Acta Informatica 9, 61–71 (1977)

Axioms for Total Correctness

Stefan Sokolowski

Institute of Mathematics, University of Gdansk, ul Wita Stwosza 57, 80-952 Gdansk, Poland

Summary. A notation for total correctness of a program with respect to input and output formulas is introduced; and Hoare's loop axiom is rearranged in such a way as to form a good inference system for the total correctness. Its consistency and completeness are shown.

1. Introduction

The usual way for proving the correctness of programs consists of two steps:

prove the partial correctness by means of inductive assertions,

prove the termination either by means of functions into a well-founded-set or by the loop counters. The full description of both can be found in [4].

Hoare in [2] introduced a system of axioms and inference rules in order to give a formal mathematical treatment to the first step. In [5] Manna and Pnueli describe an axiomatic system for proving the total correctness of programs without going through the two separate steps as above. The termination part of the proof is based on well-founded sets, and is incorporated in the loop rule.

Quite understandably the rules are not as simple as those in [2] and the special denotation for the total correctness causes increased complication in all the rules and not only in the loop rule.

I'd like to propose a different axiomatics for total correctness. The termination checking will be based on bounding of loop counters. The loop rule will be fairly clumsy (however simpler than in [5]) but all the remaining rules are identical to those of [2].

Throughout the paper the semantics of statements is determined by their results functions on the state vector as, for example, in the interpretative model in [3].

2. The Rules

First I recall Hoare's basic rules and axioms in order to describe the notation I am going to use. In this system **x** is a constant input vector, **y** is the state vector, *B* and D are statements, p, q, r, s are first order predicates over $\mathbf{x}$ and $\mathbf{y}$, g is a function over $\mathbf{x}$ and $\mathbf{y}$ yielding a new state vector and α is a quantifier-free formula over $\mathbf{x}$ and $\mathbf{y}$. It is assumed that the names of all computable functions may be used to form the terms and formulas p, q, r, s (though not α) and that the language incorporates the calculus of non-negative integers.

The notation $\{p(\mathbf{x}, \mathbf{y})\}\ B\{q(\mathbf{x}, \mathbf{y})\}$ states the partial correctness of the program B with respect to the formulas $p(\mathbf{x}, \mathbf{y})$ and $q(\mathbf{x}, \mathbf{y})$. For sake of readibility the $\mathbf{x}, \mathbf{y}$-arguments are omitted where no confusion can arise.

Hoare's System

(A) *Assignment axiom:*

$$\{p(\mathbf{x}, g(\mathbf{x}, \mathbf{y}))\}\ \mathbf{y} := g(\mathbf{x}, \mathbf{y})\ \{p(\mathbf{x}, \mathbf{y})\}$$

(R1) *Consequence rule:*

$$\frac{p \supset q,\ \{q\}\ B\{r\},\ r \supset s}{\{p\}\ B\{s\}}$$

(R2) *Composition rule:*

$$\frac{\{p\}\ B\{q\},\ \{q\}\ D\{r\}}{\{p\}\ B; D\{r\}}$$

(R3) **if-then-else-fi** *rule:*

$$\frac{\{p \wedge \alpha\}\ B\{q\},\ \{p \wedge \neg\alpha\}\ D\{q\}}{\{p\}\ \textbf{if}\ \alpha\ \textbf{then}\ B\ \textbf{else}\ D\ \textbf{fi}\ \{q\}}$$

(R4) **while-do-od** *rule:*

$$\frac{\{p \wedge \alpha\}\ B\{p\}}{\{p\}\ \textbf{while}\ \alpha\ \textbf{do}\ B\ \textbf{od}\ \{p \wedge \neg\alpha\}}$$

In order to treat total correctness I now assign a different meaning to the denotation

$$\{p\}\ B\{q\}$$

namely, let it stand for the total correctness of B with respect to p and q.

The only rule of Hoare's system that thus turns invalid is the loop rule (R4); therefore this rule needs a change. Before quoting its new form some discussion on loop counters is necessary.

Loop counters could be implemented as integer variables assigned to the while-loops, which are set to zero each time the control enters the loop and increased by one each time one run through the loop is executed. The loop counters are different for different loops and several counters may be non-zero simultaneously in the case of nested loops. They may not be assigned to nor read from within the programs.

In the sequel the existence of the counters in programs will be rather imaginary. Namely they may appear in the predicates but not in any part of the program.

The modified (R4) rule thus takes the form:

(R 4′) **while-do-od** *rule:*

$$\frac{\begin{array}{c} p(i) \wedge \neg\alpha \supset q \\ p(i) \supset i \leqq k(\mathbf{x}) \\ \{p(i) \wedge \alpha\}\ B\{p(i+1)\} \end{array}}{\{p(0)\}\ \textbf{while}\ \alpha\ \textbf{do}\ B\ \textbf{od}\ \{q\}}$$

In this rule k is a term independent of $\mathbf{y}$ yielding an integer for every interpretation. The term k is meant to denote an upper bound on the loop counter i. It has to be assumed that there is no occurrence of i in B nor in α. The full form of (R 4′), if we were not to omit the arguments of the predicates, would be the following:

(R 4′) **while-do-od** *rule:*

$$\frac{\begin{array}{c} p(\mathbf{x},\mathbf{y},i,\mathbf{l}) \wedge \neg\alpha(\mathbf{x},\mathbf{y}) \supset q(\mathbf{x},\mathbf{y},\mathbf{l}) \\ p(\mathbf{x},\mathbf{y},i,\mathbf{l}) \supset i \leqq k(\mathbf{x},\mathbf{l}) \\ \{p(\mathbf{x},\mathbf{y},i,\mathbf{l}) \wedge \alpha(\mathbf{x},\mathbf{y})\}\ B\{p(\mathbf{x},\mathbf{y},i+1,\mathbf{l})\} \end{array}}{\{p(\mathbf{x},\mathbf{y},0,\mathbf{l})\}\ \textbf{while}\ \alpha(\mathbf{x},\mathbf{y})\ \textbf{do}\ B\ \textbf{od}\ \{q(\mathbf{x},\mathbf{y},\mathbf{l})\}}$$

where $\mathbf{l}$ is the vector consisting of the loop counters for all loops that this one is nested in.

The only reason for considering a lower bound on the integer i was the attempt to stress its implementational loop-counter nature. With no harm we could omit it and take $p(j(\mathbf{x}))$ instead of $p(0)$ in

$\{p(0)\}$ **while** α **do** B **od** $\{q\}$

with the only limitation that $j \leqq k$.

To illustrate the method I bring next some simple examples of proofs of total correctness of programs by this approach. The examples are not very elaborate and the reader is certainly familiar with them.

Example 1. Factorial.

Our task is to prove the following:

```
(0)  {n≧0}
       s:=1; w:=n;
       while w≠0 do
           s:=s×w; w:=w−1
       od
     {s=n!}
```

By twice applying (A), once (R 2), once (R 1) we get

(1) $\{n \geqq 0\}\ s := 1;\ w := n\ \{s = 1 \wedge w = n \geqq 0\}$.

Let $p(i)$ be

$s \times w! = n! \wedge w + i = n \wedge w \geqq 0$

and let q be

$s = n!$

Now check the assumptions of the (R4′) rule: the first one

(2) $(s \times w! = n! \wedge w + i = n \wedge w \geqq 0) \wedge w = 0 \supset s = n!$

is obvious, the next one is satisfied by $k = n$:

(3) $s \times w! = n! \wedge w + i = n \wedge w \geqq 0 \supset i \leqq n$

indeed, the assumptions $w \geqq 0$ and $w + i = n$ yield $i \leqq n$. Checking the third assumption is done by usual means:

```
(4) {$(s \times w! = n! \wedge w + i = n \wedge w \geqq 0) \wedge w \neq 0$}
        $s := s \times w$; $w := w - 1$
    {$s \times w! = n! \wedge w + i + 1 = n \wedge w \geqq 0$}
```

Thus by the rule (R4′) applied to (2), (3), (4) we get

```
(5) {$s \times w! = n! \wedge w = n \wedge w \geqq 0$}
        while $w \neq 0$ do
            $s := s \times w$; $w := w - 1$
        od
    {$s = n!$}
```

Now prove

(6) $s = 1 \wedge w = n \geqq 0 \supset s \times w! = n! \wedge w = n \wedge w \geqq 0$

and by (R1) applied to $(n \geqq 0 \supset n \geqq 0)$, (1) and (6) and then (R2), the proof of (0) is completed.

A byproduct of the proof is an estimate on the number of iterations performed during the execution of the program: not more than n. In this case this estimate is the best possible. But in the following example the termination is proved without attempting to derive a good estimate of the program's complexity. And in fact the given bound is a gross overestimate.

Example 2. Exponentiation.

```
(0) {$x \geqq 0 \wedge y \geqq 0$}
        $a := x$; $b := y$; $c := 1$;
        while $b \neq 0$ do
            if $odd(b)$
                then $b := b - 1$; $c := c \times a$
                else $a := a \times a$; $b := b/2$
            fi
        od
    {$c = x \uparrow y$}
```

Proof.

```
(1) {$x \geqq 0 \wedge y \geqq 0$}
        $a := x$; $b := y$; $c := 1$
    {$x \uparrow y = a \uparrow b \times c \wedge b \geqq 0 \wedge a \geqq 0 \wedge b \leqq y$}
```

is straightforward by (A), (R1), (R2).

Let $p(i)$ be

$$x\uparrow y=a\uparrow b\times c\wedge b\geqq 0\wedge a\geqq 0\wedge b+i\leqq y$$

and let q be

$$c=x\uparrow y.$$

Now check the assumptions of (R 4′): the first one

(2) $(x\uparrow y=a\uparrow b\times c\wedge b\geqq 0\wedge a\geqq 0\wedge b+i\leqq y)\wedge b=0\supset c=x\uparrow y$

is obvious; the next one is satisfied by $k=y$, although in reality the program loops many less times:

(3) $x\uparrow y=a\uparrow b\times c\wedge b\geqq 0\wedge a\geqq 0\wedge b+i\leqq y\supset i\leqq y$

indeed, $b\geqq 0$ and $b+i\leqq y$ yield $i\leqq y$. Now check the third assumption:

```
(4)  $\{(x\uparrow y=a\uparrow b\times c\wedge b\geqq 0\wedge a\geqq 0\wedge b+i\leqq y)\wedge b\neq 0\}$
        if $odd(b)$
           then $b:=b-1;\ c:=c\times a$
           else $a:=a\times a;\ b:=b/2$
        fi
     $\{x\uparrow y=a\uparrow b\times c\wedge b\geqq 0\wedge a\geqq 0\wedge b+i+1\leqq y\}$
```

This can be easily done in the usual way. From (R 4′) applied to (2), (3), (4) it follows

```
(5)  $\{x\uparrow y=a\uparrow b\times c\wedge b\geqq 0\wedge a\geqq 0\wedge b\leqq y\}$
        while $b\neq 0$ do
           if $odd(b)$
              then $b:=b-1;\ c:=c\times a$
              else $a:=a\times a;\ b:=b/2$
           fi
        od
     $\{c=x\uparrow y\}$
```

and by (R 2) applied to (1), (5) we derive (0), and this completes the proof.

Example 3. Greatest common divisor (taken from [5]).

```
(0)  $\{x1>0\wedge x2>0\}$
        $y1:=x1;\ y2:=x2;$
        while $y1\neq y2$ do
           while $y1>y2$ do
              $y1:=y1-y2$
           od;
           while $y1<y2$ do
              $y2:=y2-y1$
           od
        od
     $\{y1=y2=\gcd(x1,x2)\}$
```

Proof. Let $g(y1, y2)$ denote the formula:

(1) $y1>0 \wedge y2>0 \wedge \gcd(y1, y2)=\gcd(x1, x2) \wedge y1 \leqq x1 \wedge y2 \leqq x2.$

One can easily see that each turn of the outer loop decreases $y1+y2$, so it seems reasonable to choose

(2) $g(y1, y2) \wedge y1+y2+i \leqq x1+x2$

for its invariant.

It is easy now to check two first assumptions of (R 4′):

(3) $g(y1, y2) \wedge y1+y2+i \leqq x1+x2 \wedge y1=y2 \supset y1=y2=\gcd(x1, x2)$

and

(4) $g(y1, y2) \wedge y1+y2+i \leqq x1+x2 \supset i \leqq x1+x2.$

Checking the third assumption

```
(5) {$(g(y1, y2) \wedge y1+y2+i \leqq x1+x2) \wedge y1 \neq y2$}
        **while** $y1>y2$ **do**
            $y1:=y1-y2$
        **od**;
        **while** $y1<y2$ **do**
            $y2:=y2-y1$
        **od**
    {$g(y1, y2) \wedge y1+y2+i+1 \leqq x1+x2$}
```

is more complicated. In order to do this, note that whenever during the execution of first inner loop it happens that $y1=y2$, then a decrease in $y1+y2$ has already had place. This suggests to choose

(6) $g(y1, y2) \wedge y1+y2+i \leqq x1+x2 \wedge (y1=y2 \supset y1+y2+i+1 \leqq x1+x2)$
$\wedge y1+j \leqq x1$

for invariant of this loop. For the assertion after this loop take

(7) $g(y1, y2) \wedge y1+y2+i \leqq x1+x2 \wedge (y1<y2 \vee y1+y2+i+1 \leqq x1+x2)$

As usually two first assumptions of (R 4′) are easy to check. The upper bound on the loop counter j is $x1$. The third one

```
(8) {$(g(y1, y2) \wedge y1+y2+i \leqq x1+x2 \wedge (y1=y2 \supset y1+y2+i+1 \leqq x1+x2)$
    $\wedge y1+j \leqq x1) \wedge y1>y2$}
        $y1:=y1-y2$
    {$g(y1, y2) \wedge y1+y2+i \leqq x1+x2 \wedge (y1=y2 \supset y1+y2+i+1 \leqq x1+x2)$
    $\wedge y1+j+1 \leqq x1$}
```

holds by (A), (R 1) and simple properties of $g(y1, y2)$. Thus from (R 4′) we have

```
(9) $\{g(y1,y2) \wedge y1+y2+i \leqq x1+x2 \wedge (y1=y2 \supset y1+y2+i+1 \leqq x1+x2)\}$
        while $y1>y2$ do
            $y1:=y1-y2$
        od
    $\{g(y1,y2) \wedge y1+y2+i \leqq x1+x2 \wedge (y1<y2 \vee y1+y2+i+1 \leqq x1+x2)\}$
```

For an invariant of the second inner loop take

(10) $g(y1,y2) \wedge y1+y2+i \leqq x1+x2 \wedge (y1<y2 \vee y1+y2+i+1 \leqq x1+x2)$
$\wedge\, y2+j \leqq x2$

with $x2$ for the upper bound of j. Just like before we derive for the second loop:

```
(11) $\{g(y1,y2) \wedge y1+y2+i \leqq x1+x2 \wedge (y1<y2 \vee y1+y2+i+1 \leqq x1+x2)\}$
        while $y1<y2$ do
            $y2:=y2-y1$
        od
     $\{g(y1,y2) \wedge y1+y2+i+1 \leqq x1+x2\}$
```

Now, by (R2) applied to (9) and (11), and by (R1), the derivation of (5) is completed. So, by (R4′) applied to (3), (4), (5) we have

(12) $\{g(y1,y2)\}$ *outer loop* $\{y1=y2=\gcd(x1,x2)\}$

Since $x1>0 \wedge x2>0 \supset g(x1,x2)$ holds, the proof of (0) is completed.

Example 4. Generator of prime numbers.

```
(0) $\{n \geqq 2\}$
      $r[1]:=2; r[2]:=3; k:=2;$
      while $k \neq n$ do
          $d:=r[k]+2; l:=0;$
          while $l \neq k$ do
              $l:=l+1;$
              if $r[l] | d$
                  then $d:=d+2; l:=0$
              fi
          od;
          $k:=k+1; r[k]:=d$
      od
    $\{r[1],\dots,r[n]$ are consecutive prime numbers$\}$
```

Proof. The assertions for the outer loop are

$p1(i)$: '$2 \leqq k=i+2 \leqq n$ *and* $r[1],\dots,r[k]$ *are consecutive primes*'
$q1$: '$r[1],\dots,r[n]$ *are consecutive primes*'

and the upper bound on the loop counter i is $n-2$. The assertions for the inner loop are:

$p2(i,j)$: '$2 \leqq k = i+2 < n$, *and* $0 \leqq l \leqq k$, *and* $j \leqq l + k \times d$, *and none of* $r[1], \ldots, r[l]$ *divides* d, *and there is no prime* b *such that* $r[k] < b < d$ (*i.e.* d *is the candidate for* $r[k+1]$)'

$q2(i)$: '$2 \leqq k = i+2 < n$ *and* $r[1], \ldots, r[k], d$ *are consecutive primes*'

and the upper bound on the loop counter j is $n \times (1 + \textit{the n-th prime})$.

The details are left to the reader.

3. Justification of the Loop Rule

Before the new rule may be used in practise, two questions have to be dealt with:

is it a valid rule?

does it give a full description of **while-do-od** properties?

Let B be a program and $f[B]$ – the result function of B on the state vector, i.e. B halts on a vector $\mathbf{y}$ iff $f[B](\mathbf{x}, \mathbf{y})$ is defined and $f[B](\mathbf{x}, \mathbf{y})$ is the result of computation.

Let $\alpha(x, y)$ be a quantifier-free formula and D be a program

while $\alpha(\mathbf{x}, \mathbf{y})$ **do** B **od**

The result function $f[D]$ of D is a partial function defined by

$$f[D](\mathbf{x},\mathbf{y}) = \begin{cases} \mathbf{y} & \text{iff} \quad \alpha(\mathbf{x},\mathbf{y}) = \textit{false} \\ f[D](\mathbf{x}, f[B](\mathbf{x},\mathbf{y})) & \text{otherwise} \end{cases}$$

It is undefined in each of the following cases:

$\mathbf{y}$ is undefined

$\alpha(\mathbf{x}, \mathbf{y}) = \textit{true}$ and $f[B](\mathbf{x}, \mathbf{y})$ is undefined

the condition $\alpha(\mathbf{x}, f[B](\mathbf{x}, f[B](\mathbf{x}, \ldots, f[B](\mathbf{x}, \mathbf{y}) \ldots)))$ holds for every iteration.

Theorem 1 (Validity). *If for a certain integer term* $k(\mathbf{x})$ *independent of* $\mathbf{y}$, *for certain predicates* p, q, *for a certain program section* B *and for a quantifier-free formula* α *the following conditions hold:*

(1) $p(\mathbf{x}, \mathbf{y}, i) \wedge \neg \alpha(\mathbf{x}, \mathbf{y}) \supset q(\mathbf{x}, \mathbf{y})$,

(2) $p(\mathbf{x}, \mathbf{y}, i) \supset i \leqq k(\mathbf{x})$,

(3) *if* $p(\mathbf{x}, \mathbf{y}, i) \wedge \alpha(\mathbf{x}, \mathbf{y})$ *holds then* $f[B](\mathbf{x}, \mathbf{y})$ *is defined and* $p(\mathbf{x}, f[B](\mathbf{x}, \mathbf{y}), i+1)$ *holds*

and if for a certain state vector $\mathbf{y0}$: $p(\mathbf{x}, \mathbf{y0}, 0)$ *holds, then* $f[D](\mathbf{x}, \mathbf{y0})$ *is defined and* $q(\mathbf{x}, f[D](\mathbf{x}, \mathbf{y0}))$ *holds (for simplicity the vector of global loop counters is omitted).*

Proof. Assume we are given an interpretation matching the term k with an integer $k0$. Assume $p(\mathbf{x}, \mathbf{y0}, 0)$ holds.

Case $k0 = 0$:

In this case $\alpha(\mathbf{x}, \mathbf{y0})$ is false for otherwise in view of (3)

$$p(\mathbf{x}, f[B](\mathbf{x}, \mathbf{y0}), 1)$$

would follow, which is in contradiction with (2) that reads

$$p(\mathbf{x}, \mathbf{y}, i) \supset i \leqq 0.$$

Hence by the definition of $f[D]$:

$$f[D](\mathbf{x}, \mathbf{y0}) = \mathbf{y0}.$$

From (1) we derive $q(\mathbf{x}, \mathbf{y0}) = true$ which is thus equivalent to

$$q(\mathbf{x}, f[D](\mathbf{x}, \mathbf{y0})) = true$$

and this completes the proof for this case.

Case $k0 > 0$:

Assume the theorem already proved for $k-1$. If $\alpha(\mathbf{x}, \mathbf{y0})$ is false then by the definition $f[D](\mathbf{x}, \mathbf{y0}) = \mathbf{y0}$ and by (1) $q(\mathbf{x}, \mathbf{y0})$ holds and hence $q(\mathbf{x}, f[D](\mathbf{x}, \mathbf{y0}))$ holds.

Let us assume $\alpha(\mathbf{x}, \mathbf{y0}) = true$. By (3) $f[B](\mathbf{x}, \mathbf{y0})$ is defined and $p(\mathbf{x}, f[B](\mathbf{x}, \mathbf{y0}), 1)$ holds. Let $\mathbf{y1} = f[B](\mathbf{x}, \mathbf{y0})$ and $p'(\mathbf{x}, \mathbf{y}, i) = p(\mathbf{x}, \mathbf{y}, i+1)$ be a new predicate.

It has just been proved that $p'(\mathbf{x}, \mathbf{y1}, 0)$ holds. One can easily see that the conditions (1), (2), (3) remain valid after substituting $k-1$ and p' for all occurrences of k and p respectively. By the induction assumption we get that $f[D](\mathbf{x}, \mathbf{y1})$ is defined and $q(\mathbf{x}, f[D](\mathbf{x}, \mathbf{y1}))$ holds. Since

$$f[D](\mathbf{x}, \mathbf{y0}) = f[D](\mathbf{x}, f[B](\mathbf{x}, \mathbf{y0})) = f[D](\mathbf{x}, \mathbf{y1}),$$

the proof of theorem 1 is completed.

Thus the **while-do-od** rule (R 4′) is valid.

Now let us discuss the completeness of this axiomatic system. If for a certain program L

$$\{p(\mathbf{x})\}\ L\{q(\mathbf{x}, \mathbf{y})\}$$

holds, is it possible to derive this from the axioms?

As assumed earlier, we are allowed to use all the computable functions in order to form the inductive assertions. For every statement S (either a simple assignment or a compound statement) of the program L there exist computable functions $i[S]$ and $o[S]$ over the input vector $\mathbf{x}$ and the vector $\mathbf{l}$ of loop counters for the loops containing S, such that $i[S](\mathbf{x}, \mathbf{l})$ equals the state vector $\mathbf{y}$ just before control enters S, and $o[S](\mathbf{x}, \mathbf{l})$ equals the state vector $\mathbf{y}$ just after it leaves S. Note that $i[S]$ and $o[S]$ are partial functions, since it may happen that for a certain $\mathbf{x}$ some of the loop counters are too large.

Thus

(4) $\{p(\mathbf{x}) \wedge i[S](\mathbf{x}, \mathbf{l}) = \mathbf{y}\}\ S\{p(\mathbf{x}) \wedge o[S](\mathbf{x}, \mathbf{l}) = \mathbf{y}\}$

holds for every statement S of the program L. The occurrences of $i[S]$, $o[S]$ and $\mathbf{l}$

in the assertions are fully legitimate. If S, T are two consecutively executed statements of the programm L then $o[S](\mathbf{x}, \mathbf{l}) = i[T](\mathbf{x}, \mathbf{l})$.

As soon as we derive (4) from the inference system for all statements S of L, the axiomatic proof of

(5) $\{p(\mathbf{x})\}\ L\{q(\mathbf{x}, \mathbf{y})\}$

is easily completed by (R 1) applied to

$$\{p(\mathbf{x}) \wedge i[L](\mathbf{x}, \mathbf{e}) = \mathbf{y}\}\ L\{p(\mathbf{x}) \wedge o[L](\mathbf{x}, \mathbf{e}) = \mathbf{y}\}$$

(where **e** is the empty vector of counters), and to the implication

$$p(\mathbf{x}) \wedge o[L](\mathbf{x}, \mathbf{e}) = \mathbf{y} \supset q(\mathbf{x}, \mathbf{y})$$

which is true because (5) holds.

Theorem 2 (Completeness). *If L halts for all input vectors satisfying $p(\mathbf{x})$, than for each statement S of L*

(4) $\{p(\mathbf{x}) \wedge i[S](\mathbf{x}, \mathbf{l}) = \mathbf{y}\}\ S\{p(\mathbf{x}) \wedge o[S](\mathbf{x}, \mathbf{l}) = \mathbf{y}\}$

can be derived in the system.

Proof. The proof will be by induction on the length of S. If S is an assignment then (4) takes the form

$$\{p(\mathbf{x}) \wedge i[S](\mathbf{x}, \mathbf{l}) = \mathbf{y}\}\ \mathbf{y} := g(\mathbf{x}, \mathbf{y})\ \{p(\mathbf{x}) \wedge o[S](\mathbf{x}, \mathbf{l}) = \mathbf{y})\}$$

which can be derived from (A) and (R 1) because

$$i[\mathbf{y} := g(\mathbf{x}, \mathbf{y})](\mathbf{x}, \mathbf{l}) = \mathbf{y} \supset o[\mathbf{y} := g(\mathbf{x}, \mathbf{y})](\mathbf{x}, \mathbf{l}) = g(\mathbf{x}, \mathbf{y})$$

holds.

If S is a compound statement we may assume the theorem already proved for its proper substatements. I will write down the proof only for the case when S is of the form

while α **do** B **od**

the remaining cases being much simpler. For simplicity of notation the occurrences of **x** and **l** in functions and formulas will be omitted.

Since L halts, there is a computable k denoting the number of iterations of B in S.

Let $g(k, \mathbf{y})$ denote the k-th iteration of $f[B]$ on $\mathbf{y}$, and let $r(\mathbf{y}, j)$ be the formula:

$$p \wedge \mathbf{y} = g(j, i[S]) \wedge f[S](\mathbf{y}) = f[S](i[S]) \wedge j \leqq k$$

and let $s(y)$ be the formula:

$$p \wedge \mathbf{y} = o[S].$$

The proof will be completed after the three assumptions of (R 4′) are proved to hold (in view of the induction assumption they are then derivable), because $r(\mathbf{y}, 0)$ is equivalent to $p \wedge \mathbf{y} = i[S]$.

Assumption 1.

$$(p \wedge y = g(j, i[S]) \wedge f[S](y) = f[S](i[S]) \wedge j \leqq k) \wedge \neg\alpha(y) \supset p \wedge y = o[S]$$

Proof. Since $\neg\alpha(y)$ then $f[S](y) = y$ and hence $y = f[S](y) = f[S](i[S]) = o[S]$.

Assumption 2.

$$p \wedge y = g(j, i[S]) \wedge f[S](y) = f[S](i[S]) \wedge j \leqq k \supset j \leqq k$$

Proof. Obvious.

Assumption 3.

$$\{(p \wedge y = g(j, i[S]) \wedge f[S](y) = f[S](i[S]) \wedge j \leqq k) \wedge \alpha(y)\}$$
$$\mathrm{B}$$
$$\{p \wedge y = g(j+1, i[S]) \wedge f[S](y) = f[S](i[S]) \wedge j+1 \leqq k\}$$

Proof. The effect of B is such as that of the assignment $y := f[B](y)$ thus the assumption 3 holds iff

(6) $p \wedge y = g(j, i[S]) \wedge f[S](y) = f[S](i[S]) \wedge j \leqq k \wedge \alpha(y)$

implies

(7) $p \wedge f[B](y) = g(j+1, i[S]) \wedge f[S](f[B](y)) = f[S](i[S]) \wedge j+1 \leqq k.$

First two parts of conjunction (7) are obviously implied by (6). We are only to prove that (6) implies

$$f[S](f[B](y)) = f[S](i[S]) \wedge j+1 \leqq k$$

as well.

Since $\alpha(y) = \mathit{true}$, so $f[S](y) = f[S](f[B](y))$ and hence

$$f[S](f[B](y)) = f[S](i[S]).$$

Validity of $j+1 \leqq k$ follows from the fact that k bounds the number of iterations of $f[B]$ and $y = g(j, i[S])$ – the j-th iteration of $f[B]$ on $i[S]$.

This ends the proof of Theorem 2.

The assertions constructed for the above proof are not the ones a programmer will usually use in correctness proof for a particular program, for usually the formulas $i[S]$ and $o[S]$ are very ineffective.

Acknowledgements. I am pleased to express my gratitude to my referee who must have spent much time correcting both the style and the contents of the paper.

References

1. Dijkstra, E.W.: A discipline of programming. Enlewood Cliffs, N.J.: Prentice-Hall 1976
2. Hoare, C.A.R.: An axiomatic basis for the computer programming. Comm. ACM **12**, 576–580 (1969)
3. Hoare, C.A.R., Lauer, P.E.: Consistent and complementary formal theories of the semantics of programming languages. Acta Informat. **3**, 135–153 (1974)
4. Katz, S., Manna, Z.: A closer look at termination. Acta Informat. **5**, 333–352 (1975)
5. Manna, Z., Pnueli, A.: Axiomatic approach to total correctness. Acta Informat. **3**, 243–264 (1974)
6. Wang, A.: An axiomatic basis for proving total correctness of GOTO-Programs. Nordisk Tidskr. Informationsbehandling (BIT), 88–102 (1976)
7. Sokolowski, S.: Total correctness for procedures. Mathematical Foundations of Computer Science 1977, Lecture Notes in Computer Science (J. Gruska, ed.), Vol. 53, pp. 475–483, 1977

Received June 17, 1976