 to the full paper.

With these Lemmas in hand, the remainder of the proof of Theorem 1 is straightforward. Note that if g is a program (with no free procedure identifiers) whose free location identifiers are contained in X, taking E to be empty, we have

$$mg(g,U(X,Y),\phi) \equiv cov(g,X) \equiv true.$$

Thus by Lemma 2, we have

$$(*) \quad \vdash WP(g,U(X,Y))\{g\}U(X,Y).$$

And if $P\{g\}Q$ is valid, by Lemma 1 with g=h we have

$$(**) \quad Th^*(D) \vdash WP(g,U(X,Y))\{g\}U(X,Y) \to P\{g\}Q.$$

Then from (*) and (**) we can conclude, as desired,

$$Th^*(D) \vdash P\{g\}Q.$$

5. CONCLUSIONS

We have provided an axiomatization of PROG 83 which is sound and relatively complete, which for the first time deals with global variables in a natural way, by means of covering assertions. There is only one drawback to the results presented here, namely, that we require a higher-order oracle: Th*(D).

By Clarke's results we know that this is in some sense necessary: we cannot hope to obtain a complete axiomatization for a language as rich as PROG 83 by restricting ourselves to just having an oracle for the first-order theory of the interpretation, Th(D). Using a rich oracle such as Th*(D) enables us to factor out certain difficulties in order to gain insight into how to reason about programming languages which allow procedures as parameters. Nevertheless, it is worth trying to understand just how far we can get using only Th(D) as an oracle.

First note that if we restrict to the subset of PROG 83 in which procedures only take basic variables as parameters (and thus no procedure parameters), then the above axiomatization is complete for the resulting language relative to Th(D).

Several previous papers on Hoare logics have given complete axiomatizations for subsets of ALGOL-like languages consisting of programs with the *finite range property*

(cf. [Cl,Ol,LO,THM1]). (Roughly speaking, a program has the finite range property if there is a bound on the number of distinct procedure environments which can be reached when it is run.) Indeed, as Olderog has shown [Ol], the standard Hoare axiom systems based on the copy rule cannot deal with programs which do not have the finite range property. A program with the finite range property can be shown to be equivalent to a program with no procedure parameters. Using this fact, we can show that by adding one axiom to our system, we can obtain a complete axiomatization for finite range programs relative to Th(D). (This result was obtained jointly with A. Meyer and will appear in the full paper.) This observation confirms the conjecture made in [THM1] we can replace copy-rule induction (the usual rule used to reason about recursion, which is shown in [THM1] to be semantically unsound) by the semantically sound fixed-point induction rule, and still get a complete axiomatization relative to Th(D) for finite range programs.

We can also adapt our proof system to yield an axiomatization for L4 (the language with no global variables introduced in [Cl]) which is complete relative to Th(D) in *Herbrand definable* domains (domains in which every value is representable as some term in the Herbrand universe). The proof of completeness is essentially the same as that given in [GCH]. (Note that with L4 covering assertions are unnecessary, since the variables that any procedure depends on are explicit. Moreover, with our present formalism we can avoid the use of *environment variables* which seemed to be necessary in [GCH].) The crucial point in the proof of completeness is that in Herbrand definable domains, we can transform an L4 program (which may not have finite range) into a program with no procedure parameters (which clearly does).

We can see that the finite range property is the common thread running through all the proofs of completeness relative to Th(D). But now consider the problem of finding a complete axiomatization for the type of models considered in [Cr], where not only is the domain finite, but there are only finitely many distinct memory locations. For such models, Clarke's incompleteness results do not hold since they depend crucially on a recursive procedure being able to generate an unbounded number of distinct locations. But the standard Hoare axiom systems based on the copy rule cannot lead to a relative completeness result in this model because the programming language still lacks the finite range property. However, it is relatively straightforward to transform the axiom system presented here to get a axiom system which is complete relative relative to Th*(D) (we must only modify the axioms slightly to deal with the fact that there are only finitely many locations). And note that for such models, Th*(D) is decidable. We believe there will be many other situations where the style of axiomatization presented here will also lead to interesting completeness results not obtainable by other means.

There still remains the problem of finding a concrete assertion language $\mathscr{L}$ satisfying the abstract properties pointed out, Dynamic Logic is one candidate, but in DL we can effectively find the weakest preconditions. It would be worthwhile to find a possibly weaker language in which weakest preconditions exist, but cannot always be found effectively.

ACKNOWLEDGMENTS

I would like to thank my coauthors on several other papers in this area, Albert Meyer, Boris Trakhtenbrot, Steve German, and Ed Clarke, for many stimulating conversations on this material. The notion of "covered by" was worked out in conjunction with Albert and Boris, while Steve was in large part responsible for the form of the recursion rule. I would also like to thank Sylvia Fujii for her help in preparing the manuscript.

REFERENCES

[Cl] E. M. Clarke, Programming language constructs for which it is impossible to obtain good Hoare axiom systems, *JACM* 26, 129-147 (1979).

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

[Cr] F. Cristian, Correct and robust programs, IBM RJ3753, 1983; to appear in *IEEE Trans. on Software Engineering*.

[DJ] W. Damm and B. Josko, A sound and relatively complete Hoare-logic for a language with higher type procedures, Bericht No. 77, Lehrstuhl für Informatik II, RWTH, Aachen, 1982.

[Di] E. W. Diikstra, *A Discipline of Programming*, Prentice-Hall, 1976.

[GCH] S. M. German, E. M. Clarke, and J. Y. Halpern,

Reasoning about procedures as parameters, *to appear in* Proceedings of CMU Workshop on Logics on Programs, Springer-Verlag, 1983.

[Go] G. A. Gorelick, A complete axiom system for proving assertions about recursive and nonrecursive programs, TR75, University of Toronto, 1975.

[Ha] D. Harel, *First-order Dynamic Logic*, Lecture Notes in Computer Science 68, Springer-Verlag, 1979.

[LO] H. Langmaack and E. R. Olderog, Present-day Hoare-like systems for programming languages with procedures: power, limits, and most likely extensions, *in* Proceedings of 7th International Colloquium on Automata, Languages, and Programming, 1980, pp. 363-373.

[MM] A. R. Meyer and J. C. Mitchell, Axiomatic definability and completeness for recursive programs, *in* Proceedings of the Ninth Annual Symposium on Principles of Programming Languages, 1982, pp. 337-346.

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

[THM1] B. A. Trakhtenbrot, J. Y. Halpern, and A. R. Meyer, From denotational to operational and axiomatic semantics for ALGOL-like languages: an overview, *to appear in* Proceedings of the CMU Workshop on Logics of Programs, Springer-Verlag, 1983.

[THM2] B. A. Trakhtenbrot, J. Y. Halpern, and A. R. Meyer, The semantics of local storage, *to appear in* Proceedings of the Eleventh Annual ACM Symposium on Principles of Programming Languages, 1984.