STANFORD ARTIFICIAL INTELLIGENCE LABORATORY
MEMO AIM-210

STAN-CS-73-382

AXIOMATIC APPROACH TO TOTAL CORRECTNESS OF PROGRAMS

BY

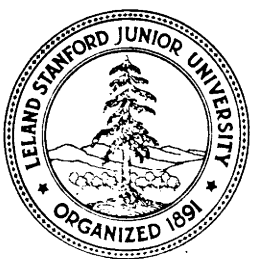
Zohar Manna and Amir Pnueli

SUPPORTED BY

ADVANCED RESEARCH PROJECTS AGENCY
ARPA ORDER NO. 457

July 1973

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY



STANFORD ARTIFICIAL INTELLIGENCE LABORATORY
MEMO AIM-210

JULY 1973

COMPUTER SCIENCE DEPARTMENT
REPORT cs-382

AXIOMATIC APPROACH TO TOTAL CORRECTNESS OF PROGRAMS

by

Zohar Manna and Amir Pnueli

Abstract: 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 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 correctness and termination at once, and in the enlarged scope of properties that can be proved by it.

This research was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense under Contract No. DAHC 15-73-C-0435.

The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

Reproduced in the USA. Available from the National Technical Information Service, Springfield, Virginia 22151.

JULY 1973

AXIOMATIC APPROACH TO TOTAL CORRECTNESS OF PROGRAMS

by

ZOHAR MANNA and AMIR PNUELI
Applied Mathematics Department
The Weizmann Institute of Science
Rehovot, Israel.

I. Introduction

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 [1969] 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 correctness and termination at once, and in the enlarged scope of properties that can be proved by it.

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.

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

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

(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 assertions -- axioms-- describing the transformation on program variables effected by simple statements, and inference rules by which assertions for small segments can be combined into one assertion for a larger segment. Among those are a composition 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

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

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.

The assertion we will be using in our method is of the form

$$< p(\bar{x}) \mid B \mid q(\bar{x},\bar{x}') >$$

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. An immediate advantage of this notation is the ability to express relations between values of variables before and after the execution. In the rest of the paper we develop the inference rules for our system which will also ensure that termination is hereditary from constituents to larger program segments.

Since we restrict ourselves to while programs, the only element endangering termination is the while statement. We attack the termination problem of the while statement by requiring the existence of a function from the program variables' domain to a well-founded set, such that on subsequent executions of the while body its value decreases. This function serves as a counter that can decrease only a finite number of times, It is this need to compare values of the counter function before and after execution of the while body which motivated us to extend the notation to relations between two sets of program variables.

If using our inference rules one is able to deduce

$$< \phi(\bar{x}) \mid P \mid \psi(\bar{x},\bar{x}') >$$

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

$$< \phi(\bar{x}) \mid P \mid T >,$$

where T is the identically true predicate.

We should remark in passing that although our rules are sufficient to show total correctness, they are by no means unique or even the best possible. Many variations and improvements probably exist.

II. The Inference Rules

All the inference rules will be described by a set of antecedents (conditions under which the rule is applicable) followed by a consequent which is the assertion deduced. Each of the antecedents is either an assertion (which should have been previously established) or a logical claim. All the logical claims are considered to be closed by universally quantifying each of their free variables on the same line.

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

(a) Assignment Rule .

$$\frac{p(\bar{x}) \wedge \bar{x}' = f(\bar{x}) \supset q(\bar{x},\bar{x}')}{< p(\bar{x}) \mid \bar{x} \leftarrow f(x) \mid q(\bar{x},\bar{x}') >}$$

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

(b) Conditional Rules

(b,) If-then-else

$$\frac{< p(\bar{x}) \wedge t(\bar{x}) \mid B_1 \mid q(\bar{x},\bar{x}') > \quad < p(\bar{x}) \wedge \neg t(\bar{x}) \mid B_2 \mid q(\bar{x},\bar{x}') >}{< p(\bar{x}) \mid \underline{\text{if}}\ t(\bar{x})\ \underline{\text{then}}\ B_1\ \underline{\text{else}}\ B_2 \mid q(\bar{x},\bar{x}') > .}$$

The rule should read as follows: If under $p(\bar{x})$ we succeeded 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.

(b2) If - do

$$\frac{\begin{array}{l} < p(\bar{x}) \wedge t(\bar{x}) \mid B \mid q(\bar{x},\bar{x}') > \\ p(\bar{x}) \wedge \sim t(\bar{x}) \supset q(\bar{x},\bar{x}) \end{array}}{< p(\bar{x}) \mid \underline{\text{if}}\ t(\bar{x})\ \underline{\text{do}}\ B \mid q(\bar{x},\bar{x}') > .}$$

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 four rules are composition rules. Some of them facilitate composition of segments while the others allow composition of predicates.

(c) Concatenation Rule

$$\frac{\begin{array}{ll} < p_1(\bar{x})\ I\ B_1\ I\ q_1(\bar{x},\bar{x}') > & (1) \\ < p_2(\bar{x})\ I\ B_2\ I\ q_2(\bar{x},\bar{x}') > & (2) \\ q_1(\bar{x},\bar{x}') \supset p_2(\bar{x}') & (3) \\ q_1(\bar{x},\bar{x}') \wedge q_2(\bar{x}',\bar{x}'') \supset q(\bar{x},\bar{x}'') & (4) \end{array}}{< p_1(\bar{x}) \mid B_1;B_2 \mid q(\bar{x},\bar{x}') > .}$$

Condition (3) ensures that the state after execution of B_1 satisfies p_2 -- the needed precondition for B_2.

Condition (4) characterizes $q(\bar{x},\bar{x}'')$ as a transfer relation between $\bar{x}$ before execution and $\bar{x}''$ after execution of $B_1;B_2$. It requires an intermediate $\bar{x}'$ which temporarily appears after execution of B_1 and before execution of B_2.

Note that by our convention (4) is universally quanitified over $\bar{x}$, $\bar{x}'$ and $\bar{x}''$.

(d) Consequence Rules

$$(d1)\quad \frac{\langle r(\bar{x}) \mid B \mid q(\bar{x},\bar{x}')\rangle \qquad p(\bar{x}) \supset r(\bar{x})}{\langle p(\bar{x}) \mid B \mid q(\bar{x},\bar{x}')\rangle}$$

$$(d2)\quad \frac{\langle p(\bar{x}) \mid B \mid s(\bar{x},\bar{x}')\rangle \qquad s(\bar{x},\bar{x}') \supset q(\bar{x},\bar{x}')}{\langle p(\bar{x}) \mid B \mid q(\bar{x},\bar{x}')\rangle}$$

The validity of the rules is obvious when we consider the meaning of the assertion.

(e) Or Rule

$$\frac{\langle p_1(\bar{x}) \mid B \mid q(\bar{x},\bar{x}') \rangle \quad \langle p_2(\bar{x}) \mid B \mid q(\bar{x},\bar{x}') \rangle}{\langle p_1(\bar{x}) \vee p_2(\bar{x}) \mid B \mid q(\bar{x},\bar{x}') \rangle}$$

This rule creates the possibility for proof by case analysis.

(f) And Rule

$$\frac{\langle p(\bar{x}) \mid B \mid q_1(\bar{x},\bar{x}') \rangle \quad \langle p(\bar{x}) \mid B \mid q_2(\bar{x},\bar{x}') \rangle}{\langle p(\bar{x}) \mid B \mid 4, \ (\bar{x},\bar{x}') \wedge q_2(\bar{x},\bar{x}') \rangle}$$

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, so that in principle we could have a stronger rule:

$$\frac{\langle p(\bar{x}) \mid B \mid q_1(\bar{x},\bar{x}') \rangle \quad \{ p(\bar{x}) \mid B \mid q_2(\bar{x}') \}}{\langle p(\bar{x}) \mid B \mid q_1(\bar{x},\bar{x}') \wedge q_2(\bar{x}') \rangle}$$

where we reserve the notation {} to 'partialcorrectness assertion'.

(g) While Rule.

$$< p(\bar{x}) \wedge t(\bar{x}) \mid B \mid q(\bar{x},\bar{x}') \wedge [\neg t(\bar{x}') \vee u(\bar{x}) \succ u(\bar{x}')] > \quad (1)$$

$$q(\bar{x},\bar{x}') \wedge t(\bar{x}') \supset p(\bar{x}') \quad (2)$$

$$q(\bar{x},\bar{x}') \wedge q(\bar{x}',\bar{x}'') \supset q(\bar{x},\bar{x}'') \quad (3)$$

$$p(\bar{x}) \wedge \neg t(\bar{x}) \supset q(\bar{x},\bar{x}) \quad (4)$$

$$< p(\bar{x}) \mid \underline{\text{while}}\ t(\bar{x})\ \underline{\text{do}}\ B \mid q(\bar{x},\bar{x}') \wedge \neg t(\bar{x}') >$$

where $(w,\prec)$ is a well-founded set and $u: X \to W$.

The above seemingly complicated rule is devised to overcome several difficulties caused by the need to prove termination. Termination of a looping while statement is essentially ensured here by Floyd's technique **[1967]**, namely, producing a function u whose values keep strictly decreasing in subsequent executions of B.

Condition (1) requires establishing a well-founded set $(W,\prec)$ with a partial order $\prec$ satisfying the descending chain condition, **i.e.**, there is no infinite chain of elements from W, a, +,a,),.. . Also required is a partial function u mapping some elements of our data domain X into elements of w , If we were able to prove that after each execution of B , $u(\bar{x}) \succ u(\bar{x}')$ (where by writing this inequality we also mean that $u(\bar{x})$ and $u(\bar{x}')$ are both defined), then clearly B cannot repeatedly execute an infinite number of times or we would violate the descending chain condition.

The demand for the existence of a descending counter which is defined for all executions of the while body B ,

can be relaxed for the case of the last execution of B . Thus if we are positive that this is the last execution of B , we may allow the counter function to become undefined or stop decreasing. Accordingly, we require in (1) the alternatives of either $\sim t(\bar{x})$ true , implying immediate termination, or the existence of the counter function which will also ultimately ensure termination.

Condition (2) 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 (1) once more and thus propagate the validity of q for all subsequent executions.

Condition (3) ensures that $q(\bar{x},\bar{x}')$ is transitive. There fore, once we showed in (1) 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 (4) 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 $q(\bar{x},\bar{x}')$.

Note that (1) establishes the termination of B itself.

In the proofs appearing in the following examples we often make use of the consequence rule within while rule derivations without explicit indication. Thus, for example, we frequently use the condition :

$$< \quad p(X) \quad \wedge \quad t(\bar{x}) \mid \; B \; \mid q(\bar{x},\bar{x}') \wedge [u(\bar{x}) \succ u(\bar{x}')] >$$

which implies condition (1) above by the consequence rule. Similarly we use the consequent:

$$< p(\bar{x}) \mid \text{while} \quad t(\bar{x}) \quad \text{do} \quad B \; \mid q(\bar{x},\bar{x}') >$$

omitting the conjunct $\sim t(\bar{x}')$.

III. Illustration of the Method

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 termiration, with only general indication of the modifications required to add correctness.

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 -- $\underline{gcd}(x_1, x_2)$ -- leaving the result in x_1. To refer to program segments we use ordinary Algol labels.

```
P:  START
      f:  while  x1 ≠ x2 do
            e:  begin
                  b:  while  x1 > x2 do  a:  x1 ← x1 - x2 ;
                  d:  while  x2 > x1 do  c:  x2 ← x2 - x1
                  end
      HALT .
```

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

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

and

$$\psi(x_1, x_2, x_1', x_2') \equiv x_1' = \underline{gcd}(x_1, x_2) .$$

We prove in detail termination only. The well-founded set we use is the domain of no&negative integers with the ordinary $<$ relation. As the termination function for all while statements we take $u(x_1, x_2) \equiv x_1 + x_2$. 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). In the second case statement a is never executed ($x_1 + x_2$ unchanged of course), while statement c is executed at least once ($x_1 + x_2$ decreasing). We will therefore analyze in our proof these two cases separately and then combine their results using the Or rule.

In all the predicates of the following assertions the conjunction $x_1 > 0 \wedge x_2 > 0$ is omitted.

Lemma al, (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

$$< x_1 > x_2 \mid a \mid x_1 + x_2 > x_1' + x_2' >$$

by the assignment rule.

Lemma bl (While Rule)

We use the while rule with the following Predicates:

$$p(\bar{x}) \equiv t(\bar{x}) \equiv x_1 > x_2 \ ,$$

$$q(\bar{x},\bar{x}') \equiv x_1+x_2 > x:+x; \ .$$

Condition (1) of the while rule is justified by Lemma al. We obtain

$$< x_1 > x_2 \mid b \mid x_1+x_2 > x_1'+x_2' > \ .$$

Note that condition (4) of the while rule is trivially satisfied because

$$p(\bar{x}) \wedge \sim t(\bar{x}) \equiv F \ .$$

Lemma cl (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

$$< x_2 > x_1 \mid c \mid x_1 + x_2 > x'_1 + x_2' > \ .$$

Lemma dl (While Rule)

Assume the following substitution:

$$p(\bar{x}) \equiv T \ , \ t(\bar{x}) \equiv x_2 > x_1 \ , \text{ and}$$

$$q(\bar{x},\bar{x}') \equiv x_1 + x_2 \geq x_1' + x_2' \ .$$

Condition (1) of the while- rule is justified by Lemma cl. We obtain

$$< T \mid d \mid x_1 + x_2 \geq x_1' + x_2' > \ .$$

Note that condition (4) is satisfied since $x_1 + x_2 \geq x_1 + x_2$.

Lemma el (Concatenation Rule)

Combine Lemmas bl and dl and use

$$x_1+x_2>x_1'+x_2' \wedge x_1'+x_2' \geq x_1''+x_2'' \supset x_1+x_2>x_1''+x_2''$$

to obtain

$$< x_1 > x_2 \mid e \mid x_1 + x_2 > x_1' + x_2' > .$$

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

Lemma a2 (Assignment Rule)

Since

$$F \wedge x_1' = x_1 - x_2 \wedge x_2' = x_2 \supset F$$

we have

$$< F \mid a \mid F > .$$

Lemma b2 (While Rule)

Take

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

$$q(\bar{x},\bar{x}') \equiv x_1' < x_2' \wedge (x_1 + x_2 = x_1' + x_2') .$$

By using a consequence of Lemma a2 we obtain

$$< x_1 < x_2 \mid b \mid x_1' < x_2' \wedge (x_1 + x_2 = x_1' + x_2') > .$$

Condition (1) is satisfied here since by the consequence rules $< F \mid a \mid F >$ implies

$$< p(\bar{x}) \wedge t(\bar{x}) \mid a \mid q(\bar{x},\bar{x}') \wedge \neg t(\bar{x}') > .$$

Note that under the initial condition $x_1 < x_2$ the while statement b never executes.

Lemma c2 (Assignment Rule)

By assignment rule

$$< x_1 < x_2 \mid c \mid x_2 + x_2 > x_1' + x_2' > .$$

Lemma- d2 (While Rule)

Take

$$p(\bar{x}) \equiv t(\bar{x}) \equiv x_1 < x_2 , \text{ and}$$

$$q(\bar{x},\bar{x}') \equiv x_1 + x_2 > x_1' + x_2' ,$$

Using Lemma c2 we obtain

$$< x_1 < x_2 \mid d \mid x_1 + x_2 > x_1' + x_2' > .$$

Lemma e2 (Concatenation Rule)

By combining Lemmas b2 and d2 we obtain

$$< x_1 < x_2 \mid e \mid x_1 + x_2 > x_1' + x_2' > .$$

Lemma e (Or Rule)

From Lemmas e1 and e2 combined we get

$$< x_1 \neq x_2 \mid e \mid x_1 + x_2 > x_1' + x_2' > .$$

Lemma f (While Rule)

Take

$$t(\bar{x}) \equiv x_1 \neq x_2 \quad , \quad p(\bar{x}) \equiv x_1 > 0 \wedge x_2 > 0 \quad , \text{ and}$$

$$q(\bar{x},\bar{x}') \equiv x_1 > 0 \wedge x_2 > 0 \quad .$$

Note that $x_1 > 0 \wedge x_2 > 0$ was implicitly assumed in all previous preconditions. Using Lemma e in condition (1) we get:

$$< x_1 > 0 \wedge x_2 > 0 \mid P \mid x_1' = x_2' > .$$

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

On trying to extend this result to prove correctness as well as termination, we run into the notion of incremental proofs, i.e., having proved some properties of the program including termination, how do we prove additional properties without repeating the whole proof process.

For this particular example, this can be solved by the following argument:

Assume that instead of any $q(\bar{x},\bar{x}')$ appearing in the assertions we used the predicate

$$q(\bar{x},\bar{x}') \wedge [\underline{gcd}(x_1,x_2) = \underline{gcd}(x_1',x_2')] .$$

It is not difficult to ascertain that all the lemmas remain valid. Consequently, we are able to prove for the complete program:

$$< x_1 > 0 \wedge x_2 > 0 \mid P \mid x_1' = x_2' \wedge \underline{gcd}(x_1,x_2) = \underline{gcd}(x_1',x_2') >,$$

i.e.,

$$< x_1 > 0 \wedge x_2 > 0 \mid P \mid x_1' = \underline{gcd}(x_1,x_2) > .$$

Generalizing the above argument, we may consider any transitive relation $s(\bar{x},\bar{x}')$ with the following properties :

$\forall \bar{x}[s(\bar{x},\bar{x})]$ and $\forall \bar{x},\bar{x}',\bar{x}''\ [\, s(\bar{x},\bar{x}') \wedge s(\bar{x}',\bar{x}'') \supset s(\bar{x},\bar{x}'')\,]$.

It is possible then to verify the following metatheorem:

Metatheorem. Suppose that $a \equiv \langle \phi(\bar{x}) \mid P \mid \psi(\bar{x},\bar{x}') \rangle$ had been proved. Let $s(\bar{x},\bar{x}')$ be a transitive relation such that for any lemma of the form $\langle p(\bar{x}) \mid B \mid q(\bar{x},\bar{x}') \rangle$ used in proving . a , where B is an assignment statement of P , it is possible to prove $\langle p(\bar{x}) \mid B \mid q(\bar{x},\bar{x}') \wedge s(\bar{x},\bar{x}') \rangle$. Then the assertion $\alpha^{+} \equiv \langle \phi(\bar{x}) \mid P \mid \psi(\bar{x},\bar{x}') \wedge s(\bar{x},\bar{x}') \rangle$ is also true for the complete program.

Thus it is sufficient to treat assignment statements in incrementing our claims. In the previous example, the only assignment statements one has to consider are

$$x_2 \leftarrow x_2 - x_1, \text{ and}$$

$$x_1 \leftarrow x_1 - x_2 ,$$

which obviously preserve the gcd function.

In order to prove the metatheorem, one has to inspect all the non-assignment rules and verify that if s was preserved in the constituents it will be preserved in the bigger segment.

Example 2: Partition (Hoare [1961])

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

$$0 \leq j < i \leq n$$

and for the rearranged array

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

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],...,A[i-1] are less than or equal to those in the upper partition A[j+1],...,A[n], where $0 \leq j < i \leq n$.

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

We will prove in detail termination only. Our proof follows the ideas presented in **Hoare's[1971]** informal proof of termination. We int **roduce** the following abbreviations :

$$a(i) \equiv \exists p[i \leq p \leq n \wedge r \leq A[p]]$$

$$\beta(j) \equiv \exists q[0 \leq q \leq j \wedge A[q] \leq r] .$$

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)

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

Clearly B(j) validity is invariant since j is not modified by this statement. From a(i) correctness we infer the existence of p which since. $A[p] \geq r$ must be $p > i$, so that we might take the same p to establish $a(i+1) = \alpha(i')$. 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 . Roth are over the domain of non -negative integers. The **alternatives** presented are that either this function is decreasing (non-increasing) or $j' < i'$ which will imply that this must be the last

execution of ℓ. Note that if the second holds true, then $j' - i'$ is not defined.

Lemma b (While Rule)

Using Lemma a with

$$p(\bar{x}) \equiv \alpha(i) \wedge \beta(j)$$

$$q(\bar{x},\bar{x}') \equiv \alpha(i') \wedge \beta(j') \wedge [i' > j' \vee j - i \geq j' - it]$$

$$u(\bar{x}) \equiv n - i ,$$

we get

$$< \alpha(i) \wedge \beta(j)$$

$$| \text{ b: } \underline{\text{while}} \ A[i] < r \ \underline{\text{do}} \ \text{a: } i \leftarrow i + 1 \ |$$

$$\beta(j') \wedge [i' > j' \vee j - i \geq j' - i'] \wedge A'[i'] \geq r > .$$

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

Lemma c (Assignment Rule)

$$< A[i] \geq r \wedge \beta(j) \wedge A[j] > r$$

$$| \text{ c: } j \leftarrow j - 1 |$$

$$\beta(j') \wedge A[i'] \geq r \wedge [i' > j' \vee j - i \geq j' - i'] \wedge j > j' > .$$

The function **ensuring** termination for the inner while d is j .

Lemma d (While Rule)

From Lemma c with

$$p(\bar{x}) \equiv A[i] \geq r \wedge \beta(j)$$

$$q(\bar{x},\bar{x}') \equiv \beta(j') \wedge A'[i'] \geq r \wedge [i' > j' \vee j - i \geq j' - i']$$

$$u(\bar{x}) \equiv j ,$$

we get

$$< A[i] \geq r \wedge \beta(j)$$

$$| \text{ d: } \underline{\text{while}} \ r < A[j] \ \underline{\text{do}} \ \text{c: } j \leftarrow j - 1 |$$

$$A'[i'] \geq r \wedge [i' > j' \vee j - i \geq j' - i'] \wedge A'[j'] \leq r > .$$

Lemma e (Concatenation Rule)

Combining Lemmas b and d we get

$$< a(i) \wedge \beta(j)$$

$$| \text{ e: } \underline{\text{begin}} \text{ b; d } \underline{\text{end}} |$$

$$A'[j'] \leq r \leq A'[i'] \wedge [i' > j' \vee j - i \geq j' - i'] > .$$

Lemma f (Assignment Rule)

$$< A[j] \leq r \leq A[i] \wedge i \leq j$$

$$| \text{ f: } A[i] \leftrightarrow A[j] |$$

$$A'[i'] \leq r \leq A^* [j'] \wedge j - i = j' - i' \wedge i' \leq j' > .$$

The condition $i \leq j$ 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)

$$< i \leq j \wedge A[i] \leq r \leq A[j]$$

$$| \text{ g: } (i,j) \leftarrow (i+1, j-1) |$$

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

This result i.s obtained by case analysis: Either $i + 1 \leq j - 1$, in which case we have $i < i' \leq j' < j$ 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, $i' > j'$.

Lemma h (Concatenation Rule)

By combining Lemmas f and g we get

$$< i \leq j \wedge A[j] \leq r \leq A[i]$$

$$| \text{ h: begin f; g end } |$$

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

Lemma k (If - do Rule)

By Lemma h we get

$$< A[j] \leq r \leq A[i]$$

$$| \text{ k: if } i \leq j \text{ do h } |$$

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

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

Lemma ℓ (Concatenation Rule)

Combining Lemmas e and k we obtain:

$$< \alpha(i) \wedge \beta(j)$$

$$| \ell: \text{ begin e; k end } |$$

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

Note that by the consequence rule this can be rewritten as

$$< \alpha(i) \wedge \beta(j)$$

$$| \ell: \text{ begin e; k end } |$$

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

which is in a form more useful for the next step.

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 $\alpha(i), \beta(j)$ both valid, we either have $i' > j'$ which ensures no more repetitions of ℓ or have $\alpha(i'), \beta(j')$ true again and a termination function $j - i$

strictly decreasing.

Lemma m (While Rule)

From lemma e with

$$p(\bar{x}) \equiv \alpha(i) \wedge \beta(j)$$

$$q(\bar{x},\bar{x}') \equiv i' \leq j' \supset [\alpha(i') \wedge \beta(j')] ,$$

we get

$$< \alpha(i) \wedge \beta(j) \mid \text{m: } \underline{\text{while}}\ i \leq j\ \underline{\text{do}}\ \ell \mid T > .$$

Lemma s (Assignment + Concatenation Rules)

Establishes the initial conditions:

$$< n \geq 2 \mid \text{s:, } r \leftarrow A[n \div 2];\ (i,j) \leftarrow (0,n) \mid \alpha(i') \wedge \beta(j') > .$$

Lemma P (Concatenation Rule)

Concatenation of lemmas m and s yields

$$< n \geq 2 \mid P \mid T > ,$$

which shows termination of P .

References

FLOYD [1967]. R. W. Floyd, "Assigning Meanings to Programs", Proc. Symp. Appl. Math. 19, American Math. Soc. (1967), pp. 19-32.

HOARE [1961]. C. A. R. Hoare, "Algorithm 65 - Find", CACM, Vol. 4, No. 7 (July 1971), p. 321.

HOARE [1969]. C. A. R. Hoare, "An Axiomatic Basis of Computer Programming", CACM, Vol. 12, No. 10 (October 1969), pp. 576-580, 583.

HOARE [1971]. C. A. R. Hoare, "Proof of a Program: FIND", CACM, Vol. 14, No. 1 (January 1971), pp. 39-45.