IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-1, NO. 1, MARCH 1975

Proving Loop Programs

SANAT K. BASU, MEMBER, IEEE, AND JAYADEV MISRA, MEMBER, IEEE

Abstract—Given a "DO WHILE" program P and a function F on a domain D, we investigate the problem of proving (or disproving) if P computes F over D. We show that if P satisfies certain natural constraints (well behaved), then there is a loop assertion, independent of the structure of the loop body, that is both necessary and sufficient for proving the hypothesis. We extend these results to classes of loop programs which are not well behaved and to FOR loops. We show the sufficiency of Hoare's DO WHILE axiom for well-behaved loop programs. Applications of these ideas to the problem of mechanical generation of assertions is discussed.

Index Terms—Assertion, DO WHILE axiom, equivalence, loop invariant, loop program, proof.

I. INTRODUCTION

PROGRAM proof techniques usually attach a set of predicates to specific points in the program so that the associated predicate is true whenever the control reaches any given point. If the predicates satisfy certain constraints, they can be used to prove that a certain predicate is true at output, from which we can prove that the program is correct (with respect to some given specification). The associated predicate is called an assertion at a particular point, following Floyd [5].

Proofs of assertions usually rely on an automatic or semiautomatic theorem prover. A modest amount of success in this area has been reported [6]. However, the generation of assertions requires not only human intervention, but demands a considerable amount of insight and ingenuity on the part of the prover. Several heuristic techniques to mechanically generate assertions have been reported [11].

The major difficulty in the mechanical generation of assertions arises whenever there is a loop structure involved, so that the control may reach the same point in the program several times with an altered set of variable values each time. Since a loop may get iterated an undetermined number of times, the assertion must be true with respect to each iteration.

In this paper, we will only deal with loops of the form WHILE B DO S, which is shown in a flow chart notation in Fig. 1. S may be any one-in one-out flow chart. We will call this flow chart schema a loop program.

The point in the loop marked with an x will be called the *reference point* of the loop. Any predicate P which is true whenever the control reaches the reference point will

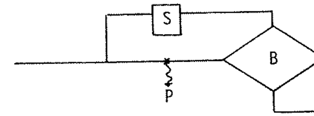


Fig. 1.

be called a loop assertion. Note that the loop assertion may depend on the program inside which the loop program is embedded. P will usually be a proposition over several quantified variables—some of the variables may be variables of the program, and some may not.

We will use the notation $Q\{T\}R$ due to Hoare [8], which stands for the logical proposition: "If the predicate Q is true before the execution of program T, then the predicate R is true after the execution of T, assuming that T terminates." Usually Q,R will be predicates over certain quantified variables whose value may be altered by the execution of T. We will call the predicate P (Fig. 1) a *loop invariant* for WHILE B DO S if and only if $P \wedge B\{S\}P$. Note that P is independent of the program inside which the loop program is embedded.

Every loop invariant, if it is true on entrance to the loop, is a loop assertion, although the converse is not true in general. We will prove this fact in a later section. We show in this paper that a loop assertion always exists which is both necessary and sufficient to prove that a loop program computes a particular function. This loop assertion is related directly to the function computed by the loop, and is independent of S (i.e., if S is replaced by S' such that the loop still computes the previous function, the loop assertion will remain unchanged). It is to be noted that proving a predicate to be a loop assertion might require global information about the program. This point is further clarified later in the paper.

Standard methods of program proving, however, rely on the existence of a loop invariant. Since a loop invariant is context independent, a specific predicate may be proven to be a loop invariant using only local properties of the program. We give the necessary and sufficient conditions for the existence of a class of loop invariants. Our proof is constructive, yielding a loop invariant in case it exists. This loop invariant, furthermore, is both necessary and sufficient for proving correctness. It is directly derivable from the function computed by the loop. We discuss the implications of these results in program proving. We prove the sufficiency of Hoare's DO WHILE axiom [8] under certain conditions.

Our results are applicable only to proving correctness, but not termination. Thus we will often use statements

Manuscript received November 1, 1974; revised January 15, 1975. This work was supported in part by the National Science Foundation under Grant GJ-36424.

The authors are with the Department of Computer Science, University of Texas at Austin, Austin, Tex. 78712.

such as "if the loop terminates, then —." Our approach is function theoretic along the lines suggested by Mills [15]; in fact, one of our basic theorems is a generalization of one due to Mills.

II. NOTATIONS, TERMINOLOGY

We will abbreviate WHILE B DO S to $W(B,S)$. Usually $W(B,S)$ will accept input in certain variables, operate on them, and terminate with output in certain variables. *We will assume that $W(B,S)$ produces its output in the input variables, i.e., input variables are transformed to output values as a result of the application of $W(B,S)$.* $W(B,S)$ will thus be a mapping from n tuples to n tuples where n is the number of input variables. Input variables to $W(B,S)$ will also be called *global* variables.

We will assume that B *is a predicate over some or all global variables.*

Any particular S will act on some of the global variables of $W(B,S)$. It may also define and operate on certain other variables, which we call the *local variables* of $W(B,S)$. We will assume that S operates on all global variables of $W(B,S)$ (even though some of the global variables may not be referred to inside S; S will be assumed to be an identity mapping for such variables). Thus S will be a mapping from m tuples to m tuples where $m \geq n$.

Along with the usual values of a variable, we accept ω (undefined) as one of the possible values. We will adopt the convention that the program does not terminate if it ever attempts to examine or use a variable whose value is ω at the moment. We furthermore assume that the program is not capable of assigning ω to a variable. All variables except the global variables are assumed to have the value ω at the beginning of the execution of $W(B,S)$.

A certain n tuple of variable values may not lead $W(B,S)$ to termination. The reason for this could be either an infinite looping or accessing a variable whose value is ω. This latter possibility can arise only inside S where a local variable gets accessed without receiving a value previously. We are thus led to the notion of the set of m tuples, which when input to $W(B,S)$, lead to termination.

Notations

$D^n(B,S)$	The set of n tuples, which when input to $W(B,S)$, lead to termination.
$D^m(B,S)$	The set of m tuples, which when input to $W(B,S)$ lead to termination.
$F_{B,S}$	The function computed by $W(B,S)$ on the global variables, i.e., for every $x \in D^n(B,S)$, $F_{B,S}(x)$ is the n tuple resulting from the application of $W(B,S)$ to x, and for $x \notin D^n(B,S)$, $F_{B,S}(x)$ is undefined. When the context is understood, we will abbreviate $F_{B,S}$ to F.
xt	Denotes an m tuple where x denotes the first n components and t the last m-n components. $x\omega$ denotes the m tuple where the last m-n components have undefined values. We will adopt the convention in describing an m-tuple (in the proper context) that the first n components refer to the global variables of $W(B,S)$ and the last $(m$-$n)$ components to the local variables of $W(B,S)$.
$\bar{y}$	Denotes the first n components of y, i.e., if $y = xt$, then $\bar{y} = x$.
$S(x)$	For $x \in D^m(B,S)$ denotes the m tuple resulting after the application of S to x. Thus $\overline{S(x\omega)}$, $x \in D^n(B,S)$ denotes the n tuple resulting after applying S to some $x\omega$.

One is usually interested in proving that a loop $W(B,S)$ computes a particular function over a certain domain D where D may be a proper subset of $D^n(B,S)$. It is to be noted that we have imposed no restrictions on the data structures the variables may represent. The results in this paper are to be interpreted in this wider context. The example below illustrates these points.

Example 1:

```
while i ≤ n do
    k = t − 1
    sum = sum + k
    i = i + 1
    t = k
end
```

Global variables are $\{i,\text{sum},t,n\}$.
Local variables are $\{k\}$.

$$B = [i \leq n]$$
$$S = [k = t - 1 \quad \text{sum} = \text{sum} + k \quad i = i + 1 \quad t = k]$$

S operates on $\{i,\text{sum},t,k,n\}$. We have added n to this list by convention.

$$D^4(B,S) = \{\langle i,\text{sum},t,n\rangle \mid i \neq \omega, \text{sum} \neq \omega, t \neq \omega, n \neq \omega\}$$
$$D^5(B,S) = \{\langle i,\text{sum},t,n,k\rangle \mid i \neq \omega, \text{sum} \neq \omega, t \neq \omega, n \neq \omega\}.$$

Consider a set $D^{(1)} = \{\langle i,\text{sum},t,n\rangle \mid n \geq 0, \text{sum} = 0, i = 0\}$ on which

$$F^{(1)} = F_{B,S}\langle i,\text{sum},t,n\rangle = \langle n, \sum_{j=t-n}^{t-1} j, t - n, n\rangle.$$

On a set $D^{(2)} = \{\langle i,\text{sum},t,n\rangle \mid n \geq 0, \text{sum} = 0, i \leq n\}$

$$F^{(2)} = F_{B,S}\langle i,\text{sum},t,n\rangle = \langle n, \sum_{j=t-n-i}^{t-1} j, t - n - i, n\rangle.$$

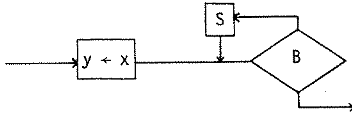


Fig. 2.

On a set $D^{(3)} = \{\langle i,\text{sum},t,n\rangle \mid n \geq 0, i \leq n\}$

$$F^{(3)} = F_{B,S}\langle i,\text{sum},t,n\rangle = \langle n,\text{sum} + \sum_{j=t-n-i}^{t-1} j, t-n-i, n\rangle.$$

$F^{(1)}$ is a restriction of $F^{(2)}$, which is a restriction of $F^{(3)}$. Note that we have referred to the components of the 4-tuple by the corresponding names of variables in the program for ease of comprehension. ■

For a given F and D, we call D *range inclusive* with respect to (wrt) F iff for $\forall x \in D$, $F(x) \in D$. Given a particular F and D, we define F', D' as follows (where D' is range inclusive wrt F'):

1) $D \subseteq D'$ and $x \in D \rightarrow [F(x) = F'(x)]$ and $F'(x) \in D'$,
2) $x \in [D' - D] \rightarrow [F'(x) = x]$,
3) if D'' satisfies 1) and 2), then $D' \subseteq D''$.

It is easily seen that a particular $W(B,S)$ computes a function F over D (which may not be range inclusive) iff it computes F' over D' [where D' is the smallest set having properties 1) and 2)]. Without specific mention, we will talk only of range inclusive domains, since it makes the statements simpler.

III. FUNDAMENTAL THEOREM OF LOOP ASSERTION

We will show in this section that $W(B,S)$ computes a certain function F over a certain domain D if and only if a particular predicate is a loop assertion. This loop assertion will use the symbols F,D and the variables x, which are the global variables of $W(B,S)$. We furthermore need the initial values of the global variables. To this end, we define an augmented schema $W'(B,S)$ as shown in Fig. 2.

y is never referred to inside S; it only saves the initial values of the global variables. (It is easy to show that a predicate $Q(x)$, which does not refer to the initial values y, is true at output iff $x \in D \wedge \neg B(x) \rightarrow Q(x)$. Thus only trivial statements can be made at output if we restrict ourselves only to the current values of the variables.)

Suppose $W(B,S)$ computes F over a certain domain D. The execution of the loop may be pictured as follows, starting with some initial values of global variables x_0. The m tuples generated during the execution are $x_0\omega, x_1t_1, x_2t_2, \cdots, x_nt_n$. Furthermore, $B(x_0), B(x_1), \cdots, B(x_{n-1})$ are true and $B(x_n)$ is false.

In this sequence, some x_j may belong to D, although it may not contain any useful information about the output —the loop might manage to store the useful information in t_j and retrieve it in subsequent iterations. The following theorem shows that the above case can never arise.

Theorem 1 (Fundamental Theorem of Loop Assertion): $W(B,S)$ computes F over a range inclusive domain D if and only if

1) input with any $x \in D$, $W(B,S)$ terminates with output in D,
2) $[x \in D \wedge \neg B(x)] \rightarrow [F(x) = x]$,
3) $\{[y \in D, x \in D] \rightarrow [F(x) = F(y)]\}$ is a loop assertion for $W'(B,S)$.

Furthermore, conditions 1), 2), and 3) are mutually independent.

Proof: We first show that if the above conditions hold with some F and range inclusive domain D, then $W(B,S)$ computes F over D. From 1), $W(B,S)$ halts for every x in D and D is range inclusive. Consider some input $y \in D$ to $W(B,S)$ for which x is the output.

From 1), $x \in D$.

From 3), $F(x) = F(y)$.

Since the loop halts at x, $B(x)$ is false. Hence from 2), $F(x) = x$. Thus we conclude that $F(y) = x$, i.e., we obtain $F(y)$ when we terminate for every $y \in D$.

We now prove the converse.

If 1) does not hold, $\exists\, y \in D$ such that the program either does not halt or halts with some $x \notin D$. In either case, it does not compute F over the range inclusive domain D. If 2) does not hold, $\exists\, y \in D$ such that $\neg B(y) \wedge F(y) \neq y$. Consider the action of $W(B,S)$ with y as input. Since $B(y)$ is false, the program halts with y as output. However, $F(y) \neq y$; hence $W(B,S)$ does not compute F.

Condition 3) asserts that whenever any intermediate $x_j \in D$, x_j must necessarily contain all the useful information regarding output. This part is easily provable in the absence of local variables (see Mills). Intuitively, if after some iteration $F(x) \neq F(y)$, the program cannot distinguish whether x is being input the first time or results during the computation of y (since there is no local variable to store this information). Hence it has to produce two different results $F(x)$ and $F(y)$ when input with x, which is impossible.

The crux of Theorem 1 is that the result is still true even in the presence of local variables. We first need to prove a lemma.

Lemma 1: Let Z be a program, which input with $x\omega$, produces $x't'$ as output. Then, input with $xt (t \neq \omega)$, it must produce $x't''$ (for some t'') as output.

Proof: Basically, the lemma says that if the program halts with $x\omega$ as input and produces some output x' in the first n components, then input with xt it will produce identical output in the first n components. The basic idea of the proof is to show that the computations must be identical with these two different inputs, and hence the program never uses any value from t.

Assume that Z is represented in a flow chart form where every line has a unique number. Initially the program control is on the line leading to the first step to be executed (predicate or function). A step of the program is the execution of a predicate or a function. Execution of predicate moves the program control to one of the two lines leading out. Execution of a function assigns values to

certain variables and moves the program control to the line leading out.

The state of computation (with any input xt) after the execution of the ith step can be characterized by two parameters $C_i(xt), L_i(xt)$. $C_i(xt)$ denotes the line number of the program control *after* the execution of the ith step (if the program terminates in less than i steps, $C_i(xt) = C_{i-1}(xt)$, i.e., the control remains on the output line permanently). $L_i(xt)$ is some set of ordered pairs of the form $\langle y, v(y) \rangle$ where y is the name of a variable and $v(y)$ $(\neq \omega)$ is its value after execution of the ith step (starting with input xt).

Consider the computation starting with $x\omega$. Initially let $L_0(x\omega)$ contain the names and values in x. Construct $L_i(x\omega)$ from $L_{i-1}(x\omega)$ as follows:

1) if the ith step was a predicate, let $L_i(x\omega) = L_{i-1}(x\omega)$;

2) if the ith step was a function, add the names and values of all the variables defined in this step to $L_{i-1}(x\omega)$ and modify the values of any existing variables in $L_{i-1}(x\omega)$ to their new values.

Starting with xt, we define $L_0(xt)$ to be the name, value pairs from x only. $L_i(xt)$ is to be obtained from $L_{i-1}(xt)$ by application of rules 1) and 2). [Substitute $L_i(xt)$, $L_{i-1}(xt)$ for $L_i(x\omega), L_{i-1}(x\omega)$ in 1) and 2).]

We will show that

$$\left.\begin{aligned} C_i(x\omega) &= C_i(xt) \\ L_i(x\omega) &= L_i(xt) \end{aligned}\right\} i \geq 0.$$

Proof is by induction on i. For $i = 0$, obviously the above is true. Assume that the above equalities are true up to $i - 1$. Now the ith step must be identical for both $x\omega$ and xt [since $C_{i-1}(x\omega) = C_{i-1}(xt)$]. Since the ith step is applied to $x\omega$, it can only use (examine or access) a variable from $L_{i-1}(x\omega)$. Since $L_{i-1}(x\omega) = L_{i-1}(xt)$, the outcome will be identical (computations and results of tests will be identical). Hence $C_i(x\omega) = C_i(xt)$ and $L_i(x\omega) = L_i(xt)$. The argument is slightly modified when the program terminates.

At termination, $L_\infty(x\omega) = L_\infty(xt)$. However, once a variable enters $L_i(x\omega)$ or $L_i(xt)$, it remains defined. x was initially in both.

Hence corresponding values are identical at the end.

(Proved.) ■

We will now show that if 3) is false, then $W(B,S)$ does not compute F over D. Suppose for some $y \in D, x \in D$, $F(x) \neq F(y)$. Then with $x\omega$ as input to $W(B,S)$, it halts with output $F(x)$. Hence with any xt as input to $W(B,S)$, it must halt with $F(x)$ as output. When input with $y\omega$, $W(B,S)$ computes some xt at an intermediate step, and hence must output $F(x)$ finally. Since $F(x) \neq F(y)$, $W(B,S)$ does not compute F over D. We have assumed that the program is not modified during its execution; only data values get modified.

We now show that conditions 1), 2), and 3) are independent. We will consider some programs which allege to compute

$$F\langle u,v \rangle = \langle u + v, 0 \rangle \qquad \text{over } D = \{\langle u,v \rangle \mid v \geq 0\}$$

while $v > 0$ **do**

$u = u - 1$

$v = v + 1$

end

The program never terminates for any $v > 0$. Conditions 2) and 3) are, however, satisfied.

while $v > 0$ **do**

if $u = 5$ **then** $v = -10$,

else $u = u + 1$

$v = v - 1$

end

end

The program terminates and satisfies 2) and 3), although the output is not always in D.

while $v > 0$ **do**

$u = u + 1$

$v = v - 1$

end

This program satisfies 1) and 3) with $F\langle u,v \rangle = \langle 2(u + v), 0 \rangle$ over $D = \{\langle u,v \rangle \mid v \geq 0\}$. However, it does not satisfy 2) since the following statement is not true:

$$[v \leq 0 \wedge v \geq 0] \rightarrow [\langle 2(u + v), 0 \rangle = \langle u,v \rangle].$$

Finally, consider

while $v > 0$ **do**

$v = v - 1$

end

with $F\langle u,v \rangle = \langle u + v, 0 \rangle$ over $\{\langle u,v \rangle \mid v \geq 0\}$. It satisfies 1) and 2), but not 3). (Proved.) ■

As an aside, we state a simple lemma.

Lemma 2: Condition 2) in Theorem 1 can be strengthened to

$$[x \in D] \rightarrow [\neg B(x) \leftrightarrow F(x) = x].$$

Proof: We have proved that $\neg B(x) \rightarrow F(x) = x$. For every $x \in D, \neg B(F(x))$ since the loop must terminate with $F(x)$. Hence

$$[x = F(x)] \rightarrow \neg B(x). \qquad \text{(Proved.)} ■$$

This lemma shows that the fixed points of F in D uniquely characterize B when $W(B,S)$ computes F over D.

We will now show that any other loop assertion which is sufficient for proving that $W(B,S)$ computes F over D must be at least logically as strong as $[x \in D, y \in D] \rightarrow [F(x) = F(y)]$. In $W'(B,S)$, we define a set R, which is a set of ordered pairs $\langle y,x \rangle$ which can possibly arise at the reference point of the loop during any iteration, starting with any x in D.

Let

$$D_y = \{\overline{S^k(y\omega)} \mid 0 \leq k \leq \min_j \neg B(\overline{S^j(y\omega)})\}$$

$$R = \{\langle y,x \rangle \mid x \in D_y,\ y \in D\}.$$

Corollary 1: Let $q(x,y)$ be a loop assertion for $W'(B,S)$, i.e.,

$$q(x,y) = \textbf{true}, \qquad \forall \langle y,x \rangle \in R.$$

Let $[q(x,y) \wedge \neg B(x)] \rightarrow [x = F(y)]$; then $\langle y,x \rangle \in R \wedge q(x,y) \rightarrow [y \in D, x \in D \rightarrow F(x) = F(y)]$.

Proof: If not, then $\exists\ \langle y^*,x^* \rangle \in R$ such that

$$q(x^*,y^*) \wedge F(x^*) \neq F(y^*).$$

Using Theorem 1, $W(B,S)$ does not compute F over D. Since $q(x,y)$ is true for all $\langle y,x \rangle \in R$, when the loop terminates, we conclude that it computes F—a contradiction. (Proved.) ■

The significant aspects of Theorem 1 are that local variables do not enter into the loop assertion; the loop assertion is completely determined by the function computed by the loop and conditions 1), 2), and 3) are necessary as well as sufficient.

We illustrate an application of Theorem 1 to prove a well-known program for computing power.

Example 2:

```
while v ≠ 0 do
        r = v/2
        t = 2 * r
        if (t ≠ v) then w = w * u
        v = r
        u = u * u
end
```

We would like to prove that the program computes $F\langle u,v,w \rangle = \langle \lambda,0,w * u^v \rangle$ over range inclusive $D = \{\langle u,v,w \rangle \mid v \geq 0\}$. Note that λ is a component value which is of no interest to us at output.

We must now prove the following.

1) The loop terminates for every $\langle u,v,w \rangle \in D$, and when it terminates, $v \geq 0$.

2) $[\langle u,v,w \rangle \in D \wedge [v = 0]] \rightarrow [F\langle u,v,w \rangle = \langle u,v,w \rangle]$ or $v = 0 \rightarrow \langle \lambda,0,w * u^v \rangle = \langle u,v,w \rangle$.

Note that this can be proven only for the second and third components, although not for the first one.

3) In the program shown below, we have to prove that $v \geq 0, b \geq 0 \rightarrow (0 = 0) \wedge (w * u^v = c * a^b)$ is a loop assertion.

```
a = u
b = v
c = w
while v ≠ 0 do
        r = v/2
        t = 2 * r
        if (t ≠ v) then w = w * u
        v = r
        u = u * u
end
```
■

It may be noted that the loop assertion in this case is identical to the one in the literature for this problem [12]. In fact, we have found this to be the case in most of the examples studied so far.

We have derived the assertion to be proven independently of the program text. Furthermore, if we can prove that this assertion is a loop assertion, we are assured that the program computes F, and if we disprove this claim, the program does not compute F. This is in contrast to the existing schemes for generating assertions where disproving the assertions does not necessarily mean that the program is incorrect.

We now present an example in which the loop assertion is nonvacuously true only at the beginning and end of the loop execution.

Example 3:

```
while v ≠ 0 do
        if first then s = u, t = v, u = 50, v = 50,
                first = false
            else s = s + 1, t = t - 1
                if t = 0 then u = s, v = t,
                first = true end
        end
end
```

Prove that $F\langle u,v,\text{first}\rangle = \langle u+v,0,\textbf{true}\rangle$ over $D = \{\langle u,v,\text{first}\rangle \mid v \geq 0, \text{first} = \textbf{true}\}$.

Note that an initial assignment statement of the form first = true can be absorbed into the domain D.

Consider $W'(B,S)$ where $a = u$, $b = v$, $c =$ first. Then the loop assertion is (assuming $b \geq 0 \wedge c = \textbf{true}$)

$$v \geq 0 \wedge \text{first} = \textbf{true} \rightarrow [u + v = a + b].$$

The antecedent of the proposition is true only on entrance and exit. During iterations, the antecedent is false and the statement is vacuously true.

Any proof of this assertion requires a look ahead (and indeed would require proving the program by some other technique). We will study the implications of this in Section V. ∎

Finally, we specialize the results to a FOR loop. Consider

```
for i = [J_k, J_{k+1}, ..., J_n) do
        S
end.
```

$[J_k,J_{k+1},\cdots,J_n)$ denotes the closed interval $[J_k,J_{k+1},\cdots,J_{n-1}]$.

Let F_k be the function computed by the above program, which we denote by FOR(i,k,J,S).

Theorem 2: FOR(i,k,J,S) computes a function F_k over a range inclusive domain D iff

1) it terminates with output in D when input with any x in D,

2) $\forall x \in D, F_n(x) = x$,

3) $[i = t] \rightarrow \{[y \in D, x \in D] \rightarrow F_k(y) = F_t(x)\}$ is a loop assertion for the program FOR$'(i,k,J,S)$ (shown below).

FOR$'(i,k,J,S)$:

```
y = x
for i = [J_k, J_{k+1}, ..., J_n) do
        S
end
```

Proof: Follows from the proof of Theorem 1. ∎

IV. COMPUTATIONS ON CLOSED DOMAIN

In the previous section, we discussed loop programs in which the intermediate values of input variables may not be in D. The only requirement was that the final value be in D when the initial value is in D.

In a large number of loop programs, it turns out that the intermediate computations (the values of input variables) remain in D or in some simple extension of D. In such a case, the results of the previous sections specialize to very simple forms which are amenable to proof by induction.

Definition: $W(B,S)$ is *closed wrt* D ($D \subseteq D^n(B,S)$) if and only if $\forall x \in D[B(x) \rightarrow \overline{S(x\omega)} \in D]$. $W(B,S)$ is said to be *closed* if and only if it is closed wrt $D^n(B,S)$.

Without loss in generality, we may augment any D with all those x's for which $B(x)$ is false. From now on, we assume that $[\neg B(x) \rightarrow x \in D]$. The following lemma is obvious.

Lemma 3: Let $W(B,S)$ be closed wrt D. Then for any $x \in D$, values of global variables after every iteration are in D. If the loop terminates, the output is in D. ∎

We now state Theorem 1 for the case of closed $W(B,S)$.

Theorem 1': Given that $W(B,S)$ is closed with respect to D, $W(B,S)$ computes F over the range inclusive domain D if and only if

1) $W(B,S)$ terminates when input with any x in D,

2) $[x \in D \wedge \neg B(x)] \rightarrow [F(x) = x]$,

3) $F(x) = F(y)$ is a *loop invariant* for $W(B,S)$.

Furthermore, 1), 2), and 3) are mutually independent.

Proof: Similar to that of Theorem 1. ∎

The importance of Theorem 1' is that $F(x) = F(y)$ is a loop invariant (and not just a loop assertion). That is, $[F(x) = F(y) \wedge B(x)] \rightarrow [F(\overline{S(x\omega)}) = F(y)]$ is a theorem, independent of the program in which $W(B,S)$ is embedded. Proposition 3) can thus be proved or disproved by a theorem prover. This specialization rules out pathological cases like the program in Example 3.

We state Theorem 1' in a more symmetric form in the following corollary.

Corollary 2: Suppose $[x \in D \wedge B(x) \rightarrow \overline{S(x\omega)} \in D]$. $W(B,S)$ computes F over the range inclusive domain D iff

1) $W(B,S)$ terminates for every $x \in D$,

2) $[B(x) \rightarrow F(x) = F(\overline{S(x\omega)})] \wedge [\neg B(x) \rightarrow F(x) = x]$, $\forall x \in D$. ∎

Thus the proof of a program with closure reduces to proof of termination and proof of proposition 2).

Example 4: Consider Example 2. We prove closure by proving $v \geq 0 \wedge v \neq 0 \rightarrow v/2 \geq 0$. We then prove termination.

Finally, the proof of proposition 2) reduces to proving

$$v \neq 0 \rightarrow \left[w * u^v = \begin{cases} w * u * (u^2)^{v/2} \text{ when } v \text{ is odd} \\ w * (u^2)^{v/2} \text{ when } v \text{ is even} \end{cases} \right]$$

$$\wedge\ v = 0 \rightarrow (w * u^v = w). \quad ∎$$

Usually the loop has initializations in the beginning, which restricts the domain over which there may not be closure. However, it is usually possible to consider a superset of the given domain and to prove that $W(B,S)$ computes a certain function over the superset whose restriction (to the given domain) is the function desired. For instance, in Example 2, restrict the domain to $D' = \{\langle u,v,w\rangle \mid v \geq 0, w = 1\}$ and prove that $W(B,S)$ computes $F\langle u,v,w\rangle = \langle \lambda,0,u^v\rangle$ over D'. $W(B,S)$ is not closed wrt D', although it is closed wrt D.

In certain extreme situations, the program can be proved

simply by locating a suitable superset with respect to which $W(B,S)$ is closed. We consider a modified version of a program due to London [14] for computing factorial without multiplication.

Example 5:

```
while R < N do
    s = 1
    v = u
    while s ≤ R do
        u = u + v
        s = s + 1
    end
    R = R + 1
end
```

Show that the program computes $F\langle R,u,N\rangle = \langle N,N!,N\rangle$ over $D = \{\langle R,u,N\rangle \mid R = 1,\ u = 1,\ N \geq 1\}$. If we identify the values of input variables that occur after every iteration, we get $D' = \{\langle R,u,N\rangle \mid u = R!, N \geq R, R \geq 1\}$. $W(B,S)$ is closed wrt D'. In fact, proving the closure wrt D' is almost enough to prove that $W(B,S)$ computes F, since $(x \in D' \wedge \neg B(x)) \rightarrow (R \geq N \wedge [u = R!, R \geq 1, N \geq R]) \rightarrow u = N!$ Some ingenuity must go into locating D'. ■

The results can be specialized to a FOR loop. The FOR loop FOR(i,k,J,S) has closure iff $[x \in D \wedge i \neq J_n \rightarrow \overline{S(x\omega)} \in D]$. Note that i is restricted to $J = [J_1,\cdots,J_n]$. We may restate Theorem 2 for this case.

Theorem 2': A FOR loop with closure computes F_k over D if and only if

1) it terminates for every $x \in D$ and $i \in J$,
2) $i \neq J_n \rightarrow [F_i(x) = F_{i+1}(S(x\omega))] \wedge [F_n(x) = x]$. ■

This may be contrasted with the approach in [10].

Example 6: Prove that the following program computes:

$$F_j\,\langle \text{sum},A,n\rangle = \langle \text{sum} + \sum_{k=j}^{n-1} A(k),A,n\rangle$$

over the domain $D = \{\langle \text{sum},A,n\rangle\}$

```
for i = j to n - 1 do
    sum = sum + A(i)
end
```

We generate the theorem to be proven.

$$i \neq n \rightarrow \left\{[\text{sum} + \sum_{k=i}^{n-1} A(k) = \text{sum} + A(i) + \sum_{k=i+1}^{n-1} A(k)] \wedge [A = A] \wedge [n = n]\right\}$$

$$\wedge \left\{[\text{sum} + \sum_{k=n}^{n-1} A(k) = \text{sum}] \wedge [A = A] \wedge [n = n]\right\}.$$

■

We will now derive a necessary and sufficient condition for the equivalence of a loop program with some other program.

Given two programs P and P', it is often desired to find if the programs are equivalent in the sense that they produce identical results for every input. There are several notions of equivalence, depending on the termination properties of P and P'. For our purposes, we define equivalence as follows.

Definition: Two programs P and P' are equivalent on an input domain D if and only if for every $x \in D$, they both halt and produce identical outputs.

Theorem 3: Let $W(B,S)$ be closed wrt D, and it halts when input with any $x \in D$. P is equivalent to $W(B,S)$ if and only if

1) P halts for every $x \in D$,
2) $\forall x \in D\ \{[B(x) \rightarrow P(x) = P(S(x\omega))] \wedge [\neg B(x) \rightarrow P(x) = x]\}$.

Proof: The theorem is identical to Corollary 1 except that P has been substituted for F. ■

We illustrate the application of the theorem with an example.

Example 7: Consider the programs shown below, which are to be proved equivalent over the input domain $D = \{v \mid v \geq 0\}$. First we must prove closure of $W(B,S)$ wrt D.

```
while v ≠ 0 do
    v = v - 1
end
```

$\underline{W(B,S)}$

```
if v ≠ 0 then
    while v ≠ 0 or v ≠ 1 do
        v = v - 2
    end
    if v = 1 then
        v = v - 1
    end
end
```

$\underline{P}$

To prove equivalence of P and $W(B,S)$ over D, we generate the following condition to be verified:

$$v \geq 0 \rightarrow [v \neq 0 \rightarrow P(v) = P(v-1)] \wedge [v = 0 \rightarrow P(v) = v].$$

This technique is most fruitful when P is a straight line program so that a symbolic computation [13] of P may be performed with x and $S(x)$ and symbolic results tested for equality. ■

V. CONDITIONS FOR THE EXISTENCE OF LOOP INVARIANT AND SUFFICIENCY OF HOARE'S DO WHILE AXIOM

Given any $W(B,S)$, we call a predicate P a loop invariant if it satisfies the following conditions.

1) P is a predicate over some or all global variables of $W(B,S)$ and possibly certain other variables (which may not be in the program). *Local variables must never occur in P.* Thus P has the form $P(x,y)$ where x is the set of global variables and y certain other variables not occurring in $W(B,S)$.

2) $P(x,y) \wedge B(x) \rightarrow P(\overline{S(x\omega)},y)$.

Every loop invariant is a loop assertion, although the converse is not true, as shown by the following example.

Example 8:

```
s = u, t = v
while v ≠ 0 do
    u = u + 1
    v = v − 1
end
```

A loop assertion is $[v = 0 \rightarrow u = s + t]$. However, this is not a loop invariant since $[v = 0 \rightarrow u = s + t] \wedge [v \neq 0] \rightarrow [v - 1 = 0 \rightarrow u + 1 = s + t]$ is not a theorem (as can be seen by substituting $v = 1, u = 5, s = 1, t = 3$ into the above statement). ■

There could be several loop invariants for any particular $W(B,S)$. Not all of them would be powerful enough to prove that $W(B,S)$ computes a particular function. Hence we define P to be an *F-adequate loop invariant* iff it is a loop invariant and

$$P(x,y) \wedge \neg B(x) \rightarrow x = F(y).$$

We have shown in the previous section that an F-adequate loop invariant exists when $W(B,S)$ is closed wrt D. We will show in this section that F-adequate loop invariants do not exist unless there is closure (or an extended form of closure) wrt $D^n(B,S)$.

We define a few terms needed to state the result.

$W(B,S)$ is *closed* if and only if it is closed wrt $D^n(B,S)$. $xt \in D^m(B,S)$ is *reachable* iff $\exists\, y \in D^n(B,S)$ such that $S^k(y\omega) = xt$ for some k and $B(\overline{S^i(y\omega)})$ for every $i < k$. Reachability of xt essentially means that xt appears as an intermediate value during iterations for some y as input.

$W(B,S)$ is *ϵ-closed* (extended closed) if and only if for any x, xt_1 and xt_2 are reachable implies that $W(B,S)$ yields identical output (in the first n components) when input with xt_1 or xt_2.

Note 1: If $x \in D^n(B,S)$, then $xt \in D^m(B,S)$ for any t. Using Lemma 1, $W(B,S)$ must yield identical output for every xt as for $x\omega$. Hence if $W(B,S)$ is closed, then it is ϵ-closed.

Note 2: The above definition of ϵ-closure can be easily generalized to take into account some rather than all the local variables of $W(B,S)$.

We give examples of closure and ϵ-closure below.

Example 9:

```
while v ≠ 0 do
    u = u + 1
    v = v − 1
end
```

$D^2(B,S) = \{\langle u,v\rangle \mid v \geq 0\}$

$W(B,S)$ is closed.

```
while v ≠ 0 do
    if v = −1 then v = t, u = u − v
        else u = u + 1, v = v − 1, t = v,
            v = −1, u = u + t
    end
end
```

$D^2(B,S) = \{\langle u,v\rangle \mid v \geq 0\}$

$D^3(B,S) = \{\langle u,v,t\rangle \mid v \geq 0 \text{ or } v = -1 \text{ and } t \geq 0\}$

The above program is not closed, since during iterations, we obtain values such as $\langle u,-1\rangle$. However, it is ϵ-closed since $\langle u,-1,t_1\rangle$ and $\langle u,-1,t_2\rangle$ lead to identical outputs (in the first two components).

```
while v ≠ 0 do
    if first then t = v, v = 50, first = false
        else u = u + 1, t = t − 1
            if t = 0 then v = t, first = true end
    end
end
```

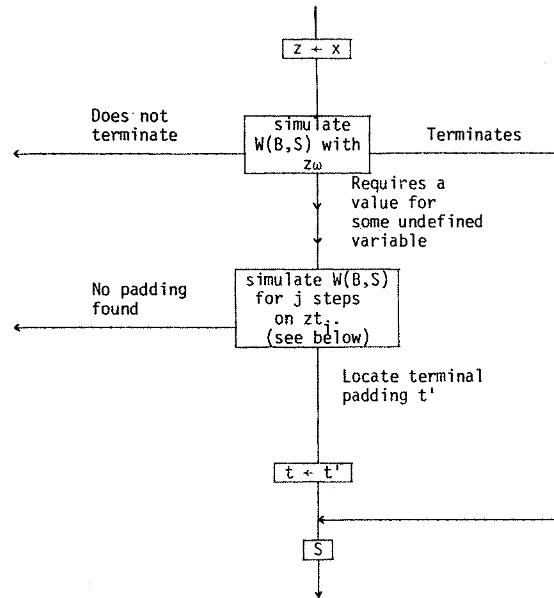


Fig. 3.

$D^3(B,S) = \{\langle u,v, \text{first}\rangle \mid u \geq 0, \text{first} = \textbf{true}\}$

$D^4(B,S) = \{\langle u,v, \text{first}, t\rangle \mid v \geq 0 \text{ and first} = \textbf{true} \text{ or first} = \textbf{false} \text{ and } t \geq 0\}$

This program is not ϵ-closed (or closed) since $\langle u, 50, \textbf{false}, t_1\rangle$ and $\langle u,50, \textbf{false}, t_2\rangle$ lead to different outputs, namely, $\langle u + t_1,0, \textbf{true}\rangle$ and $\langle u + t_2,0, \textbf{true}\rangle$. ■

Theorem 4: Let $W(B,S)$ compute F. Then $W(B,S)$ has an F-adequate loop invariant if and only if it is ϵ-closed.

Proof: If $W(B,S)$ is closed and it computes a function F over $D^n(B,S)$, then the following is an F-adequate loop invariant:

$$x \in D^n(B,S) \wedge F(x) = F(y).$$

The proof follows from Theorem 1'.

We now prove the result when $W(B,S)$ is ϵ-closed but not closed. We will construct S' such that $W(B,S')$ is equivalent to $W(B,S)$ over $D^n(B,S)$. Furthermore, $W(B,S')$ would be closed [wrt $D^n(B,S')$]. Thus there exists a loop invariant of the form

$$x \in D^n(B,S') \wedge F'(x) = F'(y)$$

for $W(B,S')$, where F' is the function computed by $W(B,S')$. It will be shown that the above loop invariant is an F-adequate loop invariant for $W(B,S)$.

The construction of S' uses a standard technique from automata theory. We first define a *terminal padding*. For any x,t' is a terminal padding if $W(B,S)$ terminates with xt' as input and no xt for any t occurs as an intermediate value during execution of $W(B,S)$ with xt'. Thus if t' is a terminal padding for x, we are assured that x will never appear again as the first n components during execution of $W(B,S)$ with xt'.

We can now describe S'. Input with any $x\omega,S'$ will determine if there is any xt for which $W(B,S)$ terminates. In case there is, it locates a terminal padding t' and inputs xt' into S. Assume that the different paddings may be indexed $t_1,t_2,\cdots$,etc.

We show S' schematically in Fig. 3.

We use the standard mapping E of pairs of integers to integers where

$$E(i,j) = \frac{(i+j-1)\ (i+j-2)}{2} + j.$$

Whenever $E(i,j)$ is enumerated, we simulate $W(B,S)$ for j steps on zt_i. During the process of simulation, we retain the last value zt' which occurs as an intermediate value. If the simulation leads to termination, t' serves as t. The reason for locating such a t' is to avoid any looping, since we are certain that $x\omega$ will never be input to S' again.

We now show that $D^n(B,S') \supseteq D^n(B,S)$. Suppose some $x \in D^n(B,S)$. Then $W(B,S)$ terminates with $x\omega$. S' will eventually discover this fact in the first simulation step and will just apply S to $x\omega$. Suppose $x \notin D^n(B\ S)$, but $xt \in D^m(B,S)$. Then S' will discover this fact in the second simulation step. Furthermore, all xt_1,xt_2 yield identical results; hence it will pad x with some terminal padding t' and ask S to execute on xt'.

Thus $x \notin D^n(B,S),xt \in D^m(B,S)xt$ was reachable implies that $x \in D^n(B,S')$. $W(B,S')$ is obviously closed. The function F' computed by $W(B,S')$ is

$$F'(x) \text{ is equal to } F(x) \text{ when } x \in D^n(B,S)$$

and $F'(x)$ is the unique value produced by $W(B,S)$ on reachable $xt \in D^m(B,S)$. It is straightforward to see that the loop invariant for $W(B,S')$ is also a loop invariant for $W(B,S)$ and that it is F-adequate.

It is possible to prove the result more simply be defining $F',D^n(B,S')$ starting from F and $D^n(B,S)$. Then one can show that $x \in D^n(B,S') \wedge F'(x) = F'(y)$ is a

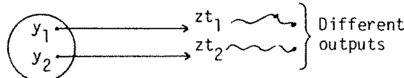


Fig. 4.

loop invariant for $W(B,S)$. *However, the above construction shows that there is a finite description for F'*, namely, F' is the function computed by $W(B,S')$, and we have demonstrated a finite description of $W(B,S')$.

To complete the proof, we show that there can be no F-adequate loop invariant when $W(B,S)$ is not ϵ-closed. Since $W(B,S)$ is not ϵ-closed, there exist $zt_1,zt_2 \in D^m(B,S)$ which are reachable (from y_1,y_2) and which yield different results. Let the input variables be denoted by x. The situation is shown pictorially in Fig. 4.

Let t' be a terminal padding for z.

Consider the following program $W(B,S')$.

while B **do**

if $x = z$ **then** $t = t'$ **end**

S

end

This program computes a function different from F since zt_1,zt_2 will yield identical results in $W(B,S')$. Now suppose there was an F-adequate loop invariant P for $W(B,S)$. P is also a loop invariant for S' since

$$P(x,y) \wedge B(x) \; \{S\} \; P(x,y)$$

$$P(x,y) \wedge B(x) \; \{\textbf{if } x = z \textbf{ then } t = t'\} \; P(x,y) \wedge B(x)$$

implies

$$P(x,y) \wedge B(x) \; \{\textbf{if } x = z \textbf{ then } t = t'; \; S\} \; P(x,y).$$

Since P was F-adequate, $P(x,y) \wedge \neg B(x) \rightarrow x = F(y)$. Thus we may conclude that $W(B,S')$ also computes F, which it does not.

We remark that the program $W(B,S')$ may not be effectively constructible. However, the existence of such a program is sufficient to prove the result. (Proved.) ■

Example 9 (Continued): An F-adequate loop invariant for the second program in Example 9 is

$$[v = 0 \wedge (u + v) = (a + b)] \wedge [v = -1 \wedge u = a + b].$$

The third program in the same example has no F-adequate loop invariant which does not involve t. ■

Theorem 4 basically shows that if the local variables carry useful information across iterations, then they must be included in the loop invariant. Conversely, if they do not carry any useful information, they need not appear in the loop invariant. We believe that a "well-behaved loop" does not employ the local variables across iterations. However, Theorem 4 can be generalized to Theorem 4', which takes care of "bad loops."

Theorem 4': Let $W(B,S)$ compute F. There exists an F-adequate loop invariant which uses the global variables and a subset of local variables if and only if $W(B,S)$ is ϵ-closed with respect to these variables.

Proof: Similar to Theorem 4. ■

Again closure (ϵ-closure) emerges as a central theme.

One simple way to assure closure is to make every variable global (by assigning them arbitrary values in the beginning). This, we believe, is self defeating in that the proof process now has to cope with complications. It is worthwhile and important to note the scope (the lifespan) of every variable.

Finally, we show that Hoare's axiom for DO WHILE defines a unique computation when $W(B,S)$ is closed.

Theorem 5: A schema $S'(B,S)$ is equivalent to $W(B,S)$ (which is closed) if and only if for every P,B,S

1) $\dfrac{P \wedge B\{S\}P}{P\{S'\}P \wedge \neg B}$ and

2) either they both halt or both diverge for every input.

Proof: It is straightforward to show that if $S'(B,S)$ is equivalent $W(B,S)$, then 1) and 2) hold. We will show that if 1) and 2) hold, then $S'(B,S)$ is equivalent to $W(B,S)$.

Let P be $x \in D^n(B,S) \wedge F_{B,S}(x) = F_{B,S}(y)$. $P \wedge B\{S\}P$ since P is a loop invariant for $W(B,S)$.

Furthermore, $\neg B(x) \wedge x \in D^n(B,S) \rightarrow F_{B,S}(x) = x$

Since $P\{S'\}P \wedge \neg B$,

$$x \in D^n(B,S) \wedge F_{B,S}(x) = F_{B,S}(y)\{S'(x)\}x \in D^n(B,S) \wedge F_{B,S}(x) = F_{B,S}(y) \wedge \neg B(x)$$

or

$$x \in D^n(B,S) \wedge F_{B,S}(x) = F_{B,S}(y)\{S'(x)\}F_{B,S}(x) = F_{B,S}(y) \wedge F_{B,S}(x) = x.$$

Since $x = y \rightarrow F_{B,S}(x) = F_{B,S}(y)$, we conclude

$$x \in D^n(B,S) \wedge (x = y)\{S'(x)\} \; x = F_{B,S}(y)$$

which says that S' computes whatever $W(B,S)$ computes. (Proved.) ■

VI. DISCUSSION

The main results of this paper can be summarized as follows.

1) There always exists a loop assertion, related to the function computed by the loop, which is both necessary and sufficient for proving correctness.

2) There does not exist an F-adequate loop invariant, in general, unless $W(B,S)$ is closed or ϵ-closed.

3) In the latter cases, the loop invariant is related to the function computed and is derivable from 1).

We believe that loop invariants intuitively capture our notion of what is provable by inductive assertion. Thus the program in Example 3 is not provable by inductive assertion (unless we let the local variables enter into the invariant). We feel that *closure of a loop (with respect to*

its global variables) is one of its most important properties. Then the program is not using the local variables across iterations. Most well-written programs seem to satisfy this requirement.

These results will have practical applications in proving programs. We have pointed out the form of the F-adequate loop invariant under closure. For a pure loop program with closure, it is now trivial to generate the assertion to be proved. Usually loops do not occur in a pure form as assumed. The initializations will usually restrict the domain over which there may not be closure. One has to discover a superset with respect to which there is closure and the function computed over the superset. This may be complicated in cases, but we hope we have convinced the reader that no blind heuristic would succeed in such a case, since the heuristic must essentially discover the superset and the function.

We remark here that once we prove that $W(B,S)$ computes F, we can prove any other partial property of $W(B,S)$ that might be required.

ACKNOWLEDGMENT

The authors are indebted to Dr. H. D. Mills of IBM, whose work inspired the present work. Thanks go to M. Conner of the University of Texas for pointing out an error in the original formulation of Theorem 1. D. Good, also of the University of Texas, has been a patient listener and an important critic during the entire course of this research. The idea of augmented schema is due to C. McGowan of Brown University.

REFERENCES

[1] S. Basu and R. Yeh, "Program verification by predicate transformation," Inst. Comput. Sci. Comput. Appl., Univ. Texas, Austin, Rep. SESLTC-1, Aug. 1974.

[2] R. Boyer and J. Moore, "Proving theorems about Lisp functions," in *Proc. 3rd Int. Conf. Artificial Intelligence*, Stanford Univ., Stanford, Calif., 1973.

[3] O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming*. London: Academic, 1972.

[4] B. Elspas, K. Levitt, R. Waldinger, and A. Waksman, "An assessment of techniques for proving program correctness," *Computing Surveys*, vol. 4, June 1972.

[5] R. W. Floyd, "Assigning meanings to programs," in *Proc. Amer. Math. Soc. Symp. Appl. Math.*, vol. 19, 1966.

[6] D. Good, R. London, and W. Bledsoe, "An interactive program verification system," Univ. Southern California, Los Angeles, USC Inform. Sci. Inst. Tech. Rep. 22, Sept. 1974.

[7] C. A. R. Hoare, "Algorithm 65, Find," *Commun. Ass. Comput. Mach.*, vol. 4, July 1961.

[8] ——, "An axiomatic approach to computer programming," *Commun. Ass. Comput. Mach.*, vol. 12, Oct. 1969.

[9] ——, "Proof of a program: FIND," *Commun. Ass. Comput. Mach.*, vol. 14, Jan. 1971.

[10] ——, "A note on the FOR statement," *BIT*, vol. 12, 1972.

[11] S. Katz and Z. Manna, "A heuristic approach to program verification," in *Proc. 3rd Int. Conf. Artificial Intelligence*, Stanford Univ., Stanford, Calif., 1973.

[12] J. King, "A program verifier," Ph.D. dissertation, Carnegie-Mellon Univ., Pittsburgh, Pa., 1969.

[13] ——, "Symbolic execution and program testing," IBM Res. Rep. RC5082, 1974.

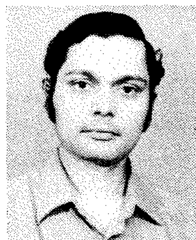
[14] R. London, private communication, 1974.

[15] H. D .Mills, "Mathematical foundations of structured programming," IBM Federal Syst. Div., Gaithersburg, Md., Rep. FSC 72-6012, Feb. 1972.

[16] M. Moriconi, "Semiautomatic synthesis of inductive predicates," Dep. Math., Univ. Texas, Austin, Rep. ATP-16, June 1974.

[17] L. C. Ragland, "A verified program verifier," Ph.D. dissertation, Univ. Texas, Austin, 1973.

[18] B. Wegbreit, "Heuristic methods for mechanically deriving inductive assertions," in *Proc. 3rd Int. Conf. Artificial Intelligence*, Stanford Univ., Stanford, Calif., 1973.



Sanat K. Basu (M'71) received the B.Sc. and M.Sc. degrees from Banaras University, India, in 1957 and 1959, respectively, and the Ph.D. degree from the University of Bombay in 1966.

He was a Fellow of the Tata Institute of Fundamental Research from 1960 to 1969, and a Research Scientist with the Department of Computer Science, Carnegie-Mellon University, from 1967 to 1969. He then joined the Electrical Engineering Department and the Computer Center of the Indian Institute of Technology, Kanpur, as an Assistant Professor. Currently he is a Visiting Associate Professor in the Department of Computer Science, University of Texas, Austin.

Dr. Basu is a member of SIGACT and CSI.

Jayadev Misra (S'71–M'72) received the B. Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, in 1969, and the Ph.D. degree in computer science from The Johns Hopkins University, Baltimore, Md., in 1972.

He worked for IBM, Federal System Division, from January 1973 to August 1974. He is currently with the Department of Computer Science, University of Texas, Austin.

Dr. Misra is a member of the Association for Computing Machinery.