CMU-CS-84-109

The Characterization Problem for Hoare Logic

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

27 February 1984

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

The Characterization Problem for Hoare Logics

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

Abstract

Research by this author and by others has shown that there are natural programming language control structures which are impossible to describe adequately by means of Hoare axioms. Specifically, we have shown that there are control structures for which it is impossible to obtain axiom systems that are sound and relatively complete in the sense of Cook. These constructs include procedures with procedure parameters under standard Algol 60 scope rules and coroutines in a language with parameterless recursive procedures.

A natural question to ask is whether it is possible to characterize those programming languages for which sound and complete proof systems can be obtained. For a wide class of programming languages and interpretations, it can be shown that P has a sound and relatively complete proof system for every expressive interpretation iff the halting problem for language P is decidable for all finite interpretations.

Nevertheless, we are still far from a completely satisfactory characterization of the programming languages that can be axiomatized in this manner. The proof system that is generated in proving the above result does not have the property of being "syntax-directed" which is distinctive of the Hoare axioms. Moreover, theoretical considerations suggest that good axioms for total correctness may exist for a wider spectrum of languages than is the case for partial correctness. In this paper we discuss these questions and others which still need to be addressed before the characterization problem can be considered solved.

1. Introduction

A key trend in program verification has been the use of axioms and rules of inference to specify the meanings of programming language constructs. This approach was first suggested by C.A.R. Hoare in 1969 [11]. Although the most complicated control structure in Hoare's original paper was the **while** statement, there has been considerable success in extending his method to other language features. Axioms have been proposed for the **goto** statement, functions, recursive procedures with value and reference parameter passing, simple coroutines, and concurrent programs. Research by Clarke [2] has shown, however, that there are natural programming language control structures which are impossible to describe adequately by means of Hoare axioms. Specifically, Clarke has shown that there are control structures for which it is impossible

[1]This research was partially supported by NSF Grant MCS-82-16706.

to obtain axiom systems that are sound and complete in the sense of Cook [5]. These constructs include procedures with procedure parameters under standard Algol 60 scope rules and coroutines in a language with parameterless recursive procedures.

A natural question to ask is whether it is possible to characterize those programming languages for which sound and complete proof systems can be obtained. The incompleteness results are established by observing that if a programming language P has a sound and relatively complete proof system for all expressive interpretations, then the halting problem for P must be decidable for finite interpretations. This condition also appears to be sufficient: For a wide class of programming languages and interpretations, it can be shown that if the halting problem for language P is decidable for all finite interpretations, then P has a proof system which will be sound and relatively complete for any expressive interpretation. Nevertheless, we are still far from a completely satisfactory characterization of the programming languages that can be axiomatized in this manner. In this paper we identify and discuss four specific issues which we believe still need to be addressed before the characterization problem can be considered solved:

1. The present version of the Characterization Theorem predicts that certain programming languages should have good Hoare proof systems, even though no natural systems have been found.

2. The Characterization Theorem should result in a usable proof system--not just an enumeration procedure. Also, the proof system should follow the syntax of the programming language (i.e. be syntax-directed) in the same way that Hoare's original system does.

3. It appears from the proof of the Characterization Theorem that certain programming languages may have good total correctness proof systems even though they do not have good partial correctness proof systems.

4. Lastly, the hypothesis of expressiveness for interpretations deserves more thought. This hypothesis is important because it determines the degree of encoding that is permitted in reasoning about programs. Is it too strong or, perhaps, not strong enough?

The paper is organized as follows: Section 2 contains a short discussion of the basic ideas of Hoare's logic and gives definitions for partial and total correctness. Soundness and relative completeness are introduced and motivated in Section 3. Expressibility and the implications of this concept are discussed in some detail in Section 4. Section 5 briefly outlines how incompleteness results are obtained for various combinations of programming language features. In Section 6 the proof of the Characterization Theorem is sketched and the limitations of this theorem are discussed. Section 7 contains a discussion of the research problems mentioned above and is the heart of the paper. Finally, Section 8 discusses the relevance of the characterization problem to programming language design.

2. Hoare Logics

The formulas in a *Hoare axiom system* are triples {P} S {Q} where S is a statement of the programming language and P and Q are formulas describing the initial and final states of the program S. The logical system in which the predicates P and Q are expressed is called the *assertion language* (AL) and in this paper will always be a first order language with *type* or *signature* Σ. Intuitively, the partial correctness formula {P} S {Q} is true iff whenever *precondition* P holds for the initial program state and S terminates, then *postcondition* Q will be satisfied by the final program state.

Although this paper is primarily concerned with partial correctness, we will occasionally need to discuss *total correctness* as well. Total correctness formulas will be triples with the syntax <P> S <Q>. Such a formula is true iff whenever the precondition P holds for some initial program state, then program S will terminate when started in this state and Q will be satisfied by the final program state.

The control structures of a programming language are specified by axioms and rules of inference for the partial correctness formulas. A typical rule of inference is

$$\frac{\{P \land b\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ S\ \{P \land \neg b\}.}$$

The predicate P is the *invariant* of the **while** loop. Proofs of correctness for programs are constructed by using the axioms together with a proof system T for the assertion language. We write $\vdash_{H,T}$\{P\} S \{Q\} if the partial correctness formula {P} S {Q} is provable using the Hoare axiom system H and the proof system T for the assertion language AL.

To discuss whether a particular Hoare axiom system adequately describes the programming language PL, it is necessary to have a definition of *truth* for partial correctness formulas which is independent of the axiom system H. The definition of truth requires two steps. First, we give an interpretation I for the assertion language AL. The interpretation I (over type Σ) specifies the primitive data objects of our programming language; it consists of a set Dom(I) (the domain of the interpretation) and an assignment of a function (respectively, predicate) over Dom(I) of the appropriate arity to each function (respectively, predicate) symbol of Σ. Typical interpretations might be the integers with the standard functions and predicates of arithmetic, or linear lists with the list processing functions car, cdr, etc. Th(I) is the set of all first-order sentences (over Σ) true in I.

Second, we provide an interpreter for the statements of the programming language. There are many ways such an interpreter may be specified--in terms of computation sequences or as the least fixed point of a continuous functional (denotational semantics). The result is a relation M[S] $\subseteq$ STATES $\times$ STATES which associates with each statement S the input-output relation on STATES $\equiv$ [VAR $\rightarrow$ ID] determined by that statement. Once the relation M has been specified, a formal definition may be given for partial correctness. The partial correctness formula {P} S {Q} is *true with respect to interpretation* I ($\models_I$ {P} S {Q}) iff for all states σ and σ', if predicate P holds for state σ under interpretation I and $(\sigma, \sigma') \in$ M[S], then Q must hold for σ' under I also. Note that by this definition the partial correctness formula {*true*} S {*false*} will hold in interpretation I iff S diverges regardless of what state it is started in.

A similar definition can also be given for total correctness. The formula <P> S <Q> is *true with respect to interpretation I* ($\models_I$ {P} S {Q}) iff for every state σ, if predicate P holds for σ under interpretation I , then there exists a state σ' such that $(\sigma, \sigma') \in$ M[S] and Q must hold for σ' under I also.

3. Soundness and Completeness

When can we be satisfied that a Hoare axiom system H adequately describes the programming language PL? There are two possible ways a Hoare axiom system may be inadequate. First, some theorem {P} S {Q} which can be proven in the axiom system may fail to hold for actual executions of the program S; in other words, there is a terminating computation of S such that the initial state satisfies P but the final state fails to satisfy Q. A way of preventing this source of error is to adopt an operational or denotational semantics for the programming language which is close to the way statements are actually executed. We then show that every theorem which can be proven using the axiom system will be true in the model of program execution that we have adopted. In the notation defined above we prove that for all P, Q, S, if $\vdash_{H,T}$ {P} S {Q} then $\models_I$ {P} S {Q}. In general, this type of *soundness* property is fairly easy to establish.

A second source of inadequacy is that the axioms for the programming language may not be sufficiently powerful to handle all combinations of the control structures of the language. However, the question of when it is safe to stop looking for new axioms is much more difficult to answer than the question of soundness. One solution is to prove a completeness theorem for the

Hoare axiom system. We can attempt to prove that every partial correctness formula which is true of the execution model of the programming language is provable in the axiom system. In general it is impossible to prove such completeness theorems; the proof system for the assertion language may itself fail to be complete. For example, when dealing with the integers for any consistent axiomatizable proof system, there will be formulas which are true of the integers but not provable within the system. Also the assertion language may not be powerful enough to express the invariants of loops. This difficulty occurs if the assertion language is Presburger arithmetic (i.e., integer arithmetic without multiplication). Note that both of the difficulties above are faults of the underlying assertion language and interpretation--not of the Hoare axiom system.

How can we talk about the completeness of a Hoare axiom system independently of its assertion language? Cook [5] gives a Hoare axiom system for a subset of Algol including the while statement and nonrecursive procedures. He then proves that if there is a complete proof system for the assertion language (e.g., all true statements of the assertion language) and if the assertion language satisfies a certain natural expressibility condition, which will be discussed in detail in the next section, then every true partial correctness assertion will be provable.

Definition 1: A Hoare axiom system H for a programming language PL is *sound* and *complete* (in the sense of Cook) iff for all AL and I, if I is *expressive* with respect to AL and PL, then

$$\models_I \{P\}\ S\ \{Q\} \Leftrightarrow\ \vdash_{H,Th(I)} \{P\}\ S\ \{Q\}$$

4. Expressibility

We say that I is *expressive* with respect to AL and PL iff for all S $\in$ PL and Q there is a formula of AL which expresses the *weakest precondition for partial correctness* (called the weakest liberal precondition in [7]) $WP[S](Q) = \{\sigma \mid \forall \sigma' \, [(\sigma, \sigma') \in M[S] \rightarrow Q[\sigma']\,]\}$. If I is expressive with respect to AL and PL, then it is not difficult to prove that $\models_I \{WP[S](Q)\}\ S\ \{Q\}$ and that if $\models_I \{P\}\ S\ \{Q\}$ then $\models_I P \rightarrow WP[S](Q)$.

Expressibility is important because it guarantees the existence of invariants for loops and recursive procedures. For example, it is easy to show that

$$\models_I WP[\textbf{while}\ b\ \textbf{do}\ S](Q) \equiv (b \wedge WP[S](WP[\textbf{while}\ b\ \textbf{do}\ S](Q)) \vee (\neg\, b \wedge Q)$$

From this identity it follows that

$$\models_I \{WP[\textbf{while } b \textbf{ do } S](Q) \wedge b\}\ S\ \{WP[\textbf{while } b \textbf{ do } S](Q)\} \qquad (1)$$

and

$$\models_I WP[\textbf{while } b \textbf{ do } S](Q) \wedge \neg b \rightarrow Q \qquad (2)$$

By using the **while** axiom and the rule of consequence, we immediately obtain

$$\models_I \{WP[\textbf{while } b \textbf{ do } S](Q)\}\ \textbf{while } b \textbf{ do } S\ \{Q\}.$$

This type of reasoning (cf [3]) shows that WP[**while** b **do** S](Q) can always be used as the invariant of a **while** loop with postcondition Q and is the essence of the relative completeness proof for a simple programming language containing the **while** statement as the only control structure.

We could have equally well defined expressibility in terms of the *weakest precondition for total correctness*

$$WT[S](Q) = \{\sigma \mid \exists\, \sigma' [\, (\sigma, \sigma') \in M[S] \wedge Q[\sigma']\,]\}$$

or in terms of the *strongest postcondition*

$$SP[S](P) = \{\sigma' \mid \exists\, \sigma[P[\sigma] \wedge (\sigma, \sigma') \in M[S]\,]\}.$$

It is shown in [2] that all of these definitions lead to the same concept.

Theorem 2: The following are equivalent:

1. I is WP-expressive with respect to PL and AL

2. I is WT-expressive with respect to PL and AL

3. I is SP-expressive with respect to PL and AL

In establishing relative completeness results for looping constructs it is more convenient to work with the weakest precondition for partial correctness. For recursive procedures, on the other hand, the strongest postcondition generally is more useful.

Not every choice of AL, PL, and I gives expressibility. Cook demonstrates this in the case where the assertion language is Presburger arithmetic. Wand [20] gives another example of the same phenomenon. More realistic choices of AL, PL, and I do give expressibility, however. If AL is the full language of number theory and I is an interpretation in which the symbols of number theory receive their usual interpretations, then I is expressive with respect to AL and PL. Also if the domain of I is finite, then expressibility is assured. Recently, German and Halpern [9] and Urzyczyn [19] have independently obtained a strong characterization of those interpretations which are expressive:

Theorem 3: Suppose that PL is an *acceptable* programming language with recursion and that I is a *Herbrand definable* interpretation which is expressive for AL and PL. Then I is either finite or *strongly arithmetic.*

The *acceptability* of the programming language is a mild technical assumption which ensures that the language is closed under certain reasonable programming constructs, and that given a program, it is possible to effectively ascertain its step-by-step computation in interpretation I by asking quantifier-free questions about I. An interpretation I over a type Σ is *Herbrand definable* (cf [4]) if every element $d \in \mathrm{dom}(I)$ is the meaning of some term of the Herbrand universe over type Σ. An interpretation I is said to be *strongly arithmetic* (cf [4]) if there exist first order formulas $Z(x)$ (for zero), $S(x,y)$ (for successor), $A(x,y,z)$ (for addition), and $M(w,y,z)$ (for multiplication) and an bijection $J:\mathrm{dom}(I) \rightarrow N$ which makes I isomorphic to a standard model of arithmetic.

5. Incompleteness Results

Are there any programming language constructs for which it is impossible to obtain good Hoare axiomatizations? An obvious place to start our search is with more complicated parameter passing mechanisms. In this section we consider the problem of obtaining a sound and complete proof system for an Algol-like language which allows procedures as parameters of procedure calls.

Theorem 4: It is impossible to obtain a Hoare proof system H which is sound and complete in the sense of Cook for a programming language PL which allows:

1. procedures as parameters of procedure calls
2. recursion
3. static scope
4. global variables
5. internal procedures as parameters of procedure calls

Proof of Theorem 4 follows immediately from Lemmas 5 and 6. Note that all of the features (i)-(v) are found in Algol 60. Moreover, the result holds even if the language PL is restricted so that *self-application* (e.g., calls of the form **call** P(. . . ,P, . . .)) is not permitted. Thus, the result also applies to Pascal where procedures are restricted so that actual procedure parameters must be either formal procedure parameters or names of procedures with no procedure formal parameters.

Lemma 5: The halting problem is undecidable for programs in a programming language PL with features (1)-(5) above for all finite interpretations I with $\mathrm{card}(\mathrm{dom}(I)) \geq 2$.

The proof of the lemma uses a modification of a technique of Jones and Muchnick [12] and is fully described in [2]. Note that the lemma does not hold for flowchart schemes or *while* schemes. In each of these cases if I is finite, the program can be viewed as a finite state machine and we may test for termination (at least theoretically) by watching the execution sequence of the program to see whether any program state is repeated. In the case of recursion one might expect that the program could be viewed as a type of pushdown automaton (for which the halting problem is also decidable). This is not the case if we allow procedures as parameters. The static scope execution rule, which states that procedure calls are interpreted in the environment of the procedure's declaration rather than in the environment of the procedure call, allows the simulation program to access values normally buried in the runtime stack without first "popping the top" of the stack. This additional power can be used to simulate an arbitrary Turing machine.

Lemma 6: If PL has a Hoare proof system which is sound and complete in the sense of Cook, then the halting problem for PL must be decidable for all finite interpretations.

Proof: Suppose that PL has a Hoare proof system which is sound and complete in the sense of Cook. Thus, for all AL and I if (a) T is a complete proof system for AL and I and (b) I is expressive with respect to PL and AL, then

$$\models_I \{P\}\ S\ \{Q\} \Leftrightarrow \vdash_{H,T} \{P\}\ S\ \{Q\}.$$

Assume further that the halting problem for PL is undecidable for some particular finite interpretation I. Observe that in this case T may be chosen in a particularly simple manner; in fact, there is a decision procedure for the truth of formulas in AL relative to I. Note also that AL is expressive with respect to PL and I, since I is finite. Thus, both hypothesis (a) and (b) are satisfied. From the definition of partial correctness, we see that $\{true\}$ S $\{false\}$ holds iff S diverges for the initial values of its global variables. We conclude that the set of programs S such that $\models_I \{true\}\ S\{false\}$ holds is not recursively enumerable. On the other hand, since

$$\models_I \{true\}\ S\ \{false\} \Leftrightarrow \vdash_{H,T} \{true\}\ S\ \{false\}$$

We can enumerate those programs S such that $\models_I\{true\}\ S\ \{false\}$ holds -- simply enumerate all possible proofs and use the decision procedure for T to check applications of the rule of consequence; this, however, is a contradiction.

If sharing (which intuitively means referring to the same program variable by two or more different names) and self application are disallowed, a sound and relatively complete Hoare proof system may be obtained by modifying any one of the five features of Theorem 4. Thus if we change from *static scope* to *dynamic scope*, a complete set of axioms may be obtained for (i) procedures with procedure parameters, (ii) recursion, (iv) global variables, and (v) internal procedures as parameters; or if we disallow internal procedures as parameters, a complete system may be obtained for (i) procedures with procedure parameters, (ii) recursion, (iii) static scope, and (iv) global variables.

Techniques similar to that used in Theorem 4 have also been used to obtain incompleteness results for programming languages which include any of the following features: (a) call-by-name parameter passing in the presence of recursive procedures, functions, and global variables; (b) coroutines with local recursive procedures that can access global variables; (c) unrestricted (PL/1 like) pointer variables with retention; (d) unrestricted pointer variables with recursion; and (e) label variables with retention.

6. The Characterization Problem

The incompleteness results are established by observing that if a programming language PL has a sound and relatively complete proof system for all expressive interpretations, then the halting problem for PL must be decidable for finite interpretations. Lipton [14] considered a form of converse: If PL is an *acceptable* programming language and the halting problem is decidable for finite interpretations, then PL has a sound and relatively complete Hoare logic for expressive and effectively presented interpretations. Lipton actually proved a partial form of the converse. He showed that given a program S and the effective presentation of I, it is possible to enumerate all the partial correctness assertions of the form {*true*} S {*false*} which are true in I. From this it easily follows that we can enumerate all true quantifier-free partial correctness assertions, since we can encode quantifier-free tests into the programs. But, it does not follow that we can enumerate all first-order partial correctness assertions, since an acceptable programming language will not in general allow first-order tests.

Clarke, German, and Halpern [4] consider acceptable programming languages which permit recursive procedure calls. They also require that the interpretation be Herbrand-definable. Under these assumptions they are able to extend the results of [2] and [14], significantly. They are able to eliminate the requirement that pre and postconditions be quantifier-free and that the interpretation be effectively presented. They further show that the set of partial correctness assertions true in I is actually (uniformly) decidable in Th(I) provided that the halting problem for P is decidable for finite interpretations. Lipton's proof, on the other hand, produces an enumeration procedure for partial correctness assertions and, thus, shows only that the set of true partial correctness assertions is r.e. in Th(I). We sketch below a proof of the main theorem of [4].

Theorem 7: Let PL be an acceptable programming language with recursion. Then the following are equivalent:

1. There is an effective procedure which for expressive, Herbrand-definable interpretations I will decide which first-order partial correctness assertions are true in I when given an oracle for Th(I).

2. PL has a decidable halting problem for finite interpretations.

Sketch of proof. The fact that (1) $\Rightarrow$ (2) follows from Lemma 6. Proof that (2) $\Rightarrow$ (1) is considerably more complicated. Assume that PL is an acceptable programming language with recursion and that I is both expressive and Herbrand-definable. By Theorem 3 we know that I is either finite or strongly arithmetic. Assume further that we are given an oracle for Th(I). We must provide an effective procedure for deciding which partial correctness assertions are true in I. The decision procedure will actually consist of two procedures M_1 and M_2 which are dovetailed. Both M_1 and M_2 are sound in the sense that they generate only true partial correctness triples; in addition, M_1 will be complete if I is strongly arithmetic, and M_2 will be complete if I is finite.

Let AX be a finite set of axioms for first-order arithmetic. We could take, for example, the nine axioms for zero, successor S(x,y), addition A(x,y,z), multiplication M(x,y,z), and less-than L(x,y,z) given in chapter 2 of [18]. There will, of course, be nonstandard models for AX, so this set of axioms will not be complete for all of standard arithmetic. Nevertheless, an interpretation which satisfies AX will have a standard part consisting of those elements of the domain of the form $S^k(0)$ for some integer k. In general, there is no first-order formula which defines the standard part, but under the hypothesis above we will show that the standard part can be defined.

The first step is to define inductively an encoding of Herbrand terms of type Σ. The details of the encoding are straightforward, and we refer the reader to [4] for details. We will use the binary predicate symbol H to denote this encoding. Thus, we want H(u,d) to be true iff u is the encoding of a Herbrand term with value d. To achieve this goal, we give an axiom ENC for H and prove that if I satisfies AX and ENC, then $\models_I H(S^k(0),d)$ iff k is the encoding of a Herbrand term whose value in I is d.

By using the encoding relation H we can explicitly give a formula which defines the standard part of I.

Lemma 8: If I satisfies AX, ENC, and is Herbrand definable then $STD(x) \equiv \exists d \forall z(H(z,d) \Rightarrow x < z)$ defines the standard part of I.

We can now describe the construction of M_1. M_1 will guess formulas $Z(x)$, $S(x,y)$, $L(x,y)$, $A(x,y,z)$, $M(x,y,z)$, and $H(x,y)$ and check using the oracle for Th(I) that AX and ENC hold in I when written in terms of these formulas. We then define STD(x) as in Lemma 8 check $\models_I \forall x[Std(x)]$. If not, M_1 continues guessing. But if $\forall x[Std(x)]$ does hold in I, then we have effectively found formulas which make I strongly arithmetic.

Lemma 9: Suppose we can effectively find formulas $Z(x)$, $S(x,y)$, $A(x,y,z)$ and $M(x,y,z)$ of type Σ which make I strongly arithmetic. Then for each $P \in PL$ we can effectively find a formula A'_p of type Σ which is equivalent to A_p in I.

Now given a pair of first-order formulas P, Q and a program S, M_1 will construct the formula

$$\forall \bar{x},\bar{y}[P(\bar{x}) \wedge A'_S(\bar{x},\bar{y}) \Rightarrow Q(\bar{y})]$$

and consult the oracle for Th(I). If this formula is true, M_1 will output {P}S{Q}; otherwise it will output $\neg$({P}S{Q}).

By making use of Theorem 3, the construction of M_2 can be made much simpler than the version in [4]. The first step is to determine how many elements are in Dom(I). P_2 will successively generate formulas of the form $F_n = \exists x_1 x_2 \ldots x_n \forall x[x=x_1 \vee x=x_2 \vee \ldots \vee x=x_n]$ for $n = 1,2,\ldots$ and submit them to the oracle for Th(I). If I is finite, then the answer *true* will be obtained for some formula F_n indicating that Dom(I) has no more than n elements. In this case every element of Dom(I) must be the value of some Herbrand term of depth $n+1$ or less. Let $t_1, t_2, \ldots, t_m$ be the Herbrand terms of depth $n+1$ or less. Consider a particular partial correctness formula {P}S{Q}. We rename the bound variables of P and Q so that all are distinct. We next replace every subformula of P and Q of the form $\forall x[W]$ by

$$(x = t_1 \rightarrow W) \wedge \ldots \wedge (x = t_n \rightarrow W)$$

and every subformula of the form $\exists x[W]$ by

$$(x = t_1 \wedge W) \vee \ldots \vee (x = t_n \wedge W)$$

to obtain a new quantifier-free partial correctness triple {P'}S{Q'} which will be true in I iff the original triple {P}S{Q} is true in I. If LOOP is a program which always diverges, then S' if ¬P' then LOOP else begin S; if Q' then LOOP end

will also be a program and will diverge on all of its inputs iff {P'}S{Q'} is true in I. Thus, by using our decision procedure for the halting problem of PL on finite interpretations we can determine whether the original triple {P}S{Q} is true or false in I.

This completes the sketch of the proof of Theorem 7. Grabowski [10] has developed a modification of the proof above which appears to avoid the hypothesis of Herbrand-definability that we have previously required of interpretations. However, Grabowski's version of the theorem does not handle total correctness.

The deficiencies of the Characterization Theorem and its proof are clear. The proof system that is produced is an enumeration procedure and could not be used in practice. Moreover, the proof system does not follow the syntax of the programming language in the same way that Hoare's original system does. This is disturbing since the theorem may guarantee a proof system for a programming language for which no natural Hoare system is known. These problems, however, are precisely the ones mentioned in the introduction as being suitable for further research; we will discuss them in detail in the next section.

7. Research Directions

7.1. Natural Axiomatizations for New Programming Languages

Although, it is difficult to say precisely what makes a proof system natural or whether one system is more natural than another, certainly no one would claim that Theorem 7 leads to a natural Hoare proof system. Since the present version of the Characterization Theorem may predict that a certain programming language should have a good Hoare proof system, even though no natural system has been found, it would seem to be of little use. We conjecture, however, that whenever this happens, additional research will always lead to a natural proof system--perhaps by extending the existing notions of what is permitted in a Hoare axiomatization. A good example is the language L4, which is obtained from the programming language in Theorem 4 when global variables are disallowed. Since L4 has generated a great deal of interesting research and since it also illustrates a number of new ideas, we consider it in some detail below.

In [2] it was argued that if use of global variables was disallowed, then denesting of internal procedures would be possible. Thus, the proof system given for the latter case in [2] could also be adapted for use with L4. This argument was shown to be incorrect by Olderog [15]. Since globally declared procedures can still be called from within an internal procedure declaration even if global variables have been disallowed, complete denesting is not always possible. For example, it is impossible to denest the internally declared procedure q in the program segment below. (We use the convention that parameters appearing after the colon ":" in a parameter list are procedure parameters.)

```
begin proc p(:f); begin proc q; begin ... f; ... end q ;
                        ... p(:q); ...
                        ... f; ...
                      end p;
        proc r; begin ... end r;
        p(:r)
end
```

Previous languages involving procedures were relatively easy to axiomatize, since they all had the *finite range* property. Informally, this property states 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. L4 does not have this property, however. This is significant since all previous axiom systems for procedures were based on Algol 60's copy rule semantics for procedure execution and since Olderog [16] was able to show that none of these axiom systems can deal adequately with infinite range.

For several years the question of whether there existed a natural Hoare proof system for L4 that was sound and complete in the sense of Cook remained open. Langmaack [13] proved that the halting problem for L4 was decidable and hence by the Characterization Theorem given in Section 6 such a proof system should exist (although perhaps not a natural one!). In 1982 Olderog [15] and Damm and Josko [6] devised proof systems for L4 which were based on the use of a higher order assertion language and the addition of *relation variables* to the programming language. Their systems did not completely solve the problem, however; in both of these papers, the axiom system is assumed to include all of the formulas valid in a certain higher order theory related to the interpretation. Moreover, because of the addition of relation variables to the programming language, their proofs required a stronger notion of expressiveness than was used originally by Cook.

A natural proof system which only uses a first order assertion language and the standard notion of expressiveness has recently been given by German, Clarke, and Halpern [4]. In order to deal with infinite range, they 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. By using these assertions it is possible to relate the semantics of a procedure with the semantics of procedures passed to it as parameters.

For example, 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. 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).

It is not obvious that this approach is sufficient to specify all procedures; indeed, this is the essence of the relative completeness proof. The proof is based on the existence of *abstract interpreter programs* which can be shown to exist whenever the interpretation is Herbrand-definable and the programming language is acceptable in the sense of Section 4. Roughly speaking, an interpreter program receives as inputs a number of ordinary variables containing an encoding of a 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.

Many of the techniques introduced in [8] appear to have applications beyond L4. For example, the more general partial correctness assertions and the way the relative completeness proof is structured may be helpful with other languages which have infinite range [1].

7.2. Syntax-Directed Proof systems

Certainly the most important research problem is to develop a version of the Characterization Theorem which provides some insight as to when a syntax-directed proof system can be obtained. One could even argue that any version the theorem which fails to address this issue does not really capture the spirit of Hoare's Logic. An important first step towards developing such a theorem has recently been made by Olderog who has obtained an interesting characterization of the formal call trees of programs in those sublanguages of Pascal for which a sound and relatively complete Hoare axiomatization can be obtained. His theorem also guarantees that a particular syntax-directed proof system will be sound and relatively complete for those sublanguages.

Let PL_{Pas} be the language obtained from Dijkstra's guarded command language by adding blocks and a Pascal-like procedure mechanism in which actual procedure parameters of a procedure call must either be formal procedure parameters or names of procedures with no formal procedure parameters. Thus, self application is not possible with programs in PL_{Pas}. We refer the reader to [16] for the formal syntax and semantics of this class of programs.

By the Incompleteness Theorem of Section 5 there is no sound and relatively complete proof system for the full language; however, there may be complete proof systems for sublanguages of $PL \subseteq PL_{Pas}$. Olderog gives a Hoare proof system H_0 which is sound for all of PL_{Pas} and then proves the following surprising result.

Theorem 10: For every admissible $PL \subseteq PL_{Pas}$ the following are equivalent:

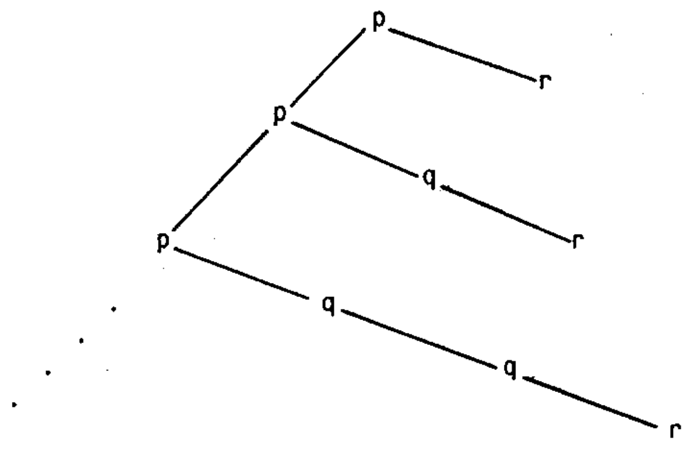
1. There exists a sound and relatively complete Hoare logic in the sense Theorem 7 for PL.

2. The halting problem for PL is decidable under finite interpretations.

3. All programs in PL have *regular* formal call trees.

4. The Hoare proof system H_0 is sound and relatively complete for PL.

A sublanguage $PL \subseteq PL_{Pas}$ is *admissible* if PL is r.e. and closed under program transformations which leave procedure structure invariant. A tree T over a finite alphabet is *regular* if the set of paths in T is a regular language or,equivalently, if there are only finitely many different patterns of subtrees. The *formal call tree* of a program S records the order in which the procedures of S are called in all possible executions of S. The formal call tree for the program skeleton in Section 7.1 is shown below and is clearly non-regular.



Hence, it follows by Olderog's Theorem that any programming language PL containing the program must fail to have a sound and relatively complete Hoare proof system. Note that this does not contradict the results of [8], since any admissible language which contains this program will also contain programs that access non-local variables, and hence the proof system of [8] would not be expected to be complete.

7.3. The Problem with Total Correctness

What happens when we attempt to extend the Characterization Theorem to apply to total correctness assertions as well as partial correctness assertions? Under the same hypothesis as in the previous proof it is possible to show that the set of true total correctness assertions is (uniformly) decidable in Th(I) iff the halting problem for PL is decidable for finite interpretations. Moreover, the set of true total correctness assertions is (uniformly) r.e. in Th(I) even if the halting problem for PL is *not decidable* for finite interpretations ([4]). This last result unexpectedly suggests that good axiom systems for total correctness may exist for a wider class of programming languages than in the case for partial correctness and is, therefore, somewhat disturbing.

Proof of the result above is similar to the proof of the Characterization Theorem in Section 6. As in the previous proof, this proof breaks into two cases depending on whether the interpretation is finite or infinite and strongly arithmetic. The infinite case is just like the infinite case for partial correctness except that we.ask the oracle for Th(I) about the formula

$$\forall \bar{x} \exists \bar{y} (P(\bar{x}) \Rightarrow A^I_S(\bar{x},\bar{y}) \wedge Q(\bar{y}))$$

in order to determine whether the total correctness assertion <P>S<Q> is true or not.

In the finite case we can use the same trick as in the previous theorem to make the pre and post conditions quantifier-free. We then use the decision procedure for the halting problem of PL programs to determine if the program S' shown below halts on all of its inputs.

if P **then begin** S; **if** $\neg$ Q **then** LOOP **end**

Alternatively, since there are only a finite number of domain elements and since we can find a finite set of Herbrand terms such that every domain element is the value of some term in the set, we can run S' on all possible combinations of its inputs. If S' halts on all of them, then we enumerate the triple <P>S<Q>.

We note that this anomaly does not occur if we require that the negation of a total correctness assertion also be a legal total correctness assertion. For example, we could augment first order logic with a special operator for the *weakest precondition for total correctness*:

$$\langle\text{formula}\rangle ::= \langle\text{atomic formula}\rangle \mid \text{WT}[\langle\text{program}\rangle](\langle\text{formula}\rangle) \mid \neg\langle\text{formula}\rangle$$

$$\mid \langle\text{formula} \vee \langle\text{formula}\rangle \mid \exists\langle\text{var}\rangle[\langle\text{formula}\rangle]\rangle.$$

Atomic formulas will have the same syntax as in standard first-order logic. The syntax of programs will not be given; however, programs are assumed to be deterministic, and all booleans in programs must be atomic formulas. Thus, in contrast to Dynamic Logic [17], we do not permit booleans to be arbitrary WT-formulas.

Let f be a formula and let S be a program. We write $I,\sigma \models f$ iff f is true in interpretation I and state σ. The obvious definition is used in all of the clauses for <formula> except the one for WT. We define $I,\sigma \models \text{WT}[S](f)$ iff $I,M[S](\sigma) \models f$. A WT-formula f is *true in I* ($I \models f$) iff $I,\sigma \models f$ for all states σ.

Theorem 11: Assume that PL is an acceptable programming language with recursion and that I is Herbrand definable and expressive with respect to AL and PL. Then the set of WT-formulas which are true in I is uniformly r.e. in Th (I) iff the halting problem for PL is decidable for finite interpretations.

Proof: Assume that PL has an undecidable halting problem for some finite interpretation I. Since finite interpretations are expressive, it follows that there will always be a formula P for WT[S](Q) which does not itself involveWT. However, it is impossible to effectively enumerate such formulas given S and Q. Thus, we cannot have a sound and relatively complete proof system for a logic that can express $\text{WT}[S](Q) \equiv P$ when P and Q do not involve WT.

For the converse we actually prove that if the halting problem for PL is decidable for finite interpretations, then the set of WT-formulas which are true in I is uniformly decidable in Th(I).

Assume that I is Herbrand-definable and expressive and that the halting problem for PL is decidable for finite interpretations. Given an oracle for Th(I), the construction used in the proof of theorem 7 can also be used to find a formula of AL which expresses WT[S](Q) whenever Q is a formula of AL and S is a program in PL.

In case I is arithmetical, we can use the formula $\exists \bar{y}[A'_S(\bar{x}, \bar{y}) \Rightarrow Q(\bar{y})]$ where $A'_S(\bar{x}, \bar{y})$ is the AL formula which expresses the input/output relation of S.

In explaining the finite case we use the same notation as in Theorem 7. Assume that S has global variables $v_1, v_2, \ldots, v_k$. Let $S'(a_{i_1}, \ldots a_{i_k})$ be the program

```
begin
          $v_1$ := $a_{i_1}$;
          .
          .
          .
          $v_k$ := $a_{i_k}$;
          S;
          if $\neg$ Q' then LOOP
end
```

Where each a_{i_j} is one of the terms $t_1, \ldots, t_n$ and Q' is the quantifier-free formula that is equivalent to Q. Next, determine whether S' will halt for each possible combination of $a_{i_1}, \ldots, a_{i_n}$. The formula for the weakest precondition will be the disjunction of all those clauses $v_1 = a_{i_1} \wedge v_2 = a_{i_2} \wedge \ldots \wedge v_k = a_{i_k}$ which correspond to initial states in which S' will halt.

Thus, given an arbitrary WT-formula, we can transform it to an equivalent formula of AL not involving WT. Start with the most deeply nested occurrence of WT, say $WT[S_1](Q_1)$, where Q_1 is a formula of AL and does not involve WT. By the observation above, we can replace $WT\,[S_1](Q_1)$ by an equivalent AL formula Q_2 not involving WP. We continue to repeat this process until all occurrences of WT are eliminated. We then ask the oracle for Th(I) about the truth of f^* where f^* is the universal closure of f.

7.4. More Powerful Notions of Expressibility

Another obvious question is whether a more powerful notion of expressibility might permit sound and relatively complete proof systems to be obtained for a wider class of programming languages than is currently the case with Cook's original definition. The answer is, trivially, yes. If, for example, we use a notion of expressibility which requires that interpretations be strongly arithmetical, then only the infinite case in Theorem 7 will apply. Since the infinite case does not use the hypothesis that the halting problem is decidable for finite interpretations, the relative decision procedure could be adapted, for example, to apply to the full language PL_{Pas}. This is unlikely to lead to a very natural proof technique because of the encoding that is necessary to obtain the formula $A^I_S(x,y)$ from program S.

Alternatively, we could simply compile PL_{Pas} into an assembly language where the runtime stack is encoded as the value of an integer variable, where the only control structures are the conditional and the while statement, and where assignments can use standard arithmetical operations of addition, multiplication, etc. This, however, is contrary to the spirit of high level programming languages. If the proof of a recursive program requires explicit reasoning about the low-level implementation of the language by means of the runtine stack, then why not simply replace the recursive procedures themselves by stack operations. The purpose of recursion in programming languages is to free the programmer from the details of implementing recursive constructs.

If a programming language requires an unnatural use of encoding in order to get an axiomatization, then perhaps it is too powerful to reason about effectively. The incompleteness results of Section 5, which depend only on finite interpretations, show that certain programming language features cannot have natural axiomatizations. In fact, we would argue that finite interpretations are often more useful than infinite interpretations for judging whether an axiomatization is natural, since they preclude the possibility that domain elements can be used to encode complicated runtime data structures such as the runtime stack or linked lists of activation records. Moreover, all of the standard partial-correctness rules (e.g. the assignment axiom, the **while** statement rule, etc.) work just as well for finite interpretations as for infinite ones.

We do not mean to imply that there is nothing to be learned from further study of expressiveness. We suggest, however, a different direction for research on this topic. Although expressiveness has been assumed by many previous researchers to get a complete axiomatization, the use they have made of this assumption (e.g. to generate the existence of loop invariants) seems more natural than its use in the proof of the characterization theorem in Section 6. Thus, we believe that perhaps the hypothesis of expressiveness should be weakened or restricted in some way. We note, however, that such a weakening would not affect the incompleteness results of Section 5.

8. Conclusion--Implications for Language Design

The fact that not every programming language can be described adequately by means of Hoare axioms does not mean that this method for reasoning about programs is less useful than operational or denotational methods. On the contrary, it is exactly because Hoare Logic is more restrictive in descriptive power that it turns out to be so useful for reasoning about programs. The increased flexibility of more operational approaches is obtained at a high price; the necessary attention to low level implementation details usually makes high-level reasoning about programs unacceptably cumbersome.

That some programming languages would be extremely hard to specify in this manner should be expected. It has been known for some time that certain language constructs make informal reasoning about a program's behavior quite difficult; this same complexity would also be expected to complicate a Hoare proof system for such a language. In this respect the programming languages of Section 5 are particularly pathological since arbitrary Turing machine computations can be simulated by the control structures of the language even in a finite interpretation.

Perhaps, the existence of a sound and relatively complete Hoare Logic could be used as a criterion for the design of programming languages suitable for program verification. At the very least such a criterion would force language designers to devise programming languages with simple, clean control structures and to consider carefully the possible unexpected interactions of adding another control structure to an already existing language.

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. "Program Invariants as Fixedpoints." *Computing 21* (1979), 273-294.

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

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

6. 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.

7. Dijkstra, E.W.. *A Discipline of Programming.* Prentice Hall, 1976.

8. German, S.M., E.M. Clarke, and J.Y. Halpern. Reasoning about procedures as parameters. Proc. CMU Conference on Logics of Programs, ACM, 1983.

9. German, S.M. and J.Y. Halpern. On the Power of the Hypothesis of Expressiveness. Tech. Rept. RJ 4079, IBM Research, 1983.

10. Grabowski, M. On the Relative Completeness of Hoare Logics. Proc 14th ACM Symp. on Principles of Programming Languages , ACM, 1984.

11. Hoare, C. A. R. "An axiomatic basis for computer programming." *Comm. ACM 12* (1969), 576-583.

12. Jones, N.D., Muchnick, S. S. "Complexity of Finite Memory Programs with Recursion." *JACM 25 No. 2* (1978), 312-321.

13. Langmaack, H. On termination problems for finitely interpreted ALGOL-like programs. Tech. Rept. Bericht Nr. 7904, Inst. f. Inform. u. Prakt. Math., Christian-Albrechtst-Univ., Kiel, 1979.

14. Lipton, R. J. A necessary and sufficient condition for the existence of Hoare logics. 18th IEEE Symp. on Foundations of Computer Science, 1-6, LNCS 85, 1977, pp. 363-373.

15. Olderog, E.-R. Correctness of PASCAL-like programs without global variables. , Mathematisches Forschungsinstitut Oberwolfach, 1982.

16. Olderog, E.-R. A characterization of Hoare's logic for programs with Pascal-like procedures. Proc 15th ACM Symp. on Theory of Computing, ACM, 1983, pp. 320-329.

17. Pratt, V.R. Semantical Considerations of Floyd-Hoare logic. Proc. 17th IEEE symp. on Foundations of Computer Science, IEEE, 1976, pp. 109-121.

18. Shoenfield, J.R.. *Mathematical Logic.* Addison Wesley, 1967.

19. Urzyczyn, P. A Necessary and Sufficient Condition in order that a Herbrand Interpretation is Expressive Relative to Recursive Programs. Institute of Mathematics, University of Warsaw, 1983.

20. Wand, M. A New Incompleteness result for Hoare's System. JACM, ACM, 1978, pp. 168-167.