The Semantics of New While Loop*

TADAO TAKAOKA
Department of Information Science, University of Ibaraki, Hitachi, 316 Japan

A new control structure called 'new while-loop', abbreviated 'nwhile' is introduced. The meaning of nwhile* B *do* S *is that if* B *is False* S *is not executed. If* B *is true,* S *is executed repeatedly while* B *is true. If* B *becomes false anywhere inside* S, *the computer goes out of loop. An example of program for the new control structure is given.

Although the nwhile structure can be given using loop and exit statements in ADA, the programmer does not have to worry where to put exit statements in the proposed nwhile loop.

Received June 1984

1. INTRODUCTION

Isomichi[1] proposed a new control structure for a while-loop for FORTRAN 77. His idea is that the computer goes out of loop as soon as the boolean condition goes untrue. The present note discusses a rigorous logical basis for the new while-loop and applies it to give a formal proof to the algorithm of partition used in quicksort.

2. LOGICAL BASIS

Let a general form of the new while-loop be as follows.

$$\textbf{nwhile } B \textbf{ do } S \qquad (1)$$

where 'nwhile' stands for 'new while'. An informal description of (1) is as follows. If the boolean expression B is false when the computer comes to (1), the computer does not execute S at all and goes to the next statement. As long as B is true, the computer repeatedly executes S, but as soon as B becomes false at any point inside S, it goes to the next statement.

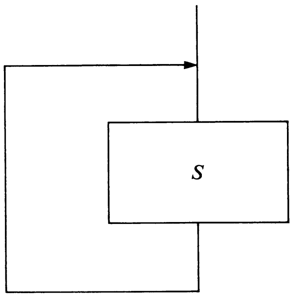


Figure 1. Loop *S*.

Similarly a new repeat (abbreviated 'nrepeat') statement is expressed as follows.

$$\textbf{nrepeat } S \textbf{ until } B \qquad (2)$$

The computer goes out of loop as soon as B becomes true. If we reverse the sense of B we have the same meaning between (1) and (2). In other words, there is no essential difference between (1) and (2), whereas in the conventional while-loop and repeat-loop, there is a great difference between testing the condition B before S and after S. Hence we have only to investigate the nwhile-loop.

First let us change program (1) into (3).

$$\textbf{loop } S \qquad (3)$$

* Part of this work was done while the author was on leave at the Department of Computer and Information Sciences, University of Alabama in Birmingham, Birmingham, Alabama, U.S.A.

where the computer goes back to the beginning of S after executing it. The flowchart for (3) is given in Fig. 1.

The semantics for (3) is given by

$$\frac{\{Q\}\,S\{Q\},\ P \supset Q}{\{P\}\ \textbf{loop}\ \{Q\}\,S} \qquad (4)$$

where the assertion P is the precondition for (3) and we have no postcondition for (3). The style of (4) is from Hoare[2] and Alagić and Arbib.[3]

Now we turn to the semantics for (1). Let $S_1, \ldots, S_n$ be assignment statements in S, which affect the condition B. Assume that $P_1, \ldots, P_n$ hold immediately before $S_1, \ldots, S_n$, respectively, in program (3) under precondition P. Under these circumstances, we have the following inference rule for (1).

$$\frac{\prod_{i=1}^{n} \{P_i \wedge B\}\,S_i\{Q_i\},\quad \prod_{i=1}^{n} (\neg B \wedge Q_i \supset Q),\quad P \wedge \neg B \supset Q}{\{P\}\ \textbf{nwhile}\ B\ \textbf{do}\ S\{Q\}} \qquad (5)$$

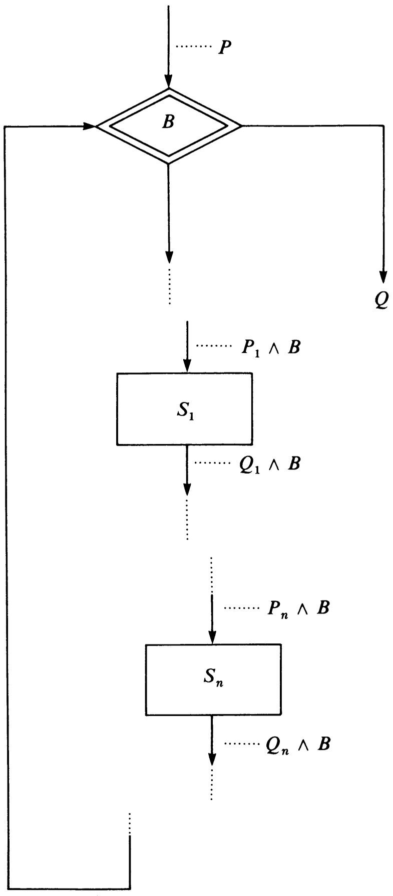
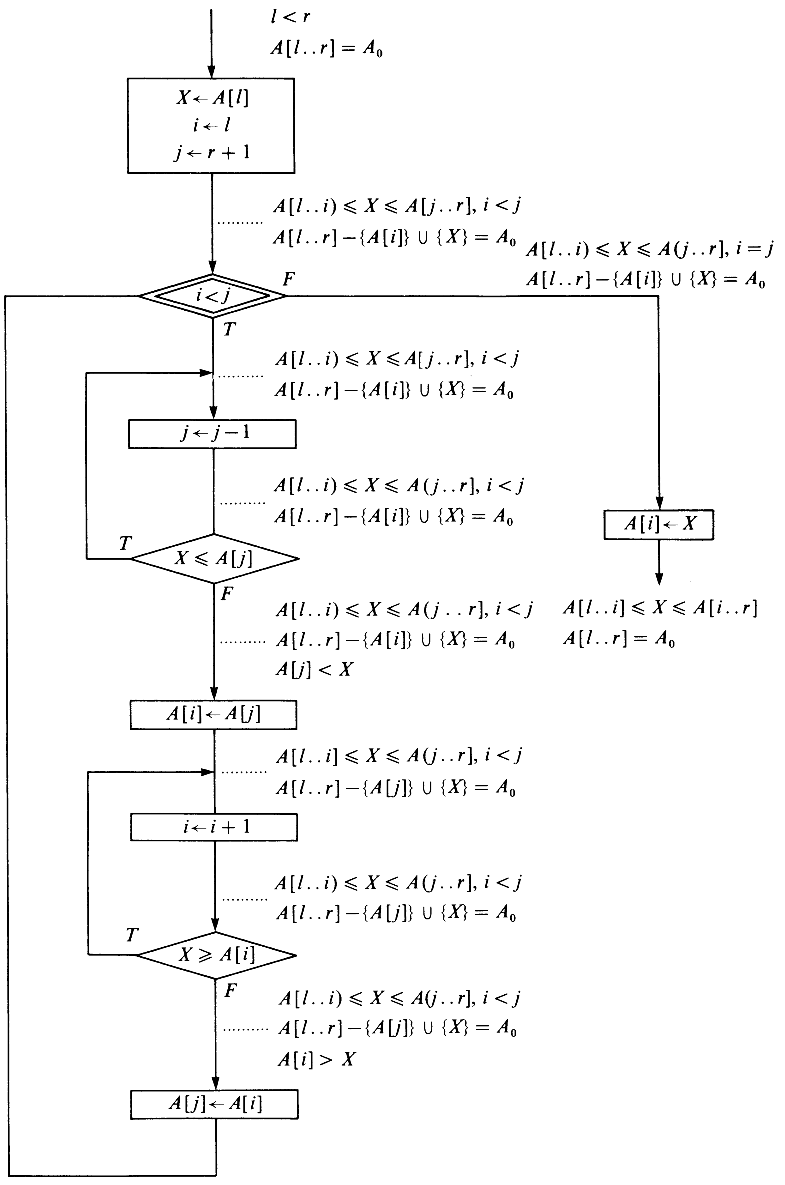


Figure 2. New while-loop.



$l < r$
$A[l..r] = A_0$
$X \leftarrow A[l]$
$i \leftarrow l$
$j \leftarrow r+1$
$A[l..i) \leq X \leq A[j..r], i < j$
$A[l..r] - \{A[i]\} \cup \{X\} = A_0$
$A[l..i) \leq X \leq A(j..r], i = j$
$A[l..r] - \{A[i]\} \cup \{X\} = A_0$
$i < j$
T
F
$j \leftarrow j-1$
$X \leq A[j]$
$A[j] < X$
$A[i] \leftarrow A[j]$
$A[i] \leftarrow X$
$A[l..i] \leq X \leq A[i..r]$
$A[l..r] = A_0$
$A[l..i] \leq X \leq A(j..r], i < j$
$A[l..r] - \{A[j]\} \cup \{X\} = A_0$
$i \leftarrow i+1$
$X \geq A[i]$
$A[i] > X$
$A[j] \leftarrow A[i]$

Figure 3. Flowchart for PARTITION.

This gives a formal semantics to the nwhile statement, and is illustrated by a flowchart in Fig. 2. This style of flowchart with assertions is borrowed from Floyd.[4] The condition in the double diamond in Fig. 2 applies to every point in the loop, but is put on the top position for simplicity. The semantics for conventional statements is from [3].

3. APPLICATION

The following is a procedure for quicksort.

```
procedure QUICK(l, r);
  var i: integer;
  if $l < r$ then
    begin
      PARTITION(l, r);
      QUICK(l, $i-1$);
      QUICK($i+1$, r)
    end                                    (6)
```

where the procedure *PARTITION* is given below.

```
procedure PARTITION(l, r);
  var x: real; j: integer;
  begin
    $x := A[l]$;
    $i := l$;  $j := r+1$;
    nwhile $i < j$ do
      begin
        repeat $j := j-1$ until $A[j] < x$;
        $A[i] := A[j]$;
        repeat $i := i+1$ until $A[i] > x$;
        $A[j] := A[i]$
      end;
    $A[i] := x$
  end                                      (7)
```

The procedure *QUICK*(l, r) sorts the portion $A[l]..A[r]$ of array A in increasing order. The procedure *PARTITION* partitions the portion $A[l]..A[r]$ in such a way that

$$\begin{gathered}\forall k\,(l \leqslant k < i)\, A[k] \leqslant x \\ \forall k\,(i < k \leqslant r)\, A[k] \geqslant x \\ A[i] = x \\ A[l\,..\,r] = A_0 \qquad (8)\end{gathered}$$

where $$A[l\,..\,r] = [A[l], \ldots, A[r]\}$$

and A_0 is the set $A[l\,..\,r]$ before the procedure *PARTITION* is executed. The above condition (8) plays the role of postcondition for *PARTITION*. Here we prove that the procedure *PARTITION* is correct. The precondition is given by

$$\begin{gathered} l < r \\ A[l\,..\,r] = A_0 \end{gathered}$$

We introduce additional notations as follows.

$$\begin{gathered} A[l\,..\,i) = \{A[l], \ldots, A[i-1]\} \\ A(j\,..\,r] = \{A[j+1], \ldots, A[r]\} \qquad (9) \\ A[l\,..\,i) \leqslant x \Leftrightarrow \forall k\,(l \leqslant k < i)\, A[k] \leqslant x \\ A(j\,..\,r] \geqslant x \Leftrightarrow \forall k\,(j < k \leqslant r)\, A[k] \geqslant x \qquad (10) \end{gathered}$$

The notations $A[l\,..\,r] \leqslant x$ and $A[j\,..\,r] \geqslant x$ are defined similarly. We have two statements which affect the condition $i < j$, that is, $j := j-1$ and $i := i+1$. Assertions P_1 and P_2 are given by

$$\begin{aligned} P_1\!: &\ A[l\,..\,i) \leqslant x \leqslant A[j\,..\,r] \\ &\ A[l\,..\,r] - \{A[i]\} \cup \{x\} = A_0 \\ P_2\!: &\ A[l\,..\,i] \leqslant x \leqslant A(j\,..\,r] \\ &\ A[l\,..\,r] - \{A[j]\} \cup \{x\} = A_0 \end{aligned}$$

The precondition for the nwhile-loop is given by

$$\begin{aligned} P\!: &\ A[l\,..\,i) \leqslant x \leqslant A[j\,..\,r], \quad i < j \\ &\ A[l\,..\,r] - \{A[i]\} \cup \{x\} = A_0 \end{aligned}$$

The assertions Q_1 and Q_2 are given by

$$\begin{aligned} Q_1\!: &\ A[l\,..\,i) \leqslant x \leqslant A(j\,..\,r], \quad i < j+1 \\ &\ A[l\,..\,r] - \{A[i]\} \cup \{x\} = A_0 \\ Q_2\!: &\ A[l\,..\,i) \leqslant x \leqslant A(j\,..\,r], \quad i-1 < j \\ &\ A[l\,..\,r] - \{A[j]\} \cup \{x\} = A_0 \end{aligned}$$

The assertions $\neg(i < j) \wedge Q_1$ and $\neg(i < j) \wedge Q_2$ imply

$$\begin{aligned} Q\!: &\ A[l\,..\,i) \leqslant x \leqslant A(j\,..\,r], \quad i = j \\ &\ A[l\,..\,r] - \{A[i]\} \cup \{x\} = A_0 \end{aligned}$$

$P \wedge \neg B$ can imply anything.

From Q with the statement $A[i] := x$ we can easily derive

$$\begin{gathered} A[l\,..\,i] \leqslant x \leqslant A[i\,..\,r] \\ A[l\,..\,r] = A_0, \quad A[i] = x \end{gathered}$$

which is the desired postcondition (8). The flowchart is illustrated in Fig. 3 annotated with necessary assertions for correctness.

Remark 1. In the above proof we assume that the elements in array A are all distinct. If we had equal elements in A, we would need to interpret sets and set operations in the above proof as multisets and operations for multisets.

Remark 2. The algorithm presented here partitions n elements with $n-1$ comparisons between data, which is theoretically minimum. Hence we have the following recurrence for the expected number $T(n)$ of comparisons for quicksort for n elements.

$$\begin{aligned} T(n) &= n - 1 + \frac{1}{n}\sum_{i=1}^{n} (T(i-1) + T(n-i)) \\ &= n - 1 + \frac{2}{n}\sum_{i=1}^{n-1} T(i) \\ T(0) &= 0, \quad T(1) = 0. \end{aligned}$$

Hence as in Aho, Hopcroft and Ullman (see p. 94),[5] we have

$$T(n) \sim 2n \log_e n.$$

In typical books, such as Knuth,[6] Aho, Hopcroft and Ullman[5] and Wirth,[7] the number of comparisons between data is greater than $n-1$, which makes the analysis of the algorithm a little harder. Also the comparison of i and j appears more than once in the partition algorithms in those textbooks, whereas it appears only once in program (7), which makes the program more readable.

Remark 3. The meaning of new while is closer to the original meaning of 'while' in English.

REFERENCES

1. Y. Isomichi, On new FORTRAN statements, *Information Processing, Japan* (Joho-Shori) **21**, (9), 1000–1001 (1980).
2. C. A. R. Hoare, An axiomatic basis for computer programming *CACM* **12**, (10), 576–580 (1969).
3. S. Alagić and M. A. Arbib, *The Design of Well-Structured and Correct Programs* Springer-Verlag, Heidelberg (1978).
4. R. W. Floyd, Assigning Meanings to Programs. In *Proc. Symp. in Applied Math.* Vol. 19, Mathematical Aspects of Computer Science, edited J. T. Schwartz, pp. 19–32 American Mathematical Society, New York (1967).
5. A. V. Aho and J. E. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass. (1974).
6. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, *Sorting and Searching*. Addison-Wesley, Reading, Mass. (1973).
7. N. Wirth, *Algorithms + Data Structures = Programs.* Prentice-Hall, Englewood Cliffs (1976).