VERIFICATION OF
NONDETERMINISTIC PROGRAMS

by

Raymond T. Yeh

TR-56 (revised) June, 1977
(original) May, 1976
I. INTRODUCTION

In this chapter we present a methodology for the verification of both deterministic and nondeterministic programs. Our method is based on the concept of predicate transformer introduced by Dijkstra [1975]. Predicate transformer is used by Dijkstra to define formal semantics of programs by a mapping which transforms a set of states after the execution of a program to the set of all possible states before the execution of the same program. Thus, the difference between the concepts of determinism and nondeterminism loses its significance in this semantic context. The focus is on the nature of the computation, and hence the concept of iteration, and not how the program iterates, is of concern in the current context.

By the very nature of the semantic definition using predicate transformer, it is implicitly assumed in our method that the termination property of a program is an inherent property of the algorithm that realizes the given program. Furthermore, the
termination and consistency can be handled by the same modus operandi. These points will be clarified later in the paper. We also show that proof of program "incorrectness" (with respect to its specification) in this framework needs no special treatment and is symmetric to a proof of correctness.

In section II, we introduce nondeterministic program constructs a la Dijkstra. As an example, we have demonstrated the simplicity of writing a nondeterministic program for the eight queens problem.

In section III, the concept of a predicate transformer and its properties are introduced.

In section IV, we give an axiomatization of semantics of program constructs based on the notion of a predicate transformer.

In section V, the concept of termination, consistency and correctness are introduced and a method for their verification, based on the semantic definitions given in section IV is given. This approach allows the proof of both the consistency and termination to be handled simultaneously. Several examples are given to illustrate this approach.

In section VI and VII, we explore the approach of characterizing the loop construct by a recursive equation and its implications in verification. The notion of output-adequate loop invariant is introduced, and its relation to fixpoints of the recursive equation explored.

II. NONDETERMINISTIC PROGRAM CONSTRUCTS

In this section, we will briefly review two constructs introduced in [Dijkstra, 1975] for nondeterministic programs. They are given in terms of an extended BNF grammar with the convention that braces "{...}"
should be interpreted as "followed by zero or more instances of the enclosed."

<guarded command>::=<guard>→<guarded list>
<guard>::=<boolean expression>
<guarded list>::=<statement>{;<statement>}
<guarded command set>::=<guarded command>{[]}<guarded command>
<alternative construct>::=if<guarded command set>fi
<repetitive construct>::=do<guarded command set>od
<statement>::=<alternative construct>|<repetitive construct>

"other statements"

where "other statements" means assignment statements or procedure calls.

The semicolons in the guarded list have the usual meaning: when the guarded list is selected for execution its statements will be executed successively in the order from left to right; a guard list will only be selected for execution in a state such that its guard is true. If the guarded command set consists of more than one guarded command, they are mutually separated by the separator "[]"; the order in which the guarded commands of a set appear in the text is semantically irrelevant.

Note that for the alternative construct "if ... fi", if either none of the guards were true in the initial state, or the guarded command set is empty, the program will "abort". Otherwise, an arbitrary guarded list with a true guard will be selected for execution.

To illustrate this construct and the nondeterminancy such a construct can bring about, we use the original example in [Dijkstra, 1975].
Example 1 - A program that for fixed x and y assigns to m the maximum value of x and y.

\[ \text{if } x \geq y \text{ then } m := x \]
\[ \text{else } y \geq x \text{ then } m := y \]
\[ \text{fi} \]

In order to define the semantics of alternative construct more easily later, we shall use the following abbreviations.

Let \( \text{IF} \) denote
\[ \text{if } B_1 \rightarrow S_1 \quad \ldots \quad B_n \rightarrow S_n \text{ fi} \]

Let \( \text{BB} \) denote
\[ B_1 \lor B_2 \lor \ldots \lor B_n \]

where each \( B_i \) is a Boolean expression and each \( S_i \) is a program statement.

There are two special cases for the alternative constructs which specialize to the usual deterministic constructs.

Case 1. IF denotes "\( \text{if } B \rightarrow S_1 \lor \overline{B} \rightarrow S_2 \text{ fi} \)". In this case, the IF construct corresponds to the usual "if \( B \) then \( S_1 \) else \( S_2 \)". We will show in a later section that these two constructs are indeed "semantically equivalent".

Case 2. IF denotes "\( \text{if } B \rightarrow S_1 \lor \overline{B} \rightarrow \text{skip fi} \)". In this case, the construct IF is equivalent to the "deterministic" construct "IF \( B \) then \( S_1 \)".

For the repetitive construct "\( \text{do } \ldots \text{ od} \)" in the case either none of the guards were true in the initial state, or the guarded comment set is empty, the statement is semantically equivalent to "skip". We will use the abbreviation DO for
\[ \text{do } B_1 \rightarrow S_1 \quad \ldots \quad B_n \rightarrow S_n \text{ od} \]
A special case to be observed here is that the construct "do B → S \_1 od" is "semantically equivalent" to the usual deterministic construct "while B do S \_1".

Example 2 - The following nondeterministic program is to find a placement of eight queens on a chessboard such that no two queens occupy the same row, column, or diagonal.

The program uses four vectors:

- ROW with integral indices in the range \{1, \ldots, 8\}
- COL with integral indices in the range \{1, \ldots, 8\}
- LD with integral indices in the range \{-7, \ldots, 7\}
- RD with integral indices in the range \{2, \ldots, 16\}

representing rows, columns, left diagonals, and right diagonals, respectively. Each vector element represents a count of the number of queens in that row, column, etc. Square (i,j) of the board is in row i, column j, left diagonal i-j, and right diagonal i+j.

The nondeterministic program is:

```
ROW:=COL:=LD:=RD:=0;

  do  ROW_1:=1+if  ROW_1+1:=1  [ ... ]
      ROW_8:=1+I:=8  fi;

      if  COL_1:=1  [ ... ]
      COL_8:=1+J:=8  fi;

      if  RD_I+J<=1 and LD_I-J>=1+
          ROW_I:=COL_J:=RD_I+J:=LD_I-J:=1
      [ true → "skip" fi
  od
```
Where if A and B are vectors, then \( A = 1 \) means that for each \( i \), the \( i \)th component \( A_i = 1 \); \( A \leq B \) means that for each \( i \), \( A_i \leq B_i \).

We note that the program iteratively places a queen on the board in a position that does not conflict with previous choices. Depending on previous choices, this may or may not be possible. If it so happens that this is possible for all eight queens the program terminates successfully. However, if for some queen this is not possible, the program will continue to search without ever terminating.

III. THE CONCEPT OF A PREDICATE TRANSFORMER

In this section, we provide the basic concept and some properties of a predicate transformer which will be used for the verification of programs in later sections.

We shall use symbols \( P, Q \) and \( R \) to denote predicates defined on the state space of a program. We use symbol \( T \) to denote the predicate which is satisfied by all states and \( F \) to denote the predicate which is satisfied by no state at all. For each predicate \( P \), we denote by \( P^* \) the set of states satisfying \( P \).

Let \( S \) be a given program. A specification of \( S \) is meant here to be a pair of predicates \( (P, Q) \) such that \( P \) is an input assertion
characterizing all legal initial states of S, and Q is an output assertion characterizing all desirable final states of S. Examples of such specifications include:

<table>
<thead>
<tr>
<th>P</th>
<th>S</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x &gt; 0$</td>
<td>$x: x + 2$</td>
<td>$x &gt; 2$</td>
</tr>
<tr>
<td>a list ${x_1, \ldots, x_n}$</td>
<td>A sort program</td>
<td>a list ${x_1', \ldots, x_n'}$ of ascending order</td>
</tr>
</tbody>
</table>

We now introduce the notation $P[S]Q$ to represent the following proposition:

"If the assertion $P$ is true (on certain initial state vector) and the program $S$ is initiated with this state vector then $S$ terminates and the assertion $Q$ is true on the final state vector." Predicates $P$ and $Q$ in $P[S]Q$ are referred to as the pre- and
post condition of S.

An example to illustrate the previously introduced notation is

\[ x = 1 \ [x: = x + 2] \ x = 3 \]

Note that

\[ x > 0 \ [\text{do } x \leftarrow 0 \ \rightarrow x: = x + 1 \ \text{od}] Q \]

is not valid for any predicate Q since the given program will not terminate.

We will use the notation WP(S, Q) to denote the \textit{weakest} precondition for all the initial states of S such that activation of S is guaranteed to lead to a properly terminating activity with the final state of the S satisfying the postcondition Q. The function WP is called a \textit{predicate transformer} since it transforms a postcondition R into a precondition WP(S, R). Alternatively, WP(S, Q)* is the largest set of initial state of S for which S terminates and Q is true at output. If we denote by \( f_S \) the function to be computed by a given program S, then we see that WP(S, Q)* = \{ x \mid f_S(x) \in Q* \}.

We proceed to list a set of properties of the "predicate transformer" for given S and Q. These properties were given in [Dijkstra, 1975] and is included here for the sake of completeness. The proofs for these properties are fairly straightforward by observing that WP(S, Q)* = \( f_S^{-1}(Q*) \).

Theorem 1: The following identities and implications hold.

1) WP(S, F) \equiv F;
2) [P \leftarrow Q] \rightarrow [WP(S, P) \rightarrow WP(S, Q)];
3) WP(S, P) \lor WP(S, Q) \rightarrow WP(S, P \lor Q)
4) WP(S, P) \land WP(S, Q) \equiv WP(S, P \land Q)
It should be noted that the converse of condition (3) in the previous theorem holds in the deterministic case.

IV. A DEDUCTIVE SYSTEM FOR PROGRAM PROOFS

In order to prove that a program is what its creator intended, it is necessary to provide an axiomatic definition as formal semantics for program constructs as well as rules of inference for the semantics of program as a whole.

Let Q be an arbitrary postcondition, the semantic definitions will be given in terms of the weakest precondition of the given construct with respect to Q. Thus, two programs $S_1$ and $S_2$ are said to be **semantically equivalent** if for any given postcondition Q, $WP(S_1,Q) \equiv WP(S_2,Q)$; i.e., they have the same predicate transformer.

We now proceed to the axiomatic definitions for various program constructs.

1. **Primitive statements**: the semantics for the two primitive statements "skip" and "abort" are given in the following:

   \[
   WP("skip",Q) \equiv Q; \quad (1)
   \]

   \[
   WP("abort",Q) \equiv F \quad (2)
   \]

2. The **axiom of assignment**: the semantics of the assignment statement "x:=E" is defined by

   \[
   WP("x:=E",Q) \equiv Q^x_E \quad (3)
   \]

   where $Q^x_E$ is obtained from Q by substituting the expression E for every occurrence of the variable x.

   There are two examples in the following to illustrate this axiom.
<table>
<thead>
<tr>
<th>Program Statement S</th>
<th>Postcondition Q</th>
<th>WP(S,Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x := x + 1 )</td>
<td>( x = 3 )</td>
<td>( x = 2 )</td>
</tr>
<tr>
<td>( x := x \ast 3 )</td>
<td>( x^2 = 36 )</td>
<td>((x \ast 3)^2 = 36)</td>
</tr>
</tbody>
</table>

In the following, we shall provide rules to define the semantics of composite program constructs.

3. The axiom of composition: the semantics of the composite statement "\( S_1; S_2 \)" is defined by

\[
WP("S_1; S_2", Q) \equiv WP(S_1, WP(S_2, Q)) \tag{4}
\]

Equation (4) expresses that the effect of the composition of two transformer is that of applying the transformer of \( S_1 \) to the predicate obtained by applying the transformer of \( S_2 \) to the original predicate \( Q \).

Example 3: Consider the following program segment \( S: S_1; S_2; S_3; \)

\( S_1: y := y + 1; \)
\( S_2: x := y + x; \)
\( S_3: \text{IF } x < 0 \text{ THEN } x := -x \text{ ELSE } x := x' \)

By (4), for any postcondition \( Q(x,y) \), we have

\[
WP(S,Q(x,y)) \equiv WP(S_1, WP(S_2, WP(S_3, Q(x,y))))
\]

\[
\equiv WP(S_1, WP(x := y + x, [x < 0 \land Q(-x,y) \lor x \geq 0 \land Q(x,y)]))
\]

\[
\equiv WP(S_1, [y + x < 0 \land Q(-(x+y),y)] \lor [y + x \geq 0 \land Q(x + y, y)])
\]

\[
\equiv [x + y + 1 < 0 \land Q(-(x + y + 1), y + 1)] \lor [y + x + 1 \geq 0 \land Q(x + y + 1, y + 1)].
\]
4. The **axiom of alternative construct**: the semantics for the statement "IF" is defined by

\[
WP(\text{IF}, Q) \equiv \text{BB}\{B\downarrow \Rightarrow WP(S_{1}, Q)\} \land \ldots
\]

\[
\land [B \uparrow \Rightarrow WP(S_{n}, Q)].
\]

(5)

**Example 4** — Consider the program that for fixed integers 

x and y, assigns to m the maximum value of x or y.

\[
\begin{align*}
\text{if } & \ x \geq y \Rightarrow m := x \\
\text{if } & \ y \geq x \Rightarrow m := y \\
\text{fi}
\end{align*}
\]

With respect to the specifications \( I \equiv [x \geq 0 \land y \geq 0], \) and 

\( Q \equiv [m = \max(x,y)], \) it is easily seen that 

\[
WP(\text{IF}, Q) \equiv [x \geq y \Rightarrow x = \max(x,y)] \land [y \geq x \Rightarrow y = \max(x,y)].
\]

There are two special cases for the alternative construct in the deterministic case.

**Case i)** IF denotes

\[
\text{if } B \Rightarrow S_{1} \mid \overline{B} \Rightarrow S_{2} \text{ fi}
\]

Using (5) and algebraic simplification, we can derive the following:

\[
WP(\text{"if } B \Rightarrow S_{1} \mid \overline{B} \Rightarrow S_{2} \text{ fi"}, Q) \equiv [B \land WP(S_{1}, Q)] \lor [\overline{B} \land WP(S_{2}, Q)]
\]

(6)

**Case ii)** IF denotes

\[
\text{if } B \Rightarrow S_{1} \mid \overline{B} \Rightarrow \text{skip} \text{ fi}
\]

It is easily seen that in this case,

\[
WP(\text{"if } B \Rightarrow S_{1} \mid \overline{B} \Rightarrow \text{skip"}, Q) \equiv [B \land WP(S_{1}, Q)] \lor [\overline{B} \land Q]
\]

(7)

From [Basu and Yeh, 1975], it is easily seen that special cases of the IF constructs are semantically equivalent to the constructs
"IF $B_1$ THEN $S_1$ ELSE $S_2$" and "If $B$ THEN $S_1$", respectively.

Example 5 - Consider the program segment $S$: IF $x < 0$ THEN $x := -x$ ELSE $x := x$;

For any postcondition $Q$, we have

$$WP(S, Q(x)) \equiv [x < 0 \land Q(\neg x)] \lor [x \geq 0 \land Q(x)]$$

Where $Q(x)$ is a predicate with variable $x$.

5. The axiom of repetitive construct: the semantics for the statement "DO" is defined by

$$WP(\text{DO}, Q) \equiv (\exists k \geq 0 : H_k(Q)) \quad (8)$$

where

$$H_0(Q) \equiv \overline{BB} \land Q; \quad \text{and}$$

$$\forall k > 0, H_k(Q) \equiv WP(IF, H_{k-1}(Q)) \lor H_0(Q) \quad (10)$$

Thus, $H_k(Q)$ is interpreted as the weakest precondition guaranteeing termination after at most $k$ selections of a guard list, leaving the system in a final state satisfying $Q$.

Note that in the deterministic case, equation (8) is reduced to the following (due to the equality of (3) in theorem 1 in this case).

Corollary 1: (Basu and Yeh) Let $S$ be the statement "While $B$ do $S_1$" then for any postcondition $Q$, we have

$$WP(S, Q) \equiv [\exists j: j \geq 0 : A_S^j(Q)] \quad (11)$$

where

$$A_S^0(Q) \equiv \overline{B} \land Q:$$

$$A_S^j + 1(Q) \equiv BWP(S_1, A_S^j(Q)), \text{ for } j \geq 0.$$
It follows from (8) and (11) that the two constructs
"do B → S₁ od" and "while B do S₁" are semantically equivalent.

It should be noted that definitions (8) and (11) are identical in the deterministic case but different in the nondeterministic case as illustrated by the following example due to Dijkstra (private communication).

\[
\text{DO: } \text{do } x > 0 \rightarrow x := x - 1 \quad \text{if } x > 9 \rightarrow x := x - 10 \text{ od}
\]

The reason for this is that equation (3) in theorem 1 is an identity in the deterministic case, but not in the nondeterministic case.

Note that equations (9) and (10) provide a way of obtaining WP(DO,Q) in that successive terms of \( H_\alpha(Q) \) can be computed until a suitable candidate for WP(DO,Q) is found. While this technique is not algorithmic, it is similar to the sequence extrapolation technique familiar in mathematics. We will now illustrate the previous definitions by the following examples.

**Example 6**—Consider the program
\[
\text{do } x \uparrow 0 \rightarrow x := x = 1 \text{ od}.
\]
With respect to a postcondition $Q \equiv (x = -1)$, we see that by the procedure given in theorem 3,

$$H_1 \equiv F, \ i \geq 0$$

Hence, $WP(\text{DO}, x = -1) \equiv F$.

For postcondition $Q' \equiv (x = 0)$, we have

$$H_0 \equiv x = 0$$
$$H_1 \equiv x = 1$$

To find the general term $H_j$, assume that for $j > 0$

$$H_j \equiv x = j$$

We find that

$$H_{j+1} \equiv WP(\text{IF}, H_j)$$

$$\equiv x = j+1$$

Hence, we have

$$WP(\text{DO}, Q') \equiv [\exists j: j \geq 0: x = j] \ ||$$

**Example 7** - Let $S'$ be the following program computing the quotient and remainder with a specification $[I, Q]$, where $I \equiv (x \geq 0 \land y > 0)$ and $Q \equiv (x \leq B < y \land x \geq 0 \land x = B + Ay)$ are the respective input and output assertions.

$$(x \geq 0 \land y > 0)$$

$S'$: \textbf{begin} $A := 0; \ B := x$

$S$: While $B \geq y$ do

\textbf{begin} $B := B - Y$

\textbf{begin} $A := A + 1$

\textbf{end}

\textbf{end}

$$(0 \leq B < y \land x \geq 0 \land x = B + Ay)$$

-14-
We apply the procedure given in corollary 2 to compute $\text{WP}(S', Q)$.

$$A^0_S(Q) \equiv 0 \leq B < y \land x \geq 0 \land x = B + Ay$$

In order to compute $\text{WP}(S, Q)$, we need to obtain the general term

$A^j_S(Q)$, for $j > 0$. This can be done by induction on $j$ with $A^0_S(Q)$
as the basis of the induction. Thus we obtain for $j > 0$,

$$A^j_S(Q) \equiv jy \leq B < (j+1)y \land x \geq 0 \land x = B + Ay$$

Hence, $\text{WP}(S, Q) \equiv (\exists n \geq 0) [ny \leq B < (n+1)y \land x \geq 0 \land x = B + Ay]$ and

$$\text{WP}(S', Q) \equiv (\exists n \geq 0) [ny \leq x < (n+1)y \land x > 0].$$

**Example 8** - Let $S'$ be the following program computing the exponential function $A^B$ of two integers with specifications $[I, Q]$, where $I \equiv [A > 0 \land B \geq 0]$ and $Q \equiv [z = A^B]$.

$$S': \ (A > 0 \land B \geq 0)$$

```
begin x: = A; y: = B; z: = 1;
S: While y \neq 0 do
begin if odd (y)
begin y: = y-1;
    z: = z\times x
end
    y: = \frac{y}{2}; x: = x\times x
end
(z = A^B)
```

-15-
Again, we shall compute a few successive terms using corollary 2
and then use induction to prove the expression of a general term
to finally obtain the expression for WP(S',Q) of the loop.

Clearly,
\[ A_S^0 (Q) \equiv [y = 0 \land z = A^B] \]

By (9), we have
\[ A_S^1 (Q) \equiv (y \not\equiv 0) \land [\{\text{even}(y) \land \frac{y}{2} = 0 \land z = A^B\} \lor \\
\{\text{odd } (y) \land \frac{y-1}{2} = 0 \land z^x = A^B\}] \]

There are two cases to be considered:

i) \[ \frac{y}{2} = 0 \equiv (y = 0 \lor y = 1) \]
This implies that \((\text{even}(y) \land \frac{y}{2} = 0) \equiv y = 0\)

ii) \[ \frac{y-1}{2} = 0 \equiv (y = 1 \lor y = 2) \]
This implies that \((\text{odd } (y) \land \frac{y-1}{2} = 0) \equiv y = 1\)

By (i) and (ii), we conclude that
\[ A_S^1 (Q) \equiv [y = 1 \land z^x = A^B] \]

Similarly, we obtain
\[ A_S^2 (Q) \equiv [y = 2 \land z^x^2 = A^B] \lor \\
[y = 3 \land z^x^3 = A^B] \]

and
\[ A_S^3 (Q) \equiv [y = 4 \land z^x^4 = A^B] \lor [y = 6 \land z^x^6 = A^B] \\
\lor [y = 5 \land z^x^5 = A^B] \lor [y = 7 \land z^x^7 = A^B] \]

By induction on \(j\), we obtain that for \(j > 0\),
\[ A_S^{j+1} (Q) \equiv \bigvee_{y=2}^{2j-1} [y \equiv 0 \land z^x^k = A^B] \]
Hence,
\[
WP(S, Q) \equiv (\exists n \geq 0)[2^n \leq y < 2^{n+1} \land z^y \times y = B^1]
\]
and,
\[
WP(S', Q) \equiv (\exists n \geq 0)[2^n \leq B < 2^{n+1}].
\]

We note that in both of the previous examples, the way we obtain \(WP(S, Q)\) is by the generation of a sequence of expression
\(A_S^j(Q)\) using corollary (7), and by "guessing" the general term
\(A_S^j(Q)\). If our guess is correct, then it can be proved by induction
on \(j\). Otherwise, more terms are generated and another guess is made.
It should be emphasized here that this technique is heuristic in
nature, and therefore, does not always provide a solution readily.
Applying this technique to programs of nested loops will immediately
expose the limitations of this technique. Some special cases of
generating sequences to approximate general terms have been studied
in [Grief and Waldinger, 1974]. However, nested loops remains the
biggest problem in this approach.

V. THE NOTIONS OF CONSISTENCY, TERMINATION, AND CORRECTNESS AND
THEIR VERIFICATION

In this section, we shall formulate the concepts of consistency,
termination, and correctness, and illustrate how to prove these
properties of programs utilizing the formal semantics defined in
terms of predicate transformer.

We first formulate the notion of termination using intuitive
arguments of Dijkstra. Consider a given program \(S\) and some post-
condition \(R\). Then each initial state \(S\) belongs to one of the following
disjoint sets according to the following three conditions:

1) $S$ will terminate after activation in a final state satisfying $Q$;

2) $S$ will terminate after activation in a final state not satisfying $Q$;

3) $S$ will not terminate.

The first two sets are characterized by $WP(S, Q)$ and $WP(S, \overline{Q})$. Since $WP(S, Q) \lor WP(S, \overline{Q}) \equiv WP(S, Q \overline{Q}) \equiv WP(S, T)$, the third set is characterized by $WP(S, T)$. Hence, termination is characterized by $WP(S, T)$.

The main goal of program verification is to develop techniques which can demonstrate formally that the logical behavior of a program is indeed what its creator had intended. In order to provide a formal proof of demonstration, it is necessary to express a programmer's intention as formal specifications, and that the logical behavior of a program in terms of the formal semantics of its constructs. The proof process then is to show that the semantics of the program as a whole satisfies its specifications. Note that while specification may include all kinds of information and properties of program, termination is not usually included.

We now introduce Hoare's notation. The symbols $P[S]Q$ have the meaning "if assertion $P$ is true on certain initial state, and the program is activated in this state, and if $S$ terminates, then the assertion $Q$ will be true on the final state.

We say a program $S$ is consistent (or partially correct) with
respect to a pair of input and output assertions $I$ and $Q$ if and only if $I[S]Q$.

Note that $P[S]Q$ always imply that $P\{S\}Q$, but the converse does not hold in general.

Finally, a program $S$ is said to be correct (or totally correct or strongly verifiable) with respect to a pair of input and output assertions $P$ and $Q$ if and only if $P[S]Q$.

The usual practice of program verification consists of two parts: 1) to prove consistency, and 2) to prove termination. Usually, the failure to prove consistency or termination does not lead to a proof of inconsistency (i.e. $P\{S\}Q$ is false) or nontermination (i.e., $WP(S,T)$ is true). Furthermore, most techniques handle consistency and termination proofs separately. It will be shown in the following that the verification method we propose here handles consistency and termination proof simultaneously, and that failure to prove correctness automatically proves incorrectness.

The following results is a straightforward from theorem 1 and the definition above.

**Theorem 4** - A program $S$ terminates with respect to an input assertion $I$ if and only if $I \rightarrow WP(S, T)$.

To illustrate the concept, let us consider example 7. It is easily seen that

$$WP(S', T) = (\exists n \geq 0) \ [ny \leq x < (n+1)y].$$

Since

$$x \geq 0 \ \land \ y > 0 \ \rightarrow (\exists n \geq 0) [ny \leq x < (n+1)y]$$
is clearly a basic property of the number theory, we conclude that $S$ terminates with respect to the input assertion.

**Theorem 5** - A program $S$ is strongly verifiable with respect to a specification $[I,Q]$ if and only if $I \rightarrow WP(S,Q)$.

It follows from theorem 5 that the problem of verifying $I[S]Q$ can be partitioned into the following subproblems.

1. Given $S$ and $Q$, obtain a candidate $R$ for $WP(S,Q)$;
2. Show that $R \equiv WP(S,Q)$;
3. Prove that $I \rightarrow R$.

To illustrate the concept of strong verification, we refer the reader back to examples 4 and 6. In example 4, we see that

$$I \equiv [x \geq 0 \land y \geq 0] \rightarrow [x \geq y \rightarrow x = \max(x,y)] \land [y \geq x + y = \max(x,y)] \equiv WP(IF,Q)$$

Hence, the program is strongly verifiable.

Similarly, one can easily show in example 6 that

$$x > 0 \rightarrow [\exists j : j \geq 0 : x = j]$$

is a tautology, and hence the program is strongly verifiable with respect to $[I,Q']$.

It should be observed here that when strong verification of a program $S$ cannot be achieved with respect to $[P,Q]$, a program is automatically incorrect with respect to $[P,Q]$ in the sense that there exists $x \in P^*$ such that either $S$ does not terminate with respect to input $x$, or that $S$ terminates but $f_S(x) \not\in Q^*$. Where $P^* = \{x | P(x)\}$.

The following results characterizes incorrectness and is a corollary of theorem 4.
Corollary 2 - A program S is incorrect with respect to a pair of input-output specification \([I,Q]\) if and only if \(I \rightarrow WP(S,Q)\) is false.

Referring again to example 6, we see that the program in example 6 is incorrect with respect to \([I,Q]\) for \(I \equiv x > 0\) since \(WP(D0,Q) \equiv F\) and \(I \rightarrow F\) is false.

Note that we can prove indeed whether the incorrectness is contributed by inconsistency or nontermination. First, nontermination can be proved if \(I \rightarrow WP(S,T)\) is false. Hence inconsistency is proved if \(I \rightarrow WP(S,Q)\) is false but \(I \rightarrow WP(S,T)\) is true. Referring again to example 6, we see that \(I \rightarrow WP(S,T)\) is true, and hence its incorrectness is due to inconsistency.

Finally, we consider examples 7 and 8. In example 7, we obtained

\[
WP(S',Q) \equiv (\exists n \geq 0)[ny \leq x < (n+1)y \land x > 0]
\]

Since it is easily shown from properties of integers that

\[
x \geq 0 \land y \geq 0 \rightarrow (\exists n)[n \leq x < (n+1)y]
\]

We have shown that \(I \rightarrow WP(S',Q)\), and hence \(S;\) is correct.

In example 8, we obtained

\[
WP(S',Q) \equiv (\exists n \geq 0)[2^n \leq B < 2^{n+1}],
\]

Again, it is easily seen that

\[
A > 0 \land B \geq 0 \rightarrow (\exists n \geq 0)[2^n \leq B < 2^{n+1}]
\]

Hence, we have proved that \(S'\) is correct.
VI. THE FIXPOINT THEOREM

In this section, we shall consider another way of viewing the repetitive construct DO for program verification.

We first consider the deterministic construct

\[ S': \text{WHILE B DO S;} \]

We note that this program is equivalent to

\[ \text{IF B THEN S;} \]
\[ \text{IF B THEN S;} \]
\[ \ldots \]
\[ \text{IF B THEN S;} \]

Where the equivalent program consists of as many "IF B THEN S" statements as the number of times S is executed in the WHILE program if it terminates. Thus, the program \( S' \) is equivalent to the program \( S'' \):

\[ \text{IF B THEN begin S;} \]
\[ \text{WHILE B DO S;} \]
\[ \text{end;} \]

By (7), we see that

\[ \text{WP}(S'',Q) \equiv (\overline{B} \land Q) \lor (B \land \text{WP}(S;S',Q)) \]

By the axiom of composition,

\[ \text{WP}(S',Q) \equiv \text{WP}(S,\text{WP}(S',Q)) \]

and hence we derive the following recursive equation

\[ \text{WP}(S',Q) \equiv [\text{BAQ}] \lor [\text{BAWP}(S,\text{WP}(S',Q))] \quad (12) \]

Similarly, in the nondeterministic case, we note that the statement DO

\[ \overline{\text{do } B_1 \rightarrow S_1 \ldots \overline{B_n \rightarrow S_n \text{ od}}} \quad (13) \]
is "semantically" equivalent to the statement
\[
\text{if } \overline{BB} \rightarrow "\text{skip}" \square BB \rightarrow \text{IF}; \ DO \ \text{fi}
\]  \hspace{1cm} (14)

where \( \overline{BB} \) is the negation of \( BB \). Hence, we arrive at the following recursive equation.

\[
WP(DO,Q) \equiv [\overline{BB}Q] \lor [BB\text{AWP}(IF,WP(DO,Q))]
\]  \hspace{1cm} (15)

We shall now justify that statements "DO" and "if \( \overline{BB} \rightarrow "\text{skip}\) \( \square BB \rightarrow \text{IF}; \ DO \text{ fi}" are indeed semantically equivalent by proving (15). First of all, we will need a lemma due to Dijkstra [1976] which states

\[
BB\text{AWP}(DO,Q) \equiv BB\text{AWP}(IF,WP(DO,Q))
\]  \hspace{1cm} (16)

then,

\[
(\overline{BB}Q) \lor [BB\text{AWP}(IF,WP(DO,Q))]
\]

\[
\equiv (\overline{BB}Q) \lor (BB\text{AWP}(DO,Q))
\]

\[
\equiv (BBQ) \lor (BB\exists i:i \geq 0:HH)
\]  \hspace{1cm} (by 16)

\[
\equiv H_0 \lor (\exists i:i \geq 0:BB \land HH)
\]

\[
\equiv H_0 \lor (\exists i:i \geq 1:BBHH)
\]

\[
\equiv \exists i:i \geq 0:BBHH
\]

\[
\equiv WP(DO,R).
\]

Let us consider a function \( \tau \) whose domain is the set of predicates defined on state space of a program such that for any such predicate \( P \),

\[
\tau(P) \equiv [\overline{BB} \land Q] \lor [BB \land WP(IF,P)]
\]  \hspace{1cm} (17)

where \( BB \) and \( IF \) refer to the repetitive construct "DO".

In the deterministic case, we have

\[
\tau(P) \equiv [\overline{B} \land Q] \lor [B \land WP(S_1,P)]
\]  \hspace{1cm} (18)

where we are referring to the construct "While B do \( S_1 \)".

We call \( P \) a fixpoint of \( \tau \) if \( \tau(P) = P \). By equation (15), we see that \( WP(DO,Q) \) is a fixpoint of \( \tau \). Indeed, we have a stronger result as
stated by the following theorem.

**Theorem 6** – The predicate $\text{WP}(D_0, Q)$ is the least fixpoint of $\tau$ in the sense that if $P$ is any other fixpoint of $\tau$, then $\text{WP}(D_0, Q) \rightarrow P$.

**Proof:** Let $P$ be a fixpoint of $\tau$. We need the following lemma:

for $i > 0$, $\text{WP}(\text{IF}, H_{i-1}(Q)) \rightarrow \text{WP}(\text{IF}, P)$ \hspace{1cm} (19)

Assuming (19) holds, then we can conclude that for $i > 0$,

$\text{WP}(\text{IF}, H_{i-1}(Q)) \lor H_0(Q) \rightarrow \text{WP}(\text{IF}, P) \lor H_0(Q)$ \hspace{1cm} (20)

Since $\text{WP}(\text{IF}, S) \rightarrow \text{BB}$, we can conclude that

$\text{WP}(\text{IF}, P) = \text{WP}(\text{IF}, P) \land \text{BB}$ \hspace{1cm} (21)

Hence, by equations (20) and (21), we have

$H_1(Q) = H_0(Q) \lor \text{WP}(\text{IF}, H_{i-1}(Q)) \rightarrow (\text{BB} \land \text{WP}(\text{IF}, P)) \lor H_0(Q)$, for $i > 0$

Therefore,

$\text{WP}(D_0, Q) = \exists j: j \geq 0: H_j(Q) \rightarrow H_0(Q) \lor (\text{BB} \land \text{WP}(\text{IF}, P)) \equiv P$

And hence the theorem is proved.

We now need to prove the lemma stated by equation (19). The proof will be by induction

(i) $\text{WP}(\text{IF}, P) \equiv \text{WP}(\text{IF}, (\text{BB} \land Q) \lor (\text{BB} \land \text{WP}(\text{IF}, P)))$

$\rightarrow \text{WP}(\text{IF}, \text{BB} \land Q) \lor \text{WP}(\text{IF}, \text{BB} \land \text{WP}(\text{IF}, P))$

$\equiv \text{WP}(\text{IF}, H_0(Q)) \lor \text{WP}(\text{IF}, \text{BB} \land \text{WP}(\text{IF}, P))$

Hence, $\text{WP}(\text{IF}, H_0(Q)) \rightarrow \text{WP}(\text{IF}, P)$.

(ii) Let us assume that for $K \geq 1$,

$\text{WP}(\text{IF}, H_K(Q)) \rightarrow \text{WP}(\text{IF}, P)$

Let $K = K+1$, we have

$\text{WP}(\text{IF}, H_{K+1}(Q)) \equiv \text{WP}(\text{IF}, \text{WP}(\text{IF}, H_K(Q)) \lor H_0(Q))$

$\rightarrow \text{WP}(\text{IF}, \text{WP}(\text{IF}, H_K(Q)) \land \text{BB} \lor H_0(Q))$

$\rightarrow \text{WP}(\text{IF}, \text{WP}(\text{IF}, P) \land \text{BB} \lor H_0(Q))$
\[ \equiv \text{WP(IF,P)} \]

Thus, we can conclude by induction that

\[ \text{for } i > 0, \text{ WP(IF,} h_{i-1} \text{)} \rightarrow \text{WP(IF,P)} \text{.} \]

and hence the lemma is proved. ||

Note that for any \( x \in P^* - \text{WP(DO,Q)*, } x \notin H_1(Q)^\ast \), for \( i > 0 \),

and hence we conclude the following result. Where \( P^* \) denotes the set
of states characterized by the predicate \( P \).

Corollary 3 - Let \( P \) be a fixpoint of \( \tau \) in (17) different from
\( \text{WP(S,Q)} \), then the program DO will not terminate when input with elements
of the set \( P^* - \text{WP(S,Q)*} \).

The previous result states than any solution to (15) is a
predicate consisting of \( \text{WP(DO,Q)} \) together with some elements which will
not cause DO to terminate. Recall that \( \text{WP(DO,T)} \) is the weakest precon-
dition for DO to terminate, this combined with the previous result leads
to the following corollary.

Corollary 4 - Let \( P \) be a fixpoint of \( \tau \) in (17), then
\[ \text{WP(DO,Q) \equiv P \land WP(DO,T).} \]

Proof: Since \( P \land WP(DO,T) \) is again a fixpoint of \( \tau \), and that
\( P \land WP(DO,T) \rightarrow WP(DO,T) \), we conclude that \( P \land WP(DO,T) \equiv WP(DO,Q) \) by
theorem 6 and corollary 3. ||

Example 9 - Consider again the program computing the quotient
and remainder given in Example 7. We will illustrate here that
\[ \text{WP(S,Q) \equiv WP(S,T) \land P, where } P \equiv (x = B + AY). \]

\( P \) is clearly a solution to equation (17) since
\[ (B < Y \land x = B + AY) \lor (B \geq y \land x = (B + Y) + (A + 1)Y) \equiv \]
\[ x = B + AY. \]

-25-
Consider now \( T \) as the postcondition. We have for \( j \geq 0, \)
\[
A_j^S(T) \equiv jY \leq B < (j+1)Y.
\]
Hence, clearly, \( WP(S,T) \land P \equiv WP(S,Q) \).

Corollary 3 and example 9 illustrate that weakest precondition
may be obtained in two separate stages: by finding a solution to
equation (15), and by determining \( WP(DO,T) \). As we pointed out
previously, \( WP(DO,T) \) is precisely the condition for termination, we
therefore are led to the fact that if \( P \) is a solution to equation
(15) then a proof of \( I \rightarrow P \), where \( I \) is the input specification
for \( S \), would be a proof that \( S \) is consistent (or partially correct)
with respect to its specifications.

Observe that the previous discussions suggest that proof of
a program can be achieved in two stages using our approach, namely,
prove the consistency and termination separately. This, of course,
is familiar in the case of using "inductive assertion" approach. However,
the difference is that there is no need to invent a well-ordered set
for the termination proof. \( WP(DO,T) \) singles out the inherent proper-
ties of an algorithm which lead to termination as illustrated in
example 9.

As a consequence of theorem 4 and corollary 4, we have the
following result.

**Corollary 5** - Let \([I,Q]\) be a pair of specifications for the program
\[
S: \text{do } B_1 \rightarrow S_1 \quad \ldots \quad B_n \rightarrow S_n \text{ od,}
\]
then \( S \) is consistent with respect to \([I,Q]\) if and only if \( I \rightarrow P \) for
some fixpoint \( P \) of \( \tau \) in (17).

**Example 10** - Consider the nondeterministic program given in
example 2 for the solution of the eight queens problem. The predicate
\[
P \equiv \text{ROW} \leq 1 \land \text{COL} \leq 1 \land \text{LD} \leq 1 \land \text{RD} \leq 1
\]
is clearly a fixpoint of $\tau$ in (17). Let the specifications $[I,Q]$ be respectively the following:

\[ I \equiv [\text{ROW}:=\text{COL}:=\text{LD}:=\text{RD}:=0] \]
\[ Q \equiv [\text{ROW}=1 \land \text{COL} \leq 1 \land \text{LD} \leq 1 \land \text{RD} \leq 1]. \]

We see that $BB \land P \Rightarrow Q$, where $BB \equiv [\text{ROW} \neq 1]$ and $WP(I,P) \equiv T$. Hence, by corollary 4, the program is consistent with respect to $[I,Q]$.

It should be noted the ease of the consistency proof. The termination proof turns out to be tedious in that $WP(\text{DO},T)$ is a precise description of various ways of placing the queens to obtain all solutions of the problem.

Example 11 - Consider the following nondeterministic programs with specifications $[I,Q]$. Show that it is strongly verifiable with respect to $[I,Q]$.

\[ I:((n > 0 \land (\forall i: 0 \leq i \leq n, f(i) \geq 0)) \]
\[ k:=0; j:=1 \]
\[ \text{do} j \neq n \rightarrow \text{if } f(j) \leq f(k) \rightarrow j:=j+1 \]
\[ \quad \text{if } f(j) \geq f(k) \rightarrow k:=j; j:=j+1 \]
\[ \text{od} \]

\[ Q:[0 \leq k \leq n \land (\forall i: 0 \leq i < n: f(k) \geq f(i))] \]

Utilizing the fixpoint theorem, we shall first find a candidate for a fixpoint of the loop equation (15). This, of course, can be done by the generation of sequence of expressions $H_0(Q), H_1(Q), \ldots$ and extract from it a term and plug it into (15) to test whether it is a fixpoint. In any event, a candidate is proposed. In this case, we propose

\[ P \equiv (\exists j: 0 \leq j < n) \land (\forall i: 0 \leq i < j: f(j) \geq f(i)) \]

to be a fixpoint of (15) which is readily verifiable.

Since it is clear that $I \Rightarrow P$, we see that the program is consistent with respect to $[I,Q]$. 

-27-
To prove termination, we proceed to compute $H_0(T), H_1(T)$, etc and use induction to obtain the expression for a general term such that

$$H_i(T) \equiv \bigvee_{k=0}^{i} j+k = n$$

Hence:

$$WP(DO,T) \equiv (\exists k \geq 0)[k + j = n]$$

and

$$WP(S,T) \equiv (\exists k \geq 0)[k + 1 = n].$$

Clearly, $I \vdash WP(S,T)$ and this together with the consistency above implies that the program $S$ is strongly verifiable. $||$

VII. THE CONCEPT OF A LOOP INVARIANT AND ITS FIXPOINT CHARACTERIZATION

Recall that in both examples 10 and 11, the fixpoints $P$ we selected turned out to be "invariant" with respect to the loop $DO$. In other words, the predicate $P$ remains to be true no matter how often the guarded command of a set is selected. We say $P$ is a loop invariant of the loop $DO$ if

$$P \land BB \Rightarrow WP(IF,P) \tag{22}$$

The following result of Dijkstra called the "Fundamental Invariance Theorem for Loop $S$" is included here for completeness.

Theorem 7 (Dijkstra) - Let a guarded command set with its derived alternative construct IF and a predicate $P$ be such that $P \land BB \Rightarrow WP(IF,P)$ holds for all states; then for the corresponding repetitive construct $DO$ we can conclude that

$$[P \land WP(DO,T)] \Rightarrow WP(DO,P \land \overline{BB})$$
The previous theorem states that if \( P \) is a loop invariant, then upon termination of the whole retetitive construct, when none of the guard is true, the program will enter a final state satisfying \( P \land \overline{BB} \).

We now pursue the relationship between loop invariants and fixpoints further. Consider a fixpoint \( P \) of equation (12). We note that if \( BB \) is true, then
\[
P \equiv BB \land WP(IF,P)
\]
From which we can conclude that \( P \land BB \land WP(IF,P) \) and hence \( P \) is a loop invariant.

In the event \( \overline{BB} \) is true, then
\[
P \equiv \overline{BB} \land Q
\]
and we can conclude from the above equation
\[
P \land \overline{BB} \rightarrow Q
\] (23)

We note that property (23) provides a sufficiency condition for deducing the output assertion upon the completion of the repetitive construct.

We will call a predicate \( P \) an output\text{-}adequate loop invariant of DO if it satisfies equations (22) and (23). The following results summarize the relationship between this type of loop invariants and fixpoints of (15).

\textbf{Theorem 8} - Any fixpoint of (15) is an output adequate loop invariant of DO.

The converse of theorem 8 does not hold since the program construct IF may not terminate.

Theorem 8 states that fixpoints of equation (17) are those loop invariants which are strong enough to imply the output assertion, and
to guarantee that the loop will terminate. For verifying programs using the inductive assertion method, "output-adequate invariants" are the desirable invariants. Thus, equation (17) provides a test for this type of invariant.

To illustrate this point, let us consider the program of computing quotient and remainder given in example 7. The loop invariant $P \equiv X = A + BY$ is clearly a fixpoint of (17). Similarly, the loop invariant for the loop in example 8 is $P \equiv \exists^* x^y = A^B$ which is again a fixpoint of (10).

VIII. CONCLUDING REMARKS

In this paper, we have utilized the concept of a "predicate transformer" introduced by Dijkstra for the verification of deterministic and nondeterministic programs.

We have studied two ways of generating the weakest preconditions; by generation of a sequence of "approximations" of the final predicate and by solving for a solution of a recursive equation. Note that these two techniques can be used to complement each other in locating the loop invariant. The generation of successive terms will provide a guideline in obtaining a candidate, and that the recursive equation provides a test for checking the adequacy of the candidate. It thus seems that these two techniques will be a useful addition to any verification system based on inductive assertion method.

We conclude this paper by observing that the results of this paper seem to indicate that the concept of defining program semantics using predicate transformer is a very useful one. However, from a
practical point of view, a basic limitation is imposed by the exploding complexity of the weakest precondition.

IX. ACKNOWLEDGMENTS

The author is indebted to Professor E. W. Dijkstra for pointing out an error in an earlier draft of this chapter.

Support from NSF under Grants GJ-36424 and DCR75-09842, and Air Force under contract grant F44620-71-C-0091 are duly acknowledged.
VIII. REFERENCES


