SIAM J. COMPUT.
Vol. 7, No. 1, February 1978

SOUNDNESS AND COMPLETENESS OF AN AXIOM SYSTEM FOR PROGRAM VERIFICATION*

STEPHEN A. COOK†

Abstract. A simple ALGOL-like language is defined which includes conditional, while, and procedure call statements as well as blocks. A formal interpretive semantics and a Hoare style axiom system are given for the language. The axiom system is proved to be sound, and in a certain sense complete, relative to the interpretive semantics. The main new results are the completeness theorem, and a careful treatment of the procedure call rules for procedures with global variables in their declarations.

Key words. program verification, semantics, axiomatic semantics, interpretive semantics, consistency, completeness

1. Introduction. The axiomatic approach to program verification along the lines formulated by C. A. R. Hoare (see, for example, [6] and [7]) has received a great deal of attention in the last few years. My purpose here is to pick a simple programming language with a few basic features, give a Hoare style axiom system for the language, and then give a clean and careful justification for both the soundness and adequacy (i.e., completeness) of the axiom system. The justification is done by introducing an interpretive semantics for the language, rather like that in [10] and [8]. These two papers also have outlined soundness arguments for axiom systems, but for somewhat different language features, axioms, and interpretive models. The completeness claim and argument presented here is new (although completeness and incompleteness proofs inspired by an earlier version of this paper [2] appear in [3], [11], [12], [13], and [14]). I have tried to choose the axioms and rules of the formal system to be as simple as possible, subject to the constraints that they be sound, complete, and in the style and spirit of Hoare's rules.

Donahue [4] presented a soundness argument for a similar axiom system, but soundness was proved in terms of mathematical semantics in the style of Dana Scott. This led to a rather different argument than that presented here.

Most of the complication in the present paper comes from handling procedure statements. The rules for procedure call statements often (in fact usually) have technical bugs when stated in the literature, and the rules stated in earlier versions of the present paper are not exceptions. In the process of trying to prove the soundness of these rules, I uncovered some of the bugs, and this led me to believe a careful and detailed proof of soundness is necessary to have any confidence that there are no further bugs. I have allowed procedure declarations to have global variables (subject to some restrictions) and this has added to the complications of the rules and their justifications.

In addition to procedure statements, the programming language used allows assignment, conditional, while, compound, and block statements, but disallows input/output statements, jumps, functions, and data structures.

* Received by the editors July 1, 1976, and in revised form March 31, 1977.

† Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A7.

The programming language is specified in § 2, the interpretive model appears in § 3, and the axiom system in § 4. The soundness of the system is proved in § 5, and the completeness is proved in § 6.

2. The language Al[$\mathscr{L}_1$, $\mathscr{L}_2$]. We do not want to specify the particular primitive relations and operations used to form expressions in our ALGOL fragment, so we assume these are the same as a given language $\mathscr{L}_1$ for the first order predicate calculus. For concreteness we could take $\mathscr{L}_1 = \mathscr{L}_N$ where nonlogical symbols in $\mathscr{L}_N$ are $\{<, =, +, \cdot, 0, 1\}$, but more generally any list $\{P_1, P_2, \cdots\}$ of predicate symbols and any list $\{f_1, f_2, \cdots\}$ of function symbols will do. In addition, we assume we are given a predicate calculus language $\mathscr{L}_2$, an extension of $\mathscr{L}_1$, which will be used for assertions. In most examples, we will take $\mathscr{L}_2 = \mathscr{L}_1 = \mathscr{L}_N$.

The programming language Al[$\mathscr{L}_1, \mathscr{L}_2$] will be a modified subset of ALGOL 60, with the following objects.

Variables. The variables (identifiers) will coincide with the variables of $\mathscr{L}_1$. All variables have the same (unspecified) type. In our example in which $\mathscr{L}_1 = \mathscr{L}_N$, we think of that type as "integer".

Declarations. a) *Procedure declarations* have the form

$$p(\bar{x} : \bar{v})\ \textbf{proc}\ K$$

where p is the procedure name, $(\bar{x} : \bar{v})$ is the formal parameter list, and K is the procedure body. $\bar{x}$ and $\bar{v}$ are disjoint lists of distinct variables, and the variables in $\bar{v}$ cannot occur to the left of any assignment statement in K, nor can they appear as actual parameters to the left of the colon (:) in any procedure call statement in K. The variables $\bar{x}$ and $\bar{v}$ are considered local to the declaration. We allow global variables in K (in addition to $\bar{x}$ and $\bar{v}$). However, the variables in $\bar{x}$ and $\bar{v}$ cannot occur globally in any procedure declaration for another procedure which could be activated by executing K. Also note the restriction on procedure calls stated below.

To avoid confusion over associating procedure names with procedure bodies, we require that no procedure name can be declared more than once in any program. In general we shall assume that some fixed procedure declaration is associated with each procedure p.

We shall assume in this paper that no procedure is recursive. That is, there is no chain of procedure names $p_1, \cdots, p_n$ such that $p_1 = p_n$, and the procedure body for p_i contains a call to p_{i+1}, $1 \leqq i < n$.

b) *Variable declarations* have the form

$$\textbf{new}\ x$$

where x is any variable. Both procedure and variable declarations occur at the beginning of blocks. A variable can occur without being declared, in which case it acts as an input to the program (it must have a value before the program is executed). Also, a given variable can be declared in any number of blocks.

Expressions. a) A *Boolean expression* is any quantifier-free formula of $\mathscr{L}_1$. For example, in the case of $\mathscr{L}_N$, $0 = 0, x + 1 < y + 1 \ \&\ z < x$ are Boolean expressions.

b) Numerical expressions are terms of $\mathscr{L}_1$. For example, in the case of $\mathscr{L}_N$, $(x + 1) \cdot (z + 1) + 1$, 0, 1 are numerical expressions.

Statements. a) *Assignment* statements have the form $x := e$, where x is a variable and z is a numerical expression.

b) *Procedure calls* have the form **call** $p(\bar{u} : \bar{e})$, where $\bar{u}$ is a list of distinct variables, $\bar{e}$ is a list of expressions containing no variable in $\bar{u}$, and no variable in $(\bar{u} : \bar{e})$ occurs as a nonlocal variable either in the procedure declaration for p or in the procedure declaration for any procedure which can be activated indirectly by activating p. (Formal parameters are local to a procedure declaration.)

c) *Conditional, while, compound,* and *block* statements are as in ALGOL 60, except we use **fi** and **od** to punctuate the end of conditional and while statements, respectively, as in ALGOL 68.

3. The interpretive model. An *interpretive model* $\mathcal{M} = \mathcal{M}[\mathcal{I}]$ for the programming language Al$[\mathcal{L}_1, \mathcal{L}_2]$ is determined by giving an interpretation $\mathcal{I}$ for the predicate calculus language $\mathcal{L}_2$ (of course $\mathcal{I}$ also interprets the language $\mathcal{L}_1$). Thus $\mathcal{I} = \langle D, \bar{P}_1, \bar{P}_2, \cdots, \bar{f}_1, \bar{f}_2, \cdots \rangle$ where D is a nonempty domain, $\{\bar{P}_1, \bar{P}_2, \cdots\}$ are the predicates on D interpreting the predicate symbols of $\mathcal{L}_2$, and $\{\bar{f}_1, \bar{f}_2, \cdots\}$ are the functions on D interpreting the function symbols of $\mathcal{L}_2$. As usual in predicate calculus interpretations, the predicates $\bar{P}_i$ and the functions $\bar{f}_i$ are assumed to be total. In our example $\mathcal{L}_N$, the natural interpretation is $\mathcal{I}_N$, in which D is the set of integers, and $<$, $=$, $+$, $\cdot$, 0, 1 are all given their standard meanings. If the function symbol $\div$ is included in the language and it is interpreted as division, then some value must be assigned to $n \div 0$. One way to do this is to add an extra element Ω, the "undefined" element to the domain D, and let $n \div 0$ be Ω. In this case, all function and predicate symbols must have their interpretations extended to be defined at Ω, although their values at Ω could be Ω. In any case, during execution of a program, all expressions which must be evaluated will have well-defined values in D, so an undefined expression is never a cause for termination.

Notation. If E is a term or formula, $t_1, \cdots, t_k$ are terms, and $y_1, \cdots, y_k$ are distinct variables, then

$$E\frac{t_1 \cdots t_k}{y_1 \cdots y_k}$$

indicates the result of simultaneously substituting $t_1, \cdots, t_k$ for free occurrences of $y_1, \cdots, y_k$, respectively, in E. In the definitions of $P(s, \delta)$ and $e(s, \delta)$ below, the role of t_i is played by $s(\delta(y_i))$. The latter object is an element of D rather than a term, so that strictly speaking a constant c_i should be introduced whose value under the interpretation is $s(\delta(y_i))$, and then

$$e(s, \delta) = \mathcal{I}\left(e\frac{c_1 \cdots c_k}{y_1 \cdots y_k}\right).$$

However, the abuse of notation below is convenient and, we hope the intended meaning is clear.

The set of *registers* $\mathcal{R}$ is the infinite set $\{X_1, X_2, \cdots\}$. A *state* of $\mathcal{M}[\mathcal{I}]$ is a total map $s: \mathcal{R} \to D$. A *variable assignment* is a one-one partial map

$$\delta: \{\text{variables of } \mathcal{L}_2\} \to \mathcal{R}$$

with a finite domain. If P is a formula of $\mathcal{L}_2$ with free variables $y_1, \cdots, y_k$, and s is

a state, δ is a variable assignment to $\{y_1, \cdots, y_k\}$, then

$$P(s, \delta) \equiv \begin{cases} \text{true} & \text{if } P\dfrac{s(\delta(y_1)) \cdots s(\delta(y_k))}{y_1 \cdots y_k} \text{ is true in } \mathscr{I}, \\ \text{false} & \text{otherwise.} \end{cases}$$

Thus P becomes either true or false in $\mathscr{I}$ when its free variables are given values according to s and δ.

If e is a term (i.e., numerical expression) with free variables $y_1, \cdots, y_k$, and δ, s are as above, then

$$e(s, \delta) = \mathscr{I}\left(e\frac{s(\delta(y_1)) \cdots s(\delta(y_k))}{y_1 \cdots y_k}\right).$$

Thus $e(\delta, s)$ is that element of D which is the value of the expression e when the free variables of e are assigned values according to s and δ.

A *procedure assignment* is a partial map

$$\pi\colon \{\text{procedure names}\} \to \{\text{procedure bodies}\} \times \{\text{formal parameter lists}\}$$

such that π has a finite domain. Thus, if the procedure declaration $p(\bar{x} : \bar{v})$ **proc** K occurs in a program, we will define $\pi(p) = \langle K, (\bar{x} : \bar{v})\rangle$.

The heart of the model $\mathscr{M}$ is the function Comp(A, s, δ, π), which assigns to a statement A, state s, variable assignment δ, and procedure assignment π, a finite or infinite sequence $\langle s_1, s_2, \cdots\rangle$ which represents the successive states of the computation determined by the statement A when the initial state is s. This computation is not defined unless δ assigns a register at least to each free (i.e., global) variable of A and π assigns a procedure body and formal parameter list at least to each procedure name associated with A which has no corresponding procedure declaration. (In general, A will be taken from the interior of a block B, and the declarations of B must be recorded in δ and π, as shown below.)

The function Comp(A, s, δ, π) is defined below by giving one defining clause for each of 8 possible forms which the statement A can take. The reader can check that every legal statement A in Al$[\mathscr{L}_1, \mathscr{L}_2]$ fits one and only one of the 8 cases. The definition is recursive, in the sense that Comp appears on the right side of the clauses. This may appear ironic in a paper on program verification, since one of the important issues in programming language semantics is interpreting recursively defined procedures. However, one does not have to understand recursive procedures in general in order to understand this specific definition. Suffice it to say that we intend Comp to be evaluated by "call by name," in the sense that occurrences of Comp are to be replaced successively by their meanings according to the appropriate clauses in the definition. Simplifications are to be made using knowledge about the model $\mathscr{M}[\mathscr{I}]$. Of course the process may not terminate, in which case an infinite sequence of states will be generated.

Notation. A^* stands for a sequence $A_1; A_2; \cdots; A_k$, $k \geqq 0$, of statements of Al$[\mathscr{L}_1, \mathscr{L}_2]$, and D^* stands for a sequence $D_1; D_2; \cdots; D_l$, $l \geqq 0$, of declarations of Al$[\mathscr{L}_1, \mathscr{L}_2]$, and A is a statement of Al$[\mathscr{L}_1, \mathscr{L}_2]$. The symbol $\hat{}$ indicates concatenation. More precisely, $C_1 \hat{} C_2$ is the concatenation of sequences C_1 and

C_2, if C_1 is finite, and $C_1 \hat{} C_2$ is C_1 if C_1 is infinite. If K is a procedure body, then

$$K\frac{\bar{u}, \bar{e}}{\bar{x}, \bar{v}}$$

indicates the result of substituting the actual parameters $\bar{u}, \bar{e}$ for the corresponding free (i.e., global) occurrences of the formal parameters $\bar{x}$ and $\bar{v}$ (respectively) in K. If any variable z in $(\bar{u}, \bar{e})$ is declared locally in K and if the formal parameter corresponding to z occurs within the scope of the declaration of z in K, then the local variable z must be renamed in K before the substitution takes place, so that no actual parameter gets "caught" by the declaration when it is substituted.

Out(A, s, δ, π) is the last state in the sequence given by Comp(A, s, δ, π), when this is a finite sequence, and is undefined otherwise.

DEFINITION. Comp$(A, s, \delta, \pi) =$

Cases A: (The value of Comp appears to the right of the arrow for each of the eight cases of the form of A given below.)

begin new x; D^*; A^* **end** $\rightarrow \langle s \rangle \hat{}$ Comp (**begin** D^*; A^* **end**, s, δ', π),

$$\text{where } \delta'(y) = \begin{cases} \delta(y), & \text{if } y \neq x, \\ X_{k+1}, & \text{if } y = x, \text{ where } X_k \text{ is the highest} \\ & \text{indexed register in the range of } \delta, \end{cases}$$

begin $p(\bar{x} : \bar{v})$ **proc** K; D^*; A^* **end** $\rightarrow \langle s \rangle \hat{}$ Comp(**begin** D^*; A^* **end**, s, δ, π'),

$$\text{where } \pi'(q) = \begin{cases} \pi(q), & \text{if } q \neq p, \\ \langle K, (\bar{x} : \bar{v}) \rangle, & \text{if } q = p. \end{cases}$$

begin A_1; A^* **end** $\rightarrow$ Comp$(A_1, s, \delta, \pi) \hat{}$ Comp(**begin** A^* **end**, Out$(A_1, s, \delta, \pi), \delta, \pi)$.

begin end $\rightarrow \langle s \rangle$.

$$x := e \rightarrow \langle s' \rangle, \text{ where } s'(X_i) = \begin{cases} s(X_i), & \text{if } \delta(x) \neq X_i, \\ e(s, \delta), & \text{if } \delta(x) = X_i. \end{cases}$$

call $p(\bar{u} : \bar{e}) \rightarrow \langle s \rangle \hat{}$ Comp$\left(K\frac{\bar{u}, \bar{e}}{\bar{x}, \bar{v}}, s, \delta, \pi\right)$, where $\pi(p) = \langle K, (\bar{x} : \bar{v}) \rangle$.

if R **then** A_1 **else** A_2 **fi** $\rightarrow \begin{cases} \langle s \rangle \hat{}\, \text{Comp}(A_1, s, \delta, \pi), & \text{if } R(s, \delta) \text{ is true}, \\ \langle s \rangle \hat{}\, \text{Comp}(A_2, s, \delta, \pi), & \text{otherwise} \end{cases}$

while R **do** A_1 **od** $\rightarrow \begin{cases} \text{Comp}(A_1, s, \delta, \pi) \hat{}\, \text{Comp}(\textbf{while } R \textbf{ do } A_1, \\ \qquad \text{Out}(A_1, s, \delta, \pi), \delta, \pi), & \text{if } R(s, \delta) \text{ is true}. \\ \langle s \rangle, \quad \text{otherwise}. \end{cases}$

where δ is defined for all variables global in A and π is defined for all undeclared procedure names in A.

Note that the clause defining the statement **call** $p(\bar{u} : \bar{e})$ means procedures have dynamic rather than static scope.

4. The axioms and rules. The axioms and rules of inference of our deductive system are basically those of Hoare [6], [7], with amendments due to Lauer [10], Igarashi, London and Luckham [9] (among others) and modified so as to reflect the structure of the recursive definition of the function Comp.

The basic object in the system is the formula $P\{A\}Q$, where P and Q are formulas in $\mathscr{L}_2$ and A is a statement of $\mathrm{Al}[\mathscr{L}_1, \mathscr{L}_2]$. Informally, $P\{A\}Q$ is true (relative to our interpretation $\mathscr{I}$) iff whenever the assertion P is true before A is executed, either A will fail to terminate, or Q will hold after A is executed.

DEFINITION. The *free set* of a statement A consists of all variables with global occurrences in A, together with variables with global occurrences in the procedure bodies of any procedures which might be activated by executing A. A formal definition can be given recursively by considering each of the possible statement types for A (as in the definition of Comp). We give four of the more interesting clauses in this definition. The free set of **begin new** x; D^*; A^* **end** consists of the union of the free sets of D^* and A^*, with x deleted. The free set of **begin** $p(\bar{x} : \bar{v})$ **proc** K; D^*; A^* **end** consists of the union of the free sets of K, D^*, and A^*, except $\bar{x}$ and $\bar{v}$ are excluded from the free set of K. The free set of $x := e$ consists of the variable x, together with all variables in e. The free set of **call** $p(\bar{u} : \bar{e})$ consists of the variables in $(\bar{u} : \bar{e})$, together with the free set of

$$K\frac{\bar{u}, \bar{e}}{\bar{x}, \bar{v}},$$

where K is the procedure body for p and $(\bar{x} : \bar{v})$ are the formal parameters for p. (Note that K and $(\bar{x} : \bar{v})$ are uniquely determined in any program by our convention of unique declarations, and see the remarks at the end of this section.)

Notation. P, Q, R, S stand for the formulas of $\mathscr{L}_2$.

$$\frac{\alpha_1, \cdots, \alpha_n}{\beta}$$

indicates the rule: from the formula(s) α_i, $i = 1, \cdots, n$, deduce the formula β.

$$\frac{D, \alpha}{\beta},$$

where D is a declaration of some procedure p, indicates the rule

$$\frac{\alpha}{\beta},$$

with the understanding that all calls of p in α and β are according to D (see the remarks at the end of this section).

The rules and axiom schemes of the system $\mathscr{H}$ consist of 1) through 11) below.

1) *Rule of variable declarations.*

$$\frac{P\frac{y}{x}\{\textbf{begin } D^*; A^* \textbf{ end}\}Q\frac{y}{x}}{P\{\textbf{begin new } x; D^*; A^* \textbf{ end}\}Q}$$

where y is not free in P or Q, and is not in the free set of D^* or A^*.

2) *Rule of procedure declarations.*

$$\frac{D, P\{\textbf{begin } D^*; A^* \textbf{ end}\}Q}{P\{\textbf{begin } D; D^*; A^* \textbf{ end}\}Q}$$

where D is any procedure declaration.

3) *Rule of compound statements.*

$$\frac{P\{A\}Q,\ Q\{\textbf{begin } A^* \textbf{ end}\}R}{P\{\textbf{begin } A; A^* \textbf{ end}\}R}$$

4) *Axiom of compound statements.*

$$P\{\textbf{begin end}\}P$$

5) *Axiom of assignment statements.*

$$P\frac{e}{x}\{x := e\}P$$

6) *Rule of conditional statements.*

$$\frac{P \,\&\, R\{A_1\}Q,\ P \,\&\, \neg R\{A_2\}Q}{P\{\textbf{if } R \textbf{ then } A_1 \textbf{ else } A_2 \textbf{ fi}\}Q}$$

7) *Rule of while statements.*

$$\frac{P \,\&\, Q\{A\}P}{P\{\textbf{while } Q \textbf{ do } A \textbf{ od}\}P \,\&\, \neg Q}$$

8) *Rule of procedure calls.*

$$\frac{p(\bar{x} : \bar{v}) \textbf{ proc } K,\ P\{K\}Q}{P\{\textbf{call } p(\bar{x} : \bar{v})\}Q}$$

9) *Rule of parameter substitution.*

$$\frac{P\{\textbf{call } p(\bar{x}' : \bar{v}')\}Q}{P\frac{\bar{u}, \bar{e}}{\bar{x}', \bar{v}'}\{\textbf{call } p(\bar{u} : \bar{e})\}Q\frac{\bar{u}, \bar{e}}{\bar{x}', \bar{v}'}}$$

provided that no variable in $\bar{u}$ (except possibly one in $\bar{x}'$) occurs free in P or Q. Here $\bar{x}'$ and $\bar{v}'$ are lists of distinct variables which may be, but need not be, the same as the formal parameters $(\bar{x} : \bar{v})$ for the procedure p. We require, of course, that the statements **call** $p(\bar{x}' : \bar{v}')$ and **call** $p(\bar{u} : \bar{e})$ be syntactically correct, which means (for **call** $p(\bar{u} : \bar{e})$) that the variables $\bar{u}$ be distinct and have no occurrence in the expressions $\bar{e}$, and no variable in $\bar{u}$ or $\bar{e}$ (other than one in $\bar{x}$ or $\bar{v}$) can be in the free set of the procedure body K of p.

10) *Rule of variable substitution.*

$$\frac{P\{\textbf{call } p(\bar{u} : \bar{e})\}Q}{P\sigma\{\textbf{call } p(\bar{u} : \bar{e})\}Q\sigma}$$

where

$$\sigma = \frac{\bar{z}'}{\bar{z}}$$

is a substitution of expressions for variables such that no variable in $\bar{z}$ or $\bar{z}'$ occurs in the free set of **call** $p(\bar{u}:\bar{e})$.

11) *Rule of consequence.*

$$\frac{P \supset R,\ R\{A\}S,\ S \supset Q}{P\{A\}Q}$$

Note that in the rule of consequence, $P \supset R$ and $S \supset Q$ are formulas of $\mathscr{L}_2$. We assume they are correct (that is their universal closures are true in the model $\mathscr{S}$) but the manner in which they are deduced is not the concern of this axiom system. This point is discussed further in later sections.

It is worth pointing out that rules 9) and 10) could be replaced by the simpler rule

$$\frac{p(\bar{x}:\bar{v})\ \textbf{proc}\ K,\ P\left\{K\frac{\bar{u},\bar{e}}{\bar{x},\bar{v}}\right\}Q}{P\{\textbf{call}\ (\bar{u}:\bar{e})\}Q}$$

and soundness and completeness could be preserved (in fact the justification would be much simpler). However, this rule is unsatisfactory because its use requires a separate proof of the hypothesis

$$P\left\{K\frac{\bar{u},\bar{e}}{\bar{x},\bar{v}}\right\}Q$$

each time a **call** statement for the procedure p appears with different actual parameters. In contrast, the present rule 8) requires the proof just once of a general property $P\{K\}Q$ of the procedure body, and rule 9) (with rules 10) and 11)) allows the deduction of suitable instances of the property for different sets of actual parameters. (The use of rule 10) will come out in the completeness argument in the last section.)

A second objection to the above alternative to rules 9) and 10) is that it spoils the pleasing principle that no substitutions for variables are made in the program text for the hypothesis of any rule.

The rules of our system are a little awkward in handling procedure declarations. This is not a real issue for our particular programming language, since we do not allow a given procedure name to have more than one declaration in a given program. If more than one such declaration were allowed (as in ALGOL 60), some device would have to be introduced in the rules to keep track of which declaration applied to a given procedure call statement. One possibility suggested in Gorelick [5], is to transform each rule

$$\frac{\alpha_1, \cdots, \alpha_n}{\beta} \quad \text{into} \quad \frac{D^*/\alpha_1, \cdots, \alpha_n}{D^*/\beta}$$

so that the context of procedure declarations D^* is made explicit at each rule

application. The rule for blocks with procedure declarations would now become

$$\frac{D^*; D/P\{\textbf{begin } D_1^*; A^* \textbf{ end}\}Q}{D^*/P\{\textbf{begin } D; D_1^*; A^* \textbf{ end}\}Q}$$

enabling us to "discharge" heretofore implicit assumptions about the context of procedure declarations. Similarly, the rule for procedures would become

$$\frac{D^*/P\{K\}Q}{D^*; p(\bar{x}:\bar{v})\ \textbf{proc}\ K/P\{\textbf{call}\ p(\bar{x}:\bar{v})\}Q},$$

where p is a new procedure name. This would be a possible way of handling the problem if one thought it were important to allow a given name to refer to two different procedures. However, in practice, this flexibility would probably cause more confusion than it would save. Furthermore, our definition of Comp would have to be significantly more complicated, requiring that the map π store the environment of the procedure body, as well as the body and formal parameters. Therefore, we shall stick to our simplifying assumption, and our simpler rules.

5. The model satisfies the axioms and rules. Most of the rules and axioms seem to be clearly valid, given the informal meaning for $P\{A\}Q$ stated above. However, there is always a worry that some condition or possibility has been overlooked. This is particularly true of rules 8)–10) for procedure calls, variations of which have been stated incorrectly in the literature several times.

How can we be sure we aren't omitting some restrictions on these rules or the use of parameters that are necessary to ensure the validity of the rules? One way is to prove that all the axioms and rules are true in our interpretive model $\mathcal{M}[\mathcal{I}]$. (At least this ensures that the axioms and rules are correct, provided it is agreed that the model is correct.)

Thus suppose we are given an interpretation $\mathcal{I}$ for $\mathcal{L}_2$, and hence a model $\mathcal{M} = \mathcal{M}[\mathcal{I}]$ for our language Al$[\mathcal{L}_1, \mathcal{L}_2]$. We say a formula P of $\mathcal{L}_2$ is *true in* $\mathcal{I}$ (or in $\mathcal{M}$) iff the closure of P is true in $\mathcal{I}$, where by closure we mean universal closure (i.e., the result of prefixing to P universal quantifiers for all free variables in P).

DEFINITION. A formula $P\{A\}Q$ is *true in* $\mathcal{M}$ (denoted by: $\models_{\mathcal{M}} P\{A\}Q$) iff for all states s, s', if $P(s, \delta)$ is true in $\mathcal{M}$ and $s' = \text{Out}(A, s, \delta, \pi)$, then $Q(s', \delta)$ is true in $\mathcal{M}$, where δ is any assignment to the free variables of P, Q, and the free set of A, and π assigns the proper bodies and parameter lists to all necessary procedure names. The subscript $\mathcal{M}$ on $\models$ is sometimes omitted.

Lemma 4, at the end of this section, states that this definition of truth is independent of δ.

DEFINITION. A formula $P\{A\}Q$ is *valid* iff it is true in all models $\mathcal{M}[\mathcal{I}]$. A rule

$$\frac{\alpha_1, \cdots, \alpha_n}{\beta}$$

is *valid* iff β is true in every model in which $\alpha_1, \cdots, \alpha_n$ are true.

Theorem 1 below states that all our axioms and rules are valid. However, they are not sufficient in the following sense: It is usually necessary to use the rule of consequence (rule 11)) to prove interesting formulas, and for this we need a

method of establishing the truth of the (universal closures of) the formulas $P \supset R$ and $S \supset Q$ in $\mathscr{L}_2$. Hence we will need to supplement our rules and axioms by a deductive system $\mathscr{D}$ for deducing formulas in $\mathscr{L}_2$ whose closures are true in $\mathscr{I}$. Of course $\mathscr{D}$ must be *sound* relative to $\mathscr{I}$, in the sense that the universal closure of every formula deducible in $\mathscr{D}$ is true in $\mathscr{I}$. We should emphasize that the soundness of $\mathscr{D}$ has meaning only relative to $\mathscr{I}$, in contrast to our system $\mathscr{H}$, whose rules and axioms are valid for all interpretations. Further discussion of the system $\mathscr{D}$ appears in § 6.

THEOREM 1. *Axioms and rules* 1) *through* 11) *for the system* $\mathscr{H}$ *are valid.*

A formal proof in the system $(\mathscr{H}, \mathscr{D})$ consists of a finite sequence of formulas, each either of the form $P\{A\}Q$, or P, with P, Q in $\mathscr{L}_2$ and A a statement in $\mathrm{Al}[\mathscr{L}_1, \mathscr{L}_2]$. Each formula is either an axiom of $\mathscr{H}$, a formula deducible in $\mathscr{D}$, or follows from earlier formulas in the sequence by one of the rules of $\mathscr{H}$. We use the notation $\vdash_{\mathscr{H},\mathscr{D}} P\{A\}Q$ to mean $P\{A\}Q$ is provable in this sense.

COROLLARY. *If* $\mathscr{D}$ *is sound relative to* $\mathscr{I}$ *and* $\vdash_{\mathscr{H},\mathscr{D}} P\{A\}Q$, *then* $\models_{\mathscr{M}[\mathscr{I}]} P\{A\}Q$.

The main tool in the proof of Theorem 1 is "induction on the definition of Comp." This principle allows us to conclude an assertion of the form "for all A, s, δ, π, if the sequence $\mathrm{Comp}(A, s, \delta, \pi)$ is finite, then it has a certain property $P(A, s, \delta, \pi)$." To make this conclusion, it is sufficient to prove, for each of the eight cases in the definition of Comp, that if A takes the form of the case, then $\mathrm{Comp}(A, s, \delta, \pi)$ satisfies P (provided it is finite), assuming as an induction hypothesis that $\mathrm{Comp}(A', s', \delta', \pi')$ (and sometimes $\mathrm{Comp}(A'', s'', \delta'', \pi'')$) satisfy P (provided they are finite), where the latter are the occurrence(s) of Comp that appear on the right hand side of the case. The justification of this principle comes directly from the definition of Comp, by a simple induction on the length of the sequence $\mathrm{Comp}(A, s, \delta, \pi)$. The principle is used in the proofs of Lemmas 1 and 3 below.

We will now prove Theorem 1 for rules 1), 9), and 10). The other cases are straightforward. For the rest of this section, "true" means true in some arbitrary model $\mathscr{M}$. Starting with rule 1, we assume the premise

$$\text{(i)} \qquad \models P\frac{y}{x}\{\textbf{begin } D^*;\ A^* \textbf{ end}\}Q\frac{y}{x}$$

where y is not free in P or Q, and is not in the free set of D^* or A^*.

In order to verify the conclusion of the rule, we assume $P(s, \delta)$ is true, where δ assigns registers to all relevant variables. Now let $s' = \mathrm{Out}(\textbf{begin new } x;\ D^*;\ A^* \textbf{ end}, s, \delta, \pi)$, where π makes the proper assignments to procedure names. According to the appropriate clause in the definition of Comp, we have

$$\text{(ii)} \qquad s' = \mathrm{Out}(\textbf{begin } D^*;\ A^* \textbf{ end}, s, \delta', \pi)$$

where δ' agrees with δ everywhere except at the variable x, to which δ' assigns a new register. Now if we define the assignment δ_1' by

$$\delta_1'(z) = \begin{cases} \delta'(z) & \text{if } z \neq y, \\ \delta(x) & \text{if } z = y, \end{cases}$$

then δ_1' is one to one (because $\delta(x)$ is not in the range of δ'), and furthermore $P(s, \delta)$ has the same truth value as

$$P\frac{y}{x}(s, \delta_1'),$$

namely true. (This is because y has no occurrence in P, x has no occurrence in

$$P\frac{y}{x},$$

and $\delta(x) = \delta_1'(y)$.) By our premise (i), we conclude

$$Q\frac{y}{x}(s_1', \delta_1')$$

is true, where

(iii) $$s_1' = \text{Out}(\textbf{begin } D^*; A^* \textbf{ end}, s, \delta_1', \pi).$$

LEMMA 1. *If δ_1 and δ_2 are two variable assignments which agree on the free set of a statement A, and if the largest-numbered registers in the ranges of δ_1 and δ_2 are the same, then* $\text{Comp}(A, s, \delta_1, \pi) = \text{Comp}(A, s, \delta_2, \pi)$, *provided the computations are finite.*

The proof is by induction on the definition of Comp. All clauses except that for variable declarations are immediate, because δ does not enter into the definition, and the exception is also easily handled.

We notice that δ' and δ_1' satisfy the hypotheses of the lemma for the statement **begin** D^*; A^* **end**, (because by our assumptions on the rule 1) y has no free occurrence in D^* or A^*), so that it follows from (ii) and (iii) that $s' = s_1'$. Therefore

$$Q\frac{y}{x}(s', \delta_1')$$

is true and hence $Q(s', \delta)$ holds by the same reasoning that showed

$$P\frac{y}{x}(s, \delta_1') \Leftrightarrow P(s, \delta).$$

This establishes the conclusion of the rule 1) and completes the proof of the validity of 1).

The rule 9) of parameter substitution is worth verifying in some detail. Assume the premises to the rule hold, so that

(1) $$\models P\{\textbf{call } p(\bar{x}' : \bar{v}')\}Q$$

Let us use the abbreviation

(2) $$\sigma = \frac{\bar{u}, \bar{e}}{\bar{x}', \bar{v}'}$$

and assume

(3) $$P\sigma(s, \delta)$$

holds for some δ which assigns registers to all relevant free variables. If we set

$$s' = \mathrm{Out}(\textbf{call}\ p(\bar{u} : \bar{e}), s, \delta, \pi), \tag{4}$$

and assume s' is defined, and $\pi(p) = \langle K, (\bar{x} : \bar{v})\rangle$, then by the definition of Comp for procedures we have

$$s' = \mathrm{Out}\left(K\frac{\bar{u}, \bar{e}}{\bar{x}, \bar{v}}, s, \delta, \pi\right). \tag{5}$$

Suppose

$$\bar{x}' = x_1', \cdots, x_m', \quad \bar{u} = u_1, \cdots, u_m, \quad \bar{v}' = v_1', \cdots, v_n', \quad \bar{e} = e_1, \cdots, e_n. \tag{6}$$

In order to use our premise (1), we must find a new pair (s_1, δ_1) such that

a) $s_1(\delta_1(z)) = s(\delta(z))$ for all $z \notin (\bar{x}', \bar{v}')$ if both $\delta(z)$ and $\delta_1(z)$ are defined,

b) $s_1(\delta_1(x_i')) = s(\delta(u_i))$, $i = 1, \cdots, m$,

c) $s_1(\delta_1(v_i')) = e_i(s, \delta)$, $i = 1, \cdots, n$,

d) $s_1(X_{m_1+i}) = s(X_{m+i})$, $i = 1, 2, \cdots$, where X_{m_1} (respectively X_m) is the highest indexed register in the range of δ_1 (respectively δ).

Let us say (s_1, δ_1) is *matched* to (s, δ) relative to σ if a)–d) are satisfied. There is no difficulty in finding such an (s_1, δ_1), since the variables in $(\bar{x}', \bar{v}')$ are all distinct.

LEMMA 2. *If (s_1, δ_1) is matched to (s, δ) relative to*

$$\sigma = \frac{\bar{u}, \bar{e}}{\bar{x}', \bar{v}'},$$

and R is an assertion (of $\mathscr{L}_2$) and e is an expression (term of $\mathscr{L}_1$), then

$$R\sigma(s, \delta) \equiv R(s_1, \delta_1), \quad \textit{and} \quad (e\sigma)(s, \delta) = e(s_1, \delta_1)$$

(assuming δ and δ_1 assign registers to all the free variables of $R\sigma$ and $e\sigma$).

Proof. We have

$$R\sigma(s, \delta) \equiv R\frac{\cdots s(\delta(u_i)) \cdots e_i(s, \delta) \cdots s(\delta(z))}{\cdots x_i' \cdots\cdots v_i' \cdots\cdots z},$$

and, using equations a)–c),

$$R(s, \delta_1) \equiv R\frac{\cdots s(\delta(u_i) \cdots e_i(s, \delta) \cdots s(\delta(z))}{\cdots x_i' \cdots\cdots v_i' \cdots\cdots z}.$$

This establishes $R\sigma(s, \delta) \equiv R(s_1, \delta_1)$, and the equation in the lemma is established in the same way.

LEMMA 3. *Suppose (s_1, δ_1) is matched to (s, δ) relative to*

$$\sigma = \frac{\bar{u}, \bar{e}}{\bar{x}', \bar{v}'},$$

and $s' = \mathrm{Out}(A\sigma, s, \delta, \pi)$, and $s_1' = \mathrm{Out}(A, s_1, \delta_1, \pi)$, where A is any statement such that $p(\bar{x}' : \bar{v}')$ **proc** *A could be a legal procedure declaration for a legal statement* **call** *$p(\bar{u} : \bar{e})$. Then (s_1', δ_1) is matched to (s', δ) relative to σ.*

The hypotheses of the lemma imply that no variable in $(\bar{u}, \bar{e})$ (except possibly one in $(\bar{x}', \bar{v}')$) occurs in the free set of A, and no variable in $\bar{v}'$ occurs on the left

side of any assignment statement in A or to the left of the colon in any procedure call statement of A. Also, no variable in $(\bar{x}' : \bar{v}')$ occurs globally in any procedure declaration of a procedure which could be activated by executing A.

We wish to apply the lemma for the case A is

$$K\frac{\bar{x}', \bar{v}'}{\bar{x}, \bar{v}}.$$

The hypotheses of the lemma are satisfied in this case because $p(\bar{x} : \bar{v})$ **proc** K is a legitimate procedure declaration, and both $(\bar{u} : \bar{e})$ and $(\bar{x}' : \bar{v}')$ are legitimate sets of actual parameters for p. These hypotheses were stated explicitly because the proof is by induction on the definition of Comp, and it is important to check that the induction hypothesis can be legally applied to the appropriate statement A' in each case. It seems that all hypotheses stated are necessary for one case or another, in order to push the argument through.

For most cases in the definition of Comp the argument is straightforward. For example, in the case of conditional and while statements, we can apply Lemma 2 to see that $R\sigma(s, \delta) \equiv R(s_1, \delta_1)$, so the same branch of the conditional and the same case of the while definition will apply for both A and $A\sigma$.

The case of a procedure call statement is more subtle. Suppose A is **call** $p'(\bar{u}' : \bar{e}')$, where the procedure declaration for p' is $p'(\bar{x}'' : \bar{v}'')$ **proc** K'. In order to apply the induction hypothesis, we must verify that

$$A' = K'\frac{\bar{u}', \bar{e}'}{\bar{x}'', \bar{v}''}$$

satisfies the hypotheses of the lemma. First, the free set of A' can have no variable in $(\bar{u}, \bar{e})$ (other than one in $(\bar{x}', \bar{v}')$), by the hereditary nature of the definition of "free set". Second, no variable v_i' in $\bar{v}'$ can occur either in $\bar{u}'$ or globally in the procedure declaration of p', so by our restrictions on procedure call statements, v_i' cannot occur to the left of any assignment statement or to the left of the colon in any procedure call statement in A'. Third, A' satisfies the final hypothesis of Lemma 3 because A does. Hence the induction hypothesis applies to A'. By definition of Comp,

$$s_1' = \mathrm{Out}(A, s_1, \delta_1, \pi) = \mathrm{Out}(A', s_1, \delta, \pi)$$

and

$$s' = \mathrm{Out}(A\sigma, s, \delta, \pi) = \mathrm{Out}\left(K'\frac{\bar{u}'\sigma, \bar{e}'\sigma}{\bar{x}'', \bar{v}''}, s, \delta, \pi\right).$$

Since no variable in $\bar{x}'$ or $\bar{v}'$ occurs globally in the procedure declaration for p', it follows that

$$K'\frac{\bar{u}'\sigma, \bar{e}'\sigma}{\bar{x}'', \bar{v}''} = K'\frac{\bar{u}', \bar{e}'}{\bar{x}'', \bar{v}''}\sigma.$$

Therefore

$$s' = \mathrm{Out}(A'\sigma, s, \delta, \pi).$$

Hence, by the induction hypothesis, (s_1', δ_1) is matched to (s', δ) relative to σ. This completes the case of procedure call statements.

In the case of an assignment statement, A is simply $x := e$, $A\sigma$ is $x\sigma := e\sigma$,

$$s'(X_i) = \begin{cases} s(X_i) & \text{if } \delta(x\sigma) \neq X_i, \\ e\sigma(s, \delta) & \text{if } \delta(x\sigma) = X_i \end{cases}$$

and

$$s_1'(X_i) = \begin{cases} s_1(X_i) & \text{if } \delta_1(x) \neq X_i, \\ e(s_1, \delta_1) & \text{if } \delta_1(x) = X_i. \end{cases}$$

To check condition a) for (s_1', δ_1), (s', δ) we note that if z is not x then z is unchanged by the assignment, so a) for the pair (s_1', δ_1), (s', δ) follows from our assumption a) for the pair (s_1, δ_1), (s, δ). If z is x, then $s_1'(\delta_1(z)) = e(s_1, \delta_1) = e\sigma(s, \delta)$ (by Lemma 2) $= s'(\delta(z\sigma)) = s'(\delta(z))$. Condition b) is proved similarly, but it is necessary to use the facts that the u_i are distinct and that the assignment statement A cannot be $u_i := e$, (unless u_i is some x_j') because no variable in $(\bar{u}, \bar{e})$ (other than one in $(\bar{x}', \bar{v}')$) is global in A. To verify condition c), we note that v_i' is not on the left side of the assignment statement A, and no variable in e_i is on the left side of the assignment statement $A\sigma$ by our restrictions that actual parameters are not global in A and no u_j can occur in e_i. Therefore the values of v_i' and e_i remain unchanged by A and $A\sigma$, respectively. Condition d) is obviously unaffected by the assignment statement.

The remaining troublesome case in the proof of Lemma 3 is that of variable declaration in the definition of Comp. In this case, A is **begin new** x; D^*; A^*; **end**, and we again assume (s_1, δ_1) is matched to (s, δ) relative to σ. We can assume that the variable x being declared does not occur in $(\bar{u}, \bar{e})$, because of our convention for renaming locally declared variables in $A\sigma$ explained before the definition of Comp. If x is in $(\bar{x}', \bar{v}')$, then let σ' be σ with the substitution for x deleted. Otherwise, let $\sigma' = \sigma$. Note that $A\sigma = A\sigma'$, because x is not free in A. We claim (s_1, δ_1') is matched to (s, δ') relative to σ', where δ_1' and δ' are the variable assignments determined from δ_1 and δ, respectively, in the first clause in the definition of Comp. The claim is staightforward to verify, using in particular condition d) from the definition of "(s_1, δ_1) is matched to (s, δ) relative to σ," to verify condition a). From the claim and the easily verified fact that the induction hypothesis applies to $A' =$ **begin** D^*; A^* **end** we can conclude (s_1', δ_1') is matched to (s', δ') relative to σ'. From this we can conclude (s_1', δ_1) is matched to (s', δ) relative to σ, where we must also use the fact that (s_1, δ_1) is matched to (s, δ) relative to σ and the contents of the register $\delta_1(x)$ (respectively $\delta(x)$) remains unchanged during the computation of $A\sigma$ under δ_1 and s_1 (respectively A under δ and s). This completes the proof of Lemma 3.

Using Lemmas 2 and 3 it is easy to complete the proof of the validity of the parameter substitution rule. We assume equations (1) to (6) hold, and select (s_1, δ_1) to match (s, δ) relative to σ. By Lemma 2 and our assumption (3) that $P\sigma(s, \delta)$ is true, it follows that $P(s_1, \delta_1)$ is true. If we set

$$s_1' = \text{Out}(\textbf{call } p(\bar{x}' : \bar{v}'), s_1, \delta_1, \pi), \tag{7}$$

then by our assumption (1), $Q(s_1', \delta_1)$ is true. On the other hand, by (7) and the definition of Comp we have

$$s_1' = \mathrm{Out}\left(K\frac{\bar{x}', \bar{v}'}{\bar{x}, \bar{v}}, s_1, \delta_1, \pi\right). \tag{8}$$

If we now take A in Lemma 3 to be

$$K\frac{\bar{x}', \bar{v}'}{\bar{x}, \bar{v}}$$

and note that then

$$K\frac{\bar{u}, \bar{e}}{\bar{x}, \bar{v}} = A\sigma,$$

it follows (using (5) and (8)) that (s_1', δ_1) is matched to (s', δ) relative to σ. Hence by Lemma 2, $Q\sigma(s', \delta)$ is true..This establishes the truth of the conclusion of rule 9), and hence the validity of rule 9).

Let us now verify rule 10), the rule of variable substitution. This rule can just as easily be verified in the more general setting

$$\frac{P\{A\}Q}{P\sigma\{A\}Q\sigma}, \quad \text{where } \sigma = \frac{\bar{z}'}{\bar{z}}$$

is a substitution of expressions for variables such that no variable in $\bar{z}$ or $\bar{z}'$ occurs in the free set of A. The reason we selected a special case of this rule for rule 10) is that this special case is precisely what is needed to prove completeness of the system $\mathcal{H}$.

To verify the more general rule, assume the premise $P\{A\}Q$, and suppose $P\sigma(s, \delta)$ is true for some state s and variable assignment δ. Let $\bar{z} = z_1, \cdots, z_k$ and $\bar{z}' = z_1', \cdots, z_k'$. Let the state s_1 be given by

$$s_1(\delta(y)) = s(\delta(y)) \quad \text{for all } y \text{ not in } \bar{z} \text{ for which } \delta(y) \text{ is defined,}$$

$$s_1(\delta(z_i)) = z_i'(s, \delta), \quad 1 \leqq i \leqq k,$$

$$s_1(X_j) = s(X_j) \quad \text{for all registers } X_j \text{ not in the range of } \delta.$$

Then (s_1, δ) is matched to (s, δ) relative to σ, in the sense defined before Lemma 2. Hence by Lemma 2, $P(s_1, \delta)$ is true. Let $s' = \mathrm{Out}(A\sigma, s, \delta, \pi)$ and $s_1' = \mathrm{Out}(A, s_1, \delta, \pi)$. Then the hypotheses of Lemma 3 are easily verified, so (s_1', δ) is matched to (s', δ) relative to σ. Since $P(s_1, \delta)$ is true and $P\{A\}Q$ holds, $Q(s_1', \delta)$ is true. By Lemma 2, $Q\sigma(s', \delta)$ is true. Since $A\sigma = A$, this establishes the truth of $P\sigma\{A\}Q\sigma$, and hence rule 10) is valid.

All other rules can be verified directly from the corresponding clause in the definition of Comp. This completes the proof of Theorem 1.

Using Lemma 3 we can give a short proof that the truth of a formula $P\{A\}Q$, as defined at the beginning of this section, is independent of the choice of δ. Of course it would have been more logical to prove this immediately after the definition, and in fact Lemmas 1 through 3 could have been proven there. However, Lemma 3 would have been very difficult to motivate.

LEMMA 4. *The truth of $P\{A\}Q$ is independent of the choice of δ.*

Proof. Suppose δ_1 and δ_2 both assign registers to the free variables of formulas P and Q, and the free set of statement A, and π makes the proper procedure assignments for A. Suppose for all states s_1 and s_1', if $P(s_1, \delta_1)$ is true and $s_1' = \text{Out}(A, s_1, \delta_1, \pi)$, then $Q(s_1', \delta_1)$ is true. We wish to show the same can be said with δ_1 replaced by δ_2. Hence let s_2 be any state, and suppose $P(s_2, \delta_2)$ is true. Choose a state s_1 such that (s_1, δ_1) is matched to (s_2, δ_2) relative to the empty (or identity) substitution σ_0. Then by Lemma 2, $P(s_1, \delta_1)$ is true. Let $s_2' = \text{Out}(A\sigma_0, s_2, \delta_2, \pi)$ (note that $A\sigma_0 = A$). Then the hypotheses of Lemma 3 are satisfied, since the lists $\bar{u}$, $\bar{e}$, $\bar{x}$, $\bar{v}$ are all empty, so (s_1, δ_1) is matched to (s_2, δ_2) relative to σ_0. By Lemma 2, $Q(s_1', \delta_1) \equiv Q(s_2', \delta_2)$, so $Q(s_2', \delta_2)$, is true. This completes the proof of Lemma 4.

6. The question of completeness of the rules. The corollary to Theorem 1 states that if $\mathscr{D}$ is a sound deductive system for the language $\mathscr{L}_2$ relative to an interpretation $\mathscr{I}$, and if $\vdash_{\mathscr{H},\mathscr{D}} P\{A\}Q$, then $P\{A\}Q$ is true in $\mathscr{M}[\mathscr{I}]$. We turn now to the converse question, and ask under what conditions every true formula $P\{A\}Q$ is provable in the system $(\mathscr{H}, \mathscr{D})$.

If we assume $\mathscr{D}$ is an axiomatic deductive system, then the formulas $P\{A\}Q$ provable in the system $(\mathscr{H}, \mathscr{D})$ are recursively enumerable. On the other hand, the formula true $\{A\}$ false is true in $\mathscr{M}[\mathscr{I}]$ iff A fails to halt for all initial values of its global variables. Therefore the true formulas cannot be recursively enumerable in case $\mathscr{L}_1$, $\mathscr{L}_2$ and $\mathscr{I}$ are such that the halting problem for $\text{Al}[\mathscr{L}_1, \mathscr{L}_2]$ is recursively unsolvable. In particular, we have

THEOREM 2. *If $\mathscr{L}_1$ is $\mathscr{L}_N$ (or $\mathscr{L}_+$: i.e., $\mathscr{L}_N$ without multiplication) and $\mathscr{D}$ is an axiomatic deductive system for $\mathscr{L}_2$, then $(\mathscr{H}, \mathscr{D})$ is incomplete in the sense that there is a formula $P\{A\}Q$ true in $\mathscr{M}[\mathscr{I}]$, but such that not $\vdash_{\mathscr{H},\mathscr{D}} P\{A\}Q$, where $\mathscr{I}$ includes the standard interpretation in the natural numbers for $\mathscr{L}_1$.*

On the other hand, one has a feeling that the axioms and rules 1)–11) (or small modifications of 1)–11)) are complete in some sense, and the incompleteness is probably due to the incompleteness of the system $\mathscr{D}$. But there is another way in which the system can fail to be complete, and that it is if the assertion language $\mathscr{L}_2$ is not powerful enough to express invariants for the loops. Let us fix the language $\mathscr{L}_1$, $\mathscr{L}_2$ and the interpretation $\mathscr{I}$ with domain $\mathscr{D}$. Suppose $P \in \mathscr{L}_2$ and A is a statement of $\text{Al}[\mathscr{L}_1, \mathscr{L}_2]$, and $\bar{x} = (\bar{x}_1, \cdots, x_n)$ is a list of all variables occurring either free in P or in the free set of A. Then we say the *post relation corresponding to* P and A is the relation $\bar{Q}(x_1, \cdots, x_n)$ on D such that $\bar{Q}(d_1, \cdots, d_n)$ is true iff there is a state s and variable assignment δ to $x_1, \cdots, x_n$ such that $d_i = s'(\delta(x_i))$, $i = 1, \cdots, n$, where $s' = \text{Out}(A, s, \delta, \pi)$, $P(s, \delta)$ is true, and π is appropriate to the context of A in the program. The formula Q in $\mathscr{L}_2$ *expresses* the relation $\bar{Q}$ iff Q has free variables $x_1, \cdots, x_n$, and

$$\mathscr{I}\left(Q\frac{d_1, \cdots, d_n}{x_1, \cdots, x_n}\right) \Leftrightarrow \bar{Q}(d_1, \cdots, d_n)$$

for all $d_1, \cdots, d_n \in D$. We let 'post(P, A)' denote a particular formula in $\mathscr{L}_2$ (say the one with the least Gödel number) which expresses the post relation corresponding to P and A.

DEFINITION. The language $\mathscr{L}_2$ is *expressive* (relative to $\mathscr{L}_1$ and $\mathscr{I}$) iff

(i) "=" is in $\mathscr{L}_1$ and receives its standard interpretation in $\mathscr{I}$,

(ii) For every formula P in $\mathscr{L}_2$ and statement A there is a formula Q in $\mathscr{L}_2$ which expresses the post relation corresponding to P and A.

LEMMA 5. *$\mathscr{L}_N$ is expressive (relative to $\mathscr{L}_N$, $\mathscr{L}_N$, and $\mathscr{I}_N$).*

Proof. Given P and A, then roughly speaking the post relation $\bar{Q}$ is true of numbers $a_1, \cdots, a_n$ iff there are initial values $b_1, \cdots, b_n$ satisfying P such that A will terminate with these as initial values, and the final values of $(x_1, \cdots, x_n)$ are $(a_1, \cdots, a_n)$. Since A describes a partial recursive function and partial recursive functions are expressible in $\mathscr{L}_N$, the lemma follows.

We remark that $\mathscr{L}_+$ (i.e., $\mathscr{L}_N$ without multiplication) is *not* expressive (relative to $\mathscr{L}_+$, $\mathscr{L}_+$, and $\mathscr{I}_+$) because every recursively enumerable (r.e.) set is the post condition corresponding to $0 = 0$ and some A. On the other hand, truth in $\mathscr{L}_+$ is decidable (by Presburger's result), so not every r.e. set is expressible in $\mathscr{L}_+$.

Let us say that the proof system $\mathscr{D}$ for $\mathscr{L}_2$ is *semantically complete relative to* $\mathscr{I}$ iff: a formula P of $\mathscr{L}_2$ is provable in $\mathscr{D}$ iff its universal closure is true under $\mathscr{I}$. Of course by the Gödel Incompleteness Theorem, no axiomatic system $\mathscr{D}$ for number theory can be semantically complete, but for the purpose of stating a completeness theorem for $\mathscr{H}$, we shall assume we have a complete nonaxiomatizable system.

THEOREM 3 (Completeness of $\mathscr{H}$). *Let $\mathscr{T}$ be a semantically complete proof system for $\mathscr{L}_2$ (relative to $\mathscr{I}$) and suppose $\mathscr{L}_2$ is expressive relative to $\mathscr{L}_1$ and $\mathscr{I}$. Then $\vdash_{\mathscr{H},\mathscr{T}} P\{A\}Q$ whenever $\models_{\mathscr{M}} P\{A\}Q$.*

COROLLARY. *Let $\mathscr{T}$ be a complete (noneffective) proof system for $\mathscr{L}_N$. Then $\vdash_{\mathscr{H},\mathscr{T}} P\{A\}Q$ if and only if $P\{A\}Q$ is true in $\mathscr{M}[\mathscr{I}_N]$.*

Proof of theorem. Given a statement A (part of a larger program) let A' be the result of substituting all procedure bodies, with formal parameters replaced by actual parameters and local variables renamed where necessary, for procedure calls repeatedly until no procedure calls remain. This process terminates because of our outlawing of recursive procedures. The theorem is proved by induction on the sum of the length of A' and the number of procedure body substitutions necessary to convert A to A'. If A is not a procedure call statement, then exactly one of the rules 1)–7) can be applied nicely (sometimes with rule 11), the rule of consequence) to prove A from previously proved statements. Rules 8), 9), and 10) are needed for procedure call statements. We will discuss several of the more interesting cases.

a) *Compound statements.* Suppose $P\{\textbf{begin } A\,;\, A^* \textbf{ end}\}R$ is true in $\mathscr{M}[\mathscr{I}]$. Let Q be a formula expressing the post relation corresponding to P and A. Then by the definitions involved, it is easy to see that $P\{A\}Q$ and $Q\{\textbf{begin } A^* \textbf{ end}\}R$ are both true, and so both are provable in the system $(\mathscr{H}, \mathscr{T})$ by the induction hypothesis. Therefore by rule 3),

$$\vdash_{\mathscr{H},\mathscr{T}} P\{\textbf{begin } A\,;\, A^* \textbf{ end}\}R.$$

The case $P\{\textbf{begin end}\}Q$ is handled by using rule 4) and the rule of consequence.

b) *Assignment statements.* Suppose $P\{x := e\}Q$ is true. Then the universal

closure of

$$P \supset Q\frac{e}{x}$$

must be true under the interpretation $\mathcal{I}$, so

$$\vdash_{\mathcal{T}} P \supset Q\frac{e}{x}.$$

But

$$\vdash_{\mathcal{K},\mathcal{T}} Q\frac{e}{x}\{x := e\}Q$$

by axiom 5), and $\vdash_{\mathcal{T}} Q \supset Q$, so $\vdash_{\mathcal{K},\mathcal{T}} P\{x := e\}Q$ by the rule of consequence.

c) *While statements.* Suppose $P\{\textbf{while } R \textbf{ do } A \textbf{ od}\}Q$ is true in $\mathcal{M}[\mathcal{I}]$. In order to apply rule 7), the rule of **while** statements, we must find a loop invariant P_1 with the properties that $P_1 \,\&\, R\{A\}P_1$ is true, and (the universal closures of) $P \supset P_1$ and $(P_1 \,\&\, \neg R) \supset Q$ are true. The induction hypothesis can then be applied to assume $\vdash_{(\mathcal{K},\mathcal{T})} P_1 \,\&\, R\{A\}P_1$, and by the completeness of $\mathcal{T}$, $\vdash_{\mathcal{T}} P \supset P_1$ and $\vdash_{\mathcal{T}} (P_1 \,\&\, \neg R) \supset Q$. Hence by rules 7) and 11), $\vdash_{\mathcal{K},\mathcal{T}} P\{\textbf{while } R \textbf{ do } A \textbf{ od}\}Q$.

Let $y_1, \cdots, y_n$ be a list of the variables in the free set of A, together with the free variables in P, R, and Q. We will construct the loop invariant P_1 with free variables $y_1, \cdots, y_n$ such that $P_1(d_1, \cdots, d_n)$ holds iff there are initial values $d_1', \cdots, d_n'$ for $y_1, \cdots, y_n$ such that $P(d_1', \cdots, d_n')$ is true, and after some finite number of passes through the while loop (i.e. after A has been executed some finite number of times with the condition R satisfied before each time) the values of $y_1, \cdots, y_n$ will be $d_1, \cdots, d_n$. More precisely, P_1 is equivalent to the infinite disjunction $Q_1 \vee Q_2 \vee \cdots$, where Q_1 is P and Q_{i+1} is $\text{post}(Q_i \,\&\, R, A)$, $i = 1, 2, \cdots$. The reader can easily verify that if such a finite P_1 can be found, then the conditions in the previous paragraph are satisfied. But in fact, P_1 is just $\exists z_1 \cdots \exists z_n P_2$, where P_2 expresses the post relation corresponding to P and

$$\textbf{while } (R \,\&\, (y_1 \neq z_1 \vee y_2 \neq z_2 \vee \cdots \vee y_n \neq z_n)) \textbf{ do } A \textbf{ od},$$

and $z_1, \cdots, z_n$ are new variables. P_2 is in the language $\mathcal{L}_2$, by our assumption that $\mathcal{L}_2$ is expressive. This completes case c).

d) *Procedure calls.* Suppose $P\{\textbf{call } p(\bar{u} : \bar{e})\}Q$ is true, where the procedure declaration for p is $p(\bar{x} : \bar{v})$ **proc** K. By definition of Comp,

$$P\left\{K\frac{\bar{u}, \bar{e}}{\bar{x}, \bar{v}}\right\}Q$$

is true. The naive argument, which only works if there are no variable clashes, is the following. Since

$$\bar{v} = \bar{e} \,\&\, P\frac{\bar{x}}{\bar{u}}\{K\}Q\frac{\bar{x}}{\bar{u}}$$

is true, it is provable by the induction hypothesis. By the rule of procedure calls

(rule 8)),

$$\bar{v} = \bar{e} \;\&\; P\frac{\bar{x}}{\bar{u}}\{\textbf{call}\ p(\bar{x} : \bar{v})\}\ Q\frac{\bar{x}}{\bar{u}}$$

is provable. By the rule of parameter substitution (rule 9)), $\bar{e} = \bar{e}\ \&\ P\{\textbf{call}\ p(\bar{u} : \bar{e})\}Q$ is provable. Finally $P\{\textbf{call}\ p(\bar{u} : \bar{e})\}Q$ is provable by the rule of consequence.

The difficulties with this argument are first, the formal and actual parameters might have variables in common, and second, P and Q may have occurrences of the formal parameters even if the first condition does not hold. To handle the second problem, we let τ be a substitution which assigns distinct new variables to all variables in $(\bar{x}, \bar{v})$ which do not occur in $(\bar{u}, \bar{e})$. We will concentrate on showing $P\tau\{\textbf{call}\ p(u : e)\}Q\tau$ is provable, and then apply the rule of variable substitution. Since the variables renamed by τ do not occur in the free set of **call** $p(\bar{u} : \bar{e})$, and since $P\{\textbf{call}\ p(\bar{u} : \bar{e})\}Q$ is true, it is intuitively clear (and follows formally from Lemmas 2 and 3) that

(1′) $$\models P\tau\{\textbf{call}\ p(\bar{u} : \bar{e})\}Q\tau.$$

To handle the first problem (along with the second problem) let $\bar{f}$ be a list of the variables occurring in the expressions $\bar{e}$, and let $\bar{f}'$ be a list of distinct new variables of the same length, and let $\bar{e}'$ be the result of substituting these new variables for the old in $\bar{e}$. By definition of Comp and (1′),

$$P\tau\left\{K\frac{\bar{u}, \bar{e}}{\bar{x}, \bar{v}}\right\}Q\tau$$

is true. We will show $P_1\{K\}Q_1$ is true, where

$$P_1 \text{ is } \bar{v} = \bar{e}' \;\&\; P\tau\frac{\bar{x}, \bar{f}'}{\bar{u}, \bar{f}}, \quad \text{and} \quad Q_1 \text{ is } Q\tau\frac{\bar{x}, \bar{f}'}{\bar{u}, \bar{f}}.$$

Suppose s_1 and δ_1 are state and variable assignments such that $P_1(s_1, \delta_1)$ is true, and δ_1 assigns registers to the free set of K and the free variables of P_1 and Q_1, and to no other variables. Thus δ_1 is not defined for any variable in $(\bar{u}, \bar{e})$ unless that variable is also in $(\bar{x}, \bar{v})$. Hence we can find s and δ so that

A) $s(\delta(z)) = s_1(\delta_1(z))$ for all variables $z \notin (\bar{x}, \bar{v})$, if $\delta_1(z)$ is defined,

B) $s(\delta(u_i)) = s_1(\delta_1(x_i))$, $i = 1, \cdots, m$,

C) $s(\delta(f)) = s_1(\delta_1(f'))$ for each variable f in $\bar{e}$ and corresponding f' in $\bar{e}'$,

and further, condition d), stated before Lemma 2, is satisfied. Hence (s_1, δ_1) is matched to (s, δ) relative to

$$\sigma = \frac{\bar{u}, \bar{e}}{\bar{x}, \bar{v}}.$$

[Condition c) is satisfied by condition C) above, and the fact that $P_1(s_1, \delta_1)$ is true, and P_1 includes the conjunct $\bar{v} = \bar{e}'$.] Thus by Lemma 3, (s_1', δ_1) is matched to (s', δ) relative to σ, where $s' = \text{Out}(K\sigma, s, \delta, \pi)$ and $s_1' = \text{Out}(K, s_1, \delta_1, \pi)$. Now by comparing the values (s_1, δ_1) given to variables in P_1 with those (s, δ) gives to the variables in $P\tau$, it is easy to see $P\tau(s, \delta)$ is true. Since $P\tau\{K\sigma\}Q\tau$ is true, it follows

that $Q\tau(s', \delta)$ is true. Again by comparing values assigned to variables, and noting property C) above holds when s and s_1 are replaced by s' and s_1' respectively (since K and $K\sigma$ leave the values of $\bar{f}'$ and $\bar{f}$ unchanged), we see that $Q_1(s_1', \delta_1)$ is true. This completes the proof that $P_1\{K\}Q_1$ is true.

By the induction hypothesis, $P_1\{K\}Q_1$ is provable. By the rule of procedure calls, $P_1\{\textbf{call } p(\bar{x} : \bar{v})\}Q_1$ is provable. By the rule 9) of parameter substitution, we can rename $\bar{v}$ by $\bar{v}'$ and rename $\bar{x}$ by $\bar{x}'$ (these new variables are distinct from the ones τ assigns) to obtain that

$$\bar{v}' = \bar{e}' \;\&\; P\tau\frac{\bar{x}', \bar{f}'}{\bar{u}, \bar{f}}\left\{\textbf{call } p(\bar{x}' : \bar{v}')\right\}Q\tau\frac{\bar{x}', \bar{f}'}{\bar{u}, \bar{f}}$$

is provable. Again by the rule 9) of parameter substitution and the substitution

$$\frac{\bar{f}}{\bar{f}'},$$

we obtain that

$$\bar{v}' = \bar{e} \;\&\; P\tau\frac{\bar{x}'}{\bar{u}}\{\textbf{call } p(\bar{x}' : \bar{v}')\}Q\tau\frac{\bar{x}'}{\bar{u}}$$

is provable. Again by the rule 9) of parameter substitution and the substitution

$$\frac{\bar{u}, \bar{e}}{\bar{x}', \bar{v}'}$$

we obtain that

$$\bar{e} = \bar{e} \;\&\; P\tau\{\textbf{call } p(\bar{u} : \bar{e})\}Q\tau$$

is provable. Finally, conjunct $\bar{e} = \bar{e}$ can be removed by the rule of consequence, and the rule 10) of variable substitution with substitution τ^{-1} can be applied to prove $P\{\textbf{call } p(\bar{u} : \bar{e})\}Q$.

Acknowledgments. I am grateful to Jim Donahue and Bob Constable for reading and pointing out errors in earlier versions of this paper. Also, Gerry Gorelick, whose M.Sc. thesis [5] attempts to extend these results to the case of recursive procedures, spent a great deal of time thinking about the ideas presented here and his comments and criticisms were especially helpful.

REFERENCES

[1] S. A. COOK AND D. C. OPPEN, *An assertion language for data structures*, Conference Record of the Second ACM Symposium on Principles of Programming Languages (Palo Alto, CA), Jan. 1975, pp. 160–166.

[2] S. A. COOK, *Axiomatic and interpretive semantics for an ALGOL fragment*, Tech. Rep. 79, Dept. of Computer Sci., Univ. of Toronto, Feb. 1975.

[3] E. M. CLARKE, JR., *Completeness and incompleteness theorems for Hoare-like axiom systems*, Ph.D. thesis, Cornell Univ., Ithaca, NY, 1976.

[4] J. E. DONAHUE, *Complementary definitions of programming language semantics*, Computer Systems Research Group Tech. Rep. CSRG-62, Univ. of Toronto, Nov. 1975.

[5] G. A. GORELICK, *A complete axiomatic system for proving assertions about recursive and non-recursive programs*, Tech. Rep. 75, Dept. of Computer Sci., Univ. of Toronto, Feb. 1975.

[6] C. A. R. HOARE, *An axiomatic basis for computer programming*, Comm. ACM, 12 (1969), pp. 576–580.

[7] ———, *Procedures and parameters: An axiomatic approach*, Symposium on Semantics of Algorithmic Languages, E. Engeler, ed., Springer-Verlag, Berlin, 1971, pp. 102–116.

[8] C. A. R. HOARE AND P. E. LAUER, *Consistent and complementary formal theories of the semantics of programming languages*, Acta Informat., 3 (1974), pp. 135–153.

[9] S. IGARASHI, R. L. LONDON AND D. C. LUCKHAM, *Automatic program verification I: A logical basis and its implementation*, Tech. Rep. AIM-200, Artificial Intelligence Laboratory, Stanford Univ., Stanford, CA, July 1973.

[10] P. E. LAUER, *Consistent formal theories of the semantics of programming languages*, Tech. Rep. TR 25.121, IBM Laboratory, Vienna, Nov. 1971.

[11] D. C. OPPEN, *On logic and program verification*, Tech. Rep. 82, Dept. of Computer Sci., University of Toronto, April 1975.

[12] D. C. OPPEN AND S. A. COOK, *Proving assertions about programs that manipulate data structures*, Proceedings of Seventh Annual ACM Symposium on Theory of Computing (Albuquerque, NM), May 1975, pp. 107–116.

[13] S. OWICKI, *A consistent and complete deductive system for the verification of parallel programs*, Proceedings Eighth Annual ACM Symposium on Theory of Computing (Hershey, PA), May 1976, pp. 73–86.

[14] V. R. PRATT, *Semantical considerations on Floyd–Hoare logic*, 17th Annual IEEE Symposium on Foundations of Computer Science (Houston, TX), Oct. 1976.