Acta Informatica 3, 243—263 (1974)
© by Springer-Verlag 1974

Axiomatic Approach to Total Correctness of Programs

Zohar Manna and Amir Pnueli

Received June 20, 1973

Summary. We present here an axiomatic approach which enables one to prove by formal methods that his program is "totally correct" (i.e., it terminates and is logically correct—does what it is supposed to do). The approach is similar to Hoare's approach [3] for proving that a program is "partially correct" (i.e., that whenever it terminates it produces correct results). Our extension to Hoare's method lies in the possibility of proving both correctness and termination by one unified formalism. One can choose to prove total correctness by a single step, or by incremental proof steps, each step establishing more properties of the program.

I. Introduction

The class of programs we treat in this paper is the class of *while programs* which are written in an Algol-like language allowing assignment statements, conditional statements, compound statements and while statements. Go to statements and procedure calls are explicitly excluded, but this restriction does not seem essential and can be removed by appropriate extension of the results presented here.

For this class of programs we present a method, based on Hoare's [3] axiomatic approach, by which both termination and correctness can be proved.

To review Hoare's notation, he uses theorems of the form

$$\{p(\bar{x})|B|q(\bar{x})\}$$

(where p, q are predicates, and B is a program segment) to mean that for every $\bar{x}$, if $p(\bar{x})$ holds prior to execution of B and the execution of B terminates, then the resulting values after execution will satisfy $q(\bar{x})$. His system consists of several basic theorems—*axioms*—describing the transformation on program variables effected by simple statements, and *inference rules* by which theorems for small segments can be combined into one theorem for a larger segment. Among those are a concatenation rule, a conditional rule, and a while rule. If starting from the axioms about the simple statements of a program P, and employing inference rules one is able to deduce

$$\{\phi(\bar{x})|P|\psi(\bar{x})\},$$

then one has shown in fact the partial correctness of P with respect to ϕ and ψ, i.e., that for every $\bar{x}$ satisfying $\phi(\bar{x})$ for which the execution of P terminates, $\psi(\bar{x})$ holds for the resulting variables' values. A formal exposition of this approach is presented in Igarashi, London and Luckham [5].

The theorems we will be using in our method are of the form

$$\langle p(\bar{x}) | B | q(\bar{x}, \bar{x}')\rangle$$

to mean that for every $\bar{x}$, if $p(\bar{x})$ holds prior to execution of B, then the execution of B terminates and, denoting the set of resulting values by $\bar{x}'$, $q(\bar{x}, \bar{x}')$ holds. We call $p(\bar{x})$ the *precondition of* B, and $q(\bar{x}, \bar{x}')$ the *postcondition of* B or the *outcome predicate of* B. If using our inference rules one is able to deduce

$$\langle \phi(\bar{x}) | P | \psi(\bar{x}, \bar{x}')\rangle$$

then one has shown in fact that P is totally correct with respect to ϕ and ψ, i.e., that for every $\bar{x}$ satisfying $\phi(\bar{x})$, the execution of P terminates and $\psi(\bar{x}, \bar{x}')$ holds between the initial values $\bar{x}$ and the resulting values $\bar{x}'$. If one is only interested in proving termination over ϕ it is sufficient to show

$$\langle \phi(\bar{x}) | P | T\rangle,$$

where T is the identically true predicate.

An apparent advantage of this notation is the ability to express relations between values of variables before and after the execution. In principle the expression of such relations is also possible in the regular Hoare's notation by the introduction of free ("snapshot") variables. Thus

$$\{p(\bar{x}) \wedge \bar{x} = \bar{x}_0 | B | q(\bar{x}_0, \bar{x})\}$$

expresses the fact that if B terminates and $\bar{x}_0$ denotes the set of values before its execution, satisfying $p(\bar{x}_0)$, then after execution $q(\bar{x}_0, \bar{x})$ holds, where $\bar{x}$ here denotes the set of values after execution. Such free variables are implicitly assumed to be universally quantified over the theorem.

To illustrate the need for expressing the relation of variables values before and after execution we present the following "do-nothing" program P over the natural numbers:

```
P:    $x_2 \leftarrow 0$;
      while $x_1 > 0$ do
      a: begin
            b: while $x_1 > 0$ do
                  begin $x_1 \leftarrow x_1 - 1$;
                        $x_2 \leftarrow x_2 + 1$
                  end;
            c: $x_2 \leftarrow x_2 - 1$;
            d: while $x_2 > 0$ do
                  begin $x_1 \leftarrow x_1 + 1$;
                        $x_2 \leftarrow x_2 - 1$
                  end
         end a
```

In order to prove termination of the above program one has to follow closely the interrelations between x_1 and x_2. Thus it is very helpful to be able to express what happens across the while statement b by

$$\langle x_1 > 0 \wedge x_2 = 0 | b | x_1' = 0 \wedge x_2' > 0 \wedge x_2' = x_1\rangle$$

and similarly for d

$$\langle x_1 = 0 \wedge x_2 \geqq 0 | \, d \, | \, x_2' = 0 \wedge x_1' = x_2 \wedge x_1' \geqq 0 \rangle .$$

These two, tied together with the subtraction in c can be combined (using the "concatenation rule") to give

$$\langle x_1 > 0 \wedge x_2 = 0 | \, a \, | \, x_2' = 0 \wedge x_1' < x_1 \wedge x_1' \geqq 0 \rangle$$

which, using the "while rule", guarantees termination of the outer loop.

There are several topics one could consider under the heading of proper termination of a program. Not only should we forbid endless loops but we should insist that each individual statement be "properly" executed, generating no faults such as overflow, underflow, zero division, subscript range violation and similar other disrupting mishaps. In this paper we chose to treat fully only one of these issues, namely, ensuring against endless execution which is a global property of the program. The other topics, which are local in nature, are mentioned in Section V at the end of the paper where we indicate how some of the relevant rules should be modified in order to guard against these faults.

In our restricted language, the while statement is the only possible loop-producing mechanism; therefore it is the while statement rule which should be studied in order to prevent endless execution.

Consider a general while statement

$$L\text{: } \textbf{while } t(\bar{x}) \textbf{ do } B.$$

The termination property of such a while statement is proved by establishing a *convergence function* $u(\bar{x})$ mapping the program variable's domain X into a range W. W is required to be a well-founded set with respect to the relation $\prec$, *i.e.*, allows no infinitely descending sequence $w_1, w_2, \ldots$ such that $w_i \in W$ and $w_i \succ w_{i+1}$ for each i. Such $u(\bar{x})$ is a convergence function for the block B if on subsequent executions of B the value of $u(\bar{x})$ decreases. Thus if we can prove, using our inference rules that

$$(*) \qquad \langle p(\bar{x}) \wedge t(\bar{x}) | \, B \, | \, p(\bar{x}') \wedge u(\bar{x}) \succ u(\bar{x}') \rangle$$

this ensures that the while statement having B as its body converges. The principle employed here for ensuring convergence is identical to the one used by Floyd (1967) in his method for proving termination of flowchart programs[1].

The need to compare values of the convergence function before and after execution as required by (*) was the main motivation to the notational extension. However, it is not sufficient to consider just a single while statement and accept (*) as the general form. In general, we have to consider a nested structure of loops within loops where information about the outer loop convergence function should be transferred across the inner loop. Thus the typical theorem to be proved for a while body will be of the form:

$$(**) \qquad \langle p(\bar{x}) \wedge t(\bar{x}) | \, B \, | \, p(\bar{x}') \wedge q(\bar{x}, \bar{x}') \wedge u(\bar{x}) \succ u(\bar{x}') \rangle ,$$

[1] One should realize that in order to accommodate the concepts of well-foundedness, functions and their definedness, some extensions to the logical formalism are unavoidable. Possible extensions are the use of three valued logic with the "undefined" value or conversion from the function calculus into relation calculus.

where $p(\bar{x})$ is the "loop-invariant" and $q(\bar{x}, \bar{x}')$ consists of information about convergence functions of outer loops and also $\bar{x}, \bar{x}'$ relations useful for proving correctness. Consequently, the general form of the conclusion to the while rule will be

$$\langle p(\bar{x}) |\ \textbf{while}\ t(\bar{x})\ \textbf{do}\ B\ | p(\bar{x}') \wedge q(\bar{x}, \bar{x}') \wedge \neg t(\bar{x}')\rangle$$

where the $u(\bar{x})$ part has been discarded (having fulfilled its task) and $p(\bar{x}')$ and $q(\bar{x}, \bar{x}')$ are retained for continuation of the proof.

We illustrate this typical case by the following simple program

```
         $i \leftarrow n$;    $s \leftarrow 0$;
      $d$: while $i > 0$ do
              $c$: begin $j \leftarrow m$;
                 $b$: while $j > 0$ do
                      $a$: begin $s \leftarrow s+1$;
                                 $j \leftarrow j-1$
                           end;
                 $i \leftarrow i-1$
                 end
```

The theorem we wish to establish for statement a is:

$$\langle [s=(n-i)\cdot m+(m-j) \wedge i>0] \wedge j>0$$
$$|a|$$
$$[s'=(n-i')\cdot m+(m-j') \wedge i'>0] \wedge i=i'>0 \wedge j>j' \geq 0\rangle$$

which when written in the general form:

$$\langle p(\bar{x}) \wedge t(\bar{x}) |\ a | p(\bar{x}') \wedge q(\bar{x}, \bar{x}') \wedge u(\bar{x}) \succ u(\bar{x}')\rangle$$

should identify:

$$p(\bar{x}) \equiv [s=(n-i)\cdot m+(m-j) \wedge i>0]$$
$$t(\bar{x}) \equiv [j>0]$$
$$q(\bar{x}, \bar{x}') \equiv [i=i'>0]$$
$$u(\bar{x}) \succ u(\bar{x}') \equiv [j>j' \geqq 0]$$

where the well-founded domain is the set of non-negative integers. Note that p is the invariant and u is the convergence function of the inner while statement b. q is the property of the convergence function of the outer while statement d which has to be established across the statement b; this together with the next statement $i \leftarrow i-1$ establishes the strict decrease of i and hence the convergence of d.

The final feature that we would like to add is to allow $u(\bar{x})$ to become undefined at the last execution of the while body B. With this addition, the required form of the hypothesis is:

(***) $$\langle p(\bar{x}) \wedge t(\bar{x}) |\ B\ | p(\bar{x}') \wedge q(\bar{x}, \bar{x}') \wedge [\neg t(\bar{x}') \vee u(\bar{x}) \succ u(\bar{x}')]\rangle.$$

Here the last conjunct of the postcondition should read: "After execution, either $t(\bar{x}')$ is false (and hence we are sure that this is the last repetition of the while body—and termination is ensured), or both $u(\bar{x})$ and $u(\bar{x}')$ are defined and $u(\bar{x})$ is strictly larger than $u(\bar{x}')$". This feature is required to provide for the recurrent case in which the "natural" convergence function becomes undefined on the last execution.

In presenting the inference rules below we will usually give first the general rule, and then present several derived rules which are convenient for use in frequently arising cases. We should remark in passing that although our rules are by no means unique or even presumably best, we tried to select the variation which will fit most conveniently and naturally the variety of different cases which arise in practice. Many other variations and approaches obviously exist.

II. Axioms and Inference Rules

All the inference rules will be described by a set of antecedents (conditions under which the rule is applicable) followed by a consequence which is the theorem deduced. Each of the antecedents is either a theorem (which should have been previously established) or a logical claim.

We first present the straightforward rules dealing with assignment, conditionals and composition and leave the while rule, which is the most complicated, to the end.

1. Assignment Axiom

$$\boxed{\dfrac{\forall \bar{x}_1, \bar{x}_2 [p(\bar{x}_1) \wedge \bar{x}_2 = f(\bar{x}_1) \supset q(\bar{x}_1, \bar{x}_2)]}{\langle p(\bar{x}) \,|\, \bar{x} \leftarrow f(\bar{x}) \,|\, q(\bar{x}, \bar{x}') \rangle.}}$$

This rule is essentially an axiom since it uses only logical claims to create a theorem. Since f is considered a basic function (not a user-defined procedure), termination is as obvious as correctness.

2. Conditional Rules

2.1. If-then-else

$$\boxed{\dfrac{\begin{array}{l}\langle p(\bar{x}) \wedge t(\bar{x}) \,|\, B_1 \,|\, q(\bar{x}, \bar{x}') \rangle \\ \langle p(\bar{x}) \wedge \neg t(\bar{x}) \,|\, B_2 \,|\, q(\bar{x}, \bar{x}') \rangle\end{array}}{\langle p(\bar{x}) \,|\, \textbf{if}\, t(\bar{x})\, \textbf{then}\, B_1\, \textbf{else}\, B_2 \,|\, q(\bar{x}, \bar{x}') \rangle.}}$$

The rule should read as follows: If under $p(\bar{x})$ we succeded in showing separately that whether we proceed with $t(\bar{x})$ true to execute B_1 or with $t(\bar{x})$ false to execute B_2, $q(\bar{x}, \bar{x}')$ holds in both cases, then clearly if we cross the combined conditional statement with $p(\bar{x})$ initially true, we come out with $q(\bar{x}, \bar{x}')$. Since the antecedents claim that both B_1 and B_2 when executed under the proper conditions terminate, the termination of the conditional statement under $p(\bar{x})$ follows.

2.2. If—do

$$\boxed{\dfrac{\begin{array}{l}\langle p(\bar{x}) \wedge t(\bar{x}) \,|\, B \,|\, q(\bar{x}, \bar{x}') \rangle \\ \forall \bar{x} [p(\bar{x}) \wedge \neg t(\bar{x}) \supset q(\bar{x}, \bar{x})]\end{array}}{\langle p(\bar{x}) \,|\, \textbf{if}\, t(\bar{x})\, \textbf{do}\, B \,|\, q(\bar{x}, \bar{x}') \rangle.}}$$

This is the one clause (empty **else**) conditional statement. Note that if we do not execute B we have to verify that $q(\bar{x}, \bar{x})$ holds.

The following five rules are composition rules. Some of them facilitate composition of segments while the others allow composition of predicates.

3. Concatenation Rules

3.1. Basic Rule

$$\boxed{\begin{array}{ll} \langle p_1(\bar{x}) \mid B_1 \mid q_1(\bar{x}, \bar{x}')\rangle & \text{(i)} \\ \langle p_2(\bar{x}) \mid B_2 \mid q_2(\bar{x}, \bar{x}')\rangle & \text{(ii)} \\ \forall \bar{x}_1, \bar{x}_2 [q_1(\bar{x}_1, \bar{x}_2) \supset p_2(\bar{x}_2)] & \text{(iii)} \\ \forall \bar{x}_1, \bar{x}_2, \bar{x}_3 [q_1(\bar{x}_1, \bar{x}_2) \wedge q_2(\bar{x}_2, \bar{x}_3) \supset q_3(\bar{x}_1, \bar{x}_3)] & \text{(iv)} \\ \hline \langle p_1(\bar{x}) \mid B_1; B_2 \mid q_3(\bar{x}, \bar{x}')\rangle. & \end{array}}$$

Condition (iii) ensures that the state after execution of B_1 satisfies p_2 — the needed precondition for B_2. Condition (iv) characterizes $q_3(\bar{x}_1, \bar{x}_3)$ as a transfer relation between $\bar{x}_1$ before execution and $\bar{x}_3$ after execution of B_1; B_2. It requires an intermediate $\bar{x}_2$ which temporarily appears after execution of B_1 and before execution of B_2.

3.2. Derived Rule

A simpler derived rule for concatenation is

$$\boxed{\begin{array}{l} \langle p_1(\bar{x}) \mid B_1 \mid p_2(\bar{x}') \wedge q_1(\bar{x}, \bar{x}')\rangle \\ \langle p_2(\bar{x}) \mid B_2 \mid p_3(\bar{x}') \wedge q_2(\bar{x}, \bar{x}')\rangle \\ \forall \bar{x}_1, \bar{x}_2, \bar{x}_3 [q_1(\bar{x}_1, \bar{x}_2) \wedge q_2(\bar{x}_2, \bar{x}_3) \supset q_3(\bar{x}_1, \bar{x}_3)] \\ \hline \langle p_1(\bar{x}) \mid B_1; B_2 \mid p_3(\bar{x}') \wedge q_3(\bar{x}, \bar{x}')\rangle. \end{array}}$$

The derivation follows from substitution of $p_2(\bar{x}') \wedge q_1(\bar{x}, \bar{x}')$ for $q_1(\bar{x}, \bar{x}')$, $p_3(\bar{x}') \wedge q_2(\bar{x}, \bar{x}')$ for $q_2(\bar{x}, \bar{x}')$, and $p_3(\bar{x}') \wedge q_3(\bar{x}, \bar{x}')$ for $q_3(\bar{x}, \bar{x}')$.

4. Consequence Rules

$$\boxed{\begin{array}{l} \langle r(\bar{x}) \mid B \mid q(\bar{x}, \bar{x}')\rangle \\ \forall \bar{x} [p(\bar{x}) \supset r(\bar{x})] \\ \hline \langle p(\bar{x}) \mid B \mid q(\bar{x}, \bar{x}')\rangle \end{array}}$$

and

$$\boxed{\begin{array}{l} \langle p(\bar{x}) \mid B \mid s(\bar{x}, \bar{x}')\rangle \\ \forall x_1, x_2 [s(\bar{x}_1, \bar{x}_2) \supset q(\bar{x}_1, \bar{x}_2)] \\ \hline \langle p(\bar{x}) \mid B \mid q(\bar{x}, \bar{x}')\rangle. \end{array}}$$

The validity of the rules is obvious when we consider the interpretation of a theorem.

5. *Or Rule*

$$\boxed{\begin{array}{l} \langle p_1(\bar{x}) \,|\, B \,|\, q(\bar{x}, \bar{x}') \rangle \\ \langle p_2(\bar{x}) \,|\, B \,|\, q(\bar{x}, \bar{x}') \rangle \\ \hline \langle p_1(\bar{x}) \vee p_2(\bar{x}) \,|\, B \,|\, q(\bar{x}, \bar{x}') \rangle. \end{array}}$$

This rule creates the possibility for proof by case analysis.

6. *And Rule*

$$\boxed{\begin{array}{l} \langle p(\bar{x}) \,|\, B \,|\, q_1(\bar{x}, \bar{x}') \rangle \\ \langle p(\bar{x}) \,|\, B \,|\, q_2(\bar{x}, \bar{x}') \rangle \\ \hline \langle p(\bar{x}) \,|\, B \,|\, q_1(\bar{x}, \bar{x}') \wedge q_2(\bar{x}, \bar{x}') \rangle. \end{array}}$$

This rule enables one to generate incremental proofs, by proving separately two independent properties, and then combining them by the *and* rule. Note that it is sufficient to prove termination for only one of the antecedents' conditions of the *and* rule.

7. *Assumption Introduction Rule*

$$\boxed{\begin{array}{l} \langle p(\bar{x}) \,|\, B \,|\, q(\bar{x}, \bar{x}') \rangle \\ \hline \langle p(\bar{x}) \wedge s(\bar{x}) \,|\, B \,|\, q(\bar{x}, \bar{x}') \wedge s(\bar{x}) \rangle. \end{array}}$$

Note that although $s(\bar{x})$ appears in the postcondition, it states a property of $\bar{x}$, before the execution. Thus its validity is ensured by the appearance of $s(\bar{x})$ in the precondition.

8. *While Rules*

8.1. Basic Rule

$$\boxed{\begin{array}{ll} \langle p(\bar{x}) \wedge t(\bar{x}) \,|\, B \,|\, q(\bar{x}, \bar{x}') \wedge [\neg t(\bar{x}') \vee u(\bar{x}) \succ u(\bar{x}')] \rangle & \text{(i)} \\ \forall \bar{x}_1, \bar{x}_2 [q(\bar{x}_1, \bar{x}_2) \wedge t(\bar{x}_2) \supset p(\bar{x}_2)] & \text{(ii)} \\ \forall \bar{x}_1, \bar{x}_2, \bar{x}_3 [q(\bar{x}_1, \bar{x}_2) \wedge q(\bar{x}_2, \bar{x}_3) \supset q(\bar{x}_1, \bar{x}_3)] & \text{(iii)} \\ \forall \bar{x} [p(\bar{x}) \wedge \neg t(\bar{x}) \supset q(\bar{x}, \bar{x})] & \text{(iv)} \\ \hline \langle p(\bar{x}) \,|\, \textbf{while } t(\bar{x}) \textbf{ do } B \,|\, q(\bar{x}, \bar{x}') \wedge \neg t(\bar{x}') \rangle. & \end{array}}$$

Termination of a looping while statement is ensured by the existence of a convergence function $u(\bar{x})$, following the arguments presented in the introduction. As mentioned there the demand for the convergence function being defined and strictly decreasing can be relaxed for the last execution of B. Accordingly, we require in (i) the alternatives of either $t(\bar{x}')$ false, implying immediate termination, or the existence and strict decrease of the convergence function which will also ultimately ensure termination. Note that (i) establishes the termination of B itself.

Condition (ii) requires that having executed B at least once, and having $t(\bar{x}')$ correct at this instance, logically establishes $p(\bar{x}')$. $p(\bar{x})$ is exactly the condition we need to use (i) once more and thus propagate the validity of q for all subsequent executions.

Condition (iii) ensures that $q(\bar{x}, \bar{x}')$ is transitive. Therefore, once we showed in (i) that it holds over one execution of B, it follows that it will hold over any number of repeated executions of B. Consequently, it will hold over the repeating while statement.

Condition (iv) deals with the case of the initially vacant while statement, where B did not execute even once. There also we wish to establish the final outcome $q(\bar{x}, \bar{x}')$.

8.2. The First Derived Rule

The first derived rule is obtained by removing $\neg t(\bar{x}')$, the "last-loop relaxation" disjunct, strengthening the hypothesis (i):

$$\begin{array}{lr}
\langle p(\bar{x}) \wedge t(\bar{x}) \mid B \mid q(\bar{x}, \bar{x}') \wedge [u(\bar{x}) \succ u(\bar{x}')]\rangle & \text{(i)} \\
\forall \bar{x}_1, \bar{x}_2 [q(\bar{x}_1, \bar{x}_2) \wedge t(\bar{x}_2) \supset p(\bar{x}_2)] & \text{(ii)} \\
\forall \bar{x}_1, \bar{x}_2, \bar{x}_3 [q(\bar{x}_1, \bar{x}_2) \wedge q(\bar{x}_2, \bar{x}_3) \supset q(\bar{x}_1, \bar{x}_3)] & \text{(iii)} \\
\forall \bar{x} [p(\bar{x}) \wedge \neg t(\bar{x}) \supset q(\bar{x}, \bar{x})] & \text{(iv)} \\
\hline
\langle p(\bar{x}) \mid \textbf{while } t(\bar{x}) \textbf{ do } B \mid q(\bar{x}, \bar{x}') \wedge \neg t(\bar{x}')\rangle. &
\end{array}$$

Obviously when we can use this rule with no excessive discomfort it is preferable to do so.

8.3. The Second Derived Rule

As a second derived while rule we have

$$\begin{array}{lr}
\langle p(\bar{x}) \wedge t(\bar{x}) \mid B \mid p(\bar{x}') \wedge q(\bar{x}, \bar{x}') \wedge [u(\bar{x}) \succ u(\bar{x}')]\rangle & \text{(i)} \\
\forall \bar{x}_1, \bar{x}_2, \bar{x}_3 [q(\bar{x}_1, \bar{x}_2) \wedge q(\bar{x}_2, \bar{x}_3) \supset q(\bar{x}_1, \bar{x}_3)] & \text{(ii)} \\
\forall \bar{x} [p(\bar{x}) \wedge \neg t(\bar{x}) \supset q(\bar{x}, \bar{x})] & \text{(iii)} \\
\hline
\langle p(\bar{x}) \mid \textbf{while } t(\bar{x}) \textbf{ do } B \mid p(\bar{x}') \wedge q(\bar{x}, \bar{x}') \wedge \neg t(\bar{x}')\rangle &
\end{array}$$

which is obtained by replacing $q(\bar{x}, \bar{x}')$ by $p(\bar{x}') \wedge q(\bar{x}, \bar{x}')$ in the first derived while rule.

III. Examples

We present below two examples for which we can prove total correctness by our method. Because of the amount of detail involved we will concentrate on proving termination.

Example 1. The following while program over the integers is supposed to compute the greatest common divisor of two positive integers x_1 and x_2 —$gcd(x_1, x_2)$—leaving the result in x_1. To refer to program segments we use ordinary Algol labels.

```
P: START
  f: while $x_1 \neq x_2$ do
        e: begin
           b: while $x_1 > x_2$ do a: $x_1 \leftarrow x_1 - x_2$;
           d: while $x_2 > x_1$ do c: $x_2 \leftarrow x_2 - x_1$
           end
  HALT.
```

We would like to prove that the program P is totally correct with respect to

$$\phi(x_1, x_2) \equiv [x_1 > 0 \wedge x_2 > 0]$$

and

$$\psi(x_1, x_2, x_1', x_2') \equiv [x_1' = gcd(x_1, x_2)].$$

We prove in detail termination only. The well-founded set we use is the domain of non-negative integers with the ordinary $<$ relation. As the convergence function for all while statements we take

$$u(x_1, x_2) \equiv [x_1 + x_2 \geqq 0].$$

Note that this example is special in the sense that the same convergence function is used for all while statements.

Our proof of termination distinguishes between two cases according to whether $x_1 > x_2$ or $x_1 < x_2$ upon entrance to the compound statement e. In the first case, statement a is executed at least once ($x_1 + x_2$ decreasing), while statement c is executed 0 or more times ($x_1 + x_2$ remaining the same or decreasing). On the second case statement a is never executed ($x_1 + x_2$ remaining the same), while statement c is executed at least once ($x_1 + x_2$ decreasing). We will therefore analyze in our proof these two cases separately (Lemmas a1 to e1 in the first case and Lemmas b2 to e2 in the second case), and then combine their results using the *or* rule.

In all the predicates of the following theorems the conjunction $x_1 > 0 \wedge x_2 > 0$ is omitted. The consequence rules are freely used without any indication.

Lemma a1 (Assignment Rule).

Since $[x_1 > x_2 \wedge x_1' = x_1 - x_2 \wedge x_2' = x_2] \supset [x_1 + x_2 > x_1' + x_2']$ we get

$$\langle x_1 > x_2 | a | x_1 + x_2 > x_1' + x_2' \rangle$$

by the assignment rule.

Lemma b1 (While Rule).

We use the first derived while rule with the following predicates:

$$p(\bar{x}) \equiv t(\bar{x}) \equiv [x_1 > x_2],$$
$$q(\bar{x}, \bar{x}') \equiv u(\bar{x}) \succ u(\bar{x}') \equiv [x_1 + x_2 > x_1' + x_2'].$$

Condition (i) of the while rule is justified by Lemma a1. We obtain

$$\langle x_1 > x_2 | b | x_1 + x_2 > x_1' + x_2' \rangle.$$

Lemma c1 (Assignment Rule).

Since

$$[x_2 > x_1 \wedge x_1' = x_1 \wedge x_2' = x_2 - x_1] \supset [x_1 + x_2 > x_1' + x_2'],$$

we get by the assignment rule

$$\langle x_2 > x_1 | c | x_1 + x_2 > x_1' + x_2' \rangle.$$

Lemma d1 (While Rule).

Assume the following substitution in the first derived rule

$$p(\bar{x}) \equiv T, \quad t(\bar{x}) \equiv [x_2 > x_1],$$
$$q(\bar{x}, \bar{x}') \equiv [x_1 + x_2 \geqq x_1' + x_2'],$$
$$u(\bar{x}) \succ u(\bar{x}') \equiv [x_1 + x_2 > x_1' + x_2'].$$

Condition (i) of the while rule is justified by Lemma c1. We obtain

$$\langle T | d | x_1 + x_2 \geqq x_1' + x_2' \rangle.$$

Note that condition (iv) is satisfied since $x_1 + x_2 \geqq x_1 + x_2$.

Lemma e1 (Concatenation Rule).

Combine Lemmas b1 and d1 and use the concatenation rule with

$$p_1(\bar{x}) \equiv [x_1 > x_2], \quad p_2(\bar{x}) \equiv T$$
$$q_1(\bar{x}, \bar{x}') \equiv q_3(\bar{x}, \bar{x}') \equiv [x_1 + x_2 > x_1' + x_2']$$
$$q_2(\bar{x}, \bar{x}') \equiv [x_1 + x_2 \geqq x_1' + x_2']$$

to obtain

$$\langle x_1 > x_2 | e | x_1 + x_2 > x_1' + x_2' \rangle.$$

We now treat the case of $x_1 < x_2$ upon entrance to e:

Lemma b2 (While Rule).

Under the intial condition $x_1 < x_2$ the while statement b never executes. Therefore we use an axiom

$$\langle F | a | F \rangle$$

which is true for every statement a. Taking

$$t(\bar{x}) \equiv [x_1 > x_2], \quad p(\bar{x}) \equiv [x_1 < x_2],$$

and

$$q(\bar{x}, \bar{x}') \equiv [x_1 + x_2 = x_1' + x_2']$$

(q is obviously reflexive), we obtain

$$\langle x_1 < x_2 | b | x_1' < x_2' \wedge (x_1 + x_2 = x_1' + x_2') \rangle.$$

Lemma c2 (Assignment Rule).

By the assignment rule

$$\langle x_1 < x_2 | c | x_1 + x_2 > x_1' + x_2' \rangle.$$

Lemma d2 (While Rule).

Take in the first derived while rule

$$p(\bar{x}) \equiv t(\bar{x}) \equiv [x_1 < x_2],$$

and

$$q(\bar{x}, \bar{x}') \equiv u(\bar{x}) \succ u(\bar{x}') \equiv [x_1 + x_2 > x_1' + x_2'].$$

Using Lemma c2 we obtain

$$\langle x_1 < x_2 | d | x_1 + x_2 > x_1' + x_2' \rangle.$$

Lemma e2 (Concatenation Rule).

We use the derived concatenation rule with

$$p_1(\bar{x}) \equiv p_2(\bar{x}) \equiv [x_1 < x_2], \quad p_3(\bar{x}) \equiv T$$
$$q_1(\bar{x}, \bar{x}') \equiv [x_1 + x_2 = x_1' + x_2']$$
$$q_2(\bar{x}, \bar{x}') \equiv q_3(\bar{x}, \bar{x}') \equiv [x_1 + x_2 > x_1' + x_2'].$$

Then by combining Lemmas b2 and d2 we obtain

$$\langle x_1 < x_2 | e | x_1 + x_2 > x_1' + x_2' \rangle.$$

Lemma e (Or Rule).

From Lemmas e1 and e2 combined we get

$$\langle x_1 \neq x_2 | e | x_1 + x_2 > x_1' + x_2' \rangle.$$

Lemma f (While Rule).

Take in the second derived while rule

$$t(\bar{x}) \equiv [x_1 \neq x_2], \quad p(\bar{x}) \equiv [x_1 > 0 \wedge x_2 > 0],$$
$$q(\bar{x}, \bar{x}') \equiv T, \quad u(\bar{x}) \succ u(\bar{x}') \equiv [x_1 + x_2 > x_1' + x_2'].$$

Note that $x_1 > 0 \wedge x_2 > 0$ was implicitly assumed in all previous preconditions. Using Lemma e in condition (i) we get:

$$\langle x_1 > 0 \wedge x_2 > 0 | P | x_1' > 0 \wedge x_2' > 0 \wedge x_1' = x_2' \rangle.$$

We have thus shown termination with the additional information that on exit $x_1' = x_2' > 0$. □

Example 2. Partition (Hoare [2]).

The purpose of the program given below is to rearrange the elements of an array A of $n+1$, $n \geqq 2$, real numbers $A[0], \ldots, A[n]$ and to find two integers i and j, such that

$$0 \leqq j < i \leqq n$$

and for the rearranged array

$$\forall a\,\forall b\,[(0 \leqq a < i \wedge j < b \leqq n) \supset A[a] \leqq A[b]].$$

In other words, we would like to rearrange the elements of A into two non-empty partitions such that those in the lower partition $A[0], \ldots, A[i-1]$ are less than or equal to those in the upper partition $A[j+1], \ldots, A[n]$, where $0 \leqq j < i \leqq n$.

```
P: START
s: $r \leftarrow A[n \div 2]$; $(i,j) \leftarrow (0,n)$;
m: while $i \leqq j$ do
      l: begin
            e: begin b: while $A[i] < r$ do a: $i \leftarrow i+1$;
                     d: while $r < A[j]$ do c: $j \leftarrow j-1$
               end;
            k: if $i \leqq j$ do h: begin f: $A[i] \leftrightarrow A[j]$;
                                      g: $(i,j) \leftarrow (i+1, j-1)$
                                 end
         end l;
   HALT.
```

We will prove in detail termination only. Our proof follows the ideas presented in Hoare's [4] informal proof of termination. We introduce the following abbreviations:

$$\alpha(i) \equiv \exists p\,[i \leqq p \leqq n \wedge r \leqq A[p]]$$
$$\beta(j) \equiv \exists q\,[0 \leqq q \leqq j \wedge A[q] \leqq r].$$

These invariants are used to ensure that while i is stepped up and j is stepped down they do not exceed the bounds of n and 0 respectively.

Lemma a (Assignment Rule).

$$\langle \alpha(i) \wedge \beta(j) \wedge A[i] < r$$
$$|a: i \leftarrow i+1|$$
$$\alpha(i') \wedge \beta(j') \wedge j - i \geqq j' - i' \wedge n - i > n - i' \rangle.$$

Clearly $\beta(j)$ validity is invariant since j is not modified by this statement. From $\alpha(i)$ correctness we infer the existence of p which since $A[p] \geqq r$ must be $p > i$, so that we might take the same p to establish $\alpha(i+1) = \alpha(i')$. The statement about $n - i$ decreasing will be used for termination of the while statement b, while the function $j - i$ will be used for proving termination of m. Both are over the domain of non-negative integers.

Lemma b (While Rule).

Using Lemma a with

$$p(\bar{x}) \equiv [\alpha(i) \wedge \beta(j)], \quad t(\bar{x}) \equiv [A[i] < r]$$
$$q(\bar{x}, \bar{x}') \equiv [j - i \geqq j' - i']$$
$$u(\bar{x}) \succ u(\bar{x}') \equiv [n - i > n - i'],$$

we get from the second derived while rule

$$\begin{aligned}&\langle \alpha(i) \wedge \beta(j)\\ &|b\colon \textbf{while } A[i] < r \textbf{ do } a\colon i \leftarrow i+1|\\ &\beta(j') \wedge A'[i'] \geqq r \wedge [j-i \geqq j'-i']\rangle.\end{aligned}$$

Note that we do not need $\alpha(i')$ any more, but will use instead the conclusion of the while's termination $A'[i'] \geqq r$ which also implies $i' \leqq n$.

Lemma c (Assignment Rule).

$$\begin{aligned}&\langle A[i] \geqq r \wedge \beta(j) \wedge A[j] > r\\ &|c\colon j \leftarrow j-1|\\ &\beta(j') \wedge A'[i'] \geqq r \wedge [j-i \geqq j'-i'] \wedge j > j'\rangle.\end{aligned}$$

The function ensuring termination for the inner while d is j, $j \geqq 0$.

Lemma d (While Rule).

Using Lemma c with

$$\begin{aligned}p(\bar{x}) &\equiv [A[i] \geqq r \wedge \beta(j)], \qquad t(\bar{x}) \equiv [r < A[j]]\\ q(\bar{x}, \bar{x}') &\equiv [j-i \geqq j'-i']\\ u(\bar{x}) \succ u(\bar{x}') &\equiv [j > j']\end{aligned}$$

we get from the second derived while rule

$$\begin{aligned}&\langle A[i] \geqq r \wedge \beta(j)\\ &|d\colon \textbf{while } r < A[j] \textbf{ do } c\colon j \leftarrow j-1|\\ &A'[i'] \geqq r \wedge A'[j'] \leqq r \wedge [j-i \geqq j'-i']\rangle.\end{aligned}$$

Lemma e (Concatenation Rule).
Taking

$$\begin{aligned}p_1(\bar{x}) &\equiv [\alpha(i) \wedge \beta(j)]\\ p_2(\bar{x}) &\equiv [A[i] \geqq r \wedge \beta(j)]\\ p_3(\bar{x}) &\equiv [A[j] \leqq r \leqq A[i]]\\ q_1(\bar{x}, \bar{x}') \equiv q_2(\bar{x}, \bar{x}') &\equiv q_3(\bar{x}, \bar{x}') \equiv [j-i \geqq j'-i'],\end{aligned}$$

and combining Lemmas b and d we get from the derived concatenation rule

$$\begin{aligned}&\langle \alpha(i) \wedge \beta(j)\\ &|e\colon \textbf{begin } b;\, d \textbf{ end}|\\ &A'[j'] \leqq r \leqq A'[i'] \wedge [j-i \geqq j'-i']\rangle.\end{aligned}$$

Lemma f (Assignment Rule).

$$\begin{aligned}&\langle A[j] \leqq r \leqq A[i] \wedge i \leqq j\\ &|f\colon A[i] \leftrightarrow A[j]|\\ &A'[i'] \leqq r \leqq A'[j'] \wedge i' \leqq j' \wedge j-i = j'-i'\rangle.\end{aligned}$$

The condition $i \leqq j$ is added since it is known to be true if we enter statement h. Clearly, after exchanging $A[i]$ and $A[j]$ the previous inequalities are reversed.

Lemma g (Assignment Rule).

$$\begin{aligned}&\langle A[i] \leqq r \leqq A[j] \wedge i \leqq j\\ &|g: (i,j) \leftarrow (i+1, j-1)|\\ &i' > j' \vee [j-i > j'-i' \wedge \alpha(i') \wedge \beta(j')]\rangle.\end{aligned}$$

This result is obtained by case analysis: Either $i+1 \leqq j-1$, in which case we have $i < i' \leqq j' < j$ and we can take $p=j$ to establish $\alpha(i')$ and $q=i$ to establish $\beta(j')$. The other case is $i+1 > j-1$ or, in other words, $i' > j'$. For convenience and identification of $p(\bar{x})$ and $q(\bar{x}, \bar{x}')$ we rewrite the outcome predicate of Lemma g as:

$$[(i' \leqq j') \supset \alpha(i') \wedge \beta(j')] \wedge [i' > j' \vee j-i > j'-i'].$$

Lemma h (Concatenation Rule).

Taking

$$\begin{aligned}p_1(\bar{x}) &\equiv [A[j] \leqq r \leqq A[i] \wedge i \leqq j]\\ p_2(\bar{x}) &\equiv [A[i] \leqq r \leqq A[j] \wedge i \leqq j]\\ p_3(\bar{x}) &\equiv [i \leqq j \supset \alpha(i) \wedge \beta(j)]\\ q_1(\bar{x}, \bar{x}') &\equiv [j-i = j'-i']\\ q_2(\bar{x}, \bar{x}') \equiv q_3(\bar{x}, \bar{x}') &\equiv [i' > j' \vee j-i > j'-i']\end{aligned}$$

and combining Lemmas f and g we get from the derived concatenation rule

$$\begin{aligned}&\langle A[j] \leqq r \leqq A[i] \wedge i \leqq j\\ &|h: \textbf{begin } f;\ g \textbf{ end}|\\ &[(i' \leqq j') \supset \alpha(i') \wedge \beta(j')] \wedge [i' > j' \vee j-i > j'-i']\rangle.\end{aligned}$$

Lemma k (If-do Rule).

By Lemma h we get

$$\begin{aligned}&\langle A[j] \leqq r \leqq A[i]\\ &|k: \textbf{if } i \leqq j \textbf{ do } h|\\ &[(i' \leqq j') \supset \alpha(i') \wedge \beta(j')] \wedge [i' > j' \vee j-i > j'-i']\rangle.\end{aligned}$$

Note that in the case where the **do** clause is skipped, we have $i > j$, so that the conclusion is still correct.

Lemma l (Concatenation Rule).

Combining Lemmas e and k we obtain from the derived concatenation rule

$$\begin{aligned}&\langle \alpha(i) \wedge \beta(j)\\ &|l: \textbf{begin } e;\ k \textbf{ end}|\\ &[(i' \leqq j') \supset \alpha(i') \wedge \beta(j')] \wedge [i' > j' \vee j-i > j'-i']\rangle.\end{aligned}$$

Now we are ready to prove termination of the encompassing while statement. We have shown, in fact, that after one execution of l starting with $\alpha(i) \wedge \beta(j)$ true, we either have $i' > j'$ which ensures no more repetitions of l, or have $\alpha(i') \wedge \beta(j')$ true again and a termination function $j-i$ nonnegative and strictly decreasing. Note that in Lemmas a to e we allowed $j-i$ to range over the domain of all integers (possibly negative); it is only on entrance to the while rule that we restrict $j-i$ to the nonnegative integers in order to establish a convergence function. Consequently, it is only now that we need the $i > j$ alternative.

Lemma m (While Rule).

By the basic while rule using Lemma l and with

$$p(\bar{x}) \equiv [\alpha(i) \wedge \beta(j)], \qquad t(\bar{x}) \equiv [i \leqq j]$$
$$\neg t(\bar{x}') \vee u(\bar{x}) \succ u(\bar{x}') \equiv [i' > j' \vee j-i > j'-i'],$$
$$q(\bar{x}, \bar{x}') \equiv [(i' \leqq j') \supset \alpha(i') \wedge \beta(j')]$$

we get

$$\langle \alpha(i) \wedge \beta(j) \,|\, m\colon \textbf{while } i \leqq j \textbf{ do } l \,|\, i' > j' \rangle.$$

Note that $q(\bar{x}, \bar{x}') \wedge t(\bar{x}') \equiv [\alpha(i') \wedge \beta(j')] \equiv p(\bar{x}')$ as required.

Lemma s (Assignment + Concatenation Rules).
Establishes the initial conditions:

$$\langle n \geqq 2 \,|\, s\colon r \leftarrow A\,[n \div 2];\; (i, j) \leftarrow (0, n) \,|\, \alpha(i') \wedge \beta(j') \rangle.$$

Lemma P (Concatenation Rule).
Concatenation of Lemmas m and s yields

$$\langle n \geqq 2 \,|\, P \,|\, i' > j' \rangle,$$

which shows termination of P. □

IV. Incremental Proofs

In the two examples brought above we restricted our efforts to prove just termination of the programs augmented by some simple properties needed to establish termination. A natural question raised by such a procedure is the following: Having proved some properties of the program including termination, how do we prove additional properties without repeating the whole proof process? Can we use some of the already established properties and proof process to facilitate augmentation of additional properties?

In answering these questions we encounter the significant notion of *incremental proofs*. According to this notion, total correctness is established by a refinement method in which one proceeds by incremental steps to add more and more verified information to our understanding of a program, each step benefitting and made easier by the already gained information.

To attack the problem in its full generality we may proceed as follows: Let the sequence of theorems proved in the last deduction be

$$\begin{aligned}
&T_1 \equiv \langle p_1(\bar{x}) \,|\, B_1 \,|\, q_1(\bar{x}, \bar{x}')\rangle \\
&\vdots \\
&T_n \equiv \langle p_n(\bar{x}) \,|\, B_n \,|\, q_n(\bar{x}, \bar{x}')\rangle \\
&T_{n+1} \equiv \langle \phi(\bar{x}) \,|\, P \,|\, \psi(\bar{x}, \bar{x}')\rangle .
\end{aligned}$$

Note that the p_k's need not be distinct. In order to increment the result predicate $\psi(\bar{x}, \bar{x}')$ we have in general to increment each of the intermediate claims (p_k or q_k). Thus we should consider a $p_k(\bar{x}) \wedge \delta p_k(\bar{x})$ instead of $p_k(\bar{x})$, a $q_k(\bar{x}, \bar{x}') \wedge \delta q_k(\bar{x}, \bar{x}')$ instead of $q_k(\bar{x}, \bar{x}')$, a $\phi(\bar{x}) \wedge \delta\phi(\bar{x})$ instead of $\phi(\bar{x})$ and a $\psi(\bar{x}, \bar{x}') \wedge \delta\psi(\bar{x}, \bar{x}')$ instead of $\psi(\bar{x}, \bar{x}')$, where we use the δ notation to denote increments in claims. The following principle may be helpful in order to repeat only *part* of the proof procedure to establish total correctness with respect to the incremented claim set:

Let us enumerate all the *logical equations* appearing as antecedents in the deduction, $E_1, E_2, \ldots, E_m$. Each E_j is then of the form:

$$\forall \bar{z}\,[U(\bar{p}, \bar{q}, \bar{z}) \supset V(\bar{p}, \bar{q}, \bar{z})]$$

where U and V are each conjunctions of p_i's, q_i's and other predicates appearing in the program. The first observation is that it is sufficient to prove the *logical equations* for the incremented claims $p_i \wedge \delta p_i$, $q_i \wedge \delta q_i$ in order to establish a valid deduction for

$$\langle \phi(\bar{x}) \wedge \delta\phi(\bar{x}) \,|\, P \,|\, \psi(\bar{x}, \bar{x}') \wedge \delta\psi(\bar{x}, \bar{x}')\rangle .$$

The second observation is that in order to prove the logical equations for the incremented claims, a certain simplification is possible. Since the p_i's and q_i's appear in U and V only as conjuncts it is possible to write

$$\begin{aligned}
U(\overline{p \wedge \delta p}, \overline{q \wedge \delta q}, \bar{z}) &= U(\bar{p}, \bar{q}, \bar{z}) \wedge \delta U(\bar{p}, \bar{q}, \bar{z}) \\
V(\overline{p \wedge \delta p}, \overline{q \wedge \delta q}, \bar{z}) &= V(\bar{p}, \bar{q}, \bar{z}) \wedge \delta V(\bar{p}, \bar{q}, \bar{z}).
\end{aligned}$$

Observing that we already proved

$$\forall \bar{z}\,[U(\bar{p}, \bar{q}, \bar{z}) \supset V(\bar{p}, \bar{q}, \bar{z})]$$

it only remains to prove

$$\forall \bar{z}\,[U(\bar{p}, \bar{q}, \bar{z}) \wedge \delta U(\bar{p}, \bar{q}, \bar{z}) \supset \delta V(\bar{p}, \bar{q}, \bar{z})]$$

for each of the logical equations. We may summarize this in the following:

Metatheorem 1. *With the notation introduced above, it is sufficient to prove*

$$\forall \bar{z}\,[U(\bar{p}, \bar{q}, \bar{z}) \wedge \delta U(\bar{p}, \bar{q}, \bar{z}) \supset \delta V(\bar{p}, \bar{q}, \bar{z})]$$

for each of the logical equations in a deduction for

$$\langle \phi(\bar{x}) \,|\, P \,|\, \psi(\bar{x}, \bar{x}')\rangle$$

in order to establish

$$\langle \phi(\bar{x}) \wedge \delta\phi(\bar{x}) \,|\, P \,|\, \psi(\bar{x}, \bar{x}') \wedge \delta\psi(\bar{x}, \bar{x}')\rangle .$$

Note for convenience that if $q(\bar{x}, \bar{x}')$ is a transitive relation so is $q(\bar{x}, \bar{x}') \wedge \delta q(\bar{x}, \bar{x}')$, provided that $\delta q(\bar{x}, \bar{x}')$ is transitive. (Note that an increment of the form $\delta q(\bar{x}, \bar{x}') = \delta p(\bar{x}')$ is always transitive.)

We illustrate this principle by the following simple example

Example 3. Consider the program P over the integers

```
a: $(y_1, y_2, y_3) \leftarrow (0, 1, 1)$;
c: while $y_2 \leqq x$ do
        b: $(y_1, y_2, y_3) \leftarrow (y_1 + 1, y_2 + y_3 + 2, y_3 + 2)$
```

which is supposed to compute in y_1 the integral square-root of a nonnegative integer x such that

$$y_1^2 \leqq x < (y_1 + 1)^2.$$

As a first step we present a deduction which shows that for any integer x the program terminates and yields a y_1 such that $x < (y_1 + 1)^2$.

We denote

$$p(\bar{y}) \equiv [y_2 = (y_1 + 1)^2 \wedge y_3 = (2y_1 + 1) \wedge y_3 > 0].$$

(1) $\forall \bar{y}' \{T \wedge [y_1' = 0 \wedge y_2' = 1 \wedge y_3' = 1] \supset p(\bar{y}')\}$ logical equation.

(2) $\langle T | \, a \, | p(\bar{y}') \rangle$ assignment, (1).

(3) $\forall \bar{y}, \bar{y}' \{[p(\bar{y}) \wedge y_2 \leqq x]$
$\wedge [y_1' = y_1 + 1 \wedge y_2' = y_2 + y_3 + 2 \wedge y_3' = y_3 + 2]$
$\supset [p(\bar{y}') \wedge (y_2' > x \vee x - y_2 > x - y_2')]\}$ logical equation.

We are about to prepare the required antecedents for use in the basic while rule with W being the non-negative integers (here $q(\bar{y}, \bar{y}') \equiv p(\bar{y}')$)

(4) Antecedent (i)
$\langle p(\bar{y}) \wedge y_2 \leqq x | \, b \, | p(\bar{y}') \wedge (y_2' > x \vee x - y_2 > x - y_2') \rangle$ assignment, (3).

(5) Antecedent (ii)
$\forall \bar{y}' [p(\bar{y}') \wedge y_2' \leqq x \supset p(\bar{y}')]$ logical equation.

(6) Antecedent (iii)
$\forall \bar{y}, \bar{y}', \bar{y}'' [p(\bar{y}') \wedge p(\bar{y}'') \supset p(\bar{y}'')]$ logical equation.

(7) Antecedent (iv)
$\forall \bar{y} [p(\bar{y}) \wedge y_2 > x \supset p(\bar{y})]$ logical equation.

(8) $\langle p(\bar{y}) | \, c \, | p(\bar{y}') \wedge y_2' > x \rangle$ basic while rule, (4)–(7)

(9) $\forall \bar{y}' [p(y') \wedge y_2' > x \supset (y_1' + 1)^2 > x]$ logical equation.

(10) $\langle p(\bar{y}) | \, c \, | (y_1' + 1)^2 > x \rangle$ Consequence, (8), (9).

(11) $\forall \bar{y}, \bar{y}' [T \wedge T \supset T]$ logical equation (needed for concatenation).

(12) $\langle T | \, p \, | (y_1' + 1)^2 > x \rangle$ Concatenation (2), (10), (11).

We would like to increment this deduction by showing that if x is non-negative then on exit also $y_1^2 \leq x$. To clarify the notation we denote:

$$\begin{aligned}\phi(\bar{y}) &\equiv T\\ p(\bar{y}) &\equiv [y_2 = (y_1+1)^2 \wedge y_3 = (2y_1+1) \wedge y_3 > 0]\\ \psi(\bar{y}, \bar{y}') &\equiv [x < (y_1'+1)^2].\end{aligned}$$

The increments respectively are:

$$\begin{aligned}\delta\phi(\bar{y}) &\equiv [x \geq 0]\\ \delta p(\bar{y}) &\equiv [x \geq 0 \wedge y_1^2 \leq x]\\ \delta\psi(\bar{y}, \bar{y}') &\equiv [(y_1')^2 \leq x].\end{aligned}$$

Tracing the logical equations in the deductions, we have to verify:

(δ1) $\forall \bar{y}, \bar{y}' [\phi(\bar{y}) \wedge \delta\phi(\bar{y}) \wedge \bar{y}' = f(\bar{y}) \supset \delta p(\bar{y}')]$,

(δ3) $\forall \bar{y}, \bar{y}' [p(\bar{y}) \wedge \delta p(\bar{y}) \wedge y_2 \leq x \wedge \bar{y}' = f(\bar{y}) \supset \delta p(\bar{y}')]$,

(δ5) $\forall \bar{y}' [p(\bar{y}') \wedge \delta p(\bar{y}') \wedge y_2' \leq x \supset \delta p(\bar{y}')]$,

(δ6) $\forall \bar{y}, \bar{y}', \bar{y}'' [p(\bar{y}') \wedge \delta p(\bar{y}') \wedge p(\bar{y}'') \wedge \delta p(\bar{y}'') \supset \delta p(\bar{y}'')]$,

(δ7) $\forall \bar{y} [p(\bar{y}) \wedge \delta p(\bar{y}) \wedge y_2 > x \supset \delta p(\bar{y})]$,

(δ9) $\forall \bar{y}, \bar{y}' [p(\bar{y}') \wedge \delta p(\bar{y}') \wedge y_2' > x \supset \delta\psi(\bar{y}, \bar{y}')]$,

(δ11) $\forall \bar{y}, \bar{y}' [T \wedge T \supset T]$.

All the above are easily verifiable. Thus we establish by the incremental step that

$$\langle x \geq 0 \,|\, P \,|(y_1')^2 \leq x < (y_1'+1)^2\rangle. \qquad \square$$

While the previous metatheorem covers the most general case, there is a special case which is so frequent, that it deserves a treatment of its own. This is the simpler case where the incrementation is uniform, i.e., we wish to modify each intermediate theorem and also the final theorem from

$$\langle p_i(\bar{x}) \,|\, B \,|q_i(\bar{x}, \bar{x}')\rangle$$

to

$$\langle p_i(\bar{x}) \wedge \pi(\bar{x}) \,|\, B \,|q_i(\bar{x}, \bar{x}') \wedge \pi(\bar{x}') \wedge \sigma(\bar{x}, \bar{x}')\rangle.$$

The increment $\pi(\bar{x})$ is uniformly added to *all* precondition predicates, while the increment $\pi(\bar{x}') \wedge \sigma(\bar{x}, \bar{x}')$ is uniformly added to *all* postcondition predicates. It is possible then to formulate the following:

Metatheorem 2. *Suppose that there exists a deduction for proving the theorem*

$$\langle \phi(\bar{x}) \,|\, P \,|\psi(\bar{x}, \bar{x}')\rangle.$$

Let $\pi(\bar{x})$ be a predicate and $\sigma(\bar{x}, \bar{x}')$ a reflexive and transitive relation, such that for any lemma in the deduction of the form

$$\langle p(\bar{x}) \,|\, \bar{x} \leftarrow f(\bar{x}) \,|q(\bar{x}, \bar{x}')\rangle$$

where $\bar{x} \leftarrow f(\bar{x})$ is an assignment statement, it is possible to prove

$$\langle p(\bar{x}) \wedge \pi(\bar{x}) \,|\, \bar{x} \leftarrow f(\bar{x}) \,|\, q(\bar{x}, \bar{x}') \wedge \pi(\bar{x}') \wedge \sigma(\bar{x}, \bar{x}') \rangle.$$

Then the incremental deduction is also valid, and consequently the theorem

$$\langle \phi(\bar{x}) \wedge \pi(\bar{x}) \,|\, P \,|\, \psi(\bar{x}, \bar{x}') \wedge \pi(\bar{x}') \wedge \sigma(\bar{x}, \bar{x}') \rangle$$

is true for the complete program.

Thus it is sufficient to treat assignment statements, or in practice, the logical equations leading to assignment statements' theorems. In order to prove the metatheorem, one has to inspect all the non-assignment rules and verify that if π and a transitive σ were preserved in the constituents they will be preserved in the bigger segment.

To illustrate the use of this metatheorem we bring the following examples:

(a) Let us extend the results obtained for example 1 to establish correctness in addition to the termination already established. The increment here is

$$\sigma(\bar{x}, \bar{x}') \equiv [gcd(x_1, x_2) = gcd(x_1', x_2')]$$
$$\pi(\bar{x}) = T.$$

σ is clearly reflexive and transitive. The only assignment stetements in the program are

$$x_1 \leftarrow x_1 - x_2$$

and

$$x_2 \leftarrow x_2 - x_1.$$

Obviously both these statements leave the *gcd* function invariant, thus establishing the validity of σ as increment for every theorem involving the assignment statements. By metatheorem 2 the following theorem is true:

$$\langle x_1 > 0 \wedge x_2 > 0 \,|\, P \,|\, x_1' = x_2' \wedge gcd(x_1, x_2) = gcd(x_1', x_2') \rangle$$

i.e., (by consequence)

$$\langle x_1 > 0 \wedge x_2 > 0 \,|\, P \,|\, x_1' = gcd(x_1, x_2) \rangle.$$

(b) Let us prove that the partition program of example 2 at most permutes the contents of the array A. Thus after execution, A' contains the same values with the same multiplicity that the initial A contains, possibly in modified order. The increment in this case is

$$\sigma(\bar{x}, \bar{x}') \equiv perm(A, A').$$

$perm(A, A')$ here is a predicate which exactly states that A' is a permuted version of A. It is obviously transitive and reflexive.

By inspecting the assignment statements in the program we observe that the only one modifying A is the exchange statement $A[i] \leftrightarrow A[j]$. Clearly, any exchange operation leaves the *perm* predicate invariant. Therefore, by metatheorem 2 we establish for the program:

$$\langle n \geqq 2 \,|\, P \,|\, i' > j' \wedge perm(A, A') \rangle.$$

V. Faults and Their Treatment

It was earlier mentioned that there exist several other faults besides endless loops. Among such faults we included: overflow, underflow, zero division, and subscript range violation.

Normally, we will not tend to call an execution in which any of these happened, a proper or correct execution.

It is actually a matter of semantics whether warding against such faults belongs to the correctness or the termination properties of a program. However, the more frequent features of programs are singled out, and standard devices formed for establishing their correctness, the easier the verification process becomes. Presumably, this was the reason, in the first place, for separating the issues of termination and partial correctness.

We would like now to indicate a modification to the rules by which one may attempt to establish that an execution is fault-free for some of the faults accounted above.

Consider, for example, an assignment statement

$$s\colon \bar{x} \leftarrow f(\bar{x}).$$

It should be understood that under this generic form we include the possibilities of a general subscript expression on the left side and a general expression on the right side of the assignment. With any *concrete* assignment statement it is possible to associate a "guard" predicate $\gamma(\bar{x})$ which ensures a faultless execution of this assignment. This predicate construction is dependent only on the form of the statement and some global information. Consequently, it is not difficult to have it generated automatically.

To illustrate this consider a concrete assignment statement:

$$s\colon A\,[i+1] \leftarrow (x + A\,[i-1]\cdot y)/z$$

and assume that A was previously declared to be an *array* $A\,[m\colon n]$. A suitable guard predicate will be

$$\begin{aligned}\gamma_s(\bar{x}) \equiv {} & (m_0 \leqq i+1 \leqq n_0) \wedge (m_0 \leqq i-1 \leqq n_0)\\ & \wedge (c \leqq |A\,[i-1]\cdot y| \leqq C) \wedge (c \leqq |x + A\,[i-1]\cdot y| \leqq C)\\ & \wedge (z \neq 0) \wedge \bigl(c \leqq |(x + A\,[i-1]\cdot y)/z| \leqq C\bigr).\end{aligned}$$

Here m_0 and n_0 are the values of m and n *on entrance* to the block at whose head A was last declared. (Accommodation of such relations lies outside our formalism and requires a major extension.) C and c are respectively the overflow and underflow thresholds for the relevant system.

Once it is decided which faults are to be insured against and the rules for constructing the guard predicate laid down, it is straightforward to modify the appropriate inference rule. Thus for example the modified assignment statement rule would look as follow

$$\frac{\forall \bar{x}_1, \bar{x}_2\{[p(\bar{x}_1) \supset \gamma_s(\bar{x}_1)] \wedge [p(\bar{x}_1) \wedge \bar{x}_2 = f(\bar{x}_1) \supset q(\bar{x}_1, \bar{x}_2)]\}}{\langle\langle p(\bar{x}) \mid s\colon \bar{x} \leftarrow f(\bar{x}) \mid q(\bar{x}, \bar{x}')\rangle\rangle}$$

where the double angular brackets notation is reserved to denote convergent fault-free correctness.

Similarly it is not difficult to conceive the construction of appropriate guard predicates for other types of statements and their associated modified inference rules.

Acknowledgement. We would like to thank V. E. Itkin for his inspiring remarks. We are particularly grateful to C. A. R. Hoare for a very detailed criticism of an early version of the paper. His challenging comments motivated many improvements towards the final form of this work.

References

1. Floyd, R. W.: Assigning meanings to programs. In: Schwartz, J. T. (ed.): Mathematical aspects of computer science. Proc. Symposia in Applied Mathematics **19**. Providence (R.I.): Amer. Math. Soc. 1967, p. 19–32
2. Hoare, C. A. R.: Algorithm 65-Find. Comm. ACM **4**, 321 (1961)
3. Hoare, C. A. R.: An axiomatic basis of computer programming. Comm. ACM **12**, 576–580, 583 (1969)
4. Hoare, C. A. R.: Proof of a program: FIND. Comm. ACM **14**, 39–45 (1971)
5. Igarashi, S., London, R. L., Luckham, D. C.: Automatic program verification I: A logical basis and its implementation. Computer Science Department, Stanford University, STAN-CS-73-365, May 1973

Zohar Manna
Amir Pnueli
Applied Mathematics Department
The Weizmann Institute of Science
Rehovot, Israel