Ten Years of Hoare's Logic: A Survey—Part I

KRZYSZTOF R. APT
Erasmus University

A survey of various results concerning Hoare's approach to proving partial and total correctness of programs is presented. Emphasis is placed on the soundness and completeness issues. Various proof systems for **while** programs, recursive procedures, local variable declarations, and procedures with parameters, together with the corresponding soundness, completeness, and incompleteness results, are discussed.

Key Words and Phrases: Hoare's logic, partial correctness, total correctness, soundness, completeness in the sense of Cook, expressiveness, arithmetical interpretation, **while** programs, recursive procedures, variable declarations, subscripted variables, call-by-name, call-by-value, call-by-variable, static scope, dynamic scope, procedures as parameters
CR Category: 5.24

1. INTRODUCTION

In 1969 Hoare [27] introduced an axiomatic method of proving programs correct. This approach was partially based on the so-called intermediate assertion method of Floyd [18]. Hoare's approach has received a great deal of attention during the last decade, and it has had a significant impact upon the methods of both designing and verifying programs. It has also been used as a way of specifying semantics of programming languages (see [17, 28, 40]).

The purpose of this paper is to present the most relevant issues pertaining to Hoare's method (namely, those of soundness and completeness) in a systematic and self-contained way. The main problem with such an exposition is that various proofs given in the literature are awkward, incomplete, or even incorrect. In many cases proof rules are introduced without any proofs of soundness or completeness at all. The field itself is enormous, since for virtually all programming constructs and notions some proof rules have been suggested. Also, for some constructs, such as recursive procedures with parameters, several alternative proof rules have been proposed.

Faced by these problems, we decided to restrict the exposition to only those constructs and notions which we found most important. In each case we selected only one, hopefully the most successful, among many possible proof systems.

This paper is an extended version of a paper presented at the Fifth Scandinavian Logic Symposium, Aalborg, Denmark, January 17-19, 1979.
Author's present address: LITP, Université Paris 7, 2, place Jussieu, 75221 Paris, France.

The choice of a semantics for the programming constructs concerned turns out to be the decisive factor for the complexity of the proofs. Therefore, we are at great pains to suggest in each case a semantics which would make the soundness and completeness proofs as simple as possible. The papers we are referring to do not necessarily provide proofs using the same semantics. However, in practically all cases the proofs can be straightforwardly translated (and simplified) into our framework. We refrain from pointing out mistakes and errors in the referenced papers. Many of them can be repaired easily, and many others cannot occur in the suggested semantical framework.

We found it convenient to divide the subject material in accordance with the constructs of programming languages: **while** statements (Section 2), recursive procedures (Section 3), local variables (Section 4), subscripted variables (Section 5), parameter mechanisms (Section 6), and procedures as parameters (Section 7). Of course, procedures are but another parameter mechanism. However, procedures as parameters deserve a separate treatment owing to the extensive results concerning them.

Several other important constructs which are also covered by Hoare's method, such as go-to's, coroutines, functions, data structures, and parallelism, are not treated in this paper. Those interested in the issues raised by these constructs are referred to [14, chap. 10] (written by A. de Bruin) and to [8, 11, 12, 25, 46, 47]. To the reader interested in a more detailed development of the subject we suggest [14]. The second part of this survey, to be contained in a separate paper, will be devoted to a discussion of various Hoare-like proof systems for nondeterministic and parallel programs.

It should be mentioned that there are several other approaches to program verification which are related to Hoare's method. These approaches are not discussed in this paper. The interested reader is referred to [22], where other methods are discussed.

Throughout the paper we assume that the reader has knowledge of some basic notions and facts from mathematical logic. We state them whenever they are applied. All of them can be found in, for example, [51].

The title of this paper was perhaps appropriate at the moment of its submission but is not so appropriate now that it appears in print. We decided to retain this title, but to keep the paper up-to-date we took the liberty of incorporating here a few results proved since 1979. Most of them concern the use of procedures as parameters and form the contents of Section 7. The reader deserves a warning that that section deals with the most complex results obtained in this area. For a proper understanding of them, a thorough knowledge of all other sections of the paper is required. Due to the lack of space, the presentation of Section 7 is rather sketchy, and no examples are provided.

2. **while** PROGRAMS

Let L denote a first-order language with equality. We use the letters a, b, x, y, z to denote the variables of L, the letters s, t to denote terms (*expressions*) of L, the letter e to denote a quantifier-free formula (a *Boolean expression*) of L, and, finally, the letters p, q, r to denote the formulas (*assertions*) of L.

Denote by $\mathcal{S}$ the least class of programs such that

1. for every variable x and expression t, $x := t \in \mathscr{S}$; and
2. if $S, S_1, S_2 \in \mathscr{S}$, then $S_1; S_2 \in \mathscr{S}$ and, for every Boolean expression e, **if** e **then** S_1 **else** S_2 **fi** $\in \mathscr{S}$ and **while** e **do** S **od** $\in \mathscr{S}$.

The elements of $\mathscr{S}$ are called ***while** programs*.

2.1 The Proof System

The basic formulas of Hoare's logic are constructs of the form $\{p\}\ S\ \{q\}$ (called *asserted programs*) where p, q are assertions and $S \in \mathscr{S}$. The formulas are not subject to Boolean operations. The intuitive meaning of the construct $\{p\}\ S\ \{q\}$ is as follows: whenever p holds *before* the execution of S and S terminates, then q holds after the execution of S.

Hoare's logic is a system of formal reasoning about the asserted programs. Its axioms and proof rules are the following.

AXIOM 1: ASSIGNMENT AXIOM

$$\{p[t/x]\}\ x := t\ \{p\}.$$

RULE 2: COMPOSITION RULE

$$\frac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}.$$

RULE 3: **if-then-else** RULE

$$\frac{\{p \wedge e\}\ S_1\ \{q\},\ \{p \wedge \neg e\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ e\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}.$$

RULE 4: **while** RULE

$$\frac{\{p \wedge e\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ e\ \mathbf{do}\ S\ \mathbf{od}\ \{p \wedge \neg e\}}.$$

As usual, $p[t/x]$ stands for the result of substituting t for the free occurrences of x in p.

2.2 An Example of a Proof

As a typical example of a proof in the system, take for L the language of Peano arithmetic augmented with the minus operation and consider the program S_0 computing the integer division of two natural numbers x and y:

$a := 0;\ b := x;\ \mathbf{while}\ b \geq y\ \mathbf{do}\ b := b - y;\ a := a + 1\ \mathbf{od}.$

We now prove that

$$\{x \geq 0 \wedge y \geq 0\}\ S_0\ \{a \cdot y + b = x \wedge 0 \leq b < y\}, \tag{1}$$

that is, that

> if x, y are nonnegative integers and S_0 terminates, then a is the integer quotient of x divided by y and b is the remainder. (*)

The proof runs as follows. By the assignment axiom,

$$\{0 \cdot y + x = x \wedge x \geq 0\}\ a := 0\ \{a \cdot y + x = x \wedge x \geq 0\} \tag{2}$$

and

$$\{a \cdot y + x = x \wedge x \geq 0\}\ b := x\ \{a \cdot y + b = x \wedge b \geq 0\}; \tag{3}$$

so, by the composition rule,

$$\{0 \cdot y + x = x \wedge x \geq 0\}\ a := 0;\ b := x\ \{a \cdot y + b = x \wedge b \geq 0\}. \tag{4}$$

Now

$$x \geq 0 \wedge y \geq 0 \rightarrow 0 \cdot y + x = x \wedge x \geq 0 \tag{5}$$

holds; so (5) and (4) imply

$$\{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{a \cdot y + b = x \wedge b \geq 0\}. \tag{6}$$

On the other hand, by the assignment axiom,

$$\begin{gathered}\{(a+1) \cdot y + b - y = x \wedge b - y \geq 0\}\ b := b - y \\ \{(a+1) \cdot y + b = x \wedge b \geq 0\}\end{gathered} \tag{7}$$

and

$$\{(a+1) \cdot y + b = x \wedge b \geq 0\}\ a := a + 1\ \{a \cdot y + b = x \wedge b \geq 0\}; \tag{8}$$

so, by the composition rule,

$$\begin{gathered}\{(a+1) \cdot y + b - y = x \wedge b - y \geq 0\} \\ b := b - y;\ a := a + 1\ \{a \cdot y + b = x \wedge b \geq 0\}.\end{gathered} \tag{9}$$

Now

$$\begin{gathered}a \cdot y + b = x \wedge b \geq 0 \wedge b \geq y \\ \rightarrow (a+1) \cdot y + b - y = x \wedge b - y \geq 0\end{gathered} \tag{10}$$

holds; so (10) and (9) imply

$$\begin{gathered}\{a \cdot y + b = x \wedge b \geq 0 \wedge b \geq y\} \\ b := b - y;\ a := a + 1\ \{a \cdot y + b = x \wedge b \geq 0\}.\end{gathered} \tag{11}$$

(11) implies, by the **while** rule,

$$\begin{gathered}\{a \cdot y + b = x \wedge b \geq 0\}\ \textbf{while}\ b \geq y\ \textbf{do}\ b := b - y; \\ a := a + 1\ \textbf{od}\ \{a \cdot y + b = x \wedge b \geq 0 \wedge b < y\}.\end{gathered} \tag{12}$$

Finally, (6) and (12) imply (1) by the composition rule.

2.3 The Rule of Consequence

Several remarks are in order. First, to justify the above proof we have to explain how we derived (6) from (5) and (4) and derived (11) from (10) and (9). These steps, although intuitively clear, lack a formal basis. The missing proof rule which we used here is the following:

RULE 5: CONSEQUENCE RULE

$$\frac{p \rightarrow p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \rightarrow q}{\{p\}\ S\ \{q\}}.$$

In the example, we used this rule for $q_1 \equiv q$, but in general the above version is needed.

This rule forces us to include assertions among the formulas of Hoare's logic. Denote the resulting system by H.

Now, to get (5) and (10), we have to augment H with a formal proof system concerning assertions. In this particular case, any elementary theory T in the underlying L, in which (5) and (10) can be proved, will do. The proofs of (5) and (10) in T, concatenated with the sequence (2)–(12), (1) of asserted programs or assertions, finally form a proof of (1) in $H \cup T$.

This interpretation is by no means satisfactory for our purposes. We do not care whether (5) and (10) are theorems of a theory T. All we need to know is that (5) and (10) are *true* in the domain of integers.

2.4 Soundness of H

Let A be a set of assertions. Let us write $A \vdash_H \{p\}\ S\ \{q\}$ to denote the fact that there exists a proof of $\{p\}\ S\ \{q\}$ in H which uses as assumptions (for the consequence rule) assertions from A. We have thus shown that (5), (10) $\vdash_H$ (1). The whole idea of the above proof is that we wish to *interpret* (1) as (*). To do this, we must first introduce the notion of the *truth* of an asserted program under an interpretation I of the language L. In this case we choose for I the standard interpretation I_0 of L with the domain of integers.

So let I be an interpretation of L with a nonempty domain D. By a *state* we mean a function assigning to each variable x a value from the domain D. We use the letters δ, τ to denote states.

The relation "under the interpretation I an assertion p is true in a state δ," written as $\models_I p(\delta)$, is defined in the usual way. If for all states $\delta \models_I p(\delta)$ holds, we say that p is *true under* I, written $\models_I p$. With each program $S \in \mathcal{S}$ we can associate a *meaning* $\mathcal{M}_I(S)$ under I, this being a partial function from states to states. It is easy to define $\mathcal{M}_I(S)$ so as to capture the intended meaning of the program.

Having done so, we can finally define the *truth* of an asserted program under I. We say that an asserted program $\{p\}\ S\ \{q\}$ is *true under I* if

$$\text{for all states } \delta, \tau, \text{ if } \models_I p(\delta) \text{ and } \mathcal{M}_I(S)(\delta) = \tau, \text{ then } \models_I q(\tau).$$

This definition is clearly a correct formalization of the informal notion of the truth of $\{p\}\ S\ \{q\}$. We can now safely state that (*) simply says that (1) is true under I_0.

The last step in the justification of (*) is the following. Call an asserted program *valid* if it is true under all interpretations I. Call a proof rule *sound* if for all interpretations I it preserves the truth under I of the asserted programs (and, in the case of the consequence rule, assertions). It is easy to prove that the axioms of H are valid and the proof rules of H are sound.

This fact implies (by induction on the length of proofs) the following theorem, which states that the proof system H is *sound*.

THEOREM 1. *For every interpretation I, set of assertions A, and asserted program φ the following holds: if all assertions from A are true under I and $A \vdash_H \varphi$, then φ is true under I.*

In other words, if $\mathrm{Tr}_I \vdash_H \varphi$, then $\models_I \varphi$, where Tr_I denotes the set of all true assertions under I.

This theorem immediately implies that (1) is true under I_0, because obviously (5) and (10) are true under I_0. (1) is actually true under any interpretation I under which (5) and (10) are true. So, for example, (1) is also true under the standard interpretation in the real numbers or in a finite set of natural numbers $\{x \mid x \le \max\}$.

2.5 Loop Invariants

Like all formal proofs, the proof of (1) is tedious and difficult to follow. We are not accustomed to following a line of reasoning expressed in such small steps. However, it is easy to observe that the whole argument boils down to one crucial step: observing that (11) holds. Once we guess the assertion $r \equiv a \cdot y + b = x \wedge b \ge 0$, to find the proof is a straightforward problem. Since (11) holds, r is called an *invariant* of the loop **while** $b \ge y$ **do** $b := b - y$; $a := a + 1$ **od**. Since (6) holds, we say that the program $a := 0$; $b := x$ *establishes r*. Since (12) holds, we say that the program **while** $b \ge y$ **do** $b := b - y$; $a := a + 1$ **od** *preserves r*.

A concise way of embedding this information into the program S is simply to *annotate* it with the desired assertion(s). To illustrate this point, we now take a different example. It is easy to see that

```
{$x \ge 0 \wedge y \ge 0$} $a := x$; $b := y$; $z := 1$;
                  while $b \ne 0$ do $b := b - 1$; $z := z \cdot a$ od {$z = x^y$}
```

is true under the interpretation I_0 once we write it as

```
{$x \ge 0 \wedge y \ge 0$} $a := x$; $b := y$; $z := 1$;
                  while $b \ne 0$ do {$z \cdot a^b = x^y$} $b := b - 1$; $z := z \cdot a$ od {$z = x^y$}.
```

Thinking in terms of establishing an invariant and preserving it has immediate implications for reasoning about programs and their design. For example, in the case of the above program, an observation that the loop **while** *even*(b) **do** $b := b/2$; $a := a \cdot a$ **od** preserves the invariant $z \cdot a^b = x^y$ leads to the following improvement:

```
{$x \ge 0 \wedge y \ge 0$} $a := x$; $b := y$; $z := 1$;
                  while $b \ne 0$ do {$z \cdot a^b = x^y$}
                                    while even($b$) do {$z \cdot a^b = x^y$}
                                                       $b := b/2$; $a := a \cdot a$
                                                 od;
                  $b := b - 1$; $z := z \cdot a$ od {$z = x^y$}.
```

In both cases it is Theorem 1 which allows us to infer that the asserted programs are true under I_0.

2.6 Termination Not Implied

It is important to note that the above proofs are not concerned with the termination of programs. Even though (1) is true under I_0, we do not have any guarantee that the program S_0 terminates. In fact, in a state in which the value of y is 0, S_0 does not terminate. While defining the meaning of programs, we left room for nontermination by allowing $\mathcal{M}_I(S)$ to be a partial function from states to states.

Actually, the termination of a program is interpretation-dependent. For example, the program **while** $x > 0$ **do** $x := x - 1$ **od** under the interpretation I_0 always terminates (is total), whereas under an interpretation in a nonstandard model of Peano arithmetic it is not total. This simple remark has direct consequences concerning the existence of sound proof systems dealing with termination, as we see in Section 2.11.

2.7 The Issue of Completeness of *H*

A natural question of not only theoretical interest is that of the completeness of the proof system H. The question of soundness concerns the correctness of the method, whereas the question of completeness concerns the scope of its applicability (under what circumstances it can be successfully applied).

The system H alone is obviously incomplete: an asserted program $\{p[t/x]\}$ $x := t$ $\{\mathbf{true}\}$ is true under every interpretation and yet is unprovable in H; there is no way to prove in H the formula $p \rightarrow \mathbf{true}$.

Supplementing H by an axiomatic system T dealing with assertions is of no help. For any axiomatic system G the set of asserted programs provable in G is recursively enumerable (r.e.) But for the language L_P of Peano arithmetic with its standard interpretation I_N the set Tr_{I_N} is not r.e. (see [51]), and for all assertions $p \models_{I_N} \{\mathbf{true}\}\ x := x\ \{p\}$ iff $\models_{I_N} p$; so the set of asserted programs true under I_N is not r.e. either. This shows that in the case of the language L_P any axiomatizable deduction system dealing with the asserted programs is incomplete.

One might think that the incompleteness comes from allowing arbitrary first order formulas as assertions. However, this is not true, as the following argument shows (see [13]). For any interpretation I and program S, $\models_I \{\mathbf{true}\}\ S\ \{\mathbf{false}\}$ iff S fails to halt for all initial values of its variables. Therefore, the following holds.

Fact. Let $\mathcal{S}_0$ be a class of programs. If L, I, and $\mathcal{S}_0$ are such that the halting problem of $\mathcal{S}_0$ for I is undecidable, then the set $\{\{\mathbf{true}\}\ S\ \{\mathbf{false}\} \mid \models_I \{\mathbf{true}\}\ S\ \{\mathbf{false}\}, S \in \mathcal{S}_0\}$ is not r.e.

Now, the halting problem of $\mathcal{S}$ for I_N is undecidable, so the restriction of the assertion language to $\{\mathbf{true}, \mathbf{false}\}$ cannot lead to completeness either.

The best one might hope for would be to prove *relative* completeness of the system H, which would be a converse of Theorem 1:

> For all interpretations I and all asserted programs φ, if $\models_I \varphi$, then $\mathrm{Tr}_I \vdash_H \varphi$.

Unfortunately, even this cannot be proved. Wand [54] exhibited a particular language L with an interpretation I and asserted program φ such that $\models_I \varphi$ and $\mathrm{Tr}_I \nvdash_H \varphi$. The incompleteness comes from the fact that the necessary intermediate assertions cannot be expressed in L with this particular interpretation.

We now present a simple argument leading to an extension of this incompleteness result. Consider the language L_+ of Presburger arithmetic, that is, the language L_P of Peano arithmetic without the multiplication operation. Let I_+ be its standard interpretation. By the result of [49], Tr_{I_+} is a recursive set. Therefore, for any axiomatic system G the set of asserted programs φ such that $\mathrm{Tr}_{I_+} \vdash_G \varphi$ is r.e.

On the other hand, the halting problem of $\mathscr{S}$ for I_+ is undecidable, since the halting problem of $\mathscr{S}$ for I_N is undecidable and multiplication can be simulated in $\mathscr{S}$ using addition. Therefore, by the Fact above, the set of asserted programs true under I_+ is not r.e. This shows that *no* axiomatic system G can be relatively complete for $\mathscr{S}$.

This argument is a special case of a general incompleteness result proved in [5]. Various other natural structures leading to incompleteness are also exhibited there. The above argument is also implicit in [13].

2.8 Completeness of H in the Sense of Cook

A way to overcome these difficulties while defining the notion of completeness has been indicated by Cook [13]. Define

$$\mathrm{post}_I(p, S) = \{\tau : \exists\delta\, [\models_I p(\delta) \wedge \mathscr{M}_I(S)(\delta) = \tau]\};$$

$$\mathrm{pre}_I(S, q) = \{\delta : \forall\tau\, [\mathscr{M}_I(S)(\delta) = \tau \rightarrow \models_I q(\tau)]\}.$$

Note that these sets are characterized by the following equivalences:

$$\models_I \{p\}\ S\ \{q\} \text{ iff } \{\delta : \models_I p(\delta)\} \subset \mathrm{pre}_I(S, q) \text{ iff } \mathrm{post}_I(p, S) \subset \{\delta : \models_I q(\delta)\}.$$

Now let $\mathscr{S}_0$ be a set of programs. Call the language L *expressive relative to I and $\mathscr{S}_0$* if for all assertions p and programs $S \in \mathscr{S}_0$ there exists an assertion q which defines $\mathrm{post}_I(p, S)$ in L (i.e., such that $\{\delta : \models_I q(\delta)\} = \mathrm{post}_I(p, S)$). If I is such that L is expressive relative to I and $\mathscr{S}_0$, we write that $I \in \mathrm{Exp}(L, \mathscr{S}_0)$.

Definition. A proof system G for $\mathscr{S}_0$ is complete *in the sense of Cook* if, for every interpretation $I \in \mathrm{Exp}(L, \mathscr{S}_0)$ and every asserted program φ, if $\models_I \varphi$, then $\mathrm{Tr}_I \vdash_G \varphi$.

The results of [13] imply that the proof system H for $\mathscr{S}$ is complete in the above sense. The proof of completeness proceeds by induction on the structure of programs. Let $I \in \mathrm{Exp}(L, \mathscr{S})$.

If $\models_I \{p\}\ x := t\ \{q\}$, then clearly $\models_I p \rightarrow q[t/x]$; so, by the assignment axiom and the consequence rule, $\mathrm{Tr}_I \vdash_H \{p\}\ x := t\ \{q\}$.

If $\models_I \{p\}\ S_1; S_2\ \{q\}$, then clearly $\models_I \{p\}\ S_1\ \{r\}$ and $\models_I \{r\}\ S_2\ \{q\}$, where r defines $\mathrm{post}_I(p, S_1)$; so, by the induction hypothesis and the composition rule, $\mathrm{Tr}_I \vdash_H \{p\}\ S_1; S_2\ \{q\}$.

The case of **if** e **then** S_1 **else** S_2 **fi** is straightforward.

If $\models_I \{p\}$ **while** e **do** S **od** $\{q\}$, then we must find a loop invariant r such that $\models_I \{r \wedge e\}\ S\ \{r\}$, $\models_I p \rightarrow r$, and $\models_I (r \wedge e) \rightarrow \neg q$. Then, by the induction hypothesis, $\mathrm{Tr}_I \vdash_H \{p\}$ **while** e **do** S **od** $\{q\}$.

Consider the set

$$C = \{\delta : \exists k, \delta_0, \ldots, \delta_k\, [\delta = \delta_k \wedge \models_I p(\delta_0) \wedge \forall i < k\, [\mathscr{M}_I(S)(\delta_i) = \delta_{i+1} \wedge \models_I e(\delta_i)]]\}.$$

Thus $\delta \in C$ iff there exists a computation which starts in a state satisfying p and which reaches state δ after some finite number of passes through the loop. It is clear that an assertion r defining C satisfies the above three conditions.

To find such an assertion, consider the list $y_1, \ldots, y_n$ of all variables which occur free in p, e, S, or q. Let r_1 be the assertion which defines $\text{post}_I(p,$ **while** $e \wedge (y_1 \neq z_1 \vee y_2 \neq z_2 \vee \cdots \vee y_n \neq z_n)$ **do** S **od**), where $z_1, \ldots, z_n$ are new variables. If $\delta \in C$, then $\models_I \exists z_1, \ldots, z_n r_1(\delta)$, where the values chosen for z_i $(i = 1, \ldots, n)$ are correspondingly $\delta(y_i)$ $(i = 1, \ldots, n)$. The implication $\models_I \exists z_1, \ldots, z_n r_1(\delta) \rightarrow \delta \in C$ is obvious. Hence $r \equiv \exists z_1, \ldots, z_n r_1$ is the desired assertion.

Clarke [9] observed that if, in the definition of expressiveness, we change the requirement of definability of $\text{post}_I(p, S)$ to that of definability of $\text{pre}_I(S, q)$, then the above proof (viz., the last case) can be simplified. Namely, for the invariant r we can simply take an assertion which defines $\text{pre}_I($**while** e **do** S **od**, $q)$. This proof also shows that, when using the requirement of definability of $\text{pre}_I(S, q)$ in the definition of expressiveness, it is not necessary to assume that the equality predicate is in the language L.

We chose here Cook's original definition of expressiveness, since the completeness result in the form just proved is used in Section 3.

2.9 Expressiveness

As indicated by Clarke [10a] and rigorously proved by Olderog [44], these two definitions of expressiveness are actually equivalent for any class of programs considered in this paper. To give an idea of the proof, assume that, for any p and S, $\text{post}_I(p, S)$ is definable. Consider a program S_0. Let $\bar{x}$ be a sequence of all variables occurring in S_0 and let $\bar{z}$ be a sequence of some new variables of the same length as $\bar{x}$. Now let q_0 be an assertion which defines $\text{post}_I(\bar{x} = \bar{z}, S_0)$. It is easy to see that, for any q, $\text{pre}_I(S_0, q)$ is definable by the formula $(\forall \bar{x}\ (q_0 \rightarrow q))[\bar{x}/\bar{z}]$.

A similar construction proves the converse implication. It is worthwhile to note that the formulas $\{\bar{x} = \bar{z}\}\ S_0\ \{q_0\}$ play an important role in the completeness proofs in Section 3.

A natural question now arises: how restrictive is the assumption of expressiveness? Observe that L_P is expressive relative to I_N and $\mathcal{S}$. Thus, any true (under I_N) asserted program can be proved in H provided we can "ask" an oracle about the truth of the assertions under I_N. Also, as Clarke [9] observed, if the domain of I is finite, then L is expressive relative to I and $\mathcal{S}$.

It turns out that these are actually the only two possibilities. The following theorem is a special case of a theorem proved by De Millo, Lipton, and Snyder (see [38]):

THEOREM 2. *If L is expressive relative to I and $\mathcal{S}$, then either*

1. *a standard model of Peano arithmetic can be defined in I, or*
2. *$\forall S \in \mathcal{S}\ \exists n$ such that S reaches at most n states in any computation over the domain of I with any initial state.*

In the previous section we saw that expressiveness is a sufficient condition for completeness. Is it a necessary condition as well? The answer is no, and the following argument due to Bergstra and Tucker [6] gives evidence for it.

There exists a nonstandard model of Peano arithmetic with an interpretation I such that $\text{Tr}_I = \text{Tr}_{I_N}$. It is a direct consequence of the compactness theorem (see [51]). It is now easy to see that, for any asserted program φ, if $\models_I \varphi$, then $\models_{I_N} \varphi$.

Thus, for any φ, $\models_I \varphi$ implies $\models_{I_N} \varphi$, which in turn implies $\mathrm{Tr}_{I_N} \vdash_H \varphi$ by the just-proved Cook completeness result; so finally $\mathrm{Tr}_I \vdash_H \varphi$ by the choice of I.

On the other hand, L_P is not expressive relative to I and $\mathscr{S}$. This follows from Theorem 2, but of course a straightforward argument can be given. Namely, consider the program $S \equiv$ **while** $x < y$ **do** $x := x + 1$ **od**. This program terminates on I when started in a state σ in which $x = 0$ iff $\sigma(y)$ is a standard natural number. Thus,

$$\mathrm{post}_I(x = 0, S) = \{\sigma : \sigma(x) = \sigma(y) \wedge y \text{ is a standard natural number}\}.$$

But this set is not definable in I by any formula q of L_P. Otherwise, we could prove by the induction axiom that $\models_I \forall x, y\ q$, which is not the case.

2.10 Complete Assertion Classes

The completeness of H and the expressibility of L_P relative to I_N and $\mathscr{S}$ imply that, if $\models_{I_N} \{p\}\ S\ \{q\}$, then there exists a proof of $\{p\}\ S\ \{q\}$ in H (from Tr_{I_N}) which uses only arithmetical assertions. In typical proofs much simpler assertions are used.

A *global* correctness property $\{p\}\ S\ \{q\}$ in practice has recursive assertions[1] p and q. The precondition p is usually some simple condition on the input variables, or even **true**. Similarly, one may expect that the postcondition q can be checked effectively by inspection of the output variables. A natural conjecture then is that all (intermediate) assertions needed in the proof of $\{p\}\ S\ \{q\}$ in H may also be chosen to be recursive.

Let A be a set of assertions. Let us write $\vdash_{H,A} \{p\}\ S\ \{q\}$ to denote the fact that there exists a proof of $\{p\}\ S\ \{q\}$ in H (from $\mathrm{Tr}_{I_N} \cap A$) in which only assertions from A occur. We call a class of assertions A *complete* (with respect to $\mathscr{S}$) if for every $p, q \in A$ and $S \in \mathscr{S}$ we have $\models_{I_N} \{p\}\ S\ \{q\}$ iff $\vdash_{H,A} \{p\}\ S\ \{q\}$. In [2] it is proved that any class of recursive assertions A which contains **true** and **false** is incomplete; so the above conjecture is false. On the other hand, the class of recursively enumerable assertions and various other natural classes are complete.

We can, however, get completeness of the class of recursive assertions for $\mathscr{S}$ if we extend the proof system H by adding to it the following proof rule concerning deletion of assignments to the auxiliary variables.

Let AV be a set of variables which appear in S' only in assignments $x := t$, where x is in AV. If p and q do not contain free variables from AV and S is obtained from S' by deleting all assignments to the variables in AV, then

$$\frac{\{p\}\ S'\ \{q\}}{\{p\}\ S\ \{q\}}.$$

This rule is from [47], where it was used in the proof system for verification of parallel programs.

2.11 Total Correctness

By distinguishing between partial and total correctness, we stress the fact that termination is not dealt with in H.

[1] That is, assertions which define a recursive set.

We say that a program S is *partially correct* under I (with respect to p and q) if $\models_I \{p\}\ S\ \{q\}$. In contrast, we say that a program S is *totally correct* under I (with respect to p and q), written $|\!\!\models_I \{p\}\ S\ \{q\}$, if additionally the termination of S is guaranteed. Thus, $|\!\!\models_I \{p\}\ S\ \{q\}$ holds iff

> for all states δ such that $\models_I p(\delta)$ there exists a state τ such that $\mathscr{M}_I(S)(\delta) = \tau$ and $\models_I q(\tau)$.

Thus H is a proof system for partial correctness. It is clear that the only proof rule of H which introduces a possibility of nontermination is the **while** rule; so to deal with total correctness that rule has to be changed.

The following refinement of the **while** rule leading to total correctness has been formulated in [23].

Rule 6: **while** Rule II. *Let $p(n)$ be an assertion with a free variable n which does not appear in S and ranges over natural numbers. Then*

$$\frac{p(n+1) \rightarrow e,\ \{p(n+1)\}\ S\ \{p(n)\}, p(0) \rightarrow \neg e}{\{\exists n\, p(n)\}\ \textbf{while}\ e\ \textbf{do}\ S\ \textbf{od}\ \{p(0)\}}.$$

Let H_0 denote the proof system obtained from H by replacing the **while** rule (Rule 4) by Rule 6. In H_0 we can easily prove total correctness of the program S_0 from Section 2.2 with respect to $x \geq 0 \wedge y > 0$ and $a \cdot y + b = x \wedge 0 \leq b < y$. Namely, take $p(n) \equiv r \wedge n \cdot y \leq b < (n+1) \cdot y$ where $r \equiv a \cdot y + b = x \wedge b \geq 0$ is the loop invariant from the proof in Section 2.2. $p(n)$ clearly satisfies the premises of the above rule for $e \equiv b \geq y$ and $S \equiv b := b - y;\ a := a + 1$.

Also, similarly to (6),

$$\{x \geq 0 \wedge y > 0\}\ a := 0;\ b := x\ \{r \wedge y > 0\}$$

holds. To conclude the proof it is now sufficient to observe that

$$\models_{I_0} r \wedge y > 0 \rightarrow \exists n\, p(n)$$

and

$$\models_{I_0} p(0) \rightarrow a \cdot y + b = x \wedge 0 \leq b < y$$

and apply the consequence rule and the composition rule.

Call a proof system G *totally sound* if, for all interpretations I and asserted programs φ, $\mathrm{Tr}_I \vdash_G \varphi$ implies $|\!\!\models_I \varphi$. We would like to require any proof system for total correctness, including H_0, to be totally sound. Note that total soundness of H_0 would imply that S_0 is totally correct under I_0 with respect to $x \geq 0 \wedge y > 0$ and $a \cdot y + b = x \wedge 0 \leq b < y$, thus completing the above reasoning. Unfortunately, any totally sound proof system is hopelessly weak, as the following theorem shows.

Theorem 3. *There does not exist a proof system G such that*

1. *G is totally sound and*
2. *$Tr_{I_0} \vdash_G \{$***true***$\}\ S_1\ \{$***true***$\}$ where $S_1 \equiv$ **while** $x > 0$ **do** $x := x - 1$ **od**.*

Proof. The proof is immediate. Namely, suppose that a proof system G satisfies 1 and 2. As we did in Section 2.9, take a nonstandard model of Peano arithmetic with an interpretation I such that $\mathrm{Tr}_I = \mathrm{Tr}_{I_0}$. Now, by 2, $\mathrm{Tr}_I \vdash_G$ {**true**} S_1 {**true**}; so, by 1, $\mid\!\models_I$ {**true**} S_1 {**true**}. The latter is, however, a contradiction, since it states that S_1 is total under an interpretation in a nonstandard model of Peano arithmetic. □

Therefore, when dealing with proof systems for total correctness we have to find another notion of soundness. One possibility is to restrict attention to one interpretation only, namely, I_N (or a minor extension of it like I_0). However, this is not a satisfactory choice, as it would force us to allow one assertion language only: that of Peano arithmetic. A more satisfactory proposal has been indicated by Harel [23].

Let L be an assertion language and let L^+ be the minimal extension of L containing the language L_P of Peano arithmetic and a unary relation $\mathrm{nat}(x)$. Call an interpretation I of L^+ *arithmetical* if its domain includes the set of natural numbers, I provides the standard interpretation for L_P, and $\mathrm{nat}(x)$ is interpreted as the relation "to be a natural number." Additionally, we require that there exist a formula of L^+ which, when interpreted under I, provides the ability to encode finite sequences of elements from the domain of I into one element. (The last requirement is needed only for the completeness proof.)

One of the examples of an arithmetical interpretation is of course I_N. It is important to note that any interpretation of an assertion language L with an infinite domain can be extended to an arithmetical interpretation of L^+. Clearly, the proof system H_0 is suitable only for assertion languages of the form L^+, and an expression such as $p(n + 1)$ is actually a shorthand for $\mathrm{nat}(n + 1) \wedge p(n + 1)$.

We now say that a proof system G for total correctness is *arithmetically sound* if, for all arithmetical interpretations I and asserted programs φ, $\mathrm{Tr}_I \vdash_G \varphi$ implies $\mid\!\models_I \varphi$.

Harel [23] showed that the proof system H_0 is arithmetically sound. He also proved that H_0 is *arithmetically complete*, that is, that an implication converse to the one above holds.

The completeness proof runs by induction on the structure of programs, and only the case of the **while** construct is different from the corresponding case in the completeness proof of H.

Assume $\mid\!\models_I$ $\{r\}$**while** e **do** S **od** $\{q\}$ where I is an arithmetical interpretation. Let n be a fresh variable. Consider the following set of states:

$$\begin{aligned} C = \{\delta : {} & \models_I \mathrm{nat}(n)(\delta) \\ & \wedge \exists \delta_0, \ldots, \delta_k\, [\delta = \delta_0 \wedge \models_I (q \wedge \neg e)(\delta_k) \\ & \qquad\qquad \wedge \forall i < k\, [\mathscr{M}_I(S)(\delta_i) = \delta_{i+1} \wedge \models_I e(\delta_i)]], \\ & \text{where } k = \delta(n)\}. \end{aligned}$$

Thus $\delta \in C$ iff $\delta(n)$ is a natural number, say k, such that the loop in **while** e **do** S **od** is executed exactly k times when started in δ and the final state satisfies q.

It can be shown (thanks to the provision for coding of finite sequences) that there exists an assertion $p(n)$ which defines C. It is easy to see that $\models_I p(n + 1) \rightarrow e$, $\mid\!\models_I \{p(n + 1)\}\ S\ \{p(n)\}$, and $\models_I p(0) \rightarrow \neg e$. Thus, by the induction hypothesis and the new rule, $\mathrm{Tr}_I \vdash_{H_0} \{\exists n\ p(n)\}$ **while** e **do** S **od** $\{p(0)\}$. To complete the proof it is now sufficient to observe that, by the assumption, $\models_I r \rightarrow \exists n\, p(n)$ and $\models_I p(0) \rightarrow q$ and to apply the consequence rule.

It should be stressed that Theorem 3 applies to the notion of arithmetical soundness as well. However, in any language of the form L^+ one can speak about "standard" natural numbers; so the formula to be proved can now be phrased as $\{\mathrm{nat}(x)\}\ S_1\ \{\mathbf{true}\}$, and this version *can* be proved. This shows that the essence of Harel's approach lies in the ability to speak in any assertion language of the form L^+ about natural numbers *together* with the restriction on the interpretations assuring that these are the "standard" natural numbers.

2.12 Bibliographical Remarks

The proof system H (with the exception of Rule 3) and the example in Section 2.2 are from [27]. Rule 3 is from [37], which also contains the first proof of soundness of (an extension of) H. The terminology of "establishing" and "preserving" an invariant, as well as the example in Section 2.5, is from [16]. The idea of annotating a program with the relevant assertions is first expressed in [37]. A different proof of Wand's incompleteness result is given in [24]. The incompleteness of the class of recursive assertions and the completeness of the class of recursively enumerable assertions mentioned in Section 2.10 are also proved in [39]. In [52] a completeness result similar to that of Section 2.11 is presented. The first proof rules for total correctness of **while** programs within the framework of Hoare's logic are presented in [42]. In [22] various proof rules for total correctness of **while** programs presented in the literature are discussed and compared.

3. PARAMETERLESS PROCEDURES

For clarity, we have separated the issues concerning procedures from those of scope and parameter mechanisms. Parameterless procedures are discussed next; the treatment of parameters is in Section 6. To simplify the discussion, we restrict our attention to the case of one procedure declaration. All results of this section can be straightforwardly generalized to the case of more than one procedure declaration.

3.1 Nonrecursive Procedures

We first consider the simpler case of a nonrecursive procedure. Assume a procedure declaration $P \Leftarrow S_0$ where $S_0 \in \mathcal{S}$ is the procedure body of P, and extend the set of programs $\mathcal{S}$ by allowing the programs to contain the calls of P. Call this extended class of program $\mathcal{S}_1$. Each procedure call P refers to the declaration $P \Leftarrow S_0$. The requirement that $S_0 \in \mathcal{S}$ implies that the procedure P is not recursive.

To deal with the procedure calls in the correctness proofs, we supplement the proof system H by the following proof rule.

RULE 7: PROCEDURE CALL RULE

$$\frac{\{p\}\ S_0\ \{q\}}{\{p\}\ P\ \{q\}}.$$

To consider the problem of soundness and completeness of the resulting system we must first extend the meaning function $\mathcal{M}_I$ to programs from $\mathcal{S}_1$. For $S \in \mathcal{S}_1$ let $S[S_0/P]$ denote the program resulting from replacing all occurrences of P in

S by S_0. In other words, $S[S_0/P]$ is the macro expansion of S. For $S \in \mathscr{S}_1 \backslash \mathscr{S}$ we define $\mathscr{M}_I(S)$ to be equal to $\mathscr{M}_I(S[S_0/P])$.

Thus $\mathscr{M}_I(P) = \mathscr{M}_I(S_0)$, which implies that the rule of procedure calls is sound. The fact that Rules 2-5 are sound in the case of $\mathscr{S}$ and the definition of $\mathscr{M}_I(S)$ for $S \in \mathscr{S}_1$ trivially imply the soundness of Rules 2-5 in the case of $\mathscr{S}_1$.

It should also be clear that the above proof system for $\mathscr{S}_1$ is complete in the sense of Cook (i.e., that $I \in \text{Exp}(L, \mathscr{S}_1)$ and $\models_I \varphi$ implies $\text{Tr}_I \vdash_{H+\text{Rule}\,7} \varphi$). The additional case of procedure calls is easily handled: if $\models_I \{p\}\ P\ \{q\}$, then $\models_I \{p\}\ S_0\ \{q\}$; thus $\text{Tr}_I \vdash_H \{p\}\ S_0\ \{q\}$ by the previous completeness result; that is, $\text{Tr}_I \vdash_{H+\text{Rule}\,7} \{p\}\ P\ \{q\}$. The proof of other cases is the same as in Section 2.8.

3.2 The Recursion Rule

When the declared procedure $P \Leftarrow S_0$ is recursive, that is, when $S_0 \in \mathscr{S}_1 \backslash \mathscr{S}$, the above system is still sound but obviously incomplete: an attempt at proving $\{p\}$ P $\{q\}$ results in an infinite regression. A way to overcome this difficulty has been suggested in [26]. The rule one should adopt is the following.

Rule 8: Recursion Rule

$$\frac{\{p\}\ P\ \{q\} \vdash \{p\}\ S_0\ \{q\}}{\{p\}\ P\ \{q\}}.$$

The reasoning presented by this rule is the following: infer $\{p\}\ P\ \{q\}$ from the fact that $\{p\}\ S_0\ \{q\}$ can be proved (using the other rules and axioms) from the assumption $\{p\}\ P\ \{q\}$. Rule 8 is actually a translation of the so-called Scott's induction rule (see [50]) into this framework.

As an example of a proof using the recursion rule, consider the procedure declaration $P \Leftarrow S_0$ for

$$S_0 \equiv \textbf{if } x = 0 \textbf{ then } y := 1 \textbf{ else } x := x - 1;\ P;\ x := x + 1;\ y := y \cdot x \textbf{ fi}.$$

We now prove $\{x \geq 0\}\ P\ \{y = x!\}$ in the system H augmented with the recursion rule.

By the recursion rule it is enough to prove

$$\{x \geq 0\}\ P\ \{y = x!\} \vdash_H \{x \geq 0\}\ S_0\ \{y = x!\}.$$

Assume

$$\{x \geq 0\}\ P\ \{y = x!\}. \tag{13}$$

By the assignment axiom,

$$\{y \cdot x = x!\}\ y := y \cdot x\ \{y = x!\} \tag{14}$$

and

$$\{y \cdot (x + 1) = (x + 1)!\}\ x := x + 1\ \{y \cdot x = x!\}; \tag{15}$$

so, by the composition rule,

$$\{y \cdot (x + 1) = (x + 1)!\}\ x := x + 1;\ y := y \cdot x\ \{y = x!\}. \tag{16}$$

Since the implication

$$y = x! \rightarrow y \cdot (x + 1) = (x + 1)! \tag{17}$$

is true, by the consequence rule and (13),

$$\{x \geq 0\}\ P\ \{y \cdot (x+1) = (x+1)!\}. \tag{18}$$

(16) and (18) imply, by the composition rule,

$$\{x \geq 0\}\ P;\ x := x+1;\ y := y \cdot x\ \{y = x!\}. \tag{19}$$

On the other hand, since the implication

$$x \geq 0 \wedge x \neq 0 \rightarrow x - 1 \geq 0 \tag{20}$$

is true, and, by the assignment axiom,

$$\{x - 1 \geq 0\}\ x := x - 1\ \{x \geq 0\}, \tag{21}$$

we get by the consequence rule

$$\{x \geq 0 \wedge x \neq 0\}\ x := x - 1\ \{x \geq 0\}. \tag{22}$$

By the composition rule we now get from (19) and (22)

$$\{x \geq 0 \wedge x \neq 0\}\ x := x - 1;\ P;\ x := x + 1;\ y := y \cdot x\ \{y = x!\}. \tag{23}$$

Since

$$x \geq 0 \wedge x = 0 \rightarrow 1 = x! \tag{24}$$

is true, and, by the assignment axiom,

$$\{1 = x!\}\ y := 1\ \{y = x!\}, \tag{25}$$

we get by the consequence rule

$$\{x \geq 0 \wedge x = 0\}\ y := 1\ \{y = x!\}. \tag{26}$$

(23) and (26) finally imply by the **if-then-else** rule

$$\{x \geq 0\}\ S_0\ \{y = x!\}, \tag{27}$$

which was to be proved.

Of course, strictly speaking, we have only proved that

$$(17), (20), (24) \vdash_{H+\text{Rule 8}} \{x \geq 0\}\ P\ \{y = x!\}.$$

3.3 Insufficiency of the Recursion Rule

However, the system H augmented with the recursion rule is not complete. As evidence we now show that there is no way to prove in it that in the case of the above procedure declaration a call of P does not change the value of x; that is, there is no way to prove that $\{x = z\}\ P\ \{x = z\}$.

Suppose by contradiction that $\{x = z\}\ P\ \{x = z\}$ can be proved in the system. We can assume that all assertions used in the proof do not have y as a free variable (otherwise, y can be everywhere replaced by, say, 0). We can also assume that the last rule applied was that of consequence. So for some p and q such that the formulas

$$x = z \rightarrow p \tag{28}$$

and

$$q \rightarrow x = z \tag{29}$$

are true (under the standard interpretation in natural numbers) $\{p\}\ P\ \{q\}$ can be proved. Consecutive applications of the consequence rule can be combined into one. Thus we can assume that in the proof of $\{p\}\ P\ \{q\}$ the last rule applied was the recursion rule. In other words, the premise $\{p\}\ P\ \{q\} \vdash \{p\}\ S_0\ \{q\}$ can be established. Hence, under the assumption of $\{p\}\ P\ \{q\}$ both $\{p \wedge x = 0\}$ $y := 1\ \{q\}$ and $\{p \wedge x \neq 0\}\ x := x - 1;\ P;\ x := x + 1;\ y := y \cdot x\ \{q\}$ can be proved.

The provability of the first formula implies that

$$p \wedge x = 0 \rightarrow q \tag{30}$$

is true (by assumption, y is not free in q).

The other formula had to be proved using the assumption $\{p\}\ P\ \{q\}$. For some p_1 and q_1 we have that $p_1 \rightarrow p$ and

$$q \rightarrow q_1 \tag{31}$$

is true (to obtain $\{p_1\}\ P\ \{q_1\}$ by the consequence rule) and both $\{p \wedge x \neq 0\}$ $x := x - 1\ \{p_1\}$ and $\{q_1\}\ x := x + 1;\ y := y \cdot x\ \{q\}$ hold. Provability of the second correctness formula implies that

$$q_1 \rightarrow q[x + 1/x] \tag{32}$$

is true. (31) and (32) imply that

$$q \rightarrow q[x + 1/x] \tag{33}$$

is true. But (29) implies that $q[x + 1/x] \rightarrow x + 1 = z$ is true; so, by (33),

$$q \rightarrow x + 1 = z \tag{34}$$

is true. From (29) and (34) we get that

$$q \rightarrow \mathbf{false} \tag{35}$$

is true. On the other hand, from (28) and (30) $x = z \wedge x = 0 \rightarrow q$ is true; so, by (35), $x = z \wedge x = 0 \rightarrow \mathbf{false}$ is true, which gives the desired contradiction.

3.4 The Proof System *G*

The system *H* augmented with the recursion rule is thus incomplete. Therefore, following Gorelick [19], we supplement this system by the following axiom and proof rules, which lead to a complete proof system.

AXIOM 9: INVARIANCE AXIOM

$$\{p\}\ P\ \{p\} \qquad \textit{where} \quad var(p) \cap var(S_0) = \emptyset.$$

RULE 10: SUBSTITUTION RULE I

$$\frac{\{p\}\ P\ \{q\}}{\{p[\bar{y}/\bar{z}]\}\ P\ \{q[\bar{y}/\bar{z}]\}} \qquad \textit{where} \quad \bar{z} \cap var(S_0) = \emptyset \qquad \textit{and} \qquad \bar{y} \cap var(S_0) = \emptyset.$$

RULE 11: SUBSTITUTION RULE II

$$\frac{\{p\}\ P\ \{q\}}{\{p[\bar{y}/\bar{z}]\}\ P\ \{q\}} \qquad \textit{where} \quad \bar{z} \cap var(S_0, q) = \emptyset.$$

Rule 12: Conjunction Rule

$$\frac{\{p\}\ P\ \{q\},\ \{p'\}\ P\ \{r\}}{\{p \wedge p'\}\ P\ \{q \wedge r\}}.$$

Here $\bar{y}$ and $\bar{z}$ denote sequences of variables of the language L. $p[\bar{y}/\bar{z}]$ stands for a simultaneous substitution of the variables from $\bar{y}$ for the variables from $\bar{z}$ in p. $\mathrm{var}(S_0)$ denotes the set of all variables which occur in S_0, and $\mathrm{var}(p)$ denotes the set of all free variables of p. It should be clear what we mean by $\mathrm{var}(S_0, q)$, etc.

Let us denote the resulting proof system by G.

3.5 An Example of a Proof in G

To see how the additional rules of G are used in actual proofs, we now prove the already mentioned formula $\{x = z\}\ P\ \{x = z\}$ where P is the factorial procedure from Section 3.2. To prove $\{x = z\}\ P\ \{x = z\}$, it is enough to establish the premise $\{x = z\}\ P\ \{x = z\} \vdash \{x = z\}\ S_0\ \{x = z\}$ of the recursion rule.

Assume

$$\{x = z\}\ P\ \{x = z\}. \tag{36}$$

By substitution rule I,

$$\{x = u\}\ P\ \{x = u\}, \tag{37}$$

and, by the invariance axiom,

$$\{u = z - 1\}\ P\ \{u = z - 1\}; \tag{38}$$

so, by the conjunction rule and the consequence rule,

$$\{x = z - 1\}\ P\ \{x = z - 1\}. \tag{39}$$

Now, applying the assignment axiom and the composition and consequence rules, we get from (39)

$$\{x = z\}\ x := x - 1; P; x := x + 1; y := y \cdot x\ \{x = z\}. \tag{40}$$

By the consequence rule we can conjoin the preassertion with the formula $x \neq 0$. Also,

$$\{x = z \wedge x = 0\}\ y := 1\ \{x = z\} \tag{41}$$

holds. By the **if-then-else** rule we now get $(x = z\}\ S_0\ \{x = z\}$; so we have established the desired premise.

This proof did not use substitution rule II. However, that rule is needed, for example, to prove $\{x \geq 0\}\ P\ \{y \geq 1\}$. The proof makes use of the already proved formulas $\{x \geq 0\}\ P\ \{y = x!\}$ and $\{x = z\}\ P\ \{x = z\}$ and an instance $\{z \geq 0\}\ P\ \{z \geq 0\}$ of the invariance axiom to get, by the conjunction rule, $\{x \geq 0 \wedge x = z \wedge z \geq 0\}\ P\ \{y = x! \wedge x = z \wedge z \geq 0\}$. Using the consequence rule, we now get $\{x = z \wedge x \geq 0\}\ P\ \{y \geq 1\}$. Finally, by substitution rule II, $(x = x \wedge x \geq 0\}\ P\ \{y \geq 1\}$; so $\{x \geq 0\}\ P\ \{y \geq 1\}$ by the consequence rule.

These proofs shed light on the way the new rules are used. Using substitution rule I, one simply renames variables not used in the program (so-called *auxiliary* *variables*). In contrast, substitution rule II is used to get rid of the auxiliary variables from the preassertion. Auxiliary variables are typically used here to "freeze" the values of the program variables before a procedure call. In proofs usually two different premises about a procedure call are needed: one derived by the recursion rule and the other one obtained by the invariance axiom. The conjunction rule replaces these two premises by one. Finally, observe that the invariance axiom can be proved straightforwardly using the recursion rule. However, since it is often used, it is useful to have it formulated separately.

3.6 Semantics of Recursive Procedures

Before we dwell on the question of the soundness and completeness of G, we have to define the meaning function $\mathscr{M}_I$ on programs from $\mathscr{S}_1 \backslash \mathscr{S}$. We do so in a way which simplifies our considerations concerning the soundness of G. The semantics we provide is usually called an approximating semantics.

Let Ω stand for a program from $\mathscr{S}$ which never halts. Let us define a program $S_0^{(n)} \in \mathscr{S}$ by induction on n:

$$S_0^{(0)} \equiv \Omega;$$

$$S_0^{(n+1)} \equiv S_0[S_0^{(n)}/P].$$

A straightforward proof by induction on the structure of S shows that, for all $S_1, S_2 \in \mathscr{S}$ and $S \in \mathscr{S}_1$, if $\mathscr{M}_I(S_1) \subset \mathscr{M}_I(S_2)$, then $\mathscr{M}_I(S[S_1/P]) \subset \mathscr{M}_I(S[S_2/P])$. This implies (by induction on n) that, for all n, $\mathscr{M}_I(S_0^{(n)}) \subset \mathscr{M}_I(S_0^{(n+1)})$. Thus, for all $S \in \mathscr{S}_1$, $\mathscr{M}_I(S[S_0^{(n)}/P]) \subset \mathscr{M}_I(S[S_0^{(n+1)}/P])$.

For $S \in \mathscr{S}_1$ we now define $\mathscr{M}_I(S)$ by putting

$$\mathscr{M}_I(S) = \bigcup_{n=0}^{\infty} \mathscr{M}_I(S[S_0^{(n)}/P]).$$

In particular,

$$\mathscr{M}_I(P) = \bigcup_{n=0}^{\infty} \mathscr{M}_I(S_0^{(n)}).$$

By the above, $\mathscr{M}_I(S)$ is a (partial) function.

3.7 Soundness of G

We wish to prove that the proof system G is sound, that is, that, for every interpretation I and correctness formula φ, if $\mathrm{Tr}_I \vdash_G \varphi$, then $\models_I \varphi$. The fact that G is not a usual proof system in the sense of first-order logic forces us to exercise some care while doing so. It is, for example, not clear to what extent we can use the fact that Rules 2-5 are sound in the case of nonrecursive procedure declarations and in what sense the recursion rule is to be proved sound.

To deal with these problems, we first transform the system G into a proof system K which uses the usual notion of proof.

The formulas of K are implications $\Phi \rightarrow \Psi$ (called correctness phrases), where Φ and Ψ are finite sets of correctness formulas. If Φ is empty, we write Ψ instead of $\Phi \rightarrow \Psi$. For each axiom φ_0 and proof rule

$$\frac{\varphi_1, \ldots, \varphi_n}{\varphi_{n+1}}$$

of G, we adopt in K an axiom $\Phi \rightarrow \varphi_0$ and a proof rule

$$\frac{\Phi \rightarrow \varphi_1, \ldots, \varphi_n}{\Phi \rightarrow \varphi_{n+1}}.$$

We also have the rule

$$\frac{\{p\}\ P\ \{q\} \rightarrow \{p\}\ S_0\ \{q\}}{\{p\}\ P\ \{q\}}$$

corresponding to the recursion rule, the collection rule

$$\frac{\Phi \rightarrow \Psi_1, \ldots, \Phi \rightarrow \Psi_n}{\Phi \rightarrow \Psi_1, \ldots, \Psi_n},$$

and the selection axiom $\Phi \rightarrow \varphi$ where $\varphi \in \Phi$. This translation of G into K corresponds to a translation of a Gentzen natural deduction system into a Gentzen sequent calculus.

In the following discussion we write $\langle P \Leftarrow S \mid \varphi \rangle$ instead of φ to indicate that each procedure call P in φ refers to the procedure declaration $P \Leftarrow S$.

Definition. Let I be an interpretation of L.

1. An implication $\langle P \Leftarrow S_0 \mid \Phi \rightarrow \Psi \rangle$ is called *I-good* if, for every n, $\langle P \Leftarrow S_0^{(n)} \mid \Phi \rightarrow \Psi \rangle$ is true under I.
2. For a nonrecursive procedure declaration $P \Leftarrow S$:
   a. $\langle P \Leftarrow S \mid \Phi \rightarrow \Psi \rangle$ is true under I if the truth under I of $\langle P \Leftarrow S \mid \Phi \rangle$ implies the truth under I of $\langle P \Leftarrow S \mid \Psi \rangle$; and
   b. $\langle P \Leftarrow S \mid \Phi \rangle$ is true under I if, for all $\varphi \in \Phi$, $\langle P \Leftarrow S \mid \varphi \rangle$ is true under I.

Definition

1. A correctness phrase is called *good* if it is I-good for all interpretations I.
2. A proof rule of K is called *good* if for all interpretations I it preserves the I-goodness of correctness phrases.
3. The proof system K is called *good* if all its axioms and proof rules are good.

Observe that, for every set of assertions T, $T \cup \{\varphi\} \vdash_G \Psi$ iff $T \vdash_K \varphi \rightarrow \Psi$. The proof runs by induction on the length of proofs. Thus, in particular, for every set of assertions T, $T \vdash_G \varphi$ iff $T \vdash_K \varphi$.

This, together with the observation that $\langle P \Leftarrow S_0 \mid \varphi \rangle$ is true under I iff it is I-good, implies the following claim:

CLAIM 1. *If the proof system K is good, then the proof system G is sound.*

Also, the following holds:

CLAIM 2

1. *If, for each n,* $\langle P \Leftarrow S_0^{(n)} \mid \varphi \rangle$ *is valid, then* $\langle P \Leftarrow S_0 \mid \Phi \rightarrow \varphi \rangle$ *is good.*
2. *If for each n a proof rule*

$$\frac{\langle P \Leftarrow S_0^{(n)} \mid \Psi_1 \rangle}{\langle P \Leftarrow S_0^{(n)} \mid \Psi_2 \rangle}$$

is sound, then the proof rule

$$\frac{\langle P \Leftarrow S_0 \mid \Phi \rightarrow \Psi_1 \rangle}{\langle P \Leftarrow S_0 \mid \Phi \rightarrow \Psi_2 \rangle}$$

is good.

Thus the axioms and proof rules of K which are translations of axioms and proof rules of H are all good, as the system H is sound in the case of a nonrecursive procedure declaration.

By Claims 1 and 2, to prove the soundness of G it is now enough to prove

a. the validity of the invariance axiom in the case of a nonrecursive procedure declaration;
b. the soundness of substitution rules I and II and the conjunction rule in the above case;
c. the goodness of the rule

$$\frac{\{p\}\ P\ \{q\} \rightarrow \{p\}\ S_0\ \{q\}}{\{p\}\ P\ \{q\}},$$

that is, that for any I, if for all n $\langle P \Leftarrow S_0^{(n)} \mid \{p\}\ P\ \{q\} \rightarrow \{p\}\ S_0\ \{q\} \rangle$ is true under I, then for all n $\langle P \Leftarrow S_0^{(n)} \mid \{p\}\ P\ \{q\} \rangle$ is true under I; and
d. the goodness of the selection axiom and the collection rule.

Proofs of a and b are straightforward. To prove c, assume that for a given I and all n

$$\langle P \Leftarrow S_0^{(n)} \mid \{p\}\ P\ \{q\} \rightarrow \{p\}\ S_0\ \{q\} \rangle \text{ is true under } I. \tag{42}$$

Clearly $\langle P \Leftarrow S_0^{(0)} \mid \{p\}\ P\ \{q\} \rangle$ is true under I. Assume now that for some n

$$\langle P \Leftarrow S_0^{(n)} \mid \{p\}\ P\ \{q\} \rangle \text{ is true under } I. \tag{43}$$

Then by (42)

$$\langle P \Leftarrow S_0^{(n)} \mid \{p\}\ S_0\ \{q\} \rangle \text{ is true under } I. \tag{44}$$

But $S_0[S_0^{(n)}/P] \equiv S_0^{(n+1)}$; so (44) implies that $\langle P \Leftarrow S_0^{(n+1)} \mid \{p\}\ P\ \{q\} \rangle$ is true under I. So, by induction, (43) holds for all n.

d is obviously true. Thus the system G is indeed sound.

3.8 Completeness of G in the Sense of Cook

We now prove the completeness of G for $\mathcal{S}_1$ in the sense of Cook. Let $\bar{x}$ be a sequence of all variables which occur in S_0 and let $\bar{z}$ be a sequence of some new variables of the same length as $\bar{x}$.

Assume that $I \in \text{Exp}(L, \mathcal{S}_1)$. There exists an assertion q_0 which defines $\text{post}_I(\bar{x} = \bar{z}, P)$. The correctness formula $\{\bar{x} = \bar{z}\}\ P\ \{q_0\}$ is called the *most general formula for P*, since any other true (under I) correctness formula about P can be derived from $\{\bar{x} = \bar{z}\}\ P\ \{q_0\}$ in G. This claim is the contents of the following lemma.

LEMMA 1. *If* $\models_I \{p\}\ S\ \{q\}$, *then* $Tr_I \cup \{\bar{x} = \bar{z}\}\ P\ \{q_0\} \vdash_G \{p\}\ S\ \{q\}$.

Proof. The proof proceeds by induction on the length of S. If $S \not\equiv P$, the proof is identical to the completeness proof for H.

Suppose that $S \equiv P$. Assume

$$\{\bar{x} = \bar{z}\}\ P\ \{q_0\}. \tag{45}$$

By the invariance axiom,

$$\{p_1[\bar{z}/\bar{x}]\}\ P\ \{p_1[\bar{z}/\bar{x}]\}, \tag{46}$$

where $p_1 \equiv p[\bar{u}/\bar{z}]$ and $\bar{u}$ is a sequence of some fresh variables of the same length as $\bar{z}$.

(45) and (46) imply by the conjunction rule that

$$\{\bar{x} = \bar{z} \wedge p_1[\bar{z}/\bar{x}]\}\ P\ \{q_0 \wedge p_1[\bar{z}/\bar{x}]\}. \tag{47}$$

We now show that

$$\models_I q_0 \wedge p_1[\bar{z}/\bar{x}] \rightarrow q_1 \qquad \text{where} \quad q_1 \equiv q[\bar{u}/\bar{z}]. \tag{48}$$

Assume $\models_I (q_0 \wedge p_1[\bar{z}/\bar{x}])(\tau)$. By the definition of q_0 there exists a state σ such that $\mathscr{M}_I(P)(\sigma) = \tau$ and $\mathscr{M}_I(\bar{x} = \bar{z})(\sigma)$. Suppose now that $\models_I \neg p_1[\bar{z}/\bar{x}](\sigma)$. Then, by the validity of the invariance axiom, $\models_I \neg p_1[\bar{z}/\bar{x}](\tau)$, since $\mathscr{M}_I(P)(\sigma) = \tau$. This contradicts our assumption, so $\models_I p_1[\bar{z}/\bar{x}](\sigma)$. Since $\models_I (\bar{x} = \bar{z} \wedge p_1[\bar{z}/\bar{x}]) \rightarrow p_1$, we now get $\models_I p_1(\sigma)$. But, by the soundness of substitution rule I, also $\models_I \{p_1\}\ P\ \{q_1\}$; so finally $\models_I q_1(\tau)$. This proves (48). (47) and (48) imply by the consequence rule that

$$\{\bar{x} = \bar{z} \wedge p_1[\bar{z}/\bar{x}]\}\ P\ \{q_1\}. \tag{49}$$

Now, by substitution rule II,

$$\{\bar{x} = \bar{x} \wedge p_1\}\ P\ \{q_1\}, \tag{50}$$

since $\bar{x} = \bar{x} \wedge p_1 \equiv (\bar{x} = \bar{z} \wedge p_1[\bar{z}/\bar{x}])[\bar{x}/\bar{z}]$. By the consequence rule,

$$\{p_1\}\ P\ \{q_1\}; \tag{51}$$

so, by substitution rule I,

$$\{p_1[\bar{z}/\bar{u}]\}\ P\ \{q_1[\bar{z}/\bar{u}]\}. \tag{52}$$

Clearly, $\models_I p \leftrightarrow p_1[\bar{z}/\bar{u}]$ and $\models_I q \leftrightarrow q_1[\bar{z}/\bar{u}]$; so, by the consequence rule,

$$\{p\}\ P\ \{q\}, \tag{53}$$

which was to be proved. □

The next lemma shows that the hypothesis $\{\bar{x} = \bar{z}\}\ P\ \{q_0\}$ used in the above lemma can actually be proved in G.

Lemma 2. $Tr_I \vdash_G \{\bar{x} = \bar{z}\}\ P\ \{q_0\}$.

Proof. The proof is immediate. By the definition of q_0, $\models_I \{\bar{x} = \bar{z}\}\ P\ \{q_0\}$; so $\models_I \{\bar{x} = \bar{z}\}\ S_0\ \{q_0\}$ since $\mathscr{M}_I(P) = \mathscr{M}_I(S_0)$. By Lemma 1, $\mathrm{Tr}_I \cup \{\bar{x} = \bar{z}\}\ P\ \{q_0\} \vdash_G \{\bar{x} = \bar{z}\}\ S_0\ \{q_0\}$; so, by the recursion rule, $\mathrm{Tr}_I \vdash_G \{\bar{x} = \bar{z}\}\ P\ \{q_0\}$. □

The completeness of G is the immediate consequence of Lemmas 1 and 2. Note that in the above proof the auxiliary variables in $\bar{z}$ were used to "freeze" the values of the variables in $\bar{x}$ before the procedure call.

3.9 Total Correctness of Recursive Procedures

Recursive procedures introduce another possibility of nontermination of programs. Clearly, the recursion rule does not provide any means to establish termination of the procedure calls, and so it is appropriate for proofs of partial correctness only.

Sokołowski [53] proposed the following refinement of the recursion rule, which leads to proofs of total correctness. This rule can also be found in Clarke [10a], where it is attributed to M. O'Donnell.

Rule 13: Recursion Rule II

$$\frac{\neg p(0),\ \{p(n)\}\ P\ \{q\} \vdash \{p(n+1)\}\ S_0\ \{q\}}{\{\exists n\, p(n)\}\ P\ \{q\}}\ .$$

Here, as in **while** rule II, $p(n)$ is an assertion with a free variable n which does not appear in S_0 and ranges over natural numbers.

The intuition behind this rule is the following. Call a computation (q, n)*-deep* if it terminates in a state satisfying q and if at any moment at most n calls of P are active in it. $\{p(n)\}\ S\ \{q\}$ is to be interpreted here as a statement that any execution of S starting in $p(n)$ is (q, n)-deep. The assumption $\{p(n)\}\ P\ \{q\}$ thus states that executions of P starting in states satisfying $p(n)$ are all (q, n)-deep and is used to show that all executions of P starting in states satisfying $p(n+1)$ are $(q, n+1)$-deep. The latter is shown by proving $\{p(n+1)\}\ S_0\ \{q\}$.

Using this rule, we can easily prove the correctness formula $\{x \geq 0\}\ P\ \{y = x!\}$ considered in Section 3.2 by taking $p(n) \equiv x \geq 0 \wedge x = n - 1$ and repeating the proof from Section 3.2.

However, a proof analogous to that in Section 3.3 shows that the system H_0 supplemented with recursion rule II is incomplete. Therefore, essentially following [23], we extend it to a proof system which is arithmetically complete. The extension is very similar to the corresponding extension of H + Rule 8 to G. We adopt substitution rule I and the following two proof rules.

Rule 14: Invariance Rule

$$\frac{\{p\}\ P\ \{q\}}{\{p \wedge r\}\ P\ \{q \wedge r\}} \qquad \textit{where} \quad var(r) \cap var(S_0) = \emptyset.$$

Rule 15: Elimination Rule

$$\frac{\{p\}\ P\ \{q\}}{\{\exists \bar{z}\, p\}\ P\ \{q\}} \qquad \textit{where} \quad \bar{z} \cap var(S_0, q) = \emptyset.$$

Call the resulting system G_0.

It should be clear that, if we substitute Rules 14 and 15 in G for the invariance axiom, substitution rule II, and the conjunction rule, then we also get a proof system for partial correctness which is complete in the sense of Cook. The completeness proof is in fact identical to that of G. The main reason for adopting a different extension here than in Section 3.4 is the fact that the invariance axiom is not valid when used for proofs of total correctness.

The arithmetical soundness of G_0 can be proved in a way analogous to the way the soundness of G was proved. When dealing with recursion rule II, one uses the premise $\neg p(0)$ to ensure that $\Vdash_I \langle P \Leftarrow S_0^{(0)} \mid \{p(0)\}\ P\ \{q\}\rangle$.

The proof of the arithmetical completeness of G_0 is "dual" to the completeness proof of G. Before presenting the proof, we introduce the following notion.

$$\mathrm{pret}_I(S, q) = \mathrm{pre}_I(S, q) \cap \{\delta : \exists\tau\,[\mathscr{M}_I(S)(\delta) = \tau]\}.$$

pret stands in the same relation to total correctness as pre does to partial correctness: we have $\Vdash_I \{p\}\ S\ \{q\}$ iff $\{\delta : \models_I p(\delta)\} \subset \mathrm{pret}_I(S, q)$.

Let $\bar{x}$ and $\bar{z}$ be defined as in Section 3.8 and let n be a fresh variable. Assume now that I is an arithmetical interpretation. It can be shown that there exists an assertion $p_0(n)$ such that for all states

$$\models_I p_0(\delta) \quad \text{iff} \quad \models_I \mathrm{nat}(n)(\delta) \ \text{ and } \ \delta \in \mathrm{pret}_I(S_0^{(k)}, \bar{x} = z), \quad \text{where} \ \ k = \delta(n).$$

The following lemmas show that $p_0(n)$ plays a role here analogous to that of q_0 in the completeness proof of G.

LEMMA 3. *If* $\Vdash_I \{p\}\ S\ \{q\}$, *then* $Tr_I \cup \{p_0(n)\}\ P\ \{\bar{x} = \bar{z}\} \vdash_{G_0} \{p\}\ S\ \{q\}$.

PROOF. The proof proceeds by induction on the length of S, and only the case $S \equiv P$ needs explanation.

Assume

$$\{p_0(n)\}\ P\ \{\bar{x} = \bar{z}\}. \tag{54}$$

By the invariance rule,

$$\{p_0(n) \wedge q_1[\bar{z}/\bar{x}]\}\ P\ \{\bar{x} = \bar{z} \wedge q_1[\bar{z}/\bar{x}]\}, \tag{55}$$

where $q_1 \equiv q[\bar{u}/\bar{z}]$ and $\bar{u}$ is a sequence of some fresh variables of the same length as $\bar{z}$. The implication

$$\bar{x} = \bar{z} \wedge q_1[\bar{z}/\bar{x}] \rightarrow q_1 \tag{56}$$

clearly holds; so, by the consequence rule,

$$\{p_0(n) \wedge q_1[\bar{z}/\bar{x}]\}\ P\ \{q_1\}. \tag{57}$$

By the elimination rule,

$$\{\exists n \exists z\ (p_0(n) \wedge q_1[\bar{z}/\bar{x}])\}\ P\ \{q_1\}. \tag{58}$$

We now show that

$$\models_I p_1 \rightarrow \exists n \exists \bar{z}\ (p_0(n) \wedge q_1[\bar{z}/\bar{x}]), \qquad \text{where} \quad p_1 \equiv p[\bar{u}/\bar{z}]. \tag{59}$$

First, note that, by the arithmetical soundness of substitution rule I, $\Vdash_I \{p_1\}\ P\ \{q_1\}$. Assume $\models_I p_1(\delta)$. By the definition of $\Vdash_I$ there exist k and a state τ such that $\mathscr{M}_I(S_0^{(k)})(\delta) = \tau$ and $\models_I q(\tau)$. Now let δ_1 be the state which agrees with δ on all variables not listed in n, $\bar{z}$ and such that $\delta_1(\bar{z}) = \tau(\bar{x})$ and $\delta_1(n) = k$. It is easy to see that $\models_I (p_0(n) \wedge q_1[\bar{z}/\bar{x}])(\delta_1)$. This shows that $\models_I \exists n \exists \bar{z}\ (p_0(n) \wedge q_1[\bar{z}/\bar{x}])(\delta)$; so (59) is proved.

(58) and (59) imply by the consequence rule that

$$\{p_1\}\ P\ \{q_1\}; \tag{60}$$

so, as in Section 3.8,

$$\{p\}\ P\ \{q\}. \qquad \Box \tag{61}$$

LEMMA 4. *$Tr_I \vdash_{G_0} \{p_0(n)\}\ P\ \{\bar{x} = \bar{z}\}$.*

PROOF. Observe that, by the definition of p_0, $\VDash_I \{p_0(n+1)\}\ P\ \{\bar{x} = \bar{z}\}$; so $\VDash_I \{p_0(n+1)\}\ S_0\ \{\bar{x} = \bar{z}\}$ as $\mathcal{M}_I(S_0) = \mathcal{M}_I(P)$. By Lemma 3, $\mathrm{Tr}_I \cup \{p_0(n)\}\ P\ \{\bar{x} = \bar{z}\} \vdash_{G_0} \{p_0(n+1)\}\ S_0\ \{\bar{x} = \bar{z}\}$. Clearly, $\models_I \neg p_0(0)$; so, by recursion rule II, $\mathrm{Tr}_I \vdash_{G_0} \{\exists n\, p_0(n)\}\ P\ \{\bar{x} = \bar{z}\}$. But the implication $p_0(n) \rightarrow \exists n\, p_0(n)$ obviously holds; so, by the consequence rule, $\mathrm{Tr}_I \vdash_{G_0} \{p_0(n)\}\ P\ \{\bar{x} = \bar{z}\}$. □

The completeness of G_0 now follows from Lemmas 3 and 4. Note that in the above proof the auxiliary variables in $\bar{z}$ were used to "freeze" the values of the variables in $\bar{x}$ after the procedure call.

It is easy to see that, following the reasoning presented above, one arrives at a dual completeness proof of the already-mentioned system H + Rules 8, 10, 14, and 15.

3.10 Bibliographical Remarks

Most of the papers dealing with procedures within the framework of Hoare's logic do not discuss parameterless procedures. In particular, Rule 8 is a special case of a rule given in [26]. The example in Section 3.2 is a modification of an example given by Hoare [26]. The semantics of recursive procedures in Section 3.6 is a translation into our framework of the corresponding definition from [50]. It is often used in the literature. The justification of the soundness of G seems to be new. The argument used in justifying c in Section 3.7 corresponds to the proof of the soundness of Scott's induction rule and often appears in the literature. The completeness proof given in Section 3.8 is a special case of the completeness proof presented in [19]. A similar completeness result was proved independently (but somewhat later) in [24]. The presentation of the proof system G_0 and its completeness proof in Section 3.9 slightly differs from that of Harel [23]. Harel's result was formulated within the context of dynamic logic. A similar completeness result was proved in [10a, 53]. The terminology of "freezing" is due to Harel, Pnueli, and Stavi [24].

4. VARIABLE DECLARATIONS

Let us now consider the least class $\mathcal{S}^{\mathrm{v}}$ of programs which, similarly to class $\mathcal{S}$, contains the assignment statements and is closed under the use of the composition (;), **while**, and **if-then-else** constructs but additionally satisfies the following condition:

if $S \in \mathcal{S}^{\mathrm{v}}$, then, for each variable x, **begin new** x; S **end** $\in \mathcal{S}^{\mathrm{v}}$.

new x stands for a declaration of a variable x which is valid within the *block* **begin new** x; S **end**; x is a *local* variable with respect to this block.

The occurrence of a variable x in a program S is called *bound* whenever it is within a subprogram of S of the form **begin new** x; S_1 **end**. An occurrence of x in S is *free* if it is not bound. Let free(S) denote the set of all variables which occur free in S. We define free(S, p) analogously.

By $S[y/x]$ we mean a substitution of y for the free occurrences of x in a program S. It is defined analogously to the substitution $p[y/x]$ for an assertion p. In particular, variable clashes which arise are resolved by appropriate renamings.

Let ω stand for a special constant to which we initialize all local variables. We might view ω as a symbol standing for "undefined." We now adopt the following proof rule:

RULE 16: VARIABLE DECLARATION RULE

$$\frac{\{p \wedge y = \omega\}\ S[y/x]\ \{q\}}{\{p\}\ \textbf{begin new}\ x;\ S\ \textbf{end}\ \{q\}} \qquad \textit{where} \quad y \notin \textit{free}(p, S, q).$$

The renaming of x for y is performed here to distinguish between the occurrences of local x in S and possible free occurrences of nonlocal x in p and q. The expression $y = \omega$ captures the idea of initialization.

4.1 Semantics for Variable Declarations

In order to pose the question of soundness of the variable declaration rule, as usual we must first define the meaning of the programs involved. We follow here the approach of Clarke [9].

To this purpose we redefine the notion of a state. By a *state* we now mean a *finite* function from the set of variables into the domain D of an interpretation I. For a state σ, if $x \notin \text{dom}(\sigma)$ and $d \in D$, then $\sigma \cup (x, d)$ stands for the extension of σ yielding d when applied to x. DROP(σ, x) stands for a state obtained from σ by deleting x from its domain.

We define the meaning of a program in the same way as before, with the only new case being that of a block. For all statements S we assume that $\mathcal{M}_I(S)(\sigma)$ is undefined if free(S) $\not\subseteq$ dom(σ).

Let $S_1 \equiv$ **begin new** x; S **end**, and suppose that free(S_1) $\subseteq$ dom(σ). We define

$$\mathcal{M}_I(S_1)(\sigma) = \text{DROP}(\mathcal{M}_I(S[y/x])(\sigma \cup (y, \underline{\omega})), y)$$

where y is the first variable not in dom(σ) and $\underline{\omega}$ is the value assigned to the constant ω.

Since we are using a new definition of a state, we have to provide a new notion of truth under I. Given an assertion p and a state σ, we define $\models_I p(\sigma)$ to hold in the event that p becomes true when all its free variables lying in dom(σ) get assigned values provided by σ and when the other free variables are universally quantified. For example, $\models_{I_N} (x = 0 \wedge z = z)((x, 0))$ holds. Thus, for $\models_I p(\sigma)$ to hold we do not need to have free(p) $\subseteq$ dom(σ). Thanks to this definition, we can now retain the definition of truth of an asserted program under I given in Section 2.4, with the only difference being that the new definition of state is now used. As a result, various former definitions and results do not need to be reconsidered with respect to the newly introduced semantics.

4.2 Soundness and Completeness of the System H + Rule 16

Soundness of Rule 16 follows from the fact that

$$\mathscr{M}_I(S[y/x])(\sigma \cup (y, d)) = \mathscr{M}_I(S[z/x])(\sigma \cup (z, d))$$
$$\text{for} \quad \text{any program } S \quad \text{and} \quad y, z \notin \text{dom}(\sigma). \qquad (62)$$

To prove this fact, one should actually strengthen the claim and rather prove that, for any program S and $y_1, \ldots, y_n, z_1, \ldots, z_n \notin \text{dom}(\sigma)$,

$$\begin{aligned}&\mathscr{M}_I(S[y_1/x_1] \cdots [y_n/x_n])(\sigma \cup (y_1, d_1) \cdots (y_n, d_n))\\ &\quad = \mathscr{M}_I(S[z_1/x_1] \cdots [z_n/x_n])(\sigma \cup (z_1, d_1) \cdots (z_n, d_n)).\end{aligned}$$

The last claim can be proved by induction on the structure of S, where only the case of blocks requires some caution due to the possibility of various variable clashes.

Soundness of the rules of H was proved (or rather stated) with respect to a different notion of state. But, since virtually the same definition of truth of an asserted program under I is now used, the same proofs of soundness apply. Thus the system H + Rule 16 is sound.

We now turn to the problem of the completeness of H + Rule 16. The definitions of pre, post, and expressiveness given in Section 2.8 should now be interpreted with respect to the new notion of state.

The system H + Rule 16 is easily seen to be complete in the sense of Cook. The case of blocks is dealt with using (62), and the other cases are the same as in the completeness proof of H.

4.3 Adding Procedures

Having settled the case of **while** programs, we now pass to the programs allowing procedure calls.

Consider an extension $\mathscr{S}_1^v$ of $\mathscr{S}^v$ in which the programs are allowed to contain calls of a nonrecursive parameterless procedure P. We assume a procedure declaration $P \Leftarrow S_0$ where $S_0 \in \mathscr{S}^v$. To provide a meaning to programs from $\mathscr{S}_1^v$, we proceed as in Section 3.1.

A program $S \in \mathscr{S}_1^v \backslash \mathscr{S}^v$ assumes the meaning assigned to it by the clause

$$\mathscr{M}_I(S) = \mathscr{M}_I(S[S_0/P]),$$

where $S[S_0/P]$ stands for the substitution of all occurrences of P by S_0. Because of possible variable clashes, we now have to define $S[S_0/P]$ carefully. $S[S_0/P]$ is defined by induction, with the main clause being

$$\textbf{begin new } x;\ S \textbf{ end } [S_0/P]$$
$$\equiv \begin{cases}\textbf{begin new } x;\ S[S_0/P] \textbf{ end} & \text{if } x \notin \text{free}(S_0);\\ \textbf{begin new } x';\ S[x'/x][S_0/P] \textbf{ end} \quad \text{where } x' \notin \text{free}(S, S_0) & \text{otherwise.}\end{cases}$$

The aim of this clause is to avoid binding any of the free occurrences of the variables in S_0 through the substitution process. The other clauses are defined in a natural way.

To prove the correctness of programs from $\mathscr{S}_1^{v}$, we now use in addition Rule 7. However, to apply Rule 16 to programs from $\mathscr{S}_1^{v}$ we have to require additionally that $y \notin \text{free}(S_0)$. It is easy to see that otherwise this rule becomes unsound. To illustrate the use of Rule 16 in conjunction with Rule 7, we give the following example. Consider the procedure declaration $P \Leftarrow x := z$ and the program $S \equiv z := 1;$ **begin new** z; $z := 0$; P **end**. We now prove $\{\textbf{true}\}\ S\ \{x = 1\}$.

By the assignment axiom,

$$\{z = 1\}\ x := z\ \{x = 1\};$$

so, by Rule 7,

$$\{z = 1\}\ P\ \{x = 1\}.$$

By the assignment axiom and the consequence and composition rules,

$$\{z = 1 \wedge y = \omega\}\ y := 0;\ P\ \{x = 1\},$$

from which we obtain the desired formula $\{\textbf{true}\}\ S\ \{x = 1\}$.

It should be noted that, according to ALGOL 60 semantics, the value of x after the execution of S should be 1 and not 0.

We now prove the soundness and completeness in the sense of Cook of the system H + Rules 7, 16. We proceed as in Section 3.1. The definition of semantics of blocks given above provides the meaning for programs from $\mathscr{S}_1^{v}$ in terms of the meaning of programs from $\mathscr{S}^{v}$. Therefore, we can easily reduce the problem to that of the soundness and completeness of H + Rule 16 for $\mathscr{S}^{v}$. The only case in both proofs which requires some explanation is that of blocks.

If $x, y \notin \text{free}(S_0)$, then, for $S \in \mathscr{S}_1^{v}$,

$$\mathscr{M}_I(S[y/x]) = \mathscr{M}_I(S[y/x][S_0/P]) = \mathscr{M}_I(S[S_0/P][y/x])$$

and

$$\mathscr{M}_I(\textbf{begin new } x;\ S \textbf{ end}) = \mathscr{M}_I(\textbf{begin new } x;\ S[S_0/P] \textbf{ end}).$$

If $y \notin \text{free}(S_0)$ and $x \in \text{free}(S_0)$, then

$$\mathscr{M}_I(S[y/x]) = \mathscr{M}_I(S[y/x][S_0/P]) = \mathscr{M}_I(S[x'/x][S_0/P][y/x'])$$

and

$$\begin{aligned}\mathscr{M}_I(\textbf{begin new } x;\ S \textbf{ end}) &= \mathscr{M}_I(\textbf{begin new } x;\ S \textbf{ end}[S_0/P])\\ &= \mathscr{M}_I(\textbf{begin new } x';\ S[x'/x][S_0/P] \textbf{ end})\\ &\qquad \text{where} \quad x' \notin \text{free}(S, S_0).\end{aligned}$$

This shows that in both cases the soundness of Rule 16 applied to programs from $\mathscr{S}_1^{v}$ indeed follows from the soundness of Rule 16 applied to programs from $\mathscr{S}^{v}$. Similar reduction takes care of the appropriate case in the completeness proof.

We now consider the case of recursive procedures. To provide a semantics in the case in which P is recursive, we proceed as in Section 3.6. If we assume now that $S_0 \in \mathscr{S}_1^{\text{v}} \backslash \mathscr{S}^{\text{v}}$, we can define a meaning of a program $S \in \mathscr{S}_1^{\text{v}} \backslash \mathscr{S}^{\text{v}}$ by putting

$$\mathscr{M}_I(S) = \bigcup_{n=0}^{\infty} \mathscr{M}_I(S[S_0^{(n)}/P]) \qquad \text{where} \quad S_0^{(0)} \equiv \Omega;$$

$$S_0^{(n+1)} \equiv S_0[S_0^{(n)}/P].$$

Using these definitions, the proof system G + Rule 16 for $\mathscr{S}_1^{\text{v}}$ is sound and complete in the sense of Cook. Indeed, the arguments used in Sections 3.7 and 3.8 can be applied here without any changes. The additional case of blocks is reduced as above to the already handled case of programs from $\mathscr{S}^{\text{v}}$.

4.4 Problems with Uninitialized Variables

In Rule 16 we incorporated the assumption that each local variable is initialized by using the formula $y = \omega$. To prove the soundness (and completeness) of this rule, we were forced later to reflect this assumption while defining semantics for blocks. But is this assumption needed?

We might equally well drop the formula $y = \omega$ from the premise of the rule and, while providing a semantics, initialize each local variable to, say, the first element of the domain of I not in range(σ). It should be clear that with such changes the claim (62) retains its validity; so both the soundness and the completeness proofs remain valid.

Why then did we not adopt this simpler solution? The answer is subtle. Consider the following correctness formula:

$$\{\textbf{true}\}\ \textbf{begin new}\ z;\ x := z\ \textbf{end};\ \textbf{begin new}\ u;\ y := u\ \textbf{end}\ \{x = y\}.$$

According to the semantics we adopted and also the semantics we have just suggested, this formula is true under any interpretation I. It is also easy to prove it in the system H + Rule 16, since clearly both

$$\{\textbf{true}\}\ \textbf{begin new}\ z;\ x := z\ \textbf{end}\ \{x = \omega\}$$

and

$$\{x = \omega\}\ \textbf{begin new}\ u;\ y := u\ \textbf{end}\ \{x = y\}$$

can be proved.

If, however, we adopt the proposal just suggested, we *cannot* find any intermediate assertion which would play the role of $x = \omega$ above. What is more, the suggested semantics results in an inexpressiveness of *any* L relative to *any* I and $\mathscr{S}^{\text{v}}$! (The case in which $|I| = 1$ should be excluded here, since the suggested semantics is then ill-defined.) To see this, note that the set post$_I$(**true**, **begin new** z; $x := z$ **end**) is not definable by any formula of L. Thus the completeness proof is indeed valid but vacuously so.

All these problems were caused here by the use of uninitialized local variables. We could avoid these difficulties by simply disallowing programs in which some local variables are uninitialized. Such a class of programs can easily be defined, and the newly suggested approach can be taken in dealing with it. This is the solution adopted in [14].

4.5 Scope Issues

Our discussion concerning local variables would not be complete without providing an answer to the following seemingly innocent question. Why did we not adopt the following rule?

RULE 17: VARIABLE DECLARATION RULE II

$$\frac{\{p[y/x] \wedge x = \omega\}\ S\ \{q[y/x]\}}{\{p\}\ \textbf{begin new}\ x;\ S\ \textbf{end}\ \{q\}} \qquad \textit{where} \quad y \notin \textit{free}(p, S, q).$$

Here the substitution is performed in assertions and not programs; consequently, the rule should be easier to use and handle.

The answer touches the issue of the *scope* of identifiers in programs. It is not difficult to see that the proof systems H + Rule 16 and H + Rule 17 are equivalent. However, if we allow procedures, the corresponding proof systems H + Rules 7, 16 and H + Rules 7, 17 are no longer equivalent. To see this, take the correctness formula $\{\textbf{true}\}\ S\ \{x = 1\}$ considered in Section 4.3. It is easy to see that it cannot be proved in H + Rules 7, 17. A straightforward proof shows rather that $\{\textbf{true}\}\ S\ \{x = 0\}$ holds.

We say that a *static scope* is assumed if each procedure call is evaluated in the environment in which the procedure has been declared. In our case this means that all free variables of the procedure body are understood to be the free variables of the program. If, on the other hand, each procedure call is evaluated in the environment in which the procedure is called, then we say that *dynamic scope* is assumed.

Using this terminology, we can say that Rule 16 leads to static scope, whereas Rule 17 leads to dynamic scope. All ALGOL-like languages assume static scope. Therefore, one might think that dealing with dynamic scope is irrelevant. However, as we see in Sections 6 and 7, a theoretical analysis of scope issues within the framework of Hoare's logic reveals important differences between these two scope assumptions and sheds light on the static scope assumption.

To conclude this discussion, we provide a semantics for blocks which leads to a dynamic scope assumption. This semantics is due to Clarke [9]. For this purpose we need a slightly refined notion of a state. By a state we mean here a finite sequence of pairs (x, d) where x is a variable and d an element of the domain D of an interpretation I. A variable can occur in more than one pair belonging to a state. By dom(σ) we now mean the set of all variables which belong to a pair from σ.

For $x \in \text{dom}(\sigma)$ let (x, d) be the last pair in σ to which x belongs. We define this d to be the value of x in state σ. $\sigma \cup (x, d)$ now stands for the result of extending σ with the element (x, d), whereas DROP(σ, x) stands for the sequence obtained from σ by deleting the last pair to which x belongs.

Assume now that $\text{free}(S_1) \subseteq \text{dom}(\sigma)$ where $S_1 \equiv \textbf{begin new}\ x; S\ \textbf{end}$. We define

$$\mathcal{M}_I(S_1)(\sigma) = \text{DROP}(\mathcal{M}_I(S)(\sigma \cup (x, \underline{\omega})), x).$$

It can be shown that Rule 17 is sound for programs from $\mathcal{S}^v$ when the above semantics is used. Also, the corresponding completeness result concerning H + Rule 17 holds.

To extend these results to the case of programs admitting procedures, we have to provide a semantics for such programs. Given a procedure declaration $P \Leftarrow S_0$ and a program S, let $S\langle S_0/P\rangle$ stand for the result of a literal replacement of each occurrence of P in S by S_0. If P is nonrecursive, we define the meaning of a program $S \in \mathscr{S}_1^v \backslash \mathscr{S}^v$ by

$$\mathscr{M}_I(S) = \mathscr{M}_I(S\langle S_0/P\rangle).$$

If P is a recursive procedure, then we put

$$\mathscr{M}_I(S) = \bigcup_{n=0}^{\infty} \mathscr{M}_I(S\langle S_0^{(n)}/P\rangle) \qquad \text{where} \quad S_0^{(0)} = \Omega \qquad \text{and} \qquad S_0^{(n+1)} = S_0\langle S_0^{(n)}/P\rangle.$$

The corresponding soundness and completeness results concerning the systems H + Rules 7, 17 and G + Rule 17 now follow by the same reasoning as was used in Section 4.3.

Note that the difficulties with the use of uninitialized variables arise in the case of dynamic scope as well.

4.6 Bibliographical Remarks

Rule 16 (without the formula $y = \omega$) is from [26]. The addition of the formula $y = \omega$ first appears in [19]. The use of substitution in programs in the definition of semantics of blocks independently appeared in [4]. The soundness and completeness of H + Rule 16, H + Rules 7, 16, and G + Rule 16 are special cases of a completeness result mentioned in [9] and proved in [1]. [14] provides a detailed proof of the soundness and completeness of the system H + Rule 16. Rule 17 (without $y = \omega$) is due to [37], where also its soundness is proved. The completeness of H + Rules 7, 17 and the completeness of G + Rule 17 are special cases of the completeness results of [13] and [19], respectively. All these results are subsumed by the results of [45].

5. SUBSCRIPTED VARIABLES

So far we have allowed assignment to simple variables only. Allowing subscripted variables in expressions and assignments leads to extension of the previous syntax. To keep things simple we restrict our attention to the case of one-dimensional arrays, omitting any specifications of the bounds.

Let $\mathscr{AV}$ be a set of array variables. We extend the syntax of L by allowing expressions of the form $a[t]$ for $a \in \mathscr{AV}$ and t being an expression, and we now allow an assignment $a[s] := t$ where $a \in \mathscr{AV}$ and s, t are expressions.

5.1 An Assignment Axiom for Subscripted Variables

In what follows we assume that conditional expressions of the form **if** $s = t$ **then** t_1 **else** t_2 **fi** (so-called *equality conditionals*) are allowed. To obtain a better picture of the problem, we consider first the case of an assignment when the subscript is a simple variable.

By $p[t/a[x]]$ we denote a substitution of an expression t for the subscripted variable $a[x]$. It is defined by induction, with the main clause being

$$a[z][t/a[x]] \equiv \textbf{if } z = x \textbf{ then } t \textbf{ else } a[z] \textbf{ fi}.$$

Essentially the following axiom was proposed in [29]:

$$\{p[t/a[x]]\}\ a[x] := t\ \{p\}.$$

Using this axiom, we can prove

$$\{x = y\}\ a[x] := 1\ \{a[y] = 1\}$$

since $(a[y] = 1)[1/a[x]] \equiv \textbf{if } y = x \textbf{ then } 1 \textbf{ else } a[y] \textbf{ fi} = 1$, and this formula is implied by $x = y$. Also, $\{\textbf{true}\}\ a[x] := 1\ \{a[x] = 1\}$ holds.

If we now allow arbitrary expressions as subscripts, we run into difficulties. Namely, the formula $\{\textbf{true}\}\ a[t] := 1\ \{a[t] = 1\}$ is no longer valid! To see this, observe that, if $a[1] = a[2] = 2$, then $\{\textbf{true}\}\ a[a[2]] := 1\ \{a[a[2]] = 1\}$ is not true.

This shows that the above definition of substitution has to be appropriately refined for the general case of subscripts being arbitrary expressions. Perhaps the simplest solution to this problem is to circumvent it. The above substitution still leads to correct results when used for arbitrary subscripted variables if applied only to assertions allowing simple variables as subscripts. The main clause of this substitution is thus

$$a[z][t/a[s]] \equiv \textbf{if } z = s \textbf{ then } t \textbf{ else } a[s] \textbf{ fi},$$

and the case in which an arbitrary expression stands for z is simply not handled.

Given now an arbitrary assertion p, let p' denote an assertion equivalent to p which is obtained by "quantifying out" all subscripts which are not simple variables. For example, if p is $a[a[2]] = 1$, then p' is $\exists z\ (a[z] = 1 \wedge z = a[2])$. We now extend the above substitution to arbitrary assertions by putting $p[t/a[s]] \equiv p'[t/a[s]]$. We finally arrive at a general form of the axiom:

AXIOM 18: ASSIGNMENT AXIOM FOR SUBSCRIPTED VARIABLES

$$\{p[t/a[s]]\}\ a[s] := t\ \{p\}.$$

Note the similarity in form between this axiom and Axiom 1.

By way of example, we now prove

$$\{a[2] = 2 \rightarrow a[1] = 1\}\ a[a[2]] := 1\ (a[a[2]] = 1\}.$$

We have

$$\begin{aligned}(a[a[2]] = 1)[1/a[a[2]]] &\equiv (\exists z\ (a[z] = 1 \wedge a[2] = z))[1/a[a[2]]]\\ &\equiv \exists z\ (\textbf{if } z = a[2] \textbf{ then } 1 \textbf{ else } a[z] \textbf{ fi} = 1\\ &\qquad \wedge \textbf{if } 2 = a[2] \textbf{ then } 1 \textbf{ else } a[2] \textbf{ fi} = z).\end{aligned}$$

The last formula is implied by the assertion $a[2] = 2 \rightarrow a[1] = 1$. Namely, if $a[2] = 2$ holds, then choose $z = 1$, and otherwise choose $z = a[2]$. Thus, by the consequence rule and Axiom 18 above, we get the desired result.

Axiom 18 is also complete in the following sense: if $\models_I \{p\}\ a[s] := t\ \{q\}$, then $\models_I p \rightarrow q[t/a[s]]$. So, if $\models_I \{p\}\ a[s] := t\ \{q\}$, then $\mathrm{TR}_I \vdash_{\text{Rule 5 + Axiom 18}} \{p\}\ a[s] := t\ \{q\}$, where TR_I is the set of all assertions of the extension of L which are true under I.

5.2 Bibliographical Remarks

The final form of the axiom is motivated by the solutions given in [15] and (for the simpler case) in [43]. Validity and completeness of the axiom follow from [15], from which the example is taken as well (see also [14]). Different solutions for the assignment to subscripted variables are given in [28, 30, 48] and [15] (also to be found in [14]). [30] also provides a solution for the case of an assignment of the form **if** e **then** x **else** y **fi** $:= t$ and an assignment to pointers. [20] and [21] provide solutions for the case of a multiple assignment to subscripted variables.

6. PARAMETER MECHANISMS

Parameter mechanisms are among the most troublesome issues in the framework of Hoare's logic. One of the reasons is that parameter passing is always modeled syntactically by some form of variable substitution in a program, and this leads to various subtle problems concerning variable clashes. These difficulties are especially acute in the presence of recursion and static scoping.

In contrast to our exposition of the previous sections, the presentation cannot be complete here, as not all problems have been solved in this area. In the subsequent discussion we attempt to clarify which particular issues lead to difficulties and indicate what still remains to be done in this area.

We begin our presentation with a treatment of the parameter mechanism of call-by-name in the presence of the dynamic scope assumption. These results do not concern most usual features of programming languages. However, techniques introduced to deal with them form an adequate basis to study more common parameter mechanisms under the assumption of static scope. Therefore, it is useful to treat these features first.

6.1 Call-by-Name

6.1.1 *Nonrecursive Procedures.* Consider a procedure declaration of the form

$$P \Leftarrow \langle \mathbf{name}(\bar{x} : \bar{v}) \mid S_0 \rangle,$$

where $(\bar{x} : \bar{v})$ is the formal parameter list and $S_0 \in \mathscr{S}^{\mathrm{v}}$ 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 S_0. S_0 does not contain procedure calls; so P is not recursive.

In the extension of $\mathscr{S}^{\mathrm{v}}$, called $\mathscr{S}^{\mathrm{v}}_2$, we allow procedure calls of the form $P(\bar{u} : \bar{t})$ where $\bar{u}$ is a list of distinct variables, $\bar{t}$ is a list of expressions containing no variable in $\bar{u}$, and no variable in $(\bar{u} : \bar{t})$ different from formal parameters occurs free in the procedure body S_0.

All procedure calls mentioned below are assumed to satisfy the above restrictions.

$S_0[\bar{u}, \bar{t}/\bar{x}, \bar{v}]$ indicates the result of simultaneous substitution of the actual parameters $\bar{u}$, $\bar{t}$ for the corresponding free occurrences of the formal parameters $\bar{x}$ and $\bar{v}$ in S_0.

Following [13], we now supplement the proof system H + Rule 16 by the following three proof rules.

RULE 19: PROCEDURE CALL RULE II

$$\frac{\{p\}\ S_0\ \{q\}}{\{p\}\ P(\bar{x}:\bar{v})\ \{q\}}.$$

RULE 20: PARAMETER SUBSTITUTION RULE

$$\frac{\{p\}\ P(\bar{x}':\bar{v}')\ \{q\}}{\{p[\bar{u}, \bar{t}/\bar{x}', \bar{v}']\}\ P(\bar{u}:\bar{t})\ \{q[\bar{u}, \bar{t}/\bar{x}', \bar{v}']\}} \quad \textit{where} \quad \bar{u} \cap \mathit{free}(p, q) \subseteq \bar{x}'.$$

RULE 21: VARIABLE SUBSTITUTION RULE

$$\frac{\{p\}\ P(\bar{u}:\bar{t})\ \{q\}}{\{p[\bar{s}/\bar{z}]\}\ P(\bar{u}:\bar{t})\ \{q[\bar{s}/\bar{z}]\}}$$

where no variable in $\bar{s}$ *or* $\bar{z}$ *occurs free in* $S_0[\bar{u}, \bar{t}/\bar{x}, \bar{v}]$.

Call the resulting system C.

The last two rules are rather difficult to understand because of the restrictions imposed on the substituted expressions and variables. To get a better idea of how these proof rules are used, consider an example proof in C.

Assume the declaration $P \Leftarrow \langle \mathbf{name}(x:v) \mid x := v;\ a := v\rangle$. According to the imposed restrictions, the calls $P(y:y+1)$ or $P(y:a+1)$ are disallowed, but calls $P(z:y+1)$, $P(v:y+1)$, or $P(v:x+1)$ are allowed. We now prove

$$\{x = z\}\ P(v:y+1)\ \{v = y+1 \wedge a = y+1 \wedge x = z\}.$$

To this purpose we have to rename the formal parameter x of the procedure occurring free in the assertions. Therefore, we first prove

$$\{u = z\}\ P(v:y+1)\ \{v = y+1 \wedge a = y+1 \wedge u = z\}.$$

We have

$$\{u = z\}\ x := v;\ a := v\ \{x = v \wedge a = v \wedge u = z\};$$

so, by procedure call rule II,

$$\{u = z\}\ P(x:v)\ \{x = v \wedge a = v \wedge u = z\}.$$

Now, by the parameter substitution rule,

$$\{u = z\}\ P(x:v')\ \{x = v' \wedge a = v' \wedge u = z\};$$

so once again, by the same rule,

$$\{u = z\}\ P(v:y+1)\ \{v = y+1 \wedge a = y+1 \wedge u = z\}.$$

Finally, by the variable substitution rule,

$$\{x = z\}\ P(v:y+1)\ \{v = y+1 \wedge a = y+1 \wedge x = z\}.$$

The reader can check that in all steps the corresponding restrictions were obeyed. Note that the direct step from the call $P(x:v)$ to $P(v:y+1)$ is not allowed. Namely, the parameter substitution rule requires here that the actual parameter v in the call $P(v:y+1)$ be identical to x in $P(x:v)$, as v occurs free in the assertion $x = v \wedge a = v \wedge u = z$.

It is worthwhile to note that the restrictions mentioned in the substitution rules are necessary. To see this, consider the procedure declaration $P \Leftarrow \langle \textbf{name}(x:)\, | \, x := 0\rangle$. We now have $\{u = 1\}\ P(x:)\ \{u = 1\}$, but of course $\{u = 1\}\ P(u:)\ \{u = 1\}$ is not true. This shows that the restriction in the parameter substitution rule is needed.

Also, $\{x = 1\}\ P(x:)\ \{x = 1\}$ is not true; so the corresponding restriction in the variable substitution rule is necessary as well.

This artificial example provides a better insight into the nature and use of the substitution rules. Note that the variable substitution rule was used here to rename free occurrences of the formal parameters in the assertions of the correctness formula to be proved. The step from $P(x:v)$ to $P(v:y+1)$ had to be split in two; so the parameter substitution rule had to be used twice here.

We now continue with the discussion of the system C.

Let S be a program from $\mathscr{S}_2^{\mathrm{v}} \backslash \mathscr{S}^{\mathrm{v}}$. We define the meaning of S by putting

$$\mathscr{M}_I(S) = \mathscr{M}_I(S\langle S_0/P\rangle)$$

where $S\langle S_0/P\rangle \in \mathscr{S}^{\mathrm{v}}$ is the result of the literal replacement of each procedure call $P(\bar{u}:\bar{t})$ occurring in S by $S_0[\bar{u}, \bar{t}/\bar{x}, \bar{v}]$.

The results of [13] imply that the proof system C is sound and complete in the sense of Cook, where the definition of the meaning of blocks from Section 4.5 is used. The proofs are delicate, mainly due to the possibility of various variable clashes in the substitution rules.

The only new case in the completeness proof is that of the procedure calls. We present here an argument which only works if the formal and actual parameters have no variables in common and if the assertions p and q do not have free occurrences of formal parameters. Note that in the above example the first issue forced us to use the parameter substitution rule twice, and the second problem was resolved by the use of the variable substitution rule. These difficulties are resolved in a similar way in the completeness proof of the general case, which is a careful refinement of the argument presented below.

Suppose

$$\models_I \{p\}\ P(\bar{u}:\bar{t})\ \{q\}.$$

Then, by definition of $\mathscr{M}_I$,

$$\models_I \{p\}\ S_0[\bar{u}, \bar{t}/\bar{x}, \bar{v}]\ \{q\}.$$

From this it follows due to the above restrictions that

$$\models_I \{\bar{v} = \bar{t} \wedge p[\bar{x}/\bar{u}]\}\ S_0\ \{q[\bar{x}/\bar{u}]\},$$

as no variable clashes arise here.

S_0 does not contain procedure calls; so, by the completeness of H + Rule 17,

$$\mathrm{Tr}_I \vdash_C \{\bar{v} = \bar{t} \wedge p[\bar{x}/\bar{u}]\}\ S_0\ \{q[\bar{x}/\bar{u}]\}.$$

By the rule of procedure calls,

$$\mathrm{Tr}_I \vdash_C \{\bar{v} = \bar{t} \wedge p[\bar{x}/\bar{u}]\}\ P(\bar{x}:\bar{v})\ \{q[\bar{x}/\bar{u}]\}.$$

By the parameter substitution rule,

$$\mathrm{Tr}_I \vdash_C \{\bar{t} = \bar{t} \wedge p\}\ P(\bar{u} : \bar{t})\ \{q\};$$

so, finally, by the consequence rule,

$$\mathrm{Tr}_I \vdash_C \{p\}\ P(\bar{u} : \bar{t})\ \{q\}.$$

Note that the parameter substitution rule can be applied here, since, by the assumption, $\bar{u} \cap \mathrm{free}(\bar{t}) = \varnothing$ and, by the imposed restriction, $\bar{u} \cap (\bar{x} \cup \bar{v}) = \varnothing$; so no variable from $\bar{u}$ occurs free in the assertions from the premise of the rule.

Note that Rules 19–21 could be replaced here by the simpler rule

$$\frac{\{p\}\ S_0[\bar{u}, \bar{t}/\bar{x}, \bar{v}]\ \{q\}}{\{p\}\ P(\bar{u} : \bar{t})\ \{q\}},$$

and soundness and completeness would be preserved.

If this rule were adopted, the restrictions concerning the procedure calls would be unneeded. However, if this rule were used, its hypothesis would have to be verified each time a procedure call with different actual parameters appeared. As a result, the actual proofs would be longer in general than the corresponding proofs in the proof system C, where it is sufficient to prove a general property $\{p\}\ S_0\ \{q\}$ of the procedure body just once.

6.1.2 *Recursive Procedures.* [19] contains an extension of the above result to the case of a recursive procedure. The relevant proof system is the following modification of the system G:

a. in the recursion rule P is replaced by $P(\bar{x} : \bar{v})$;
b. the invariance axiom takes the form

$$\{p\}\ P(\bar{u} : \bar{t})\ \{p\}$$

where p has no free variable occurring free in $S_0[\bar{u}, \bar{t}/\bar{x}, \bar{v}]$;
c. substitution rule I is replaced by the variable substitution rule;
d. substitution rule II takes the form

$$\frac{\{p\}\ P(\bar{u} : \bar{t})\ \{q\}}{\{p[\bar{s}/\bar{z}]\}\ P(\bar{u} : \bar{t})\ \{q\}} \qquad \text{where} \quad \bar{z} \cap \mathrm{free}(S_0[\bar{u}, \bar{t}/\bar{x}, \bar{v}], q) = \varnothing;$$

e. in the conjunction rule P is replaced by $P(\bar{u} : \bar{t})$; and
f. the parameter substitution rule is added.

Our definition of the meaning of programs containing procedure calls is similar to the definition in Section 4.5 (so using the dynamic scope requirement). Since the procedures now have parameters, we have to be careful so as to perform the appropriate substitutions of the actual parameters for formals in the proper order. Therefore, we proceed in a slightly different manner.

By induction on n we define a sequence of procedure declarations D_n

$$P_n \Leftarrow \langle \mathbf{name}\ (\bar{x} : \bar{v}) \mid S_0^{(n)} \rangle \qquad \text{where} \quad S_0^{(0)} \equiv \Omega \qquad \text{and} \qquad S_0^{(n+1)} \equiv S_0[P_n/P].$$

$S[P_n/P]$ stands for the result of substituting the procedure identifier P by P_n in a program S. We define the meaning of a program S by putting

$$\mathcal{M}_I(S) = \bigcup_{n=0}^{\infty} \mathcal{M}_I(S[P_n/P]),$$

where the context of the procedure declarations D_n is assumed. In particular,

$$\mathcal{M}_I(P(\bar{u}:\bar{t})) = \bigcup_{n=0}^{\infty} \mathcal{M}_I(P_n(\bar{u}:\bar{t})).$$

Due to these definitions,

$$\mathcal{M}_I(S[P_n/P]) = \mathcal{M}_I(S^{[n]}) \qquad \text{where} \quad S^{[n]} \equiv S[P_n/P]\langle S_0^{(n)}/P_n\rangle \cdots \langle S_0^{(0)}/P_0\rangle.$$

Observe that $S^{[n]}$ is the result of repeated literal replacement of each procedure call $P(\bar{u}:\bar{t})$ by $S_0[\bar{u}, \bar{t}/\bar{x}, \bar{v}]$ performed from the "top" being S to the depth n, followed by the literal replacement of each procedure call by Ω.

Strictly speaking, the above definitions require an extension of the considered syntax by allowing a system of nonrecursive procedure declarations $D_1, \ldots, D_n$. The results of [13] cover such a case.

The soundness proof of the above system can now be established following the reasoning used in Section 3.7 to prove the soundness of G. Due to the soundness of an extension of C dealing with a system of nonrecursive procedures, it is sufficient to prove the validity of the invariance axiom, the soundness of substitution rule II and the conjunction rule in the case of a system of nonrecursive procedure declarations, and the goodness of the recursion rule. Of course, in all cases we mean the modified versions of the axioms and proof rules. All proofs are straightforward.

The completeness proof is analogous to the completeness proof of G given in Section 3.8. For the most general formula for P we now choose the correctness formula $\{\bar{z}' = \bar{z}\}\ P(\bar{x}:\bar{v})\ \{q_0\}$ where $\bar{z}'$ is a sequence of all variables which occur free in S_0, $\bar{z}$ is a sequence of some new variables of the same length as $\bar{z}'$, and q_0 is an assertion which defines $\text{post}_I(\bar{z}' = \bar{z}, P(\bar{x}:\bar{v}))$. In the proof of a lemma corresponding to Lemma 1 from Section 3.8, we now have to tackle the case of procedure calls with actual parameters different from the formal ones. The other cases are the same as before. The argument used by Cook [13] in the completeness proof of C shows the following implicitly.

There exist two assertions p_1 and q_1 which depend on p, q, S_0, and $\bar{u}$, $\bar{t}$ such that

1. the proof rule

$$\frac{\{p\}\ P(\bar{u}:\bar{t})\ \{q\}}{\{p_1\}\ P(\bar{x}:\bar{v})\ \{q_1\}}$$

is sound in the case of a nonrecursive procedure declaration, and

2. $\{p_1\}\ P(\bar{x}:\bar{v})\ \{q_1\} \vdash_{C\text{-Rule 19}} \{p\}\ P(\bar{u}:\bar{t})\ \{q\}$.

In the special case of the completeness proof of C which we considered here, we can take $p_1 \equiv \bar{v} = \bar{t} \wedge p[\bar{x}/\bar{u}]$ and $q_1 \equiv q[\bar{x}/\bar{u}]$. That conditions 1 and 2 are satisfied immediately follows from the argument presented here.

Thanks to the arguments used in Section 3.7, the above proof rule is also sound in the case of a recursive procedure declaration. Assume now that

$$\models_I \{p\}\ P(\bar{u}:\bar{t})\ \{q\}.$$

By the above,

$$\models_I \{p_1\}\ P(\bar{x}:\bar{v})\ \{q_1\}.$$

Repeating the reasoning of Lemma 1, we get

$$\{\bar{z}' = \bar{z}\}\ P(\bar{x}:\bar{v})\ \{q_0\} \vdash \{p_1\}\ P(\bar{x}:\bar{v})\ \{q_1\};$$

hence, due to 2,

$$\{\bar{z}' = \bar{z}\}\ P(\bar{x}:\bar{v})\ \{q_0\} \vdash \{p\}\ P(\bar{u}:\bar{t})\ \{q\},$$

which was to be proved.

The rest of the proof is the same as in Section 3.7.

The systems presented in this subsection assume dynamic scope. However, the relevant results should also hold when static scope is assumed. The main problem with such proofs is that the soundness of Rule 16, the first variable declaration rule, in the presence of parameter mechanisms becomes much more difficult to prove.

6.2 Call-by-Value and Call-by-Variable

6.2.1 *Nonrecursive Procedures.* In this section we consider the parameter mechanisms of call-by-value and call-by-variable, which can be found in the programming language PASCAL and other languages. We allow local variables as well as subscripted variables. Consider a declaration

$$P \Leftarrow B \qquad \text{where} \quad B \equiv \langle \mathbf{val}\ x;\ \mathbf{var}\ y \mid S_0 \rangle$$

of a nonrecursive procedure P. x and y are the formal value and variable parameters, respectively, and S_0, as usual, is the procedure body.

To provide a meaning for procedure calls and to deal with procedure calls in a proof system, we introduce the following notation:

$$B[t, z] \equiv \mathbf{begin\ new}\ u;\ u := t;\ S_0[u/x][z/y]\ \mathbf{end},$$

$$B[t, a[s]] \equiv \mathbf{begin\ new}\ u_1, u_2;\ u_1 := t;\ u_2 := s;\ S_0[u_1/x][a[u_2]/y]\ \mathbf{end},$$

where u is the first simple variable $\not\equiv x, y$ and not free in S_0, t, or z (and analogously for u_1, u_2).

The above notation assumes a straightforward extension of the former definitions in that it uses substitution of a subscripted variable for a simple one in a program and uses a declaration of two local variables u_1, u_2 in one block.

Let v stand for a variable which is either simple or subscripted. We define the meaning of procedure calls by putting

$$\mathcal{M}_I(P(t, v)) = \mathcal{M}_I(B[t, v]),$$

and, consequently, for a program S containing calls of P,

$$\mathcal{M}_I(S) = \mathcal{M}_I(S[S_0/P])$$

where $S[S_0/P]$ is the result of substituting each procedure call $P(t, v)$ occurring in S by $B[t, v]$. $S[S_0/P]$ is defined analogously to the way it is defined in Section 4.3. We assume that $\mathcal{M}_I$ is defined in an appropriate way for programs using subscripted variables and not containing procedure calls.

We now adopt the following proof rule:

$$\frac{\{p\}\ B[t, v]\ \{q\}}{\{p\}\ P(t, v)\ \{q\}}.$$

This proof rule allows us to deal with arbitrary procedure calls. The construct $B[t, v]$ captures in a syntactic way the transmission of actual parameters to the procedure body. The following artificial example shows how various subtleties concerning the treated parameter mechanisms are handled here.

Consider the declaration $P \Leftarrow \langle \mathbf{val}\ x;\ \mathbf{var}\ y \mid i := i + 1; y := x + 1; x := 0\rangle$. We now show that

$$\{x = 1 \wedge i = 0\}\ P(x, a[i])\ \{i = 1 \wedge a[0] = 2 \wedge x = 1\}.$$

We have

$B[x, a[i]]$

$$\equiv \mathbf{begin\ new}\ u_1, u_2; u_1 := x; u_2 := i; i := i + 1; a[u_2] := u_1 + 1; u_1 := 0\ \mathbf{end}.$$

Using the assignment axioms, we now get

$$\{x = 1 \wedge i = 0\}\ u_1' := x; u_2' := i; i := i + 1; a[u_2'] := u_1' + 1; u_1' := 0$$
$$\{i = 1 \wedge a[0] = 2 \wedge x = 1\}.$$

Applying the introduced proof rule, we get the desired formula.

We now supplement the proof system H + Rule 16 + Axiom 18 by the last rule. The soundness of the resulting system can be established in the same way as the soundness of H + Rules 7, 16 discussed in Section 4.3. Note that the new rule is obviously sound. However, we have to prove anew the soundness of the former rules, since they are now used for a bigger class of programs, namely, those containing subscripted variables. It should be clear that the above system is also complete in the sense of Cook. (We assume here that the notions of expressibility and completeness are extended in a proper way to cover the case of programs and assertions using subscripted variables.)

6.2.2 *Recursive Procedures.* Assume now that the procedure P is recursive. Our definition of the meaning of programs containing procedure calls is analogous to the definition in Section 6.1 but now using the static scope requirement.

The corresponding recursion rule now takes the form

$$\frac{\{p_i\}\ P(t_i, v_i)\ \{q_i\}_{i=1,\ldots,n} \vdash \{p_i\}\ B[t_i, v_i]\ \{q_i\}_{i=1,\ldots,n}}{\{p_1\}\ P(t_1, v_1)\ \{q_1\}}.$$

The hypothesis of this rule states that the formulas $\{p_i\}\ B[t_i, v_i]\ \{q_i\}$ for $i = 1, \ldots, n$ can be proved from the assumptions $\{p_i\}\ P(t_i, v_i)\ \{q_i\}_{i=1,\ldots,n}$ using other proof rules. These assumptions are needed to deal with the (inner) calls from the procedure body, or, more precisely, from $B[t_1, v_1]$. The conclusion of the rule states that $\{p_1\}\ P(t_1, v_1)\ \{q_1\}$ can be proved (without any assumptions), but of course all formulas $\{p_i\}\ P(t_i, v_i)\ \{q_i\}_{i=1,\ldots,n}$ can be taken here as conclusions.

To get a complete proof system, we now proceed similarly to the way we did in Section 6.1. We take an extension of the proof system H + Axiom 18 + Rule 16 which, apart from the above recursion rule, contains the corresponding versions of the invariance axiom, the variable substitution rule, substitution rule II, the conjunction rule, and the parameter substitution rule. The only new rule is the following rewrite rule:

$$\frac{\{p\}\ S'\ \{q\}}{\{p\}\ S\ \{q\}}.$$

S' denotes here a program such that $S' \approx S$ and no bound variable of S' occurs free in S_0. In turn, $S_1 \approx S$ means that S_1 is obtained from S by replacing some blocks **begin new** z; S_2 **end** occurring as subprograms in S_1 by **begin new** u; $S_2[u/z]$ **end** where $u \notin \text{free}(S, S_0)$.

[14] contains a proof of soundness of the above proof system. The proof can be simplified if we proceed exactly as before, making use of the soundness of an extension of the system H + Rule 18 + Rule 16 dealing with a system of nonrecursive procedures.

In [14] it is also proved that the above proof system is complete in the sense of Cook. We present here a sketch of the proof for the case when the procedure body S_0 contains only one procedure call.

Let

$$p(t, v) \equiv \bar{z}' = \bar{z} \wedge \forall u\ (\bar{a}'[u] = \bar{a}[u])$$

where $\bar{z}'$ and $\bar{a}'$ are correspondingly the sequences of all simple and array variables which occur free in S_0, t, or v and $\bar{z}$ and $\bar{a}$ are corresponding sequences of fresh simple and array variables. Let $q(t, v)$ be an assertion defining $\text{post}_I(p(t, v), P(t, v))$.

The only interesting case in the above completeness proof is that of procedure calls. In a manner similar to the way the completeness of G was proved, one can prove that, if $\models_I \{p\}\ P(t, v)\ \{q\}$, then

$$\text{TR}_I \cup \{p(t, v)\}\ P(t, v)\ \{q(t, v)\} \vdash \{p\}\ P(t, v)\ \{q\};$$

so it is enough to prove

$$\text{TR}_I \vdash \{p(t, v)\}\ P(t, v)\ \{q(t, v)\}.$$

The proof runs as follows.

LEMMA 5. *Let S be a program and let $P(t_i, v_i)_{i=1,\ldots,n}$ be all procedure calls occurring in S'. If $\models_I \{p\}\ S'\ \{q\}$, then*

$$TR_I \cup \{p(t_i, v_i)\}\ P(t_i, v_i)\ \{q(t_i, v_i)\}_{i=1,\ldots,n} \vdash \{p\}\ S'\ \{q\}.$$

PROOF. The proof proceeds by induction on the structure of programs. The above remark indicates how to proceed in the case in which S is a procedure call. The only other nontrivial case in the proof is that of blocks. Assume S' is **begin new** x; S_1 **end**.

If $\models_I$ **begin new** x; S_1 **end**, then $\models_I S_1[y/x]$ for some fresh variable y. Procedure calls in $S_1[y/x]$ are of the form $P(t_i', v_i')$ for $i = 1, \ldots, n$ where $t_i' \equiv t_i[y/x]$ and $v_i' \equiv v_i[y/x]$. By the induction hypothesis,

$$\mathrm{TR}_I \cup \{p(t_i', v_i')\}\ P(t_i', v_i')\ \{q(t_i', v_i')\}_{i=1,\ldots,n} \vdash \{p\}\ S_1[y/x]\ \{q\}.$$

Hence, by the variable declaration rule,

$$\mathrm{TR}_I \cup \{p(t_i', v_i')\}\ P(t_i', v_i')\ \{q(t_i', v_i')\}_{i=1,\ldots,n} \vdash \{p\}\ S'\ \{q\}.$$

By the definition of S', x is not free in S_0. Therefore, $\{p(t_i, v_i)[y/x]\}\ P(t_i', v_i')$ $\{q(t_i, v_i)[y/x]\}$ can be derived from $\{p(t_i, v_i)\}\ P(t_i, v_i)\ \{q(t_i, v_i)\}$ using the corresponding parameter substitution rule. Also, since $x \notin \mathrm{free}(S_0)$ and y is a fresh variable,

$$\models_I p(t_i, v_i)[y/x] \leftrightarrow p(t_i', v_i')$$

and

$$\models_I q(t_i, v_i)[y/x] \leftrightarrow q(t_i', v_i').$$

The last three facts imply the claim. □

It is this case in the proof of Lemma 5 which forces us to work with S' instead of directly with S. Note that, if x were free in S_0, then we could not apply the corresponding parameter substitution rule. Also, observe that, if dynamic scope were assumed here, then we could use Rule 17 instead of Rule 16. Consequently, we could work directly with S, and the rewrite rule would be unneeded in the proof system.

COROLLARY. *For any procedure call* $P(t_1, v_1)$,

$$TR_I \cup \{p(t_i, v_i)\}\ P(t_i, v_i)\ \{q(t_i, v_i)\}_{i=1,2,3,4} \vdash \{p(t_i, v_i)\}\ B[t_i, v_i]'\ \{q(t_i, v_i)\}_{i=1,2,3}$$

where, for $i = 2, 3, 4$, $P(t_i, v_i)$ *is the procedure call occurring in* $B[t_{i-1}, v_{i-1}]'$.

LEMMA 6. $\{p(t_4, v_4)\}\ P(t_4, v_4)\ \{q(t_4, v_4)\}$ *can be derived from* $\{p(t_3, v_3)\}$ $P(t_3, v_3)\ \{q(t_3, v_3)\}$ *using the corresponding parameter substitution rule.*

The proof distinguishes eight different cases depending on the form of v_1 and v_0, where $P(t_0, v_0)$ is the inner call of S_0.

Now, by the above corollary, Lemma 6, and the rewrite rule, we get

$$\mathrm{TR}_I \cup \{p(t_i, v_i)\}\ P(t_i, v_i)\ \{q(t_i, v_i)\}_{i=1,2,3} \vdash \{p(t_i, v_i)\}\ B[t_i, v_i]\ \{q(t_i, v_i)\}_{i=1,2,3};$$

so, by the recursion rule,

$$\mathrm{TR}_I \vdash \{p(t_1, v_1)\}\ P(t_1, v_1)\ \{q(t_1, v_1)\}.$$

It should be noted that the proof of Lemma 6 leads to a veritable combinatorial explosion of cases to be dealt with when S_0 contains more than one procedure call and/or there are more than two formal parameters.

Some other parameter mechanisms can be described in a way similar to the above discussion. By way of illustration, consider a declaration of a recursive procedure P:

$$P \Leftarrow B_0 \quad \text{where} \quad B_0 \equiv \langle \mathbf{val}\ x;\ \mathbf{res}\ y \mid S_0 \rangle$$

and where y is a formal result parameter as used in ALGOL W (see [55]).

We define

$$B_0[t, z] \equiv \textbf{begin new } u;\ u := t;\ S_0[u/x];\ z := y \textbf{ end},$$

$$B_0[t, a[s]] \equiv \textbf{begin new } u_1, u_2;\ u_1 := t;\ u_2 := s;\ S_0[u_1/x];\ a[u_2] := y \textbf{ end}.$$

The corresponding proof system is sound and complete in the sense of Cook. The proofs are virtually the same as in the case of call-by-variable. The only difference is that the case of procedure calls in the completeness proof is now easier to handle and does not lead to a combinatorial explosion of the cases in the proof of Lemma 6. The reason is that a call-by-result parameter, in contrast to a call-by-variable parameter, does not lead to a substitution in the procedure body. As a result, $\{p(t_3^{(i)}, v_3^{(i)})\}\ P(t_3^{(i)}, v_3^{(i)})\ \{q(t_3^{(i)}, v_3^{(i)})\}$ can be derived from $\{p(t_2^{(i)}, v_2^{(i)})\}\ P(t_2^{(i)}, v_2^{(i)})\ \{q(t_2^{(i)}, v_2^{(i)})\}$ using the corresponding substitution rule.

Here $P(t_2^{(i)}, v_2^{(i)})$ is the ith procedure call occurring in $B[t_1, v_1]'$, and $P(t_3^{(i)}, v_3^{(i)})$ is the ith procedure call occurring in $B[t_2^{(j)}, v_2^{(j)}]'$ for some j.

6.2.3 *A Discussion.* One of the basic disadvantages of the proof systems dealt with in this section is the fact that each procedure call requires a separate proof of the premise concerning the body of the procedure in question. It should be possible to remove this deficiency by following the approach presented in the previous section and imposing appropriate restrictions on the actual parameters.

A useful observation in this respect is that procedure declarations

$$P \Leftarrow \langle \textbf{val } x;\ \textbf{var } y \mid S_0 \rangle$$

and

$$P \Leftarrow \langle \textbf{name}(y{:}x) \mid S_0' \rangle \qquad \text{where} \quad S_0' \equiv \textbf{begin new } u;\ u := x;\ S_0[u/x] \textbf{ end}$$

where u is a fresh variable lead to equivalent procedure calls when no subscripted variables are allowed. Therefore, the proof systems from the previous section dealing with the second declaration can be readily adopted to deal with programs in the context of the first declaration. Thus, in effect the study of call-by-value and call-by-variable can be reduced to the study of call-by-name. What remains to be done here is to incorporate subscripted variables and the static scope assumption into this framework.

Another point concerns the use of the renaming mechanism denoted here by the "′" sign. First, note that we could use a slightly different version of the recursion rule, obtained by replacing $B[t_i, v_i]$ in the recursion rule by $B[t_i, v_i]'$. After this change, the rewrite rule needs to be applied only once: as the last step of the proof. Thus, for any program S the whole proof deals in fact with programs of the form S_1'. But for such programs it does not matter which of the two variable declaration rules is applied. We conclude that, when using the refined version of the recursion rule, we can adopt variable declaration rule II provided that the rewrite rule is applied exactly once, namely, as the last step of the proof.

If we now drop from the recursion and rewrite rule the "′" sign, we get a sound and complete proof system dealing with the dynamic scope assumption. Thus we can treat static and dynamic scope in a uniform way here. At the level of semantics, a similar uniformity can be found by distinguishing between two forms of substitution: $[\ldots/P]$ and $\langle\ldots/\mathrm{P}\rangle$, defined in Sections 6.2 and 6.1, respectively.

6.3 Bibliographical Remarks

The division $\bar{x}:\bar{v}$ of formal name parameters and the restriction on the procedure calls in Section 6.1 are from [26]. Rules 20 and 21 are from [13]; they are refinements of the corresponding proof rules from [26] where global variables in procedure bodies (i.e., free variables different from formal parameters) are disallowed. Both Cook [13] and Gorelick [19] proved slightly stronger completeness results. The definition in Section 6.1.2 of the meaning of calls of recursive procedures is from [29]. The restrictions on procedure calls used in [19] are lifted in [7], where the static scope is also assumed. A recent paper of Gries and Levin [21] deals with related issues but only for the case of nonrecursive procedures. The notation "$B[t, v]$" and the corresponding recursion rule in Section 6.2.2 are from [3]. The definition of semantics of programs containing procedure calls suggested in Section 6.2 is partially motivated by [45]. Clarke [10] relates various completeness results concerning recursive procedures with parameters called by name to the existence of fixed points of some operators.

7. PROCEDURES AS PARAMETERS

7.1 Clarke's Incompleteness Result

A satisfactory treatment of procedures having procedures as parameters is impossible in full generality within the framework of Hoare's logic. This rather astonishing result was proved by Clarke [9] and is the contents of the following theorem.

THEOREM 4. *There exists no Hoare's proof system which is sound and complete in the sense of Cook for a programming language which allows*

1. *procedures as parameters in procedure calls,*
2. *recursion,*
3. *static scope,*
4. *global variables in procedure bodies, and*
5. *local procedure declarations.*

The proof follows the line of incompleteness results proved in Section 2.7. First, the following crucial lemma is proved.

LEMMA 7. *The halting problem is undecidable for programs in a programming language with features* (1) *to* (5) *above for all finite interpretations I with* $|I| \geq 2$.

Now take a finite interpretation I with $|I| \geq 2$. It is easy to see that Tr_I is a recursive set. Thus the set of asserted programs φ such that $\mathrm{Tr}_I \vdash_W \varphi$ in a Hoare's system W is recursively enumerable. Also, as observed in Section 2.9, the assertion language is expressive relative to I and the class of programs considered. On the other hand, Lemma 7 and the Fact from Section 2.7 imply that the set of all asserted programs from the above programming language which are true under I is not recursively enumerable. Therefore, no Hoare's system W for this programming language can be complete in the sense of Cook.

7.2 Copy Rules

Under what restrictions, then, is it possible to get sound and complete Hoare's systems dealing with procedures as parameters? Clarke [9] stated that, if any of the above features 1 to 5 is disallowed, then there exists a natural Hoare's system for the corresponding programming language which is sound and complete in the sense of Cook. Unfortunately, the corresponding proofs for the cases in which 1, 2, 4, or 5 is disallowed are not worked out there. Also, some additional restrictions to be discussed in Section 7.7 (concerning sharing and self-application) are imposed on the language.

A detailed analysis of these and related issues is provided in [45], where most of the missing proofs are supplied in a uniform way. In the subsequent discussion we allow procedure declarations of the form

$$P \Leftarrow \langle \mathbf{proc}\ \bar{R};\ \mathbf{var}\ \bar{y} \mid S_0 \rangle$$

where P is the name of the declared procedure, $\bar{R}$ is the list of distinct formal procedure parameters, and $\bar{y}$ is the list of distinct formal parameters called by variable.

Subscripted variables are not allowed here; consequently, only simple variables can be used as actual parameters called by variable. With such restrictions imposed on the language, call-by-variable is of course equivalent to call-by-name. In blocks, systems of declarations of local procedures are allowed in addition to declarations of local variables. Call this class of programs $\mathscr{S}^{\mathrm{P}}$.

A uniformity similar to the one exemplified in Section 6.2.3 forms an important aspect of Olderog's considerations. However, the situation is more complicated here because procedures are allowed as parameters.

Uniformity is reached by employing the notion of *copy rule*. A copy rule is a relation between two programs differing only by an injective bound renaming of some local identifiers. By an *identifier* we mean here a simple variable or a procedure name. By idf(S) we denote the set of all identifiers occurring in S. *Injective bound renaming* (written as $S \underset{\text{inj}}{\approx} S'$) is defined as follows: $S \underset{\text{inj}}{\approx} S'$ iff $S \approx S'$ (bound renaming as defined in the previous section but now referring to all identifiers) holds and additionally the renaming is injective.

Now let Id be a set of identifiers. Olderog [45] considers three copy rules:

1. The ALGOL 60 copy rule $\mathscr{C}_{60}$: $(S, \mathrm{Id})\, \mathscr{C}_{60}\, S_1$ iff $S_1 \underset{\text{inj}}{\approx} S$ and no identifier bound in S_1 occurs in Id.
2. The "most recent" copy rule $\mathscr{C}_{\mathrm{mr}}$: $(S, \mathrm{Id})\, \mathscr{C}_{\mathrm{mr}}\, S_1$ iff $S_1 \underset{\text{inj}}{\approx} S$, no simple variable bound in S_1 occurs in Id, and procedure names have *not* been renamed.
3. The naive copy rule $\mathscr{C}_{\mathrm{n}}$: $(S, \mathrm{Id})\, \mathscr{C}_{\mathrm{n}}\, S_1$ iff $S_1 \equiv S$.

Note that, according to this terminology, $(S, \mathrm{free}(S_0))\, \mathscr{C}_{60}\, S'$ and $(S, \mathrm{free}(S_0))\, \mathscr{C}_{\mathrm{mr}}\, S'$ hold for programs discussed in Section 6.2.2.

7.3 The Proof System $(O, \mathscr{C})$

These copy rules are incorporated into the proof system as a parameter; this is similar to the case of static and dynamic scope assumptions discussed in Section 6.2.3.

The proof system $(O, \mathscr{C})$ has a structure similar to that of the systems discussed in Section 6. But since we are now dealing with declarations of local procedures, we cannot take a fixed procedure declaration $P \Leftarrow \langle \ldots | S_0 \rangle$ such that every procedure name P occurring within an asserted program $\{p\}\ S\ \{q\}$ refers to this declaration. Instead, we augment each program S with a context E, being a sequence of procedure declarations with different procedure names. Thus, we now consider formulas $\{p\}\ \langle E | S \rangle\ \{q\}$ instead of simply $\{p\}\ S\ \{q\}$. Throughout this section it is always assumed that for every procedure name occurring freely in S there exists a corresponding declaration $P \Leftarrow \langle \ldots | S_0 \rangle$ in E. We say that a procedure call $S \equiv P(\bar{R}_1, \bar{y}_1)$ is *incorrect with respect to E* if the declaration of P in E requires different actual parameters. To cover the case of incorrect procedure calls, the following new axiom is introduced.

RULE 22: AXIOM OF INCORRECT PROCEDURE CALLS

$$\{p\}\ \langle E | P(\bar{R}_1, \bar{y}_1) \rangle\ \{q\}$$

where the procedure call $P(\bar{R}_1, \bar{y}_1)$ is incorrect with respect to E.

This axiom is valid because an incorrect procedure call gets a nowhere defined function as its meaning. The need for such an axiom arises from the fact that the execution of a syntactically correct program can lead to an incorrect procedure call in the case in which procedures are allowed as parameters.

A copy rule $\mathscr{C}$ is used in two proof rules. The first of them is the recursion rule, which now has the following form.

RULE 23: RECURSION RULE III

$$\frac{\{p_i\}\ \langle E_i | P_i(\bar{R}_i, \bar{y}_i) \rangle\ \{q_i\}_{i=1,\ldots,n} \vdash \{p_i\}\ \langle E_i | B_{i\mathscr{C}} \rangle\ \{q_i\}_{i=1,\ldots,n}}{\{p_1\}\ \langle E_1 | P_1(\bar{R}_1, \bar{y}_1) \rangle\ \{q_1\}}$$

where, for some S_i $(i = 1, \ldots, n)$,

1. $P_i \Leftarrow \langle \mathbf{proc}\ \bar{R}'_i;\ \mathbf{var}\ \bar{y}'_i | S_i \rangle \in E_i$ *with* $|\bar{R}_i| = |\bar{R}'_i|, |\bar{y}_i| = |\bar{y}'_i|$; *and*
2. $(S_i[\bar{R}_i/\bar{R}'_i][\bar{y}_i/\bar{y}'_i], Id_i)\ \mathscr{C}\ B_{i\mathscr{C}}$ *where* $Id_i = idf(E_i, P_i(\bar{R}_i, \bar{y}_i))$.

This rule deals with n different procedures, and of course all formulas $\{p_i\}$ $\langle E_i | P_i(\bar{R}_i, \bar{y}_i) \rangle\ \{q_i\}$ can be taken here as conclusions.

The second rule which refers to the copy rule is the following rewrite rule.

RULE 24: REWRITE RULE

$$\frac{\{p\}\ \langle \varnothing | S' \rangle\ \{q\}}{\{p\}\ S\ \{q\}}$$

where $S' \equiv S$ in the case of the naive copy rule $\mathscr{C}_n$ and $S' \approx S$ where S' is distinguished *(different defining occurrences of identifiers are denoted differently) in the case of the $\mathscr{C}_{60}$ and $\mathscr{C}_{mr}$ copy rules.*

Among all axioms and proof rules of the system, only the rewrite rule allows us to pass to a program S without any context of procedure declarations. Therefore, to prove a property of a program S we are forced to use the rewrite rule exactly once, and this as the last step in the proof.

The next step in the development of the system $(O, \mathscr{C})$ is the introduction of the following rule.

RULE 25: RULE OF BLOCKS

$$\frac{\{p[y/x] \wedge x = \omega\}\ \langle add(E, E_1) \mid S_1\rangle\ \{q[y/x]\}}{\{p\}\ \langle E \mid \textbf{begin new } x;\ E_1;\ S_1 \textbf{ end}\rangle\ \{q\}}$$

where $add(E, E_1)$ is the system of procedure declarations obtained from E by first deleting from it all declarations referring to a procedure name also declared in E_1 and then adding E_1 to it.

The discussion of the proof system from Section 6.2.2 given in Section 6.2.3 aimed to provide a better understanding of the decisions standing behind the choice of the above three rules.

To deal with the constructs present in the **while** programs, we adopt an appropriately modified system H in which each program S is replaced by $\langle E \mid S\rangle$. From Section 3.3 we know that proofs concerning procedure calls require some additional axioms and proof rules. An additional set of axioms and proof rules similar to the one used in Sections 6.1.2 and 6.2.2 is adopted here. This is a bit surprising in view of the fact that procedure parameters are now allowed. In particular, the invariance axiom, the conjunction rule, and substitution rules I and II are used (all referring to constructs of the form $\langle E \mid S \rangle$).

The final rule is a substitution rule corresponding to the parameter substitution considered in Section 6.1.

RULE 26: SUBSTITUTION RULE III

$$\frac{\{p\}\ \langle E \mid S\rangle\ \{q\}}{\{p[\bar{y}/\bar{x}]\}\ \langle add(E_1, E)[\bar{y}/\bar{x}] \mid S[\bar{y}/\bar{x}]\rangle\ \{q[\bar{y}/\bar{x}]\}}$$

where the substitution $\bar{y}/\bar{x}$ is injective when restricted to the subset $free(p, q) \cup idf(E, S)$ of $\bar{x}$. Here $\bar{x}$ can contain procedure names; x_i is a simple variable iff y_i is a simple variable.

This rule is stronger than the parameter substitution rule from Section 6.1 in the case in which both are restricted to procedure calls with actual parameters being simple variables. The reason for this strengthening is that no restrictions on actual parameters in procedure calls are imposed here, in contrast to Section 6.1.

It is instructive to check that the arguments from Section 6.1.1 showing the necessity of restrictions in Rules 20 and 21 do not indicate that the above rule is unsound. Both substitutions considered there, namely, $[u, u/x, u]$ and $[x, x/x, u]$, are not injective; so the argument does not apply here.

Note also that the above rule admits extending the procedure environment in the conclusion. Intuitively, this is allowed because the newly added procedures will never be called. The subsequent proofs are not affected if the program S is restricted in Rule 26 to be a procedure call.

7.4 Semantic Issues

We now pass to semantic issues. Each copy rule $\mathscr{C}$ generates a corresponding semantics $\mathcal{M}_{I,\mathscr{C}}$ for the programs from $\mathscr{S}^{\text{P}}$. The case of procedure calls is now more complicated because declarations of local systems of procedures are allowed. In particular, it is difficult to retain the approach of Section 6.1. Therefore, we proceed in a somewhat different way. We first define by induction on i a sequence $\mathcal{M}^i_{I,\mathscr{C}}$ of approximating semantics. The crucial clause concerns procedure calls. We put

$$\mathcal{M}^i_{I,\mathscr{C}}(E \mid P(\bar{R}, \bar{y}))$$

$$= \begin{cases} \mathcal{M}^{i-1}_{I,\mathscr{C}}(E \mid B_{\mathscr{C}}) & \text{if } \; i \geq 1, \quad P \Leftarrow \langle \textbf{proc } \bar{R}'; \textbf{var } \bar{y}' \mid S \rangle \in E \\ & \qquad\qquad\quad \text{with } |\bar{R}| = |\bar{R}'|, |\bar{y}| = |\bar{y}'|; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Here, analogously to Rule 23, $(S[\bar{R}/\bar{R}'][\bar{y}/\bar{y}'], \text{Id})\mathscr{C}\, B_{\mathscr{C}}$ where Id $=$ idf$(E, P(\bar{R}, \bar{y}))$.

The semantics of blocks is defined by the clause

$$\mathcal{M}^i_{I,\mathscr{C}}(E \mid \textbf{begin new } x; E_1; S_1 \textbf{ end})(\sigma) = \text{DROP}(\mathcal{M}^i_{I,\mathscr{C}}(\text{add}(E, E_1) \mid S_1)(\sigma'), x)$$

where $\sigma' = \sigma \cup (x, \underline{\omega})$.

Here the definition and use of states of Section 4.5 are adopted. The other clauses are defined as usual. We now put

$$\mathcal{M}_{I,\mathscr{C}}(E \mid S) = \bigcup_{i=0}^{\infty} \mathcal{M}^i_{I,\mathscr{C}}(E \mid S)$$

and

$$\mathcal{M}_{I,\mathscr{C}}(S) = \mathcal{M}_{I,\mathscr{C}}(\varnothing \mid S')$$

where S' is defined as in Rule 24.

As opposed to the approach taken in Section 4.1, here all renamings of local identifiers (necessary to satisfy scope requirements) are done first when we replace S by $(\varnothing \mid S')$ and subsequently at every step where the copy rule $\mathscr{C}$ is applied, that is, where $P(\bar{R}, \bar{y})$ is replaced by $B_{\mathscr{C}}$.

We now define

$$\models_{I,\mathscr{C}} \{p\}\ \langle E \mid S \rangle\ \{q\} \quad \text{iff} \quad \text{for all states } \sigma, \tau, \models_I p(\sigma)$$
$$\text{and } \mathcal{M}_{I,\mathscr{C}}(E \mid S)(\sigma) = \tau \text{ implies} \models_I q(\tau).$$

The soundness proof of $(O, \mathscr{C})$ with respect to the $\mathcal{M}_{I,\mathscr{C}}$ semantics proceeds through the same steps as the ones originally defined in Section 3.7 and later repeated in Sections 4.3, 6.1.2, and 6.2.2. Note that the last clause in the definition of $\mathcal{M}_{I,\mathscr{C}}$ assures the soundness of Rule 24. The only complicated case in the proof is that of the soundness of Rule 26.

We now relate copy rules to scope assumptions. In the presence of declarations of local procedures, the ALGOL 60 copy rule leads to a semantics with the static scope assumption. For programs satisfying the "most recent" property (see [41]) the same result is already achieved by using the "most recent" copy rule. Finally, the naive copy rule leads to a semantics with the dynamic scope assumption. Due to the lack of space, we do not elaborate more precisely on these issues.

7.5 The Characterization Theorem

From the theorem which opened this section we know that we cannot expect $(O, \mathscr{C})$ to be complete in the sense of Cook when $\mathscr{C}$ is the ALGOL 60 copy rule. Olderog [45] found a rather simple criterion which, when imposed on a true asserted program, guarantees its provability in $(O, \mathscr{C})$. To define this condition, we first introduce two notions.

Given a program S, we write $P \rightarrow_S Q$ iff P and Q are non-formal procedure names from S such that Q occurs freely in the procedure body of P. By a *reference chain* of S we mean a sequence of the form

$$P_1 \rightarrow_S P_2 \rightarrow_S \cdots \rightarrow_S P_n \qquad \text{where} \quad P_i \not\equiv P_j \quad \text{for} \quad i \neq j.$$

Given two programs S_1 and S_2, we write $S_1 \rightarrow_{\mathscr{C}} S_2$ if S_2 can be obtained by a single application of the copy rule $\mathscr{C}$, that is, if S_2 results from a literal replacement of some call $P(\bar{R}, \bar{y})$ in S_1 by a modified procedure body $B_{\mathscr{C}}$ defined as in the formulation of Rule 23. Let $\rightarrow_{\mathscr{C}}^*$ stand for the transitive closure of $\rightarrow_{\mathscr{C}}$. We now say that a program S is *$\mathscr{C}$-bounded* if, for some constant k, whenever $S \rightarrow_{\mathscr{C}}^* S_1$, then the lengths of reference chains of S_1 are bounded by k. Intuitively speaking, a program is $\mathscr{C}$-bounded if it cannot be expanded using the copy rule $\mathscr{C}$ to programs with arbitrarily long reference chains.

Equipped with this notion, we can formulate the following characterization theorem due to Olderog [45].

THEOREM 5. *Let I be an interpretation such that the assertion language is expressive relative to I and $\mathscr{S}^p$. Then the following statements are equivalent*:

1. $Tr_I \vdash_{(O,\mathscr{C})} \{p\}\ S\ \{q\}$.
2. $\models_{I,\mathscr{C}} \{p\}\ S\ \{q\}$ *and S is $\mathscr{C}$-bounded.*

Note that the implication 1 → 2 is a strengthening of the soundness theorem concerning $(O, \mathscr{C})$. The implication 2 → 1 is a completeness theorem. The proof deals with constructs of the form $\langle E \mid S \rangle$ and proceeds by induction on their structure. The step to programs S is obtained by using the last clause in the definition of semantics, $\mathscr{M}_{I,\mathscr{C}}$, and the rewrite rule.

All cases in the proof are dealt with in a way analogous to the handling of the cases in the previous proofs. As usual, the nontrivial case is that of procedure calls. The proof is a generalization of the techniques used so far. First, we choose the most general formula for a procedure call $P(\bar{R}, \bar{y})$.

Let $\bar{z}$ be a sequence of all simple variables occurring free in $\langle E \mid P(\bar{R}, \bar{y}) \rangle$ where $P(\bar{r}, \bar{y})$ is a correct procedure call with respect to E. Let $\bar{z}'$ be a sequence of fresh variables of the same length as $\bar{z}$. Put $p(\bar{R}, \bar{y}) \equiv \bar{z} = \bar{z}'$. Choose $q(\bar{R}, \bar{y})$ to be an assertion which defines $\text{post}_I(p(\bar{R}, \bar{y}), \langle E \mid P(\bar{R}, \bar{y}) \rangle)$.

In a manner similar to the way we proved the completeness of G we can prove that, if $\models_{I,\mathscr{C}} \{p\}\ \langle E \mid P(\bar{R}, \bar{y})\rangle\ \{q\}$, then

$$\mathrm{Tr}_I \cup \{p(\bar{R}, \bar{y})\ \langle E \mid P(\bar{R}, \bar{y})\rangle\ \{q(\bar{R}, \bar{y})\} \vdash_{(O,\mathscr{C})} \{p\}\ \langle E \mid P(\bar{R}, \bar{y})\rangle\ \{q\}.$$

Thus, as in Section 6.2.2, the problem reduces to proving the most general formula, that is, to proving that

$$\mathrm{Tr}_I \vdash_{(O,\mathscr{C})} \{p(\bar{R}, \bar{y})\}\ \langle E \mid P(\bar{R}, \bar{y})\rangle\ \{q(\bar{R}, \bar{y})\}. \tag{63}$$

Due to the lack of space, we can only present a rough sketch of the proof.

Given a procedure call $P(\bar{R}, \bar{y})$ correct with respect to E, let $\mathscr{R}_{\mathscr{C}}(\langle E \mid P(\bar{R}, \bar{y})\rangle)$ denote the set of all constructs of the form $\langle E' \mid P'(\bar{R}, \bar{y}')\rangle$ such that

1. $P'(\bar{R}', \bar{y}')$ is a correct procedure call with respect to E' and
2. it can be obtained by a formal expansion of $\langle E \mid P(\bar{R}, \bar{y})\rangle$ using symbolic execution and the copy rule $\mathscr{C}$.

For example, if E is $P \Leftarrow \langle \varnothing \mid$ **begin new** x; E_1; P **end**$\rangle$, then $\mathscr{R}_{\mathscr{C}}(\langle E \mid P\rangle)$ is $\{\langle E \mid P\rangle, \langle \mathrm{add}(E, E_1) \mid P\rangle\}$ because $\mathrm{add}((\mathrm{add}(E, E_1)), E_1) = \mathrm{add}(E, E_1)$. Symbolic execution is incorporated here by symbolically elaborating the block.

Applying this terminology to the procedure declaration E considered in the completeness proof in Section 6.2.2, we have $\{\langle E \mid P(t_i, v_i)\rangle : i = 1, \ldots, 4\} \subseteq \mathscr{R}_{\mathscr{C}_{60}}(\langle E \mid P(t_1, v_1)\rangle)$. Intuitively speaking, the set $\mathscr{R}_{\mathscr{C}}$ is the set of all correct procedure calls which could possibly occur during the execution of the program **begin** E; $P(R, y)$ **end**. Each such call has an appropriate procedure environment E' in which it is called. This set is usually infinite.

It turns out, however, that, if the program **begin** E; $P(\bar{R}, \bar{y})$ **end** is $\mathscr{C}$-bounded, then this set possesses a finite subset from which all other elements can be derived by a substitution conforming to the restrictions of Rule 26. Lemma 6 shows that $\{\langle E \mid P(t_i, v_i)\rangle_{i=1,\ldots,3}\}$ is such a subset of $\mathscr{R}_{\mathscr{C}_{60}}(\langle E \mid P(t_1, v_1)\rangle)$.

Once such a subset has been found, reasoning analogous to that in Section 6.2.2 can be applied. Namely, take the set A_1 of most general formulas for the elements of this subset. Let B be the set of corresponding correctness formulas concerning the bodies $B_{i\mathscr{C}}$ related to procedure calls from A_1. In turn, let A_2 be the set of most general formulas for the procedure calls taken from the bodies $B_{i\mathscr{C}}$. A lemma corresponding to the Corollary in Section 6.2.2 states that $\mathrm{Tr}_I \cup A_1 \cup A_2 \vdash_{(O,\mathscr{C})} B$. Now, by the choice of A_1, all formulas from A_2 can be derived from A_1 by Rule 26. Thus $\mathrm{Tr}_I \cup A_1 \vdash_{(O,\mathscr{C})} B$. By Rule 23, $\mathrm{Tr}_I \vdash_{(O,\mathscr{C})} A_1$. In particular, (63) holds as desired.

7.6 Applications of the Characterization Theorem

The characterization theorem can now be applied to various classes of programs from $\mathscr{S}^{\mathrm{p}}$ for which the assumption of $\mathscr{C}$-boundedness can be established. We list several such classes without going into further details.

1. $\mathscr{C} = \mathscr{C}_{60}$ (the ALGOL 60 copy rule, i.e., static scope):
   a. All programs disallowing one of the features 1, 2, or 5 from Theorem 4.
   b. All programs disallowing feature 4 from Theorem 4 referring to procedure names.

2. $\mathscr{C} = \mathscr{C}_{\mathrm{mr}}$ (the "most recent" copy rule): All programs satisfying the "most recent" property.
3. $\mathscr{C}= \mathscr{C}_{\mathrm{n}}$ (the naive copy rule, i.e., dynamic scope): All programs from $\mathscr{S}^{\mathrm{p}}$.

It should be noted that not all programs from $\mathscr{S}^{\mathrm{p}}$ are $\mathscr{C}_{60}$-bounded. Olderog [45] exhibits such a program. Of course, the existence of such programs follows from Theorems 4 and 5.

7.7 Decidability Issues

When studying such subclasses, it is sensible to ask whether they are properly defined, that is, whether they form decidable subsets of $\mathscr{S}^{\mathrm{p}}$. The classes listed in 1 and 3 above obviously satisfy this requirement. Also, by a theorem of Kandzia [31], the class listed in 2 is a decidable subset of $\mathscr{S}^{\mathrm{p}}$.

These decidability results should be contrasted with the restrictions imposed on parameters in Section 6.1. Recall that, according to these restrictions, all actual simple variables are to be distinct and different from global variables of the considered procedure bodies (here, those from E).

We say that a program is *sharing-free* if all procedure calls arising during its execution satisfy the above restriction. In [9] all programs are assumed to be sharing-free. In general, the restriction "sharing-free" is dangerous in light of decidability requirements: if we interpret "sharing-free" as allowing only sharing-free programs in our subclass $\mathscr{S}$ of $\mathscr{S}^{\mathrm{p}}$, then $\mathscr{S}$ is in general undecidable. This follows from a result of Langmaack [35] stating that for $\mathscr{S}^{\mathrm{p}}$ the *formal reachability* of procedures is undecidable when the ALGOL 60 copy rule is applied. Fortunately, these problems do not arise in Section 6.1 because, by the result of [33], sharing is a decidable property for programs without procedures as parameters.

Another possible interpretation of "sharing-free" is to allow arbitrary programs but restrict the application of Rule 26 to sharing-free procedure calls. But this in turn makes the substitution rule itself undecidable; so the set of provable asserted programs is not r.e. Hence, the only proper way to solve these difficulties is to formulate a proof rule which can deal with sharing.

Also, in [9] no *self-application* is allowed (e.g., procedure calls of the form $P(\ldots P \ldots)$ are disallowed).

7.8 Lipton's Theorem and Its Implications

The proof of Theorem 4 related decidability of the halting problem for all finite interpretations to the existence of a complete Hoare's system for the language in question. Lipton [38] showed that these two properties are in fact equivalent for a wide class of programming languages, thereby proving a conjecture of Clarke. Unfortunately, details of the proof are not fully worked out.

Recently Langmaack [32] provided a rigorous proof of the theorem for the case of ALGOL-like programming languages. This proof is based on the usual static scope semantics of ALGOL-like programs (defined by the ALGOL 60 copy rule), whereas Lipton uses a more general notion of programming language merely requiring the semantics to be defined by a certain type of interpreter. The version proved by Langmaack can be stated as follows.

THEOREM 6. *For any "acceptable" ALGOL-like programming language PL the following are equivalent*:

1. *PL has uniformly decidable halting problems for finite interpretations.*
2. *PL has a sound and complete Hoare's logic provided the assertion language allows quantifier-free formulas only.*

PL is called *acceptable* if PL is closed under certain program transformations such as replacing basic statements in a program $S \in \mathrm{PL}$ by an arbitrary program $S' \in \mathrm{PL}$. Roughly speaking, the existence of Hoare's logic means here that the set $\{\varphi : \models_I \varphi\}$ is uniformly recursively enumerable in Tr_I for interpretations I satisfying the expressiveness condition.

Note that the existence of a Hoare's system which is sound and complete in the sense of Cook implies the existence of sound and complete Hoare's logic.

Theorem 6 implies that for all toy programming languages considered in Sections 2–6 of this paper there exists a sound and complete Hoare's logic. However, it must be noted that the above theorem does not provide any useful axiomatization of the corresponding Hoare's logics. Also, quantifiers are disallowed in the assertions. In contrast, all proof systems considered in this paper are natural and can be used straightforwardly to prove the correctness of programs.

An interesting question is whether there exists an application of Theorem 6 which shows the existence of a sound and complete Hoare's logic for a programming language with no known sound and complete Hoare's proof system. The answer is positive. Consider the class of all programs from $\mathscr{S}^{\mathrm{p}}$ which disallow self-application and global simple variables in procedure bodies. Langmaack [34] proved that this class of programs satisfies condition 1 of Theorem 6 in the case of the ALGOL 60 copy rule. By Theorem 6 there exists a sound and complete Hoare's logic for this class of programs.

The problem of finding a natural Hoare's proof system for this class of programs is offered in [36] as a challenge to researchers in this area. It should be noted that the characterization theorem does not apply here, since not all programs in this class are $\mathscr{C}_{60}$-bounded. An example of such a program that is not $\mathscr{C}_{60}$-bounded is given in [36, 45]. There is as yet no proof system available in which the partial correctness of this program can be studied.

7.9 Bibliographical Remarks

Axiom 22 and Rule 25 are due to Clarke [9]. Rule 23 and the semantics $\mathscr{M}^i_{I,\mathscr{C}}$ are modifications of the corresponding versions used in [9]. The construct $\mathscr{R}_{\mathscr{C}}(\ldots)$ is used implicitly in [19] and explicitly in [9], where it is called a *range* of a statement. In [1] a completeness result concerning a language disallowing procedures with parameters and with features 2–5 from Theorem 4 is proved. The discussion of sharing in Section 7.7 is due to a private communication from Langmaack and Olderog. In [36] they give an overview of the results discussed in this section.

ACKNOWLEDGMENTS

We thank D. Harel, P. E. Lauer, E. W. Mayr, and J. Zucker for their comments on earlier versions of this paper. One of the referees made extensive comments and suggestions concerning the paper which significantly influenced the final version. Professor H. Langmaack and E. R. Olderog provided detailed suggestions concerning Section 7.

REFERENCES

1. APT, K.R. A sound complete Hoare-like system for a fragment of PASCAL. Rep. IW 97/78, Mathematisch Centrum, Amsterdam, 1978.
2. APT, K.R., BERGSTRA, J.A., AND MEERTENS, L.G.L.T. Recursive assertions are not enough—Or are they? *Theor. Comput. Sci. 8* (1979), 73–87.
3. APT, K.R., AND DE BAKKER, J.W. Semantics and proof theory of PASCAL procedures. In *Lecture Notes in Computer Science*, vol. 52: *Proc. 4th Colloq. Automata, Languages and Programming.* Springer-Verlag, New York, 1977, pp. 30–44.
4. APT, K.R., AND DE BAKKER, J.W. Exercises in denotational semantics. In *Lecture Notes in Computer Science*, vol. 45: *Proc. 5th Symp. Mathematical Foundations of Computer Science.* Springer-Verlag, New York, 1976, pp. 1-11.
5. BERGSTRA, J.A., AND TUCKER, J.V. Some natural structures which fail to possess a sound and decidable Hoare-like logic for their while-programs. *Theor. Comput. Sci.*, to appear; earlier version appeared as Rep. IW 136/80, Mathematisch Centrum, Amsterdam, 1980.
6. BERGSTRA, J.A., AND TUCKER, J.V. Expressiveness and the completeness of Hoare's logic. Rep. IW 149/80, Mathematisch Centrum, Amsterdam, 1980.
7. CARTWRIGHT, R., AND OPPEN, D. Unrestricted procedure calls in Hoare's Logic. In Conf. Rec., 5th Ann. ACM Symp. Principles of Programming Languages, Tucson, Ariz., Jan. 23–25, 1978, pp. 131–140.
8. CLARKE, E.M., JR. Proving correctness of coroutines without history variables. *Acta Inf. 13* (1980), pp. 169–188.
9. CLARKE, E.M., JR. Programming language constructs for which it is impossible to obtain good Hoare axiom systems. *J. ACM 26*, 1 (Jan. 1979), 129–147.
10. CLARKE, E.M., JR. Program invariants as fixed points. In Proc. 18th IEEE Symp. Foundations of Computer Science, 1977, pp. 18–29.

10a. CLARKE, E.M., JR. Completeness and incompleteness theorems for Hoare-like axiom systems. Ph.D. dissertation, Computer Science Dep., Cornell Univ., 1976.

11. CLINT, M. Program proving: Coroutines. *Acta Inf. 2* (1973), 50–63.
12. CLINT, M., AND HOARE, C.A.R. Program proving: Jumps and functions. *Acta Inf. 1* (1971), 214–224.
13. COOK, S.A. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput. 7*, 1 (1978), 70–90.
14. DE BAKKER, J.W. *Mathematical Theory of Program Correctness.* Prentice-Hall, Englewood Cliffs, N.J., 1980.
15. DE BAKKER, J.W. Correctness proofs for assignment statements. Rep. IW 55/76, Mathematisch Centrum, Amsterdam, 1976.
16. DIJKSTRA, E.W. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, N.J., 1976.
17. DONAHUE, J.E. *Lecture Notes in Computer Science*, vol. 42: *Complementary Definitions of Programming Language Constructs.* Springer-Verlag, New York, 1976.
18. FLOYD, R.W. Assigning meanings to programs. In *Proc. AMS Symp. Applied Mathematics*, vol. 19. American Mathematical Society, Providence, R.I., 1967, pp. 19–31.
19. GORELICK, G.A. A complete axiomatic system for proving assertions about recursive and non-recursive programs. Tech. Rep. 75, Dep. Computer Science, Univ. Toronto, 1975.
20. GRIES, D. The multiple assignment statement. *IEEE Trans. Softw. Eng. SE-4* (March 1978), 89–93.
21. GRIES, D., AND LEVIN, G. Assignment and procedure call proof rules. *ACM Trans. Program. Lang. Syst. 2*, 4 (Oct. 1980), 564–579.
22. HAREL, D. Proving the correctness of regular deterministic programs: A unifying survey using dynamic logic. *Theor. Comput. Sci. 12* (1980), 61–81.
23. HAREL, D. *Lecture Notes in Computer Science*, vol. 68: *First-Order Dynamic Logic.* Springer-Verlag, New York, 1979.

24. Harel, D., Pnueli, A., and Stavi, J. Completeness issues for inductive assertions and Hoare's method. Tech. Rep., Dep. Computer Science, Univ. Tel Aviv, Israel, 1976.
25. Hoare, C.A.R. Proof of correctness of data representations. *Acta Inf. 1* (1972), 271–281.
26. Hoare, C.A.R. Procedures and parameters: An axiomatic approach. In *Lecture Notes in Mathematics*, vol. 188: *Semantics of Algorithmic Languages*. Springer-Verlag, New York, 1971, pp. 102–116.
27. Hoare, C.A.R. An axiomatic basis for computer programming. *Commun. ACM 12*, 10 (Oct. 1969), 576–580, 583.
28. Hoare, C.A.R., and Wirth, N. An axiomatic definition of the programming language PASCAL. *Acta Inf. 2* (1973), 335–355.
29. Igarashi, S., London, R.L., and Luckham, D.C. Automatic program verification I: A logical basis and its implementation. *Acta Inf. 4* (1975), 145–182.
30. Janssen, T.M.V., and van Emde Boas, P. On the proper treatment of referencing, dereferencing and assignment. In *Lecture Notes in Computer Science*, vol. 52: *Proc. 4th Colloq. Automata, Languages and Programming*. Springer-Verlag, New York, 1977, pp. 282–300.
31. Kandzia, P. On the "most recent" property of ALGOL-like programs. In *Lecture Notes in Computer Science*, vol. 14: *Proc. 2d Colloq. Automata, Languages and Programming*. Springer-Verlag, New York, 1974, pp. 97–111.
32. Langmaack, H. A proof of a theorem of Lipton on Hoare Logic and applications. Ber. 8003, Inst. Inf. Prakt. Math., Univ. Kiel, W. Germany, 1980.
33. Langmaack, H. On a theory of decision problems in programming languages. In *Lecture Notes in Computer Science*, vol. 75: *Proc. Int. Conf. Mathematical Studies of Information Processing*, Springer-Verlag, New York, 1979, pp. 538–558.
34. Langmaack, H. On termination problems for finitely interpreted ALGOL-like programs. Ber. 7904, Inst. Inf. Prakt. Math., Univ. Kiel, W. Germany, 1980.
35. Langmaack, H. On correct procedure parameter transmission in higher programming languages. *Acta Inf. 2* (1973), 110–142.
36. Langmaack, H., and Olderog, E.R. Present-day Hoare-like systems for programming languages with procedures: Power, limits and most likely extensions. In *Lecture Notes in Computer Science*, vol. 85: *Proc. 7th Colloq. Automata, Languages and Programming*, Springer-Verlag, New York, pp. 363–373.
37. Lauer, P.E. Consistent formal theories of the semantics of programming languages. Tech. Rep. TR.25.121, IBM Lab. Vienna, Austria, 1971.
38. Lipton, R.J. A necessary and sufficient condition for the existence of Hoare Logics. In Proc. 18th IEEE Symp. Foundations of Computer Science, 1977, pp. 1–6.
39. Lomazova, I.A. O složnosti induktivnyh usloviĭ dlja verifikacii arifmetičeskih programm (On the complexity of inductive assertions for the verification of arithmetical programs). In Materialy Wsesojuznoĭ Naučnoĭ Studenčeskoĭ Konferencii, Matematika, Novosibirsk State Univ., Novosibirsk, U.S.S.R., 1978, pp. 85–94.
40. London, R.L., Guttag, J.V., Horning, J.J., Lampson, B.W., Mitchell, J.G., and Popek, G.J. Proof rules for the programming language Euclid. *Acta Inf. 10* (1978), 1–26.
41. McGowan, C.L. The "most recent" error: Its causes and correction. In Proc. ACM Conf. Proving Assertions About Programs; published as joint issue of *SIGPLAN Notices* (ACM) 7, 1 (Jan. 1972), and *SIGACT Newsl.* (ACM) 14 (Jan. 1972), 191–202.
42. Manna, Z., and Pnueli, A. Axiomatic approach to total correctness of programs. *Acta Inf. 3* (1974), 253–263.
43. Meyer, A.R., and Parikh, R. Definability in dynamic logic. In Conf. Proc. 12th Ann. ACM Symp. Theory of Computing, Los Angeles, Calif., April 28–30, 1980, pp. 1–7.
44. Olderog, E.R. General equivalence of expressivity definitions using strongest postconditions resp. weakest preconditions. Ber. 8007, Inst. Inf. Prakt. Math., Univ. Kiel, West Germany, 1980.
45. Olderog, E.R. Sound and complete Hoare-like calculi based on copy rules. Ber. 7905, Inst. Inf. Prakt. Math., Univ. Kiel, West Germany, 1980; also *Acta Inf.*, to appear.
46. Oppen, D.C., and Cook, S.A. Proving assertions about programs that manipulate data structures. In Conf. Rec., 7th Ann. ACM Symp. Theory of Computing, Albuquerque, N.M., May 5–7, 1975, pp. 107–116.
47. Owicki, S., and Gries, D. An axiomatic proof technique for parallel programs I. *Acta Inf. 6* (1976), 319–340.

48. PRATT, V.R. Semantical considerations on Floyd–Hoare logic. Proc. 17th IEEE Symp. Foundations of Computer Science, 1976, pp. 109–121.

49. PRESBURGER, M. Uber die Vollständigkeit eines gewissen Systems der Arithmetic ganzer Zahlen, in welchen die Addition als einzige Operation hervortritt. In C.R. 1er Congr. de Mathématiciens de Pays Slavs, 1929.

50. SCOTT, D., AND DE BAKKER, J.W. A theory of programs: Notes of an IBM Vienna seminar, 1969. Unpublished.

51. SHOENFIELD, J.R. *Mathematical Logic.* Addison-Wesley, Reading, Mass., 1967.

52. SOKOŁOWSKI, S. Axioms for total correctness. *Acta Inf. 9* (1977), 61–72.

53. SOKOŁOWSKI, S. Total correctness for procedures. In *Lecture Notes in Computer Science*, vol. 53: *Proc. 6th Symp. Mathematical Foundations of Computer Science.* Springer-Verlag, New York, 1977, pp. 475–483.

54. WAND, M. A new incompleteness result for Hoare's system. *J. ACM 25*, 1 (Jan. 1978), 168–175.

55. WIRTH, N., AND HOARE, C.A.R. A contribution to the development of ALGOL. *Commun. ACM 9*, 6 (June 1966), 413–432.

Received September 1979; revised January 1981; accepted May 1981