

# A New Verification Rule and Its Applications

J. C. HUANG, MEMBER, IEEE

**Abstract**—This paper describes a verification rule for loop programs, and shows how it can be used in conjunction with the invariant-relation theorem to facilitate verification of programs.

**Index Terms**—Consistency, decomposition, invariant-relation theorem, loop program, predicate transformation, program verification, subgoal induction, verification rule.

## I. INTRODUCTION

A COMMON task in program verification is to show that, for a given program  $S$ , if a certain “precondition”  $Q$  is true before the execution of  $S$  then a certain “postcondition”  $R$  is true after the execution, provided that  $S$  terminates. This logical proposition is commonly denoted by  $Q\{S\}R$  (a notation due to Hoare [5]) for short. If we succeeded in showing that  $Q\{S\}R$  is a theorem (i.e., always true), then to show that  $S$  is *consistent* [6] or *partially correct* [7], with respect to some input predicate  $I$  and output predicate  $\phi$ , is to show that  $I \rightarrow Q$  and  $R \rightarrow \phi$  (see, e.g., [4], [7], [10]).

In this conceptual framework, verification of the consistency can be carried out in two ways. Given  $S$ ,  $I$ , and  $\phi$  we may first let  $R \equiv \phi$  and show that  $Q\{S\}\phi$  for some predicate  $Q$ , and then show that  $I \rightarrow Q$ . Alternatively, we may let  $Q \equiv I$  and show that  $I\{S\}R$  for some predicate  $R$ , and then show that  $R \rightarrow \phi$ . In the first approach the basic problem is to find as weak<sup>1</sup> as possible a condition  $Q$  such that  $Q\{S\}\phi$  and  $I \rightarrow Q$ . A possible solution is to use the method of predicate transformation due to Basu and Yeh [1] and Dijkstra [3] to find the weakest precondition. In the second approach the problem is to find as strong as possible a condition  $R$  so that  $I\{S\}R$  and  $R \rightarrow \phi$ . This problem is fundamental to the method of inductive assertions (see, e.g., [7], [10]), and may become quite difficult when a loop structure is involved. A possible solution is to apply the so-called invariant-relation theorem [2], [3].

Specifically, if  $S$  is a loop program of the form “while  $B$  do  $S_1$ ,” and if  $P$  is a predicate such that  $(P \wedge B)\{S_1\}P$ , then  $P$  is said to be a loop invariant of  $S$ . The invariant-relation theorem states that if  $P$  is a loop invariant of  $S$  then  $P\{S\}(\sim B \wedge P)$ . Thus, the consistency of a loop program may be verified by finding a suitable loop invariant. However, there are cases in which  $\sim B \wedge P$  is not strong enough to prove the consistency. Why is that so, and how can a stronger postcondition be found to make the verification possible? We offer the answers in this paper.

Manuscript received June 19, 1979; revised January 25, 1980. This work was supported by the National Science Foundation under Grant MCS76-81717.

The author is with the Department of Computer Science, University of Houston, Houston, TX 77004.

<sup>1</sup>A condition  $P$  is said to be weaker (stronger) than another condition  $P'$  if the set defined by  $P$  is a superset (subset) of that defined by  $P'$ .

To fix the idea, let us consider the following loop program:

```
while  $x \leq 100$  do  $x := x + 10$ ;
```

if  $x$  is assigned a value, say  $c$ , before this loop is executed, then  $x \geq c$  is obviously a loop invariant. By the invariant-relation theorem we can thus conclude that  $x > 100 \wedge x \geq c$  after the loop terminates, if it does. However, it is easy to see that if  $c > 100$  then the loop will not be executed and, therefore,  $x = c$ . On the other hand, if  $c \leq 100$  then the loop will be iterated at least once, and when it terminates, we not only know that  $x > 100$ , but also know that  $x \leq 110$ . In other words, this loop program is consistent with respect to the output predicate:  $c > 100 \wedge x = c \vee c \leq 100 \wedge x > 100 \wedge x \leq 110$ . Obviously, this predicate is not implied by  $x > 100 \wedge x \geq c$ , the postcondition obtainable from an application of the invariant-relation theorem. Thus, the use of this theorem alone is not sufficient to verify the consistency of the program. The reason is that there are two important cases concerning the values of  $x$  prior to the execution of this loop program, viz.  $x > 100$  and  $x \leq 100$ . We can be much more specific about the condition that will be true upon an execution of the program if we treat each case separately. The postcondition directly derivable from an application of the invariant-relation theorem is true for both cases, and, therefore, is less specific (i.e., weaker).

In general, it is advantageous to consider a loop program for two possible cases: the case in which the loop body will not be iterated at all, and the case in which the loop body will be iterated at least once. The idea is similar to that of program verification by subgoal induction [8], and has been utilized to find the weakest precondition [1] of a “while” statement. The main result of this work is that, for the case in which the loop body will be iterated at least once, we may be able to derive from the program text a postcondition that can not be obtained by an application of the invariant-relation theorem. We shall formulate this result as an inference rule (called a verification rule) in the next section. We shall also combine this result with the invariant-relation theorem to formulate several verification rules. In Section III we show how these verification rules can be utilized to facilitate program verification.

## II. THE MAIN RESULT

In the following we shall use  $\vdash P$  to denote the fact that  $P$  is a theorem (i.e., always true). A verification rule will be stated in the form “if  $\vdash P$  then  $\vdash Q$ ,” which says that if proposition  $P$  has been proved as a theorem then  $Q$  also is thereby proved as a theorem.

Now consider the proposition:  $B\{\text{repeat } S \text{ until } \sim B\}R$ . We know that  $B$  becomes false if and when the “repeat” statement terminates. Other than this fact, what can we

say about the postcondition R? Note that S will be executed at least once, and B is true just before S is executed for the last time (before the "repeat" statement terminates). Therefore,  $B\{\text{repeat } S \text{ until } \sim B\}R$  is a theorem if the weakest precondition<sup>2</sup> of S with respect to R, denoted by  $\text{wp}(S, R)$ , is logically equivalent to B. This can be stated as a verification rule

$$\text{if } \vdash \text{wp}(S, R) \equiv B \text{ then } \vdash B\{\text{repeat } S \text{ until } \sim B\}R. \quad (1)$$

To make use of this rule we need to find, for given B and S, a predicate R such that  $\vdash \text{wp}(S, R) \equiv B$ . For this purpose, it is convenient to let  $R \equiv \text{wp}(S^{-1}, B)$ , where  $S^{-1}$  is a sequence of statements. As the result,  $\text{wp}(S, R) \equiv \text{wp}(S, \text{wp}(S^{-1}, B)) \equiv \text{wp}(S; S^{-1}, B)$ , and (1) becomes

$$\text{if } \vdash \text{wp}(S; S^{-1}, B) \equiv B \text{ then } \vdash B\{\text{repeat } S \text{ until } \sim B\}\text{wp}(S^{-1}, B). \quad (2)$$

This verification rule represents the main result of this work. The problem of finding a predicate R such that  $\text{wp}(S, R) \equiv B$  is now transformed into that of finding a sequence  $S^{-1}$  of statements such that  $\text{wp}(S; S^{-1}, B) \equiv B$ . The latter is found to be more tractable.

Note that, if S;  $S^{-1}$  is a sequence of assignment statements,  $\text{wp}(S; S^{-1}, B) \equiv B$  is true as long as an execution of S;  $S^{-1}$  will not alter the value of any variable that occurs in B. Thus, the required  $S^{-1}$  may be found by noting that, if S contains a statement that changes the value of a variable in B,  $S^{-1}$  should contain a statement that restores the old value of that variable. The following examples should be helpful in visualizing how to find  $S^{-1}$  for given B and S.

$$\begin{array}{ll} \text{a)} & B: x \leq 100 \\ & S: x := x + 10; \\ & S^{-1}: x := x - 10; \\ \text{wp}(S^{-1}, B): & x \leq 110 \end{array}$$

$$\text{if } \vdash (P \wedge B)\{S\}Q \text{ and } \vdash (Q \wedge B)\{S\}Q \text{ and } \vdash \text{wp}(S; S^{-1}, B) \equiv B \text{ then } \vdash (P \wedge B)\{\text{repeat } S \text{ until } \sim B\}\text{wp}(S^{-1}, B) \wedge \sim B \wedge Q. \quad (6)$$

$$\begin{array}{ll} \text{b)} & B: x > c \\ & S: x := a; \\ & S^{-1}: \text{(does not exist)} \\ \text{wp}(S^{-1}, B): & ? \\ \text{c)} & B: x < c \\ & S: y := y - 1; z := x + y; \\ & S^{-1}: \text{empty statement, "skip," or } x := x; \\ \text{wp}(S^{-1}, B): & x < c \\ \text{d)} & B: x > a \\ & S: y := y + 2; w := w + 1; x := x + y; \\ & S^{-1}: x := x - y; \\ \text{wp}(S^{-1}, B): & x > a + y. \end{array}$$

$$\text{if } \vdash (Q \wedge B)\{S\}Q \text{ and } \vdash \text{wp}(S; S^{-1}, Q \wedge B) \equiv (Q \wedge B) \text{ then } \vdash (Q \wedge B)\{\text{repeat } S \text{ until } \sim B\}\text{wp}(S^{-1}, Q \wedge B) \wedge \sim B \wedge Q. \quad (8)$$

<sup>2</sup>As defined by Dijkstra [3], the weakest precondition of S with respect to R is the condition that defines the largest set of initial state of S for which S terminates and R is true at the output.

Note that  $S^{-1}$  does not exist if S assigns a constant value to a variable in B as exemplified by case b) above. The problem of finding  $S^{-1}$  becomes more involved whenever S contains a conditional statement or a repetitive statement. We shall comment on this problem further in the next section.

We shall now consolidate (2) with other known results to formulate more useful verification rules. First, we observe that  $\sim B$  is true when the statement "repeat S until  $\sim B$ " terminates. Therefore, the postcondition in (2) can be strengthened as follows:

$$\text{if } \vdash \text{wp}(S; S^{-1}, B) \equiv B \text{ then } \vdash B\{\text{repeat } S \text{ until } \sim B\}\text{wp}(S^{-1}, B) \wedge \sim B. \quad (3)$$

Next, we shall combine the present result with the invariant-relation theorem. The "repeat S until  $\sim B$ " version of this theorem (see, e.g., [9]) can be stated as follows:

$$\text{if } \vdash P\{S\}Q \text{ and } \vdash (Q \wedge B)\{S\}Q \text{ then } \vdash P\{\text{repeat } S \text{ until } \sim B\}(\sim B \wedge Q). \quad (4)$$

Here Q is the loop predicate. Combining (3) and (4) we obtain the following verification rule:

$$\text{if } \vdash P\{S\}Q \text{ and } \vdash (Q \wedge B)\{S\}Q \text{ and } \vdash \text{wp}(S; S^{-1}, B) \equiv B \text{ then } \vdash (P \wedge B)\{\text{repeat } S \text{ until } \sim B\}\text{wp}(S^{-1}, B) \wedge \sim B \wedge Q. \quad (5)$$

Note that the above verification rule can not be applied unless B is the precondition of the "repeat" statement, and, therefore, we can replace  $\vdash P\{S\}Q$  in the antecedent by  $\vdash (P \wedge B)\{S\}Q$ . This will not affect the validity of (5) because any predicate Q having the property that  $\vdash (P \wedge B)\{S\}Q$  and  $\vdash (Q \wedge B)\{S\}Q$  will be true when the "repeat" statement terminates, provided  $P \wedge B$  is true initially. However, this may, in theory at least, increase the applicability of this rule because we strengthen the precondition from P to  $P \wedge B$ . Thus, we can write

It is possible that P is identical to Q. In that case, (6) becomes

$$\text{if } \vdash (Q \wedge B)\{S\}Q \text{ and } \vdash \text{wp}(S; S^{-1}, B) \equiv B \text{ then } \vdash (Q \wedge B)\{\text{repeat } S \text{ until } \sim B\}\text{wp}(S^{-1}, B) \wedge \sim B \wedge Q. \quad (7)$$

Recall that (2) was established based on the observation that B is true before S is executed for the last time. Now if Q is a loop invariant and is also true before the "repeat" statement is executed then it is also true before S is executed for the last time. Therefore, we can replace predicate B in  $\text{wp}(S; S^{-1}, B) \equiv B$  and  $\text{wp}(S^{-1}, B)$  by  $Q \wedge B$ , and (7) becomes

In summary, we have established a new verification rule (2). This result is combined with other known results to form additional verification rules, viz. (3) and (5)-(8). We shall demonstrate the utility of these rules in the next section.

## III. APPLICATIONS

We shall assume that programs are written in a structured ALGOL-like language that includes the following constructs:

1) assignment statements:  $x := e$ ;

INTDIV: **begin**  $q := 0; r := x$ ;  
**while**  $r \geq y$  **do begin**  $r := r - y; q := q + 1$  **end end.**

2) conditional statements: **if**  $B$  **then**  $S$  **else**  $S'$ ; (note:  $S$  and  $S'$  are statements, and  $B$  is a predicate; the "else" clause may be omitted);

3) repetitive statements: **while**  $B$  **do**  $S$ ; or, **repeat**  $S$  **until**  $B$ .

To prove that a program  $S$  is consistent (partially correct) with respect to an input predicate  $Q$  and an output predicate  $R$  is to show that  $Q\{S\}R$  is a theorem. This can be accomplished by using the following result [1]:

if  $\vdash Q \rightarrow wp(S, R)$  then  $\vdash Q\{S\}R$  (9)

whenever  $wp(S, R)$  can be readily computed. It may become difficult to compute if  $S$  contains a repetitive construct. In that case we shall use the verification rules established in the preceding section to obtain the proof.

For convenience, we shall use  $Q\{S_0\}P\{S_1\}R$  as the shorthand notation for  $Q\{S_0\}P \wedge P\{S_1\}R$ . It can be shown that the following relation holds:

$$Q\{S_0; S_1\}R \equiv (Q \wedge wp(S_0, P))\{S_0\}P\{S_1\}R \vee (Q \wedge wp(S_0, \sim P))\{S_0\}\sim P\{S_1\}R. \quad (10)$$

Note that, by (9),  $(Q \wedge wp(S_0, P))\{S_0\}P$  and  $(Q \wedge wp(S_0, \sim P))\{S_0\}\sim P$  are always true. Also note that, by the basic verification rule (see, e.g., [7], [10]),  $(Q \wedge wp(S_0, P))\{S_0\}P\{S_1\}R$  implies  $(Q \wedge wp(S_0, P))\{S_0; S_1\}R$  and  $(Q \wedge wp(S_0, \sim P))\{S_0\}\sim P\{S_1\}R$  implies  $(Q \wedge wp(S_0, \sim P))\{S_0; S_1\}R$ . Since  $Q1\{S\}R1 \vee Q2\{S\}R2 \equiv (Q1 \vee Q2)\{S\}(R1 \vee R2)$ , and  $wp(S_0, P) \vee wp(S_0, \sim P) \equiv T$ , the truth of (10) immediately follows.

We can use (10) to "decompose"  $Q\{S_0; S_1\}R$  into two component propositions. Each component has a precondition that is more restricted than that of the original proposition. In fact, the precondition may become so restricted that it is always false (i.e., becomes a contradiction). In that case, we say that component is *trivial* because it is trivially a theorem, and, thus, can be disregarded in the process.

Since  $(Q \wedge wp(S_0, P))\{S_0\}P$  and  $(Q \wedge wp(S_0, \sim P))\{S_0\}\sim P$  are always true, we need only to show that  $P\{S_1\}R$  and  $\sim P\{S_1\}R$  in order to verify  $Q\{S_0; S_1\}R$ . This can be utilized to facilitate the verification process as follows. Suppose  $P$  implies that  $S_1$  is equivalent to some sequence  $S'_1$  of statements. In that case,  $P\{S_1\}R \equiv P\{S'_1\}R$ . Thus, if  $S'_1$  is simpler than  $S_1$ , we can simplify the problem by verifying  $P\{S'_1\}R$  instead. In particular, if  $S_1$  begins with a conditional statement or a "while" statement, we have the following relations:

$$\begin{aligned} \sim P\{\text{if } P \text{ then } S \text{ else } S'; \sigma\}R &\equiv \sim P\{S'; \sigma\}R \\ P\{\text{if } P \text{ then } S \text{ else } S'; \sigma\}R &\equiv P\{S; \sigma\}R \\ \sim P\{\text{while } P \text{ do } S; \sigma\}R &\equiv \sim P\{\sigma\}R \\ P\{\text{while } P \text{ do } S; \sigma\}R &\equiv P\{\text{repeat } S \text{ until } \sim P; \sigma\}R. \end{aligned} \quad (11)$$

Note that the first three relations can be used to reduce the complexity of expressions that we have to deal with, and the

last relation can be used to make the verification rules established in Section II applicable.

To illustrate, let us consider the following program for performing integer division:

Suppose we wish to verify that program INTDIV is consistent with respect to input predicate:  $x \geq 0 \wedge y > 0$  and output predicate  $x = r + q * y \wedge r < y \wedge r > 0$ . This can be accomplished by showing that

$$(x \geq 0 \wedge y > 0)\{\text{INTDIV}\}(x = r + q * y \wedge r < y \wedge r \geq 0) \quad (12)$$

is a theorem. Since this program contains a "while" statement we can decompose (12) into two components by inserting assertions  $r \geq y$  and  $r < y$  before the "while" statement as prescribed by (10), and then use (11) to simplify the resulting expressions. As the result, we obtain two components of (12) shown below (note: here and elsewhere we shall list a proposition of the form " $P_0\{S_0\}P_1\{S_1\}P_2 \cdots P_{n-1}\{S_{n-1}\}P_n$ " vertically, omit the curly brackets around program segments, and italicize predicates):

$$\begin{aligned} x \geq 0 \wedge y > 0 \wedge x < y, \\ q := 0; r := x; \\ r < y, \\ x = r + q * y \wedge r < y \wedge r \geq 0, \end{aligned} \quad (13)$$

and

$$\begin{aligned} x \geq 0 \wedge y > 0 \wedge x \geq y, \\ q := 0; r := x; \\ r \geq y, \\ \text{repeat } r := r - y; q := q + 1 \text{ until } r < y; \\ x = r + q * y \wedge r < y \wedge r \geq 0. \end{aligned} \quad (14)$$

The first component (13) can be readily verified as a theorem by using (9). The second component (14) can be verified by applying one of the verification rules established in Section II to the "repeat" statement. In this case,  $B$  is  $r \geq y$  and  $S$  is  $r := r - y; q := q + 1$ . If we let  $S^{-1}$  be  $r := r + y$  then  $wp(S; S^{-1}, B) \equiv B$ , and, thus,  $wp(S^{-1}, B) \equiv (r \geq 0)$ . Since  $wp(S^{-1}, B) \wedge \sim B$  is  $r \geq 0 \wedge r < y$ , we may be able to apply either rule (5), (6), or (7) if  $Q$  is  $x := r + q * y$ . As a rule, we shall first try to apply (7) because it does not require a search for predicate  $P$ . We need to show that  $(Q \wedge B)\{S\}Q$ , i.e.,

$$\begin{aligned} x = r + q * y \wedge r \geq y, \\ r := r - y; q := q + 1; \\ x = r + q * y, \end{aligned}$$

is a theorem, which is obviously the case. We also need to show that  $Q \wedge B$  is true before the "repeat" statement is executed. However, because of the way we construct (14),  $B$  has been guaranteed to be true. Therefore, we only need to show that  $Q$  is true. This can be accomplished by showing that the following is a theorem:

$$\begin{aligned} x \geq 0 \wedge y > 0 \wedge x \geq y, \\ q := 0; r := x; \\ x = r + q * y. \end{aligned}$$

The reader can easily verify that this is indeed the case. Now, since both components of (12) are a theorem, (12) is also a theorem, and, hence, program INTDIV is consistent.

It must be pointed out at this juncture that the postcondition of a "repeat" statement will not be always perfectly of the form:  $wp(S^{-1}, B) \wedge \sim B \wedge Q$ . Therefore, in applying the verification rules, the task of identifying a condition that corresponds to  $Q$  may not be as straightforward as illustrated in this example. In general, some deliberations may be required to find a predicate  $Q$  such that  $wp(S^{-1}, B) \wedge \sim B \wedge Q$  is logically equivalent to the desired postcondition.

Having explained how the verification rules can be applied by using a simple example, we shall now demonstrate their utility by using an example program which is identified as being difficult to prove in [7]. Consider the following program for computing the McCarthy's 91 function:

```

MC91F: begin y := 1;
        while x ≤ 100 do begin x := x + 11; y := y + 1 end;
        while y ≠ 1 do begin x := x - 10; y := y - 1;
            while x ≤ 100 do begin x := x + 11;
                y := y + 1
            end
        end;
        z := x - 10;
    end.

```

The specification of this program can be stated as follows:

if  $x \leq 100$  then  $z = 91$  else  $z = x - 10$ .

Thus, to prove that this program is consistent is to verify that

$$(x \leq 100)\{MC91F\}(z = 91) \quad (15)$$

and

$$(x > 100)\{MC91F\}(z = x - 10) \quad (16)$$

are both a theorem.

We shall begin by inserting assertions  $x \leq 100$  and  $x > 100$  before the first "while" statement to decompose (15) into two components as prescribed by (10). Since both components contain a "while" statement, we can repeat the process to obtain four components for (15), however, only one of which is nontrivial and is listed below;

```

x ≤ 100,
y := 1;
x ≤ 100,
repeat x := x + 11; y := y + 1 until x > 100;
y ≠ 1,
repeat x := x - 10; y := y - 1;
    while x ≤ 100 do begin x := x + 11; y := y + 1 end
until y = 1;
z := x - 10;
z = 91.

```

Next, we observe that (3) can be applied to the first "repeat" statement to obtain a postcondition concerning the values of  $x$ . In this case  $B$  is  $x \leq 100$  and  $S$  is  $x := x + 11; y := y + 1$ . If we let  $S^{-1}$  be  $x := x - 11$  then  $wp(S; S^{-1}, B) \equiv B$ , and, thus,  $wp(S^{-1}, B) \equiv (x \leq 111)$ . Hence, by (3), the following proposition is a theorem:

$$\begin{aligned} &x \leq 100, \\ &\text{repeat } x := x + 11; y := y + 1 \text{ until } x > 100; \\ &x \leq 111 \wedge x > 100. \end{aligned} \quad (18)$$

Note that the postcondition of this proposition becomes part of the precondition of the second "repeat" statement in (17). Furthermore, by inspection, we can see that the desired postcondition of this statement is  $y = 1 \wedge x = 101$ . Thus, the remaining task is to show that the following proposition is a theorem:

$$\begin{aligned} &x \leq 111 \wedge x > 100 \wedge y \neq 1, \\ &\text{repeat } x := x - 10; y := y - 1; \\ &\quad \text{while } x \leq 100 \text{ do begin } x := x + 11; y := y + 1 \text{ end} \\ &\text{until } y = 1; \\ &y = 1 \wedge x = 101. \end{aligned} \quad (19)$$

This can be accomplished by an application of (8). In this case,  $B$  is  $y \neq 1$ ,  $Q$  is  $x \leq 111 \wedge x > 100$ , and  $S$  is  $x := x - 10; y := y - 1$ ; while  $x \leq 100$  do begin  $x := x + 11; y := y + 1$  end.

To apply (8), first we need to show that  $(Q \wedge B)\{S\}Q$  is a theorem.  $(Q \wedge B)\{S\}Q$  can be decomposed into two components shown below:

$$\begin{aligned} &x \leq 111 \wedge x > 100 \wedge y \neq 1 \wedge x > 110, \\ &x := x - 10; y := y - 1; \\ &x > 100, \\ &x \leq 111 \wedge x > 100, \end{aligned}$$

and

$$\begin{aligned} &x \leq 111 \wedge x > 100 \wedge y \neq 1 \wedge x \leq 110, \\ &x := x - 10; y := y - 1; \\ &x \leq 100, \\ &\text{repeat } x := x + 11; y := y + 1 \text{ until } x > 100; \\ &x \leq 111 \wedge x > 100. \end{aligned}$$

The first component can be readily verified by using (9). The second component is obviously a theorem because the last three lines are identical to (18), which has been proved as a theorem previously. Thus, we have shown that  $x \leq 111 \wedge x > 100$  is a loop invariant.

Next, we need to find  $S^{-1}$  such that  $wp(S; S^{-1}, Q \wedge B) \equiv (Q \wedge B)$ . In general, the problem becomes very difficult when  $S$  contains a repetitive statement. However, we might be able to simplify the problem by making use of the fact that  $Q \wedge B$  is true before  $S$  is executed. If  $Q \wedge B$  implies that  $S$  is equivalent to some simpler program segment  $S'$ , we can replace  $S$  by  $S'$  in the process. In this case,  $Q$  can be expressed as  $x = 111 \vee x < 111 \wedge x > 100$ . If  $Q$  is  $x = 111$ ,  $S$  can be simplified to  $x := x - 10; y := y - 1$ . Thus,  $S^{-1}$  will be  $x := x + 10; y := y + 1$ , and  $wp(S^{-1}, Q \wedge B)$  will be  $x =$

$101 \wedge y \neq 0$ . If  $Q$  is  $x < 111 \wedge x > 100$  then  $S$  is equivalent to a single statement:  $x := x + 1$ . Thus,  $S^{-1}$  will be  $x := x - 1$ , and  $\text{wp}(S^{-1}, Q \wedge B)$  will be  $x < 112 \wedge x > 101 \wedge y \neq 1$ . Combining both cases, we obtain

$$\text{wp}(S^{-1}, Q \wedge B) \equiv x = 101 \wedge y \neq 0 \vee x < 112 \wedge x > 101 \wedge y \neq 1.$$

Now, by (8), we can conclude that (19) is a theorem because

$$\text{wp}(S^{-1}, Q \wedge B) \wedge \sim B \wedge Q \equiv x = 101 \wedge y = 1.$$

Hence, we have shown that (15) is a theorem.

To complete the verification of MC91F, we need to prove that (16) is a theorem. This proposition can be similarly decomposed into four components. However, it turns out that only one of these is nontrivial, and is listed below:

$x > 100,$   
 $y := 1;$   
 $x > 100,$   
 $z := x - 10;$   
 $z = x - 10.$

The truth of this proposition is obvious. Thus, we have shown that MC91F is consistent.

In summary, we have described a method for decomposing a proposition of the form  $Q\{R\}S$  into components so that the verification rules established in the preceding section can be readily applied. The utility of those rules are then demonstrated by using examples.

#### REFERENCES

- [1] S. K. Basu and R. T. Yeh, "Strong verification of programs," *IEEE Trans. Software Eng.*, vol. SE-1, pp. 339-345, Sept. 1975.
- [2] R. Conway and D. Gries, *An Introduction to Programming*. Cambridge, MA: Winthrop, 1975, pp. 322-324.
- [3] E. W. Dijkstra, "Guarded commands, nondeterminacy and formal derivation of programs," *Commun. ACM*, vol. 18, pp. 453-457, Aug. 1975.
- [4] S. L. Gerhart, "Knowledge about programs: A model and case study," Dep. Comput. Sci., Duke Univ., Durham, NC, Tech. Rep. CS-1, 1975.
- [5] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, pp. 576-580, Oct. 1969.
- [6] R. L. London, "Perspectives on program verification," in *Current Trends in Programming Methodology, Vol. II: Program Validation*, R. T. Yeh, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1977, pp. 151-172.
- [7] Z. Manna, *Mathematical Theory of Computation*. New York: McGraw-Hill, 1974.
- [8] J. H. Morris, Jr. and B. Wegbreit, "Program verification by subgoal induction," in *Current Trends in Programming Methodology, Vol. II: Program Validation*, R. T. Yeh, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1977, pp. 197-227.
- [9] N. Wirth, *Systematic Programming: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1973.
- [10] R. T. Yeh, Ed., *Current Trends in Programming Methodology, Vol. II: Program Validation*. Englewood Cliffs, NJ: Prentice-Hall, 1977.



J. C. Huang (S'61-M'62) received the M.S. degree from Kansas State University, Manhattan, in 1962, and the Ph.D. degree from University of Pennsylvania, Philadelphia, in 1969.

From 1962 to 1966 he was an Engineer at the Western Electric Company, Allentown, PA, where he worked on the design and development of high-speed automatic test equipment for integrated circuits and thin-film devices. In 1969, he joined the faculty of the Department of Computer Science, University of Houston, Houston, TX, where he is currently an Associate Professor. He has done consulting and has numerous publications on the subject of program analysis and testing. His current research interests include software testing, programming methodology, and architecture of real-time systems.

Dr. Huang is a member of the Association for Computing Machinery, Phi Kappa Phi, and Sigma Xi. He served on the Program Committee of the 1978 Workshop on Software Testing and Test Documentation, and was the Conference Chairman and the Program Committee Chairman of the Seventh Texas Conference on Computing Systems.