AN APPROACH TO PROGRAM VERIFICATION

by

Raymond T. Yeh
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Introduction

Program verification is concerned with proving formally that i) the logical behavior of a given program is consistent with its (formal) specifications, and ii) that the given program will eventually terminate for any input satisfying the input specification. Clearly, in order to develop any mathematical proofs for programs, a deductive system must first be created in which these proofs of programs can be developed. For this purpose, we propose to define formally semantics of program constructs as axioms, and give inference rules so that semantics of a program segment can be derived from that of the constructs included in the segment.

In this paper, formal semantics of program constructs are defined by predicate transformer introduced by Dijkstra [3,4]. A predicate transformer is 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 concept of determinism no longer has its significance in this semantic context. Rather, the focus is on the nature of the computation, and hence the concept of iteration, and not how the program iterates, becomes a dominant concern here.

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.

Program Constructs

In this section, we will briefly review constructs introduced by Dijkstra [4] 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 guarded 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 nondeterminacy such a construct can bring about, we use Dijkstra's original example [4].

Example 1 - A program that for fixed x and y assigns to m the maximum value of x and y.

if $x \geq y \rightarrow m := x$
[] $y \geq x \rightarrow m := y$
fi

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

Let IF denote

$$\text{if } B_1 \rightarrow S_1 [] \ \ldots [] \ B_n \rightarrow S_n \text{ fi}$$

Let BB denote

$$B_1 \vee B_2 \vee \ldots \vee B_n$$

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

We observe here two special cases for the alternative constructs which specialize to deterministic constructs.

Case 1. IF denotes "if $B \rightarrow S_1 [] \ \overline{B} \rightarrow S_2$ 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 "if $B \rightarrow S_1 [] \ \overline{B} \rightarrow$skip fi". In this case, IF corresponds to the "if B then S_1" construct.

For the repetitive construct "do ... 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". Again, we will use the abbreviation DO for

$$\underline{\text{do}}\ B_1 \rightarrow S_1 \,[]\, \ldots \,[]\, B_n \rightarrow S_n\ \underline{\text{od}}$$

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,...,8}

COL with integral indices in the range {1,...,8}

LD with integral indices in the range {-7,...,7}

RD with integral indices in the range {2,...,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) the board is in row i, column j, left diagonal i-j, and right diagonal i+j.

The nondeterministic program is:

```
Line
1  ROW:=COL:=LD:=RD:=0;
2  do ROW≠1→if ROW1≠1→I:=1 [] ... []
3                ROW8≠1→I:=8 fi;
4             if COL1≠1→J:=1 [] ... []
5                COL8≠1→J:=8 fi;
6             if RDI+J<1 and LDI-J<1→
7                ROWI:=COLJ:=RDI+J:=LDI-J:=1
8             || true → "skip" fi
9  od
```

Where if A and B are vectors, then A=1 means that for each i, A_i=1; A ≤ 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. It should be noted the simplicity of this program using the nondeterministic constructs.

The Concept of a Predicate Transformer

In this section, we define the basic concepts and derive 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.

For a given S, we denote by f_S the underlying function that S computes[1]. Thus S terminates with respect to an input assertion I if $f_S(x)$ is defined for all $x \in I^*$.

Given a pair of input-out specification [P,Q] and a program S, we introduce Hoare's notation P{S}Q [8] to represent the following proposition:

"If the assertion P is true (on certain initial state vector) and the program S when initiated with this state vector terminates then the assertion Q will be true on the final state vector." Predicates P and Q are referred to as the pre- and post-condition of S. 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." Clearly, P[S]Q always implies P{S}Q.

We will use the notation WP(S,Q) to denote the weakest pre-condition for the initial state 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 predicate transformer by Dijkstra [4] since it transforms a postcondition R into a precondition WP(S,Q). It is easily seen that if P[S]Q, then P → WP(S,Q). Alternatively, WP(S,Q)* is the largest set of initial state of S for which S terminates and Q is true at output.

We next proceed to list a set of properties of the "predicate transformer" for given S and Q. These properties were given in [4] and is included here for the sake of completeness.

Theorem 1: The following identities and implications hold.

1) WP(S,F) ≡ F;

2) [P→Q] → [WP(S,P)→WP(S,Q)];

3) WP(S,P∨Q) ≡ WP(S,P) ∨ WP(S,Q);

4) WP(S,P∧Q) ≡ WP(S,P) ∧ WP(S,Q).

Utilizing the concept of weakest precondition, we can derive a simple formal characterization of termination in the following.

Corollary 1: A program S terminates with respect to an input assertion I if and only if I → WP(S,T).

A Deductive System for Program Proofs

In order to prove that a program is strongly verifiable, we will provide axiomatic definitions as formal semantics for program constructs, as well as a rule of inference for the meaning of a 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. Before giving the axiomatic definitions we need to define the concept of semantic equivalence.

Two programs S_1 and S_2 are said to be semantically equivalent if for any given postcondition Q, WP(S_1,Q) ≡ WP(S_2,Q); i.e., they have the same predicate transformer.

---

[1] Note that this does not mean we know what f_S is for given S. We simply use the notation to denote whatever function is being computed by S.

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

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

$$WP("x:=E",Q) \equiv Q_E^x \qquad (1)$$

where Q_E^x is obtained from Q by substituting the expression E for every (free) occurrence of the variable x.

2. The axiom of empty statement: the semantics for the statement "skip" is defined by

$$WP("skip",Q) \equiv Q. \qquad (2)$$

In order to define the semantics of the alternative construct we recall two abbreviations.

Let IF denote

$$\underline{if}\ B_1 \to S_1\ []\ \cdots\ []\ B_n \to S_n\ \underline{fi};$$

Let BB denote

$$B_1 \vee B_2 \vee \cdots \vee B_n$$

3. The axiom of alternative construct: the semantics for the statement "IF" is defined by

$$WP(IF,Q) \equiv BB \wedge [B_1 \to WP(S_1,Q)] \wedge \cdots \wedge [B_n \to WP(S_n,Q)]. \qquad (3)$$

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

Case i) IF denotes

$$\underline{if}\ B \to S_1\ []\ \overline{B} \to S_2\ \underline{fi}$$

Using (3) and algebraic simplification, we can derive the following:

$$WP("\underline{if}\ B \to S_1\ []\ \overline{B} \to S_2\ \underline{fi}",Q) \equiv [B \wedge WP(S_1,Q)] \vee [\overline{B} \wedge WP(S_2,Q)]$$

Case ii) IF denotes

$$\underline{if}\ B \to S_1\ []\ \overline{B} \to skip\ \underline{fi}$$

It is easily seen that in this case,

$$WP("if B \to S_1\ []\ \overline{B} \to skip",Q) \equiv [B \wedge WP(S_1,Q)] \vee [\overline{B} \wedge Q]$$

From [2], 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.

Now, we give an inference rule:

4. The axiom of composition: the semantics of the composite statements $\{S_1;S_2\}$ is defined by

$$WP("S_1;S_2",Q) \equiv WP(S_1,WP(S_2,Q)) \qquad (4)$$

We will illustrate the above definitions by the following example:

Example 3 - Consider the program that for fixed integers x and y, assigns to m the maximum value of x or y.

$$\underline{if}\ x \geq y \to m:=x$$
$$\quad y \geq x \to m:=y$$
$$\underline{fi}$$

With respect to the specifications $I \equiv [x \geq 0 \wedge y \geq 0]$, and $Q \equiv [m = \max(x,y)]$, it is easily seen that

$$WP(IF,Q) \equiv [x \geq y \to x = \max(x,y)] \wedge [y \geq x \to y = \max(x,y)].$$

To derive the semantics for the repetitive construct, we note that the statement DO

$$\underline{do}\ B_1 \to S_1\ [] \cdots []\ B_n \to S_n\ \underline{od}$$

is semantically equivalent to the statement

$$\underline{if}\ \overline{BB} \to "skip"\ []\ BB \to IF;\ DO\ \underline{fi} \qquad (5)$$

where $\overline{BB}$ is the negation of BB.

Applying previous axioms to (5) and simplying, we have

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

$$WP(DO,Q) \equiv [\overline{BB} \wedge Q] \vee [BB \wedge WP(IF,WP(DO,Q))] \qquad (6)$$

As a special case, we consider the construct "$\underline{do}\ B \to S_1\ \underline{od}$". We have

$$WP(DO,Q) \equiv [\overline{B} \wedge Q] \vee [B \wedge WP(S_1,WP(DO,Q))] \qquad (7)$$

It is easily seen from [2] that the two constructs "$\underline{do}\ B \to S_1\ \underline{od}$" and "while B do S_1" are semantically equivalent.

Theorem 3 - For any post condition Q, we have

$$WP(DO,Q) \equiv [\exists\, k:k \geq 0, H_k(Q)] \qquad (8)$$

where

$$H_0(Q) \equiv \overline{BB} \wedge Q, \text{ and}$$
$$H_k(Q) \equiv WP(IF,H_{k-1}(Q)), \text{ for } k > 0.$$

Note that intuitively $H_k(R)$ can be interpreted as the weakest precondition guaranteeing termination after exactly k selections of a guarded list, leaving the system in a final state satisfying Q.

Corollary 2: Let S be the statement "While B do S_1" then for any postcondition Q, we have

$$WP(S,Q) \equiv [\exists\, j:j \geq 0:A_S^j(Q)] \qquad (9)$$

where

$$A_S^0(Q) \equiv \overline{B} \wedge Q:$$
$$A_S^{j+1}(Q) \equiv B \wedge WP(S_1,A_S^j(Q)), \text{ for } j \geq 0.$$

Note that equations (8) and (9) provides a way of obtaining WP(DO,Q) in that successive terms of $H_k(Q)$ 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 theorems by the following examples.

Example 4 - Consider the program

$$\underline{do}\ x \neq 0 \to x:=x - 1\ \underline{od}.$$

With respect to the specifications [I,Q], where $I \equiv [x > 0]$ and $Q \equiv [x=-1]$, we obtain $WP(DO,Q) \equiv F$. With respect to [I,Q'], where $Q' \equiv [x = 0]$, we obtain

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

Strong Verification of Programs

In this section, we shall demonstrate the usefulness of formal definition of semantics by predicate transformer in program verification.

The usual practice of program verification consists of two parts: 1) to prove that behavior of a

computer program is consistent with respect to its specification; 2) to prove that the program terminates. Demonstration of the consistency between a program and its specification is usually referred to as proving the "partial correctness" of the program whereas proving both (1) and (2) is called a "total correctness" proof [3].

We say S is consistent (or partially correct or verifiable) with respect to [P,Q] if P{S}Q, and that S is strongly verifiable (or correct) with respect to [P,Q] if P[S]Q.

The following result is a straight forward consequence of Theorem 1 and the definition above.

Theorem 4 - 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 4 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 3 and 4. In example 3, we see that

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

Hence, the program is strongly verifiable.

Similarly, one can easily show in example 4 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 $f_S(x) \notin Q^*$.

The following results characterizes incorrectness and is a corollary of theorem 4.

Theorem 5 - 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.

Refering again to example 4, we see that the program in example 4 is incorrect with respect to [I,Q] since $WP(DO,Q) \equiv F$ and $I \rightarrow F$ is false.

In the following, we shall verify two programs which are frequently used as examples in demonstrating other verification techniques.

Example 5 - Let S' be the following program computing the quotient and remainder. Prove that it is strongly verifiable with respect to [I,Q], where $I \equiv (x \geq 0 \wedge y \geq 0)$ and $Q \equiv (x \leq B < y \wedge x \geq 0 \wedge x = B + Ay)$.

```
S': (x ≥ 0 ∧ y > 0)
   begin A: = 0; B: = x;
     S: While B ≥ y do
            begin B: = B-Y;
                  A: = A+1
            end
   end
(0 ≤ B < y ∧ x ≥ 0 ∧ x = B + Ay)
```

To prove the program is strongly verifiable, we will use corollary 2 to obtain a candidate for WP(S,Q). Using (9), it is clear that

$$A_S^0(Q) \equiv 0 \leq B < Y \wedge X \geq 0 \wedge X = B + AY$$

in order to compute WP(S,Q), we need to obtain the general term $A_S^j(Q)$, for $j > 0$. This can be done by induction on j with $A_S^0(Q)$ as the basis of the induction. Thus we obtain for $j > 0$,

$$A_S^j(Q) \equiv jy \leq B < (j+1)y \wedge x \geq 0 \wedge x = B + AY$$

Hence, $WP(S,Q) \equiv (\exists n \geq 0)\ [ny \leq B < (n+1)y \wedge x \geq 0 \wedge x = B+AY]$

and

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

Since it is easily shown from properties of integers that

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

we have proved that $I \rightarrow WP(S',Q)$, and hence S' is strongly verifiable. ||

Example 6 - Let S' be the following program computing the exponential function A^B of two integers. We will prove that it is strongly verifiable with respect to [I,Q], where $I \equiv [A > 0 \wedge B \geq 0]$ and $Q \equiv [z = A^B]$.

```
(A > 0 ∧ B ≥ 0)
S': begin x: = A; y: = B; z: = 1;
S : While y ≠ 0 do
       begin if odd (y)
             begin y: = y-1;
                   z: = z*x
             end
          y: = y/2; x: = x*x
       end
(z = A^B)
```

Again, we shall compute a few successive terms using equation (9) 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,

$$A_S^0(Q) \equiv [y = 0 \wedge z = A^B]$$

By (9), we have

$$A_S^1(Q) \equiv (y \neq 0) \wedge \{[\text{even}(y) \wedge \tfrac{y}{2} = 0 \wedge z = A^B] \vee [\text{odd}(y) \wedge \tfrac{y-1}{2} = 0 \wedge z*x = A^B]\}$$

After simplification, we have

$$A_S^1(Q) \equiv [y = 1 \quad z*x = A^B]$$

Similarly, we obtain

$$A_S^2(Q) \equiv [y = 2 \wedge z*x^2 = A^B] \vee [y = 3 \wedge z*x^3 = A^B]$$

By induction on j, we obtain that for $j \geq 0$,

$$A_S^{j+1}(Q) \equiv \bigvee_{y=2^{j-1}}^{2^j-1} [y \geq 0 \wedge z*x^k = A^B]$$

Hence,

$$WP(S,Q) \equiv (\exists n \geq 0)[2^n \leq y < 2^{n+1} \wedge z*x^y = A^B]$$

and

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

Clearly, it is true that

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

and hence the program is strongly verifiable. ||

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 equation (9), 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.

The Fixpoint Theorem

In the previous section we have shown that the weakest precondition of a repetitive construct is a solution to a recursive equation (6). We will show in this section how solutions of (8) are useful for program verification.

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

$$\tau(P) \equiv [\overline{BB} \wedge Q] \vee [BB \wedge WP(IF,(P)] \qquad (10)$$

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

In the deterministic case, we have

$$\tau(P) \equiv [\overline{B} \wedge Q] \vee [B \wedge WP(S_1,P)] \qquad (11)$$

where we are refering to the construct "While B do S_1".

We say P is a fixpoint of (10) if $\tau(P) \equiv P$. By (6), we see that WP(DO,Q) is a fixpoint of (10). In fact, we have the following stronger result. (The proof is similar to that given in [1] for the deterministic case and hence will not be repeated here).

Theorem 6 - The predicate WP(DO,Q) is the least fixpoint of τ in the sense that if P is any other fixpoint of τ, then $WP(DO,Q) \rightarrow P$.

Corollary 4 - Let P be a fixpoint of τ in (10) different from WP(S,Q), then the program DO will not terminate when input with elements of the set P* - WP(S,Q)*. Where P* denotes the set of states characterized by the predicate P.

Recall that WP(S,T) characterizes all the initial states which causes a program S to terminate, we have the following straight forward result.

Corollary 5 - Let P be a fixpoint of τ in (10), then $WP(DO,Q) \equiv P \wedge WP(DO,T)$.

Example 7 - Consider again the program computing the quotient and remainder given in Example 5. We will illustrate here that $WP(S,Q) \equiv WP(S,T) \wedge P$, where $P \equiv (x = B + AY)$.

P is clearly a solution to equation (7) since

$$(B<Y \wedge x = B+AY) \vee (B \geq Y \wedge x = (\overline{B}+Y) + (A+1)Y) \equiv x = B+AY.$$

Consider now T as the postcondition. We have for $j \geq 0$,

$$A_S^j(T) \equiv jY \leq B < (j+1)Y.$$

Hence, clearly, $WP(S,T) \wedge P \equiv WP(S,Q)$. ||

Corollary 5 and Example 7 illustrate that weakest precondition may be obtained in two separate stages: by finding a solution to equation (6) or (7), and by determining WP(S,T). As we pointed out previously WP(S,T) is precisely the condition for termination, we therefore are led to the fact that if P is a solution to equation (6) or (7), then a proof of $I \rightarrow P$ where I is the input specification for S, would be 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 similar to the conventional "inductive assertion" proof. However, the difference is that there is no need to invent a well-ordered set for the termination proof. WP(S,T) singles out the inherent properties of an algorithm which lead to termination as illustrated in example 7.

As a consequence of theorem 4 and corollary 5, we have the following result.

Corollary 6 - Let [I,Q] be a pair of specifications for the program

$$S: \underline{do}\ B_1 \rightarrow S_1 \ [] \ldots [] B_n \rightarrow S_n\ \underline{od},$$

then S is consistent with respect to [I,Q] if and only if $I \rightarrow P$ for some fixpoint P of τ in (10).

Example 8 - Consider the nondeterministic program given in example 2 for the solution of the eight queens problem. The predicate

$$P \equiv ROW \leq 1 \wedge COL \leq 1 \wedge LD \leq 1 \wedge RD \leq 1$$

is clearly a fixpoint of τ in (10). Let the specifications [I,Q] be respectively the following:

$$I \equiv [ROW:=COL:=LD:=RD:=0]$$

$$Q \equiv [ROW=1 \wedge COL \leq 1 \wedge LD \leq 1 \wedge RD \leq 1].$$

We see that $\overline{BB} \wedge P \rightarrow Q$, where $BB \equiv [ROW \neq 1]$ and $WP(I,P) \equiv T$. Hence, by corollary 6, the program is consistent with respect to [I,Q].

Example 9 - Consider the following nondeterministic programs with specifications [I,Q]. Show that it is strongly verifiable with respect to [I,Q].

$$I:[(n > 0 \wedge (\forall i: 0 \leq i \leq n, f(i) \geq 0)]$$

k:=0; j:=1

DO [do j ≠ n → if f(j) ≤ f(k) → j:=j + 1 [] f(j) ≥ f(k) → k:=j;j:=j+1 fi od] IF

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

Utilizing the fixpoint theorem, we shall first find a candidate for a fixpoint of the loop equation (6). 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 (6) to test whether it is a fixpoint. In any event, a candidate is proposed. In this case, we propose

$$P \equiv (\exists j{:}0 \leq j < n) \wedge (\forall i{:}0 \leq i < j{:}f(j) \geq f(i)) \quad (12)$$

to be a fixpoint of (10), which is readily verifiable by substituting (12) into (10).

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

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

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

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

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. The problem of demonstrating a candidate predicate to be a loop invariant of a deterministic program can be shown to be equivalent to the problem of demonstrating this candidate to be a fixpoint of a recursive equation. If we could prove this candidate to be the least fixpoint, then it could be used to prove the consistency and termination or to prove the incorrectness of the program. 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.

Acknowledgments

It is a pleasure to acknowledge the support of NSF Grants GJ-36424 and DCR 75-09842, and Air Force under contract grant F44620-71-C-0091.

References

1. Basu, S. K., and J. Misra, "Proving Loop Programs", Trans. on Software Engineering, vol SE-1, no. 1, 76-86, (1975).

2. Basu, S. K., and R. T. Yeh, "Strong Verification of Programs", Trans. on Software Engineering, vol. SE-1, no. 3, 339-345, (1975)

3. Dijkstra, W. E., "A Simple Axiomatic Basis for Programming Language Constructs", (1974).

4. Dijkstra, W. W., "Guarded Commands, Non-determinancy and A Calculus for the Derivation of Programs", Proc. 1975 International Conf. on Reliable Software, 2.0-2.13; CACM, vol. 18, no. 8 (1975) 453-457.

5. Floyd, R. W., "Assigning Meaning to Programs", Proc. Amer. Math. Soc. Symposia in Appl. Math., 19: 19-31, (1967).

6. Fusaoka, A., and R. Waldinger, "Program Writing Using Sequences", Stanford Research Institute Technical Report, (Jan. 1974).

7. Grief, I. and R. J. Waldinger, "A more Mechanical Heuristic Approach to Program Verification", Technical Report, (1974).

8. Hoare, C. A. R., "An Axiomatic Basis for Computer Programming", CACM, vol. 12, no. 10, 576-583, (1967).

9. Kleene, S. C., "Introduction to Meta Mathematics", Amer. Elesvier Publishing Co., (1971).

10. London, R. L., "A View of Program Verification", Proc. 1975 International Conference on Reliable Software, 534-545.

11. Manna, Z., Mathematical Theory of Computation, McGraw-Hill Inc., (1974).

12. Manna, Z., and J. Vuillemin, "Fixpoint Approach to the Theory of Computation", CACM, vol. 15, no. 7, 528-536, (1972).

13. Manna, Z., and Z. Pnueli, "Formalization of Properties of Functional Programs", J. ACM, vol. 17, no. 3, 536-555, (1970).

14. Reynolds, C. and R. T. Yeh, "Verification of Non-deterministic Programs", Proc. 11th Asilomar Conf. on Circuits, Systems, and Computers, 315-319 (Dec. 1975).

15. Wegbreit, B., "Inductive Assertion Synthesizer", Trans. Software Engineering, vol. SE-1, no. 1, 68-75, (1975).