PROGRAM VERIFICATION TABLEAUS

E.A. Ashcroft
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

CS-76-01

January 1976

PROGRAM VERIFICATION TABLEAUS

by

E.A. Ashcroft

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

Abstract

A method is described for the presentation of formal partial-correctness proofs in a concise and readable way.

Key phrases: Program proving, Formal systems, Partial correctness.

CR Categories: 5.24.

This research was supported in part by a grant from the Canadian National Research Council.

1. Introduction and Summary

Formal proofs of programs, using Hoare's axiomatic method, can be presented in a readable way, by making such proofs look like proofs by Floyd's method of attaching assertions to programs. The method is simple and natural(and rather obvious), but it requires that the formal theory only produce proofs with certain structural properties.

Basically, the method of Floyd [3], for proving properties of programs, is as follows. The program in question is drawn as a flowchart and then "assertions" are attached to some of the edges of the flowchart (at least one edge per loop). These assertions are intended to express properties that will be true whenever control passes along the corresponding edges; such assertions are said to be valid. To check a set of attached assertions for validity it suffices to prove that, for each assertion, if any values of the variables in the program satisfy the assertion then moving to the next edge at which an assertion is attached will give values to the variables that will satisfy this next assertion.

Valid assertions are useful because they can tell us what is happening as the program computes, and also because such an assertion attached to the last edge of the flowchart will express some property of the result of the program (assuming termination).

For example, in Fig.1 is shown a flowchart for computing N^M, where N and M are integers and M is non-negative. (÷ denotes integer division.) The assertions attached to the flowchart are shown to be valid by checking the following four conditions:

(i) $M \geq 0 \Rightarrow M \geq 0 \;\&\; 1 \times N^M = N^M$

(ii) $Y > 0 \;\&\; \mathrm{odd}(Y) \;\&\; Z \times X^Y = N^M \Rightarrow Y \div 2 \geq 0 \;\&\; Z \times X \times (X \times X)^{Y \div 2} = N^M$

(iii) $Y > 0 \;\&\; \neg\mathrm{odd}(Y) \;\&\; Z \times X^Y = N^M \Rightarrow Y \div 2 \geq 0 \;\&\; Z \times (X \times X)^{Y \div 2} = N^M$

(iv) $Y = 0 \;\&\; Z \times X^Y = N^M \Rightarrow Z = N^M$.

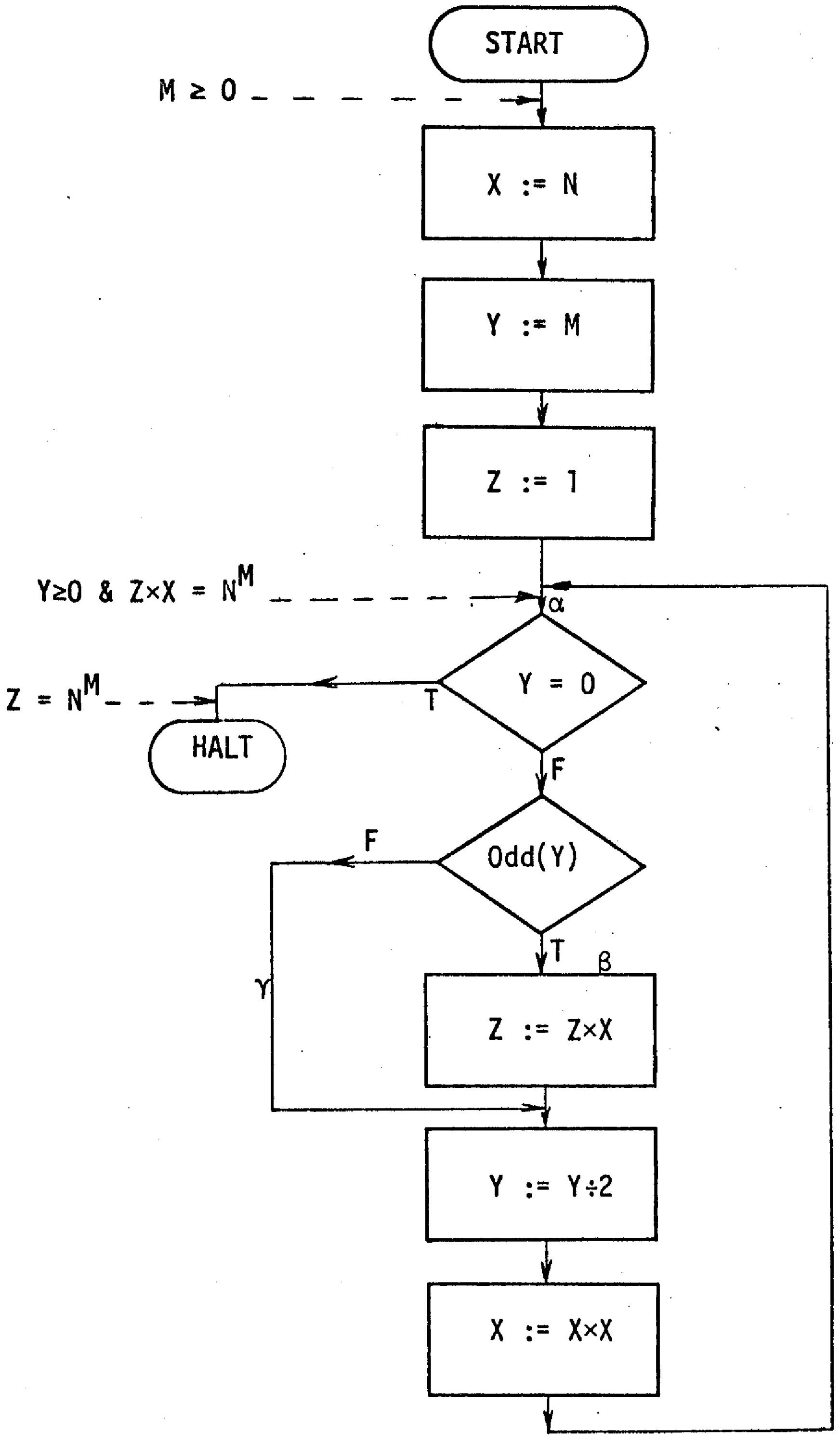

Fig.1 Flowchart for an exponentiation program, with attached assertions.

These correspond to going from the START to α, from α back to α via β, from α back to α via γ, and from α to the HALT. The conditions are easily shown to be true using the following properties of integers:

(a) $y \geq 0 \ \& \ \mathrm{Odd}(y) \Rightarrow y = 2(y \div 2)+1$

(b) $\neg\mathrm{Odd}(y) \Rightarrow y = 2(y \div 2)$

(c) $x^{2y} = (x \times x)^y$

(d) $x^y = x \times x^{y-1}$

(e) $y \geq 0 \Rightarrow (y \div 2) \geq 0.$

(Here we are ignoring the differences between computer arithmetic and "true" arithmetic. Therefore, the program is only correct if we assume there is no integer overflow, a fragile assumption for an exponentiation program.)

The method is basically simple and easy to understand, but suffers from a certain informality.

With the advent of the axiomatic approach of Hoare [4], program proving became a mathematically respectable occupation. In Hoare's approach, program text is used, rather than flowcharts, and in fact such text is incorporated directly into formulas expressing properties of the text. If R and Q are assertions and S is a program fragment, {R} S {Q} means that if R is true and S is executed then if S terminates Q will be true.*
Hoare's formal system allows such formulas (which will henceforth be called "properties") to be proved as theorems.

---

* This is a variant of Hoare's notation due to Manna and Vuillemin [5]. It leads to more legible verification tableaus, as will be seen.

As a program, the flowchart of Fig.1 will be

```
S1:X := N;
S2:Y := M;
S3:Z := 1;
S4:while Y ≠ 0 do
   S5:if Odd(Y)
         then Z := Z×X
         else null fi;
   S6:Y := Y÷2;
   S7:X := X×X
      end
```

(The labels $S_1 \ldots S_6$ will be referred to later.)

If we call this text P, then the property proved earlier is expressed by the property

$$\{M \geq 0\}\ P\ \{Z = N^M\}.$$

This can be proved in Hoare's system as follows (those unfamiliar with Hoare's system should refer to Section 3).

Proof

$$\left.\begin{array}{ll} (1) & Y > 0 \ \&\ \mathrm{Odd}(Y) \Rightarrow Y-1 = 2(Y \div 2) \\ (2) & X^Y = X \times X^{Y-1} \\ (3) & Y > 0 \Rightarrow Y \div 2 \geq 0 \\ (4) & \neg\mathrm{Odd}(Y) \Rightarrow Y = 2(Y \div 2) \end{array}\right\} \text{(Properties of integers)}$$

(5) $Z \times X^{Y} = N^{M} \;\&\; Y > 0 \;\&\; \mathrm{Odd}(Y) \Rightarrow Z \times X \times X^{2(Y \div 2)} = N^{M} \;\&\; Y > 0$

(from (1) and (2))

(6) $\{Z \times X \times X^{2(Y \div 2)} = N^{M} \;\&\; Y > 0\}\; Z := Z \times X\; \{Z \times X^{2(Y \div 2)} = N^{M} \;\&\; Y > 0)$

(assignment axiom)

(7) $\{Z \times X^{Y} = N^{M} \;\&\; Y > 0 \;\&\; \mathrm{Odd}(Y)\}\; Z := Z \times X\{Z \times X^{2(Y \div 2)} = N^{M} \;\&\; Y > 0)$

(preconsequence rule on (5) & (6))

(8) $Z \times X^{Y} = N^{M} \;\&\; Y > 0 \;\&\; \neg\mathrm{Odd}(Y) \Rightarrow Z \times X^{2(Y \div 2)} = N^{M} \;\&\; Y > 0$

(from (4))

(9) $\{Z \times X^{2(Y \div 2)} = N^{M} \;\&\; Y > 0\}$ null $\{Z \times X^{2(Y \div 2)} = N^{M} \;\&\; Y > 0\}$

(null axiom)

(10) $\{Z \times X^{Y} = N^{M} \;\&\; Y > 0 \;\&\; \neg\mathrm{Odd}(Y)\}$ null $\{Z \times X^{2(Y \div 2)} = N^{M} \;\&\; Y > 0\}$

(preconsequence rule on (8) & (9))

(11) $\{Z \times X^{Y} = N^{M} \;\&\; Y > 0\}$ if $\mathrm{Odd}(Y)$ then $Z := Z \times X$
else null fi $\{Z \times X^{2(Y \div 2)} = N^{M} \;\&\; Y > 0\}$

(conditional rule on (7) & (10))

It is now getting tedious writing out the statements of the program, so in the rest of the proof the labels in the program will be used to denote the statements that they label. For example, Line (11) can be written

(11) $\{Z \times X^{Y} = N^{M} \;\&\; Y > 0\}\; S_5\; \{Z \times X^{2(Y \div 2)} = N^{M} \;\&\; Y > 0\}$.

(12) $Z \times X^{2(Y \div 2)} = N^{M} \;\&\; Y > 0 \Rightarrow Z \times X^{2(Y \div 2)} = N^{M} \;\&\; Y \div 2 \geq 0$

(from (3))

(13) $\{Z \times X^{2(Y \div 2)} = N^{M} \;\&\; Y \div 2 \geq 0\}\; S_6\; \{Z \times X^{2Y} = N^{M} \;\&\; Y \geq 0\}$

(assignment axiom)

(14) $\{Z \times X^{2(Y \div 2)} = N^M \ \& \ Y > 0\} \ S_6 \ \{Z \times X^{2Y} = N^M \ \& \ Y \geq 0\}$

(preconsequence rule on (12) & (13))

(15) $\{Z \times X^{Y} = N^M \ \& \ Y > 0\} \ S_5;S_6 \ \{Z \times X^{2Y} = N^M \ \& \ Y \geq 0\}$

(concatenation rule on (11) & (15))

(16) $X^{2Y} = (X \times X)^Y$ (property of exponentiation)

(17) $\{Z \times X^{2Y} = N^M \ \& \ Y \geq 0\} \ S_7 \ \{Z \times X^{Y} = N^M \ \& \ Y \geq 0\}$

(assignment axiom and (16))

(18) $\{Z \times X^{Y} = N^M \ \& \ Y > 0\} \ S_5;S_6;S_7 \ \{Z \times X^{Y} = N^M \ \& \ Y \geq 0\}$

(concatenation rule on (15) & (17))

(19) $Y \geq 0 \ \& \ Y \neq 0 \Rightarrow Y > 0$

(20) $\{Z \times X^{Y} = N^M \ \& \ Y \geq 0\} \ S_4 \ \{Z \times X^{Y} = N^M \ \& \ Y = 0\}$

((19) & while rule on (18))

(21) $Z \times X^{Y} = N^M \ \& \ Y = 0 \Rightarrow Z = N^M$

(22) $\{Z \times X^{Y} = N^M \ \& \ Y \geq 0\} \ S_4 \ \{Z = N^M\}$ (postconsequence rule on (20) & (21))

(23) $\{X^{Y} = N^M \ \& \ Y \geq 0\} \ S_3 \ \{Z \times X^{Y} = N^M \ \& \ Y \geq 0\}$

(assignment axiom)

(24) $\{X^{Y} = N^M \ \& \ Y \geq 0\} \ S_3;S_4 \ \{Z = N^M\}$ (concatenation rule on (23) & (22))

(25) $\{X^{M} = N^M \ \& \ M \geq 0\} \ S_2 \ \{X^{Y} = N^M \ \& \ Y \geq 0\}$

(assignment axiom)

(26) $\{X^{M} = N^M \ \& \ M \geq 0\} \ S_2;S_3;S_4 \ \{Z = N^M\}$

(concatenation rule on (25) & (24))

(27) $\{N^{M} = N^M \ \& \ M \geq 0\} \ S_1 \ \{X^{M} = N^M \ \& \ M \geq 0\}$

(assignment axiom)

(28) $M \geq 0 \Rightarrow N^M = N^M \ \& \ M \geq 0$

(29) $\{M \geq 0\}\ S_1\ \{X^M = N^M \ \& \ M \geq 0\}$ (preconsequence rule on (27) & (28))

(30) $\{M \geq 0\}\ S_1;S_2;S_3;S_4\ \{Z = N^M\}$ (concatenation rule on (29) & (26))

□

The use of labels in place of statements made the proof easier to write (but not easier to read). Nevertheless, the repetition of statements and assertions necessitated by the many applications of the concatenation and consequence rules makes writing formal proofs such as these rather a chore. One may be forgiven for reverting to the less formal method of Floyd, with its appealing imagery of attaching assertions to the program itself.

But we can combine the 'pictorial' quality of Floyd's method with the formal theory of Hoare by using Verification Tableaus. A verification tableau is simply a program (not a flowchart) with assertions inserted between and around statements. The rules for correctly inserting these assertions are easily checked, and such a verification tableau is equivalent to a completely formal proof using Hoare's formal system. In fact, a proof can be constructed for any verification tableau and vice versa.

The following is the verification tableau of the formal proof given above.

```
$\{M \geq 0\}$
$\{M \geq 0 \;\&\; N^M = N^M\}$                      X := N
$\{M \geq 0 \;\&\; X^M = N^M\}$                      Y := M
$\{Y \geq 0 \;\&\; X^Y = N^M\}$                      Z := 1
$\{Y \geq 0 \;\&\; Z\ X^Y = N^M\}$                   while Y $\neq$ 0 do
    $\{Y > 0 \;\&\; Z \times X^Y = N^M\}$                if Odd(Y)
                                                          then
        $\{Y > 0 \;\&\; Odd(Y) \;\&\; Z \times X^Y = N^M\}$
        $\{Y > 0 \;\&\; Odd(Y) \;\&\; Z \times X \times X^{Y-1} = N^M\}$
        $\{Y > 0 \;\&\; Z \times X \times X^{2(Y \div 2)} = N^M\}$           Z := Z×X
            $\{Y > 0 \;\&\; Z \times X^{2(Y \div 2)} = N^M\}$
                                                          else
        $\{Y > 0 \;\&\; \neg Odd(Y) \;\&\; Z \times X^Y = N^M\}$
        $\{Y > 0 \;\&\; Z \times X^{2(Y \div 2)} = N^M\}$              null
            $\{Y > 0 \;\&\; Z \times X^{2(Y \div 2)} = N^M\}$                    fi
    $\{Y > 0 \;\&\; Z \times X^{2(Y \div 2)} = N^M\}$
    $\{Y \div 2 \geq 0 \;\&\; Z \times X^{2(Y \div 2)} = N^M\}$       Y := Y÷2
    $\{Y \geq 0 \;\&\; Z \times X^{2Y} = N^M\}$                  X := X×X
    $\{Y \geq 0 \;\&\; Z \times X^Y = N^M\}$                   end
$\{Y = 0 \;\&\; Z \times X^Y = N^M\}$
$\{Z = N^M\}$
```

This is a verification tableau because

(i) for all consecutive formulas {R}{Q} we have

$R \Rightarrow Q$ (i.e. R implies Q)

(ii) for all formulas surrounding assignments

$\{R\}\ x_i := e\ \{Q\}$

we have that R is $Q_e^{x_i}$.

(iii) for formulas around conditional statements

$\{R\}$ if B then $\{R_1\} \ldots \{Q_1\}$ else $\{R_2\} \ldots \{Q_2\}$ fi $\{Q\}$

we have that

R_1 is R & B, R_2 is R & $\neg B$ and $Q_1 = Q_2 = Q$.

(iv) for formulas around while statements

$\{R\}$ while B do $\{R_1\} \ldots \{Q_1\}$ end $\{Q\}$

we have

Q_1 is R, R_1 is R & B and Q is R & $\neg B$.

Tableaus are merely a notational convenience, yet the concise and natural way in which they present proofs seems to make program proving simpler. They are a contribution not to the theory but to the practice of program verification.

Some programming languages, such as ALGOL-W, already have facilities for annotating programs. It is interesting to note that it would be quite feasible to have a compiler which takes tableaus as its data rather than programs, and checks conditions (ii), (iii) and (iv) before compiling code. If a tableau is then not a verification tableau, i.e. is not tantamount to a proof of correctness, it must be because of two consecutive formulas $\{R\}\{Q\}$ for which $R \not\Rightarrow Q$. Such a compiler, which only accepts annotated programs, would force the programmer to think about proving his programs, and would make him bring his errors out into the open, in the form of formulas that follow textually but not logically.

The rest of the paper gives the formal definition of verification tableaus and proves their equivalence to formal proofs.

2. Formal Description of Programs and Tableaus

We will use conventional BNF grammars to describe the programming language and the corresponding language of "annotated programs" or tableaus. As for programs, in practice certain layout conventions will be used when writing tableaus, to improve readability. These are not part of the formal description.

We will assume four basic syntactic categories: <variable>, <expression>, <logical expression> and <formula>. Formulas are the "assertions" that can be used to annotate programs, and we assume that logical expressions are formulas, and that the set of formulas is closed under negation "¬", conjunction "&" and the substitution of expressions for free occurrences of variables.

Programs

```
<program> ::= <statementlist>
<statementlist> ::= <statement>|
                    <statementlist>;<statementlist>
<statement> ::= null|<assignment>|<while>|
                     <conditional>
<assignment> ::= <variable> := <expression>
<while> ::= while <logical expression>
                do <statementlist> end
<conditional> ::= if <logical expression>
                  then <statementlist>
                  else <statementlist> fi
```

The syntactic ambiguity caused by the production for <statementlist> is unimportant. (Ambiguity in a grammar is important only if syntactical analysis is used for semantic purposes. Here, semantically, the composition of statements is associative.) The ambiguity is there to allow a neat tie-up with the formal theory of the next section. (In proofs we do not want to be restricted to considering ";" as only associating to the left or only to the right.)

Tableaus

```
<tableau> ::= {<formula>} <STATEMENTLIST> {<formula>}
<STATEMENTLIST> ::= {<formula>} <STATEMENTLIST>|
                    <STATEMENTLIST> {<formula>}|
                    <STATEMENT>|  .
                    <STATEMENTLIST>{<formula>} <STATEMENTLIST>
<STATEMENT> ::=  null|<ASSIGNMENT>|<WHILE>|
                      <CONDITIONAL> .
<ASSIGNMENT> ::= <variable> := <expression>
<WHILE> ::= while <logical expression> do
                  {<formula>} <STATEMENTLIST> {<formula>} end
<CONDITIONAL> ::= if <logical expression> then
                  {<formula>} <STATEMENTLIST> {<formula>} else
                  {<formula>} <STATEMENTLIST> {<formula>} fi
```

The similarity between these two grammars is obvious. Essentially, the grammar for tableaus inserts one or more formulas (within curly brackets) before and after every statementlist (and hence every statement) in a program.

(We drop the semicolons in the formal definition of tableaus, just to simplify the proofs in this paper. In practice, tableaus are constructed by adding formulas (within curly brackets) to programs, and the semicolons are left in. But as far as the tableau is concerned the semicolons don't matter, which simply means that the positioning of formulas relative to semicolons is irrelevant.)

Given any STATEMENTLIST S in a tableau, we can strip off the formulas (and curly brackets) and, after possibly inserting some semi-colons, we obtain a statementlist which we denote $P(S)$.

Conversely, if we take a statementlist S and precede and follow every statement by one or more formulas within curly brackets, we will obtain a STATEMENTLIST S' say (if we ignore the semicolons). This process is called "annotating" the statementlist S, and the formulas are an "annotation" of S; the STATEMENTLIST S' is said to be "annotated".

3. A Formal Proof Theory for Programs

We will use the formal theory of Hoare [4]. Some minor changes have been made, in particular, the use of the notation {R} S {Q} where Hoare uses R{S}Q.

In the following P, Q and R are formulas, B is a logical expression, S_1 and S_2 are statementlists and $Q_e^{x_i}$ denotes the formula resulting from replacing all free occurrences of variable x_i by expression e.

Axioms

null $\{R\}\ \underline{\text{null}}\ \{R\}$

assignment $\{Q_e^{x_i}\}\ x_i := e\ \{Q\}$

Rules of inference

conditional $\dfrac{\{R \& B\}S_1\{Q\}, \{R \& \neg B\}S_2\{Q\}}{\{R\}\ \underline{\text{if}}\ B\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}}\{Q\}}$

while $\dfrac{\{R \& B\}S_1\{R\}}{\{R\}\ \underline{\text{while}}\ B\ \underline{\text{do}}\ S_1\ \underline{\text{end}}\ \{R \& \neg B\}}$

concatenation $\dfrac{\{R\}S_1\{P\}, \{P\}S_2\{Q\}}{\{R\}S_1;\ S_2\{Q\}}$

preconsequence $\dfrac{R \Rightarrow P, \{P\}S_1\{Q\}}{\{R\}S_1\{Q\}}$

postconsequence $\dfrac{\{R\}S_1\{P\},\ P \Rightarrow Q}{\{R\}S_1\{Q\}}$

4. Verification Tableaus

For a tableau T to be a verification tableau, that is, essentially a proof of the program P(T), the annotation of P(T) must be valid. The conditions for an annotation to be valid are as follows:

(i) For any two contiguous formulas, {R}{Q}, we require that $R \Rightarrow Q$.

(ii) For any STATEMENT S enclosed by formulas R and Q, {R} S {Q}, we require that

(a) if S is null then R and Q are identical;

(b) if S is an ASSIGNMENT

x_i := e

then R is $Q_e^{x_i}$;

(c) if S is a WHILE

while B do $\{R_1\}$ S_1 $\{Q_1\}$ end

(for STATEMENTLIST S_1) then

R_1 is R&B, Q_1 is R and Q is R&¬B;

(d) if S is a CONDITIONAL

```
if B then {R₁} S₁ {Q₁}
     else {R₂} S₂ {Q₂} fi
```

(for STATEMENTLISTS S_1 and S_2) then

R_1 is R&B, R_2 is R&¬B and both Q_1 and Q_2 are Q.

Notice that in checking for validity, the only real work is in checking condition (i); condition (ii) simply requires testing identity of strings (formulas). Notice also that the use of if ... then ... else ... fi rather than simply if ... then ... else makes it easy to identify STATEMENTS.

For instance, without the fi,

{R} if B then ... else {P} S_1 {Q}{U}{V}

could mean

{R} if B then ... else {P} S_1 {Q}{U} fi {V}

or {R} if B then ... else {P} S_1 {Q} fi {U}{V}

Deciding which was meant would add an extra complication to checking validity.

A similar remark could be made about the use of end in the while statement.

5. Equivalence of Verification Tableaus and Formal Proofs

We show that for every formal proof of a program (using the formal theory of section 3), there is a verification tableau, and vice versa.

THEOREM 1 For any statementlist S, and formulas A and Q, if {R} S {Q} is provable then there is a STATEMENTLIST S' such that P(S')= S and {R} S' {Q} is validly annotated.

Proof By induction on the structure of the proof of {R} S {Q}. In the simplest proofs, {R} S {Q} is simply an axiom, either {R} null {R} or $\{Q_e^{x_i}\}$ x_i := e {Q}. In both cases {R} S {Q} is validly annotated and P(S) = S (conditions (ii)(a) & (b) in section 4). Otherwise, the last step in the proof of {R} S {Q} must use one of the five rules of inference, and these five cases are checked as follows.

Conditional: S must be of the form if B then S_1 else S_2 fi and we must have previously proved

{R & B} S_1 {Q} and {R & ¬B} S_2 {Q}.

By the induction hypothesis there are STATEMENTLISTS S_1' and S_2' such that $P(S_1') = S_1$ and $P(S_2') = S_2$ and {R & B} S_1' {Q} and {R & ¬B} S_2' {Q} are validly annotated. In that case letting S' be

```
if B then {R & B} $S_1'$ {Q}
     else {R & ¬B} $S_2'$ {Q} fi
```

we see that P(S') = S and {R} S' {Q} is validly annotated (see condition (ii)(d) in section 4).

while: Here S is while B do S_1 end, Q is R & $\neg$B, and we have previously proved

$\{R \,\&\, B\}\ S_1\ \{R\}$.

By the induction hypothesis $\{R \,\&\, B\}\ S_1'\ \{R\}$ is validly annotated and $P(S_1') = S_1$. We let S' be while B do $\{R \,\&\, B\}\ S_1'\ \{R\}$ end, so that $P(S') = S$, and we see that $\{R\}\ S_1'\ \{R \,\&\, \neg B\}$ is validly annotated (condition (ii)(c)).

concatenation: Here S is $S_1;S_2$ and for some formula P we have already proved that

$\{R\}\ S_1\ \{P\}$ and $\{P\}\ S_2\ \{Q\}$.

By the induction hypothesis $\{R\}\ S_1'\ \{P\}$ and $\{P\}\ S_2'\ \{Q\}$ are validly annotated and $P(S_1') = S_1$ and $P(S_2') = S_2$. Then, letting $S' = S_1'\ \{P\}\ S_2'$ we see that $P(S') = S_1;\ S_2 = S$ and $\{R\}\ S'\ \{Q\}$ (=$\{R\}\ S_1'\ \{P\}\ S_2'\ \{Q\}$) is validly annotated.

preconsequence : Here there must be a formula P such that we have previously proved $R \Rightarrow P$ and $\{P\}\ S\ \{Q\}$. By the induction hypothesis there is a STATEMENTLIST S" such that $\{P\}\ S''\ \{Q\}$ is validly annotated and $P(S'') = S$. Letting $S' = \{P\}S''$ we see that $P(S') = S$ and $\{R\}\ S\ \{Q\}$ is validly annotated (by condition (i)).

postconsequence: As for preconsequence, but with $S' = S''\{P\}$.

This completes the proof. □

COROLLARY 1 For every proof of a program there is a corresponding valid annotation of the program, i.e. a verification tableau.

THEOREM 2 For every STATEMENTLIST S if $\{R\}\ S\ \{Q\}$ is validly annotated then there is a proof of $\{R\}\ P(S)\ \{Q\}$.

Proof By structural induction on STATEMENTLIST S. If S is a STATEMENT there are the following cases:

(i) S is null. Then if $\{R\}\ S\ \{Q\}$ is validly annotated Q must be identical to R, and $\{R\}\ P(S)\ \{Q\}$ is then an axiom.

(ii) S is an assignment

x_i := e.

Then R must be $Q_e^{x_i}$, and again

{R} P(S){Q} is an axiom.

(iii) S is a WHILE

while B do {R & B} S_1 {R} end

and Q is R & ¬B.

By the induction hypothesis, since {R & B} S_1 {R} is validly annotated there is a proof of {R & B} $P(S_1)$ {R}. Using the while rule there is then a proof of {R} P(S) {R & ¬B}.

(iv) S is a CONDITIONAL

```
if B then {R & B} S_1 {Q}
     else {R & ¬B} S_2 {Q} fi.
```

By the induction hypothesis there are proofs of {R & B} $P(S_1)$ {Q} and {R & ¬B} $P(S_2)$ {Q}. Using the conditional rule there is then a proof of {R} P(S) {Q}.

If S is not a STATEMENT there are three cases:

(a) S is {P}S' for formula P and STATEMENTLIST S'. Since {R} S {Q} is validly annotated, $R \Rightarrow P$ and by the induction hypothesis there is a proof of {P} P(S') {Q}. By the preconsequence rule there is then a proof of {R} P(S) {Q}, since P(S') = P(S).

(b) S is S'{P}. This case is treated as in (a), but using the post-consequence rule.

(c) S is S_1 {P} S_2 for formula P and STATEMENTLISTS S_1 and S_2. Since {R} S {Q} is validly annotated, {R} S_1 {P} and {P} S_2 {Q} are also. Then by the induction hypothesis there are proofs of {R} $P(S_1)$ {P} and {P} $P(S_2)$ {Q}. Using the concatenation rule there is then a proof of {R} $P(S_1)$; $P(S_2)$ {Q}, i.e. {R} $P(S)$ {Q}.

□

6. Generalizations

We have shown how to construct verification tableaus for only the simplest of Hoare's formal theories. Other more complicated language features can be handled by the axiomatic approach. Can we define classes of verification tableaus for all these systems?

Some of the things which make the definition of verification tableaus go so smoothly in this paper are non-essential, for instance, the use of the {R} S {Q} notation rather than R {S} Q, and the use of *if then else fi* rather than *if then else*. Even the close correspondence between the grammars for programs and tableaus and the rules of inference is not essential, though it does simplify the proofs of Theorems 1 and 2.

The essential property of a verification tableau is that each piece of program appears once, so that the tableau can represent the proof of only one property for each piece of program. It happens that, for the formal system presented here, this is also true of proofs. We can prove two properties of some piece of program, but there is no way to use both these properties to prove one property of the whole program.

This is no longer true if for example we add to the formal system either of the rules:

$$\text{conjunction: } \frac{\{R_1\}\ S\ \{Q_1\}, \{R_2\}\ S\ \{Q_2\}}{\{R_1\ \&\ R_2\}\ S\ \{Q_1\ \&\ Q_2\}}\ .$$

$$\text{disjunction: } \frac{\{R_1\}\ S\ \{Q_1\}, \{R_2\}\ S\ \{Q_2\}}{\{R_1 \vee R_2\}\ S\ \{Q_1 \vee Q_2\}}\ .$$

For such augmented systems Theorem 1 would not apply. These rules are not necessary since we can show by induction on proof lengths that for example, if $\{R_1\}$ S $\{Q_1\}$ and $\{R_2\}$ S $\{Q_2\}$ are provable in the formal system of Section 3 so is $\{R_1 \vee R_2\}$ S $\{Q_1 \vee Q_2\}$.

In a proof of a program P, any property {R} S {Q} that is proved will be called a basic property (of the given proof) if S is a constituent of P and the property is not proved using other properties of this constituent S. (Thus properties derived using the consequence rules are not basic properties.) Given any Hoare-style axiom system, proofs can be represented by verification tableaus if

(i) only properties of constituents of programs are needed,

and (ii) only one basic property per constituent is needed.

For example, suppose we add to the programming language the very useful repeat statement:

```
repeat <statementlist>;
        exit on <logical expression>;
       <statementlist> end.
```

The statement **repeat** S_1; **exit on** B; S_2 **end** is equivalent to S_1; **while** ¬B **do** S_2; S_1 **end**. This statement helps overcome loop initialisation and exit problems (see Conway and Gries [2], p.54), and can be used for removing go-to statements from programs without increasing the size of programs (see Ashcroft and Manna [1]).

A straightforward attempt at formulating a rule of inference for the repeat statement gives:

$$\underline{\text{repeat}}: \quad \frac{\{R\}\ S_1\ \{Q\}, \{Q \& \neg B\} S_2; S_1 \{Q\}}{\{R\}\ \textbf{repeat}\ S_1; \textbf{exit on}\ B; S_2\ \textbf{end}\ \{Q \& B\}} \ .$$

Unfortunately, this rule violates both the above conditions; $S_2;S_1$ is not a program constituent, and also, since proving $\{Q \mathbin{\&} \neg B\}S_2;S_1\{Q\}$ will involve proving $\{P\}\ S_1\ \{Q\}$ for some formula P, we may have to prove two basic properties of S_1 to use the rule.

It can be shown however that the following rule is just as powerful:

$$\underline{\text{repeat}'}:\ \frac{\{R\}\ S_1\ \{Q\},\{Q \mathbin{\&} \neg B\}\ S_2\ \{R\}}{\{R\}\ \underline{\text{repeat}}\ S_1;\underline{\text{exit}}\ \underline{\text{on}}\ B;S_2\ \underline{\text{end}}\ \{Q \mathbin{\&} B\}}\ .$$

This rule satisfies the conditions and therefore proofs using it can be represented by verification tableaus. (The necessary modification of the definition of tableaus, and the changes needed in the proofs of Theorems 1 and 2 are straightforward exercises.)

References

[1] Ashcroft, E.A., and Manna, Z., "Translating Program Schemas to While Schemas", SIAM J., Comput., vol.4, No.2, June 1975, pp.125-146.

[2] Conway, R., and Gries, D., "An Introduction to Programming", Second Edition, Winthrop, 1975.

[3] Floyd, R.W., "Assigning Meaning to Programs", in Proc. Symposia in Appl. Math. (1966), vol.19, Amer. Math. Soc. 1967, pp.19-32.

[4] Hoare, C.A.R., "An Axiomatic Basis for Computer Programming", Comm. ACM 12 (1969), pp.581.

[5] Manna, Z., and Vuillemin, J., "Fixpoint Approach to the Theory of Computation:, Comm. ACM 15 (1972), pp.528-536.