PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

The Characterization Problem for Hoare Logics [and Discussion]

E. M. Clarke, P. Aczel, J. V. Tucker and J. C. Shepherdson

Phil. Trans. R. Soc. Lond. A 1984 **312**, 423-440
doi: 10.1098/rsta.1984.0068

Phil. Trans. R. Soc. Lond. A **312**, 423–440 (1984) [423]
Printed in Great Britain

The characterization problem for Hoare logics

By E. M. Clarke, Jr
Department of Computer Science, Carnegie – Mellon University, Schenley Park, Pittsburgh, Pennsylvania 15213, *U.S.A.*

Research by myself and by others has shown that there are natural programming language control structures that 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. Moreoever, theoretical considerations suggest that good axioms for total correctness may exist for a wider spectrum of languages than for partial correctness. In this paper we discuss these questions and others that 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 (Hoare 1969). 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 **go to** statement, functions, recursive procedures with value and reference parameter passing, simple coroutines, and concurrent programs. Research by Clarke (1979*a*) has shown, however, that there are natural programming language control structures that are impossible to describe adequately by means of Hoare axioms. Specifically, Clarke has shown that there are control structures for which it is impossible to obtain axiom systems that are sound and complete in the sense of Cook (1978). 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 that 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 that 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 organization of the paper is now described. Section 2 contains a short discussion of the basic ideas of Hoare logic and gives definitions for partial and total correctness. Soundness and relative completeness are introduced and motivated in §3. Expressibility and the implications of this concept are discussed in some detail in §4. Section 5 briefly outlines how incompleteness results are obtained for various combinations of programming language features. In §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, §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 *pre-condition* P holds for the initial program state and S terminates, then *post-condition* 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 $\langle P\rangle\,S\,\langle Q\rangle$. 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

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

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

The predicate P is the *invariant* of the **while** loop. Proofs of correctness for programs are constructed by using the axioms together with the 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 by 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 that 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; its 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 langage. 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 $STATES \equiv [VAR \to Dom(I)]$ 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' under I, 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 $\langle P\rangle S\langle Q\rangle$ 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, 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. One way of preventing this source of error is to adopt operational or denotational semantics for the programming language, which is close to the way statements are actually executed. We then show that every theorem that can be proven by 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 that 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 that 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 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 (1978) gives a Hoare axiom system for a subset of ALGOL including the while statement and non-recursive procedures. He then proves that if there is a complete proof system for the assertion language (for example, 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 that expresses the *weakest precondition for partial correctness* (called the weakest liberal precondition in Dijkstra (1976)) $WP[S](Q) = \{\sigma \mid \forall \sigma'\ [(\sigma, \sigma') \in M[S] \rightarrow Q[\sigma']]\}$. 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

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

and

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

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. Clarke (1979*b*)) shows that $WP[\textbf{while}\ b\ \textbf{do}\ S](Q)$ 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 defined expressibility in terms of the *weakest precondition for total correctness*

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

or in terms of the *strongest post-condition*

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

It is shown in Clarke (1979*a*) that all of these definitions lead to the same concept.

THEOREM 1. *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 pre-condition for partial correctness. For recursive procedures, on the other hand, the strongest post-condition generally is more useful.

Not every choice of AL, PL, and I gives expressibility. Cook demonstrates this for the case where the assertion language is Presburger arithmetic. Wand (1978) 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 & Halpern (1983) and Urzyczyn (1983) have independently obtained a strong characterization of those interpretations that are expressive.

THEOREM 2. *Suppose that* PL *is an* acceptable *programming language with recursion and that* I *is a* Herbrand-definable *interpretation that 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 that 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. Clarke (1983)) 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. Clarke (1983)) 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 a bijection $J : \mathrm{Dom}(I) \to N$, 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 that allows procedures as parameters of procedure calls.

THEORM 3. *It is impossible to obtain a Hoare proof system* H *that is sound and complete in the sense of Cook for a programming language* PL *that 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 theorm 3 follows immediately from lemmas 1 and 2. Note that all of the features (1)–(5) are found in ALGOL 60. Moreover, the result holds even if the language PL is restricted so that *self-application* (for example, 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 1. *The halting problem is undecidable for programs in a programming language* PL *with features* (1)–(5) *for all finite interpretations* I *with card* (Dom(I)) $\geqslant 2$.

The proof of the lemma uses a modification of a technique of Jones & Muchnick (1978) and is fully described in Clarke (1979*a*). 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. For recursion one might expect that the program could be viewed as a type of push-down automaton (for which the halting problem is also decidable). This is not so 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 run-time stack without first 'popping the top' of the stack. This additional power can be used to simulate an arbitrary Turing machine.

LEMMA 2. *If* PL *has a Hoare proof system that 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 that 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

$$\models_{\mathrm{I}} \{\mathrm{P}\}\,\mathrm{S}\,\{\mathrm{Q}\} \Leftrightarrow \vdash_{\mathrm{H,T}} \{\mathrm{P}\}\,\mathrm{S}\,\{\mathrm{Q}\}.$$

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 hypotheses (*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 $\models_{\mathrm{I}} \{true\}\,\mathrm{S}\,\{false\}$ holds is not recursively enumerable. On the other hand, since

$$\models_{\mathrm{I}} \{true\}\,\mathrm{S}\,\{false\} \Leftrightarrow \vdash_{\mathrm{H,T}} \{true\}\,\mathrm{S}\,\{false\},$$

we can enumerate those programs S such that $\models_{\mathrm{I}} \{true\}\,\mathrm{S}\,\{false\}$ 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 3. So if we change from *static scope* to *dynamic scope*, a complete set of axioms may be obtained for (1) procedures with procedure parameters, (2) recursion, (4) global variables, and (5) internal procedures as parameters; or if we disallow internal procedures as parameters, a complete system may be obtained for (1) procedures with procedure parameters, (2) recursion, (3) static scope, and (4) global variables.

Techniques similar to that used in theorem 3 have also been used (Clarke 1979*a*) to obtain incompleteness results for programming languages that include any of the features: (*a*) call-by-name parameters 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 (1977) 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 $\{true\}\,\mathrm{S}\,\{false\}$ that are true in I. From this it easily follows that we can enumerate all true quantifer-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 *et al.* (1983) consider acceptable programming languages that permit recursive procedure calls. They also require that the interpretation be Herbrand-definable. Under these assumptions they are able to extend the results of Clarke (1979*a*) and Lipton (1977), significantly. They are able to eliminate the requirement that pre- and post-conditions 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 probelm 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 is r.e. in Th(I). We sketch below a proof of the main theorem of Clarke *et al.* (1983).

Theorem 4. *Let* PL *be an acceptable programming language with recursion. Then the following are equivalent.*

(1) *There is an effective procedure that 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) $\Leftrightarrow$ (2) followed from lemma 2. 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 2 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 ch. 2 of Shoenfield (1967). There will, of course, be non-standard models for AX, so this set of axioms will not be complete for all of standard arithmetic. Nevertheless, an interpretation that satisfies AX will have a standard part consisting of those elements of the domain of the forms $S^k(0)$ for some integer k. In general, there is no first-order formula that 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 Clarke *et al.* (1983) 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 that defines the standard part of I.

LEMMA 3. *If* I *satisfies* AX, ENC, *and is Herbrand-definable then* $\mathrm{Std}(x) \equiv \exists d \forall z(H(z, d) \Rightarrow x < z)$ *defines the standard part of* I.

We can now describe the construction of M_1, which 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 3 check $\models_I \forall x[\mathrm{Std}(x)]$. If not, M_1 continues guesses. But if $\forall x[\mathrm{Std}(x)]$ does hold in I, then we have effectively found formulas that make I strongly arithmetic.

LEMMA 4. *Suppose we can effectively find formulas* $Z(x)$, $S(x, y)$, $A(x, y, z)$ *and* $M(x, y, z)$ *of type* S *that make* I *strongly arithmetic. Then for each* $P \in PL$ *we can effectively find a formula* A'_p *of type* Σ *that is equivalent to* A_p *in* I.

Now given a pair of first-order formulas P and 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 2, the construction of M_2 can be made much simpler than the version in Clarke *et al.* (1983). 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. For 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

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

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 that always diverges, then S′

if $\neg$ 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 4. Grabowski (1984) has developed a modification of the proof above, which appears to avoid the hypothesis of Herbrand-definability that we have previously required in 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 4 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 3 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 Clarke (1979*a*) 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 Clarke (1979*a*) could also be adapted for use with L4. This argument was shown to be incorrect by Olderog (1982). 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 that for each program, there is a bound on the number of distinct *procedure environments*, or associations between procedure names and bodies, that can be reached. L4 does not have this property, however. This is significant since all previous axiom systems for procedures were based on the ALGOL 60 copy rule semantics for procedure execution and since Olderog (1983) 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 (1979) proved that the halting problem for L4 was decidable and hence, by the characterization theorem given in §6, such a proof system should exist (although perhaps not a natural one!). Olderog (1982) and Damm & Josko (1982) 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 that only uses a first order assertion language and the standard notion of expressiveness has recently been given by German *et al.* (1983). 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) multiples y by v, then for the same procedure r and value v, the call p(x: r) multiples x by v^2. Observe how the environment variable v, appearing in the post-conditions of the calls r(y) and p(x: r), is used to express the relation 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 §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 that encode the procedures.

Many of the techniques introduced in German *et al.* (1983) 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 that have infinite range de Bakker *et al.* (1981).

7.2. *Syntax-directed proof systems*

Certainly the most important research problem is to develop a version of the characterization theorem that provides some insight as to when a syntax-directed proof system can be obtained. One could even argue that any version of the theorem that fails to address this issue does not really capture the spirit of Hoare 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 sublanguage 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 $\mathrm{PL}_{\mathrm{Pas}}$ 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 $\mathrm{PL}_{\mathrm{Pas}}$. We refer the reader to Olderog (1983) for the formal syntax and semantics of this class of programs.

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

THEOREM 5. *For every admissible* $\mathrm{PL} \subseteq \mathrm{PL}_{\mathrm{Pas}}$ *the following are equivalent.*

(1) *There exists a sound and relatively complete Hoare logic in the sense of theorem* 4 *for* PL.

(2) *The halting problem of* 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 $\mathrm{PL} \subseteq \mathrm{PL}_{\mathrm{Pas}}$ is *admissible* if PL is r.e. and closed under program transformations that 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 §7.1 is shown in figure 1 and is clearly non-regular. Hence, it follows by the Olderog 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 German *et al.* (1983), since any admissible language that contains this program will also contain programs that access non-local variables, and hence the proof system of German *et al.* (1983) would not be expected to be complete.

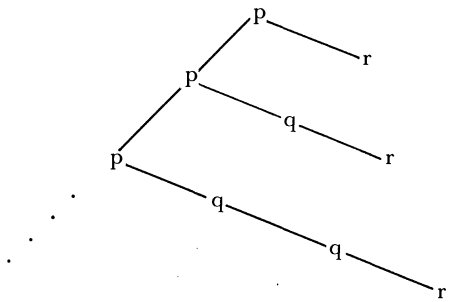

FIGURE 1. The formal call tree for the program skeleton in §7.1.

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. Moreoever, 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 (Clarke *et al.* 1983). This last result unexpectedly suggests that good axiom systems for total correctness may exist for a wider class of programming languages than for partial correctness and is, therefore, rather disturbing.

Proof of the result above is similar to the proof of the characterization theorem in §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

$$\forall \bar{x} \exists \bar{y} \, (\mathrm{P}(\bar{x}) \Rightarrow \mathrm{A}_{\mathrm{S}}^{\mathrm{I}} (\bar{x}, \bar{y}) \wedge \mathrm{Q}(\bar{y}))$$

to determine whether the total correctness assertion $\langle \mathrm{P} \rangle \, \mathrm{S} \, \langle \mathrm{Q} \rangle$ is true or false.

For the finite interpretation 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** ¬ 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 $\langle \mathrm{P} \rangle \, \mathrm{S} \, \langle \mathrm{Q} \rangle$.

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*:

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

Atomic formulas will have the same syntax as for 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 Pratt (1976), we do not permit booleans to be arbitrary WT-formulas.

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

THEOREM 6. *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 that 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) that does not itself involve WT. 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 $\mathrm{WT[S](Q)} \equiv \mathrm{P}$ 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 that 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 4 can also be used to find a formula of AL that 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 $\exists \bar{y}[\mathrm{A}'_{\mathrm{S}}(\bar{x}, \bar{y}) \Rightarrow \mathrm{Q}(\bar{y})]$, where $\mathrm{A}'_{\mathrm{S}}(\bar{x}, \bar{y})$ is the AL formula that expresses the input–output relation of S.

In explaining the finite case we use the same notation as that in theorem 4. Assume that S has global variables $\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_k$. Let $\mathrm{S}'(\mathrm{a}_{\mathrm{i}_1}, \ldots \mathrm{a}_{\mathrm{i}_k})$ be the program

begin
$\mathrm{v}_1 := \mathrm{a}_{\mathrm{i}_1}$;
.
.
.
$\mathrm{v}_k := \mathrm{a}_{\mathrm{i}_k}$;
S;
if $\neg$ Q′ **then** LOOP
end

where each $\mathrm{a}_{\mathrm{i}_j}$ is one of the terms $\mathrm{t}_1, \ldots, \mathrm{t}_n$ and Q′ is the quantifier-free formula that is equivalent to Q. Next, determine whether S′ will halt for each possible combination of $\mathrm{a}_{\mathrm{i}}, \ldots, \mathrm{a}_{\mathrm{i}_n}$. The formula for the weakest precondition will be the disjunction of all those clauses $\mathrm{v}_1 = \mathrm{a}_{\mathrm{i}_1} \wedge \mathrm{v}_2 = \mathrm{a}_{\mathrm{i}_2} \wedge \ldots \wedge \mathrm{v}_k = \mathrm{a}_{\mathrm{i}_k}$ that 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 that requires interpretations to be strongly arithmetical, then only the infinite case in theorem 4 will apply. Since the infinite case does not use the hypothesis that the halting problem is decidable for finite interpretations, the relative decision procedures 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 $A_S^I(x, y)$ from program S.

Alternatively, we could simply compile PL_{Pas} into an assembly language where the run-time 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 run-time 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 to get an axiomatization, then perhaps it is too powerful to reason about effectively. The incompleteness results of §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 run-time data structures such as the run-time stack or linked lists of activation records. Moreover, all of the standard partial-correctness rules (for example, 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 (for example, to generate the existence of loop invariants) seems more natural than its use in the proof of the characterization theorem in §6. So, 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 §5.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

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 behaviour 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 §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.

This research was partly supported by N.S.F. Grant no. MCS-82-16706.

References

de Bakker, J. W., Klop, J. W., Meyer, J.-J. C. 1981 Correctness of programs with function procedures. *Tech. Rep.* no. IW 170/81. Amsterdam: Mathematisch Centrum.

Clarke, E. M. 1979*a* Programming language constructs for which it is possible to obtain good Hoare-like axioms. *J. Ass. comput. Mach.* **26**, 129–147.

Clarke, E. M. 1979*b* Program invariants as fixedpoints. *Computing* **21**, 273–294.

Clarke, E. M. Jr, German, S. & Halpern, J. Y. 1983 Effective axiomatization of Hoare logics. *J. Ass. comput. Mach.* **30**, 612–636.

Cook, S. A. 1978 Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.* **7**, 70–90.

Damm, W. & Josko, B. 1982 A sound and relatively complete Hoare-logic for a language with higher type procedures. *Tech. Rep. Bericht*, no. 77. RWTH Aachen: Lehrstuhl fur Informatik II.

Dijkstra, E. W. 1976 *A discipline of programming*. Englewood Cliffs, N.J.: Prentice Hall.

German, S. M., Clarke, E. M. & Halpern, J. Y. 1983 Reasoning about procedures as parameters. In *Proc. CMU Conference on Logics of Programs*, pp. 206–221. New York: Ass. comput. Mach.

German, S. M. & Halpern, J. Y. 1983 On the power of the hypothesis of expressiveness. *IBM Research Tech. Rep.* no. RJ 4079.

Grabowski, M. 1984 On the relative completeness of Hoare logics. In *Proc. 14th Ass. comput. Mach. Symp. on Principles of Programming Languages*, pp. 258–262. New York: Ass. comput. Soc.

Hoare, C. A. R. 1969 An axiomatic basis for computer programming. *Commun. Ass. comput. Mach.* **12**, 576–583.

Jones, N. D. & Muchnick, S. S. 1978 Complexity of finite memory programs with recursion. *J. Ass. comput. Mach.* **25**, 312–321.

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

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

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

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

Olderog, E.-R. 1983 A characterization of Hoare's logic for programs with Pascal-like procedures. In *Proc 15th Ass. comput. Mach. Symp. on Theory of Computing*, pp. 320–329. New York: Ass. comput. Mach.
Pratt, V. R. 1976 Semantical considerations of Floyd–Hoare logic. In *Proc. 17th IEEE symp. on Foundations of Computer Science*, pp. 109–121. London: IEEE.
Shoenfield, J. R. 1968 *Mathematical logic*. Reading, Massachusetts: Addison Wesley.
Urzyczyn, P. 183 A necessary and sufficient condition in order that a Herbrand interpretation is expressive relative to recursive programs. Institute of Mathematics, University of Warsaw.
Wand, M. 1978 A new incompleteness result for Hoare's system. *J. Ass. comput. Mach.* **25**, 167–168.

Discussion

P. Aczel (*Mathematics Department, Manchester University, U.K.*). What does the *possession of a sound and relatively complete Hoare-type axiomatization* mean for the proof of correctness of real programs? Is there hope of doing it completely automatically, or even with the injection of suitable formulas, for example loop invariants, by a human? If so is the same true for Professor Clarke's non-syntax directed axioms?

E. M. Clarke. When we have a *sound and relatively complete axiomatization*, we know that our proof rules are adequate for reasoning about any possible combination of the various constructs of the programming language. We have reduced the problem of verifying a program to the problem of finding invariants and proving theorems in the assertion language. These problems may be quite difficult, but at least we know that in principle we will not get into trouble when a programmer uses some feature of the language in an unusual way.

P. Aczel (*Mathematics Department, Manchester University, U.K.*). In using Hoare style proof rules to verify programs by hand the predicates, i.e. the pre-conditions and post-conditions of correctness assertions, need only be formulated in precise, but perhaps informal, standard mathematical language. On the other hand the notion of relative completeness of the proof rules requires the involvement of formal languages for expressing these predicates.

In view of the fact that this involvement is seemingly cancelled out by relativizing to arbitrary expressive formal languages, would it not be better to avoid their use in the first place when formulating the notion of completeness of the proof rules? Much of the syntactic detail could then be banished from the completeness proofs.

E. M. Clarke. I agree with you to some extent. We could simply treat a predicate as a collection of program states and not concern ourselves with the syntactic representation of predicates. This approach was used by de Bakker & Meertens (1973). They were able to show the soundness and completeness of proof rules for the simple looping control structures and for parameterless recursive procedures. It seems to me, however, that this approach breaks down when we attempt to handle more complicated language features such as blocks with local variable declarations or recursive procedures with variable parameters. One of the rules for procedures states that execution of statement S leaves P invariant (i.e. P{S}P) provided that all of the variables free in P are inactive in S. This rule seems to require that we know something about the syntactic representation of P. There are many other similar rules in the literature.

Reference

de Bakker, J. W. & Meertens, L. G. L. T. 1973 *On the completeness of the inductive assertion method*. Amsterdam: Mathematisch Centrum.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

J. V. Tucker (*Department of Computer Studies, University of Leeds, U.K.*). I will comment on the technical idea of completeness, and on the role of completeness theorems in the theory of program verification.

Hoare logics concern control structures at a given level of data abstraction; this data abstraction is specified by a set, T, of axioms for the operations invoked in the programs of interest. A soundness theorem confirms that a formal proof of a specified program $\{p\}\,S\,\{q\}$, in Hoare logic based on T, guarantees the validity of $\{p\}\,S\,\{q\}$ for every implementation I satisfying the specification T. However, the 'converse' idea of completeness, in the sense of Cook, considers the validity of $\{p\}\,S\,\{q\}$ in a *single* implementation I of T, rather than its validity for the class of *all* implementations of T as might be expected for a data type specification. It is known that this latter completeness property also fails to obtain for first-order Hoare Logics in general.

The theorems about completeness in the sense of Cook have further deficiencies. First, the use of the set Th(I), of all true first-order statements about I, as an oracle, spoil the axiomatic nature of Hoare logics. For example, in the important case of arithmetic N, the set Th(N) is highly non-computable (being of Turing degree of unsolvability O^{ω}; infinitely more complex than the halting problem for the Turing machine, as it were). Secondly, the expressiveness concept becomes awkward to apply for two-sorted data types. For example, it is known that two independent copies of N can form a two-sorted structure that is not first-order expressive. Thus, proving programs with two types of variable can be problematical in first-order Hoare logic.

How should these completeness theorems be interpreted? I think that a completeness theorem for a given syntax-directed Hoare logic merely confirms that, in some limited sense, the logic possesses sufficient rules for its underlying programming constructs. A completeness theorem for a general Hoare logic, in Professor Clarke's sense, confirms that some logical system with sufficient rules is possible. As we have seen, in Professor Clarke's very interesting lecture, for some language constructs that is the meagre extent of our present knowledge.

I believe that a different attitude to completeness is also important. In general, a completeness theorem for a logical system L with respect to some semantics M can be interpreted as a confirmation that L syntactically or proof-theoretically characterizes M. Thus, the lack of a general completeness theorem for a consistent Hoare logic implies that the semantics of the programming language is not the semantics about which the logic is reasoning. If a Hoare logic is used to define a programming language (as was originally envisaged in the writings of Floyd, Hoare and Wirth) then various non-standard semantics of the language must be examined. I have worked on these problems of completeness with Dr J. A. Bergstra (Centre for Mathematics and Computer Science, Amsterdam).

In conclusion, I am tempted to speculate that research into the 'model theory' of Hoare logics will create a more agreeable conceptual framework for the attractive characterization theorems of Professor Clarke and his collaborators.

Reference

Bergstra, J. A. & Tucker, J. V. 1982 *J. Comput. Syst. Sci.* **25**, 267–284.

E. M. Clarke. The use of the Th(I) in relative completeness proofs is simply a technical device for factoring out the complexity of proving theorems in the assertion language from the more general problem of proving programs correct. It permits us to investigate the adequacy of the

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

Hoare axioms for reasoning about the various constructs of the programming language. With this limited objective in view, I do not see why it should 'spoil the axiomatic nature of the Hoare Logic'.

I agree that interesting technical problems with expressiveness occur for two-sorted types. I very much enjoyed Dr Tucker's paper on this topic.

I also agree with Dr Tucker's interpretation of the incompleteness results for certain programming languages in terms of the existence of non-standard semantics for those languages.

J. C. Shepherdson (*School of Mathematics, University of Bristol, U.K.*). Can Professor Clarke give a definition of what is meant by 'syntax-directed' rules?

E. M. Clarke. I think that it would not be terribly difficult to give a definition of the term *syntax-directed proof rule*. For example, one could imagine a very general definition using an *attribute grammar* for the programming language under consideration. An attribute grammar is a context-free grammar extended by attaching attributes to the symbols of the grammar. Associated with each production of the grammar is a set of semantic equations where each equation defines one attribute as the value of semantic function applied to other attributes in the production. For proof rules in a Hoare logic the attributes would be first-order formulas involving pre- and post-conditions of statements associated with the various symbols of the grammar.

The problem of actually finding a sound and relatively complete syntax-directed proof system for a given programming language seems much harder, and apart from Olderog's work little progress has been made on this question.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES