Computer Science Department
Indiana University
Lindley Hall 101
Bloomington, Indiana 47401

Key Words and Phrases:
structured programming, control structures, looping, while statement, until statement, exit statement, flowchart, ease of programming.

CR Categories: 4.4.2, 4.4.9, 5.5.2

TECHNICAL REPORT NO. 10

A CASE FOR WHILE-UNTIL

DANIEL P. FRIEDMAN
STUART C. SHAPIRO

APRIL 5, 1974

A CASE FOR WHILE-UNTIL

Daniel P. Friedman
Stuart C. Shapiro
Indiana University

Abstract

A new control structure construct, the while-until, is introduced as a syntactic combination of the while and the until. Examples are shown indicating that use of the while-until can lead to structured programs that are conceptually more manageable than those attainable without it. The while-until statement is then extended to a value returning expression which is shown to be more powerful than either the while or the until.

A major suggestion of structured programming is to employ looping control structures in order to break the program down into conceptually manageable units. The purpose of this paper is to propose an additional looping control structure construct (the while-until) that, in certain instances, yields program loops that are closer to the conceptual organization of the segment than is possible with the existing constructs. The while-until as a statement will be shown to be equivalent to the existing looping control structures. The while-until as a value (Boolean) returning expression will be shown to be a more powerful control structure than the while or until structures discussed by Dijkstra [1].

The existing constructs that we are concerned with are

<u>while</u> β <u>repeat</u> s

and

<u>repeat</u> s <u>until</u> β .

Dijkstra [1] presents these graphically as in Figures 1 and 2.

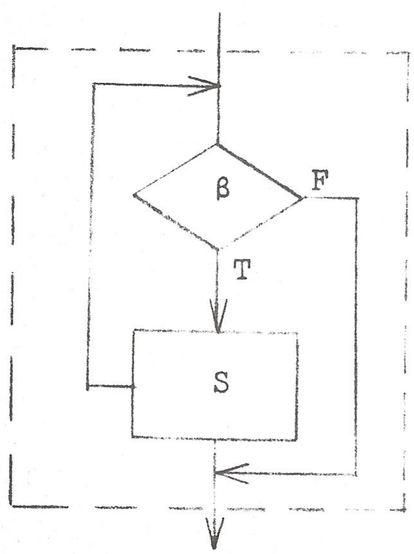

Fig. 1 <u>while</u> β <u>repeat</u> s

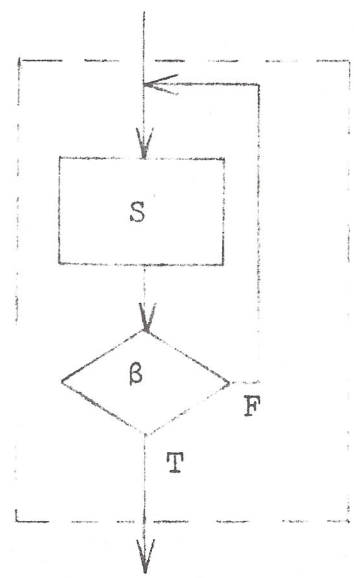

Fig. 2 <u>repeat</u> s <u>until</u> β

The syntactic construct we are proposing is

<u>while</u> β_1 <u>repeat</u> s <u>until</u> β_2

which is presented graphically in Figure 3.

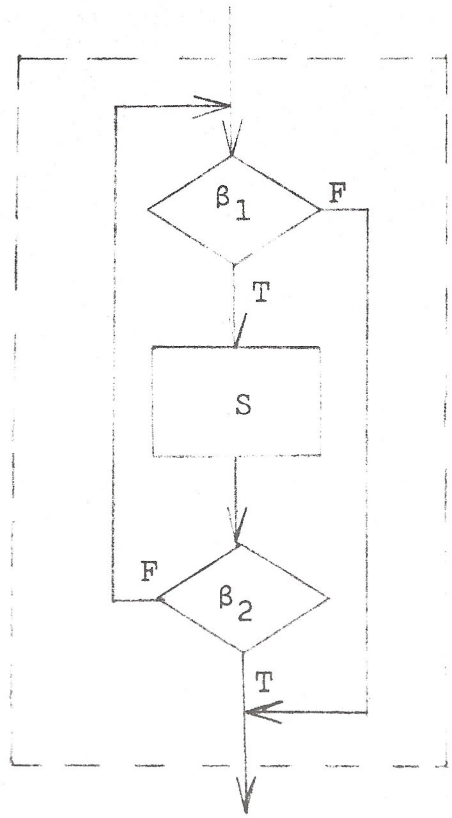

Fig. 3 while β_1 repeat s until β_2

The while-until does not involve nesting, but is some other combination [2] of the features of the while and the until loops. The while-until may be replaced by the until or the while as the only looping structure since

while β_1 repeat s until β_2

is equivalent to

if β_1 then repeat s until if(β_2 then true else $\neg\beta_1$)

and also to

<u>if</u> β_1 <u>then</u> <u>begin</u> s; <u>while</u> <u>if</u>(β_2 <u>then</u> <u>false</u> <u>else</u> β_1) <u>repeat</u> s <u>end</u>

If <u>escape</u> (or <u>break</u> or <u>exit</u>) were employed, another equivalent form is

A: <u>while</u> β_1 <u>repeat</u>

<u>begin</u> s; <u>if</u> β_2 <u>then</u> <u>escape</u> A <u>end</u>

<u>end</u> A

In those cases where Figure 3 is the desired control structure it appears that the <u>while-until</u> yields clearer, more understandable code than any of the above alternatives. We can paraphrase the semantic content of the <u>while-until</u> as: "while it is possible to try, keep trying until you succeed." The <u>while</u> and the <u>until</u> loops can be defined in terms of the <u>while-until</u> in the following way:

<u>while</u> β <u>repeat</u> s $=_{def}$ <u>while</u> β <u>repeat</u> s <u>until</u> <u>false</u>

and

<u>repeat</u> s <u>until</u> β $=_{def}$ <u>while</u> <u>true</u> <u>repeat</u> s <u>until</u> β

The <u>while-until</u> is a natural control structure for searching, since every search terminates either by finding the desired element or by determining that it is not present. As an example, we show its use for a binary search:

```
comment Find item A in table T[1:N] ;
low :=0;
high := N + 1;
while low < high - 1 repeat
     try := (low + high) / 2;
     if T[try] < A then low := try else high := try
until T[try] = A;
```

An appropriate application for the while-until occurs whenever a loop includes two operations, one of which requiring a test prior to its execution and the other requiring a test which can only be performed after its execution. An example of this is: copy a file up to and including the end-of-file mark onto an output file, however, nothing may be written on the output file unless there is enough space for a record.

```
comment Copy file INPUT onto the file OUTPUT;
while   Spaceleft(OUTPUT) repeat
            Inbuffer(INPUT, b);
            Outbuffer(OUTPUT, b)
until   Eof(INPUT);
```

In languages in which statements are expressions having values, for example LISP [3], ALGOL 68 [4] and BLISS [5], the while-until can be assigned a value in an especially useful way. We define the value of the while-until expression to be the value of the last evaluated

Boolean. That is, the value of

$$\underline{\text{while}}\ \beta_1\ \underline{\text{repeat}}\ s\ \underline{\text{until}}\ \beta_2$$

is false if and only if the loop terminates due to the evaluation of β_1 (see Figure 4). A non-Boolean value could be returned on certain termination conditions (e.g., exit in BLISS or predicates in LISP).

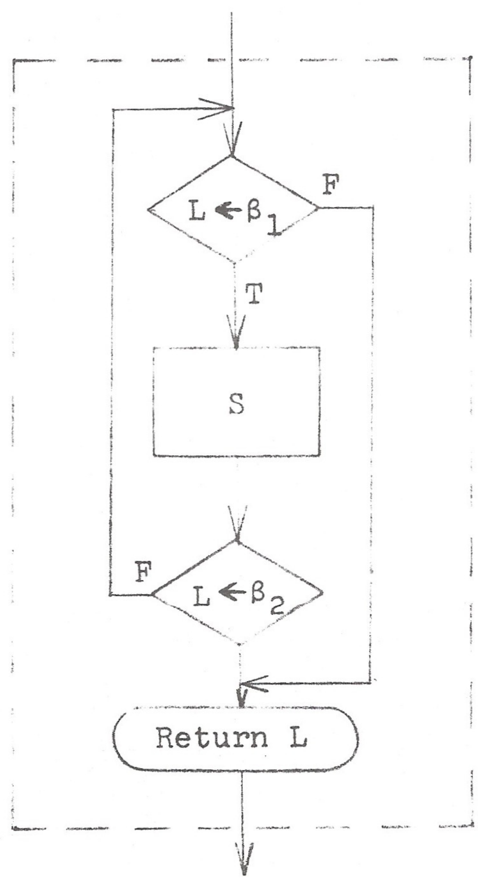

Fig. 4 value returning while-until

There are two ways in which the loop may be terminated: the programmer will want to ascertain which of the two Booleans caused termination. This is precisely the information provided by the value of the while-until.

Using the value of the while-until, we may easily incorporate the above search routine into an insertion.

```
comment T[1:M] is a table containing N < M active
    elements. Insert A in T if it is not already
    present;
low := 0;
high := N + 1;
if (while low < high - 1 repeat
    try := (low + high) / 2
    if T[try] < A then low := try else high := try
  until T[try] = A)
  then Insertafter(A,T,low);
```

The previously presented copy routine can be incorporated into an algorithm that uses up to N output files, depending on the length of the input file:

```
comment Place one copy of file INPUT onto OUTPUT[1:N]
    as needed;
i := 1;
while i ≤ N repeat
    Open(OUTPUT[i])
    if (¬while Spaceleft(Output [i]) repeat
```

```
            Inbuffer(INPUT, b);
            Outbuffer(OUTPUT[i], b)
          until   Eof(INPUT))
        then Close(OUTPUT[i]); i := i + 1
until   Eof(INPUT);
```

Earlier, we showed that the <u>while-until</u> statement is definable in terms of just the <u>while</u> or just the <u>until</u>. This is not true, however, for the <u>while-until</u> expression. Peterson, Kasami, and Tokura [6], p. 506, have shown that "There exist flowcharts that cannot be translated into [<u>if</u> and <u>until</u>] programs with single-level exits, even if node splitting is allowed." Their example of such a flowchart is shown in Figure 5.

The following program using value-returning <u>while-until</u> and <u>if</u> expression is a translation of this flowchart.

```
s; if A  then begin a1;
                   if (while (while B repeat b1
                             until ¬(while ¬C repeat c2 until true))
                    repeat c1
                    until ¬(while (while D repeat d1
                                     until ¬(while ¬A repeat a2 until true))
                           repeat a1 until true))
                   then d2 else b2
               end
         else begin a2;
                    if (while (while D repeat d1
                                 until ¬(while ¬A repeat a2 until true))
                       repeat a1
                       until ¬(while (while B repeat b1
                                        until ¬(while ¬B repeat c2 until true))
                              repeat c1 until true))
                    then b2 else d2
               end
```

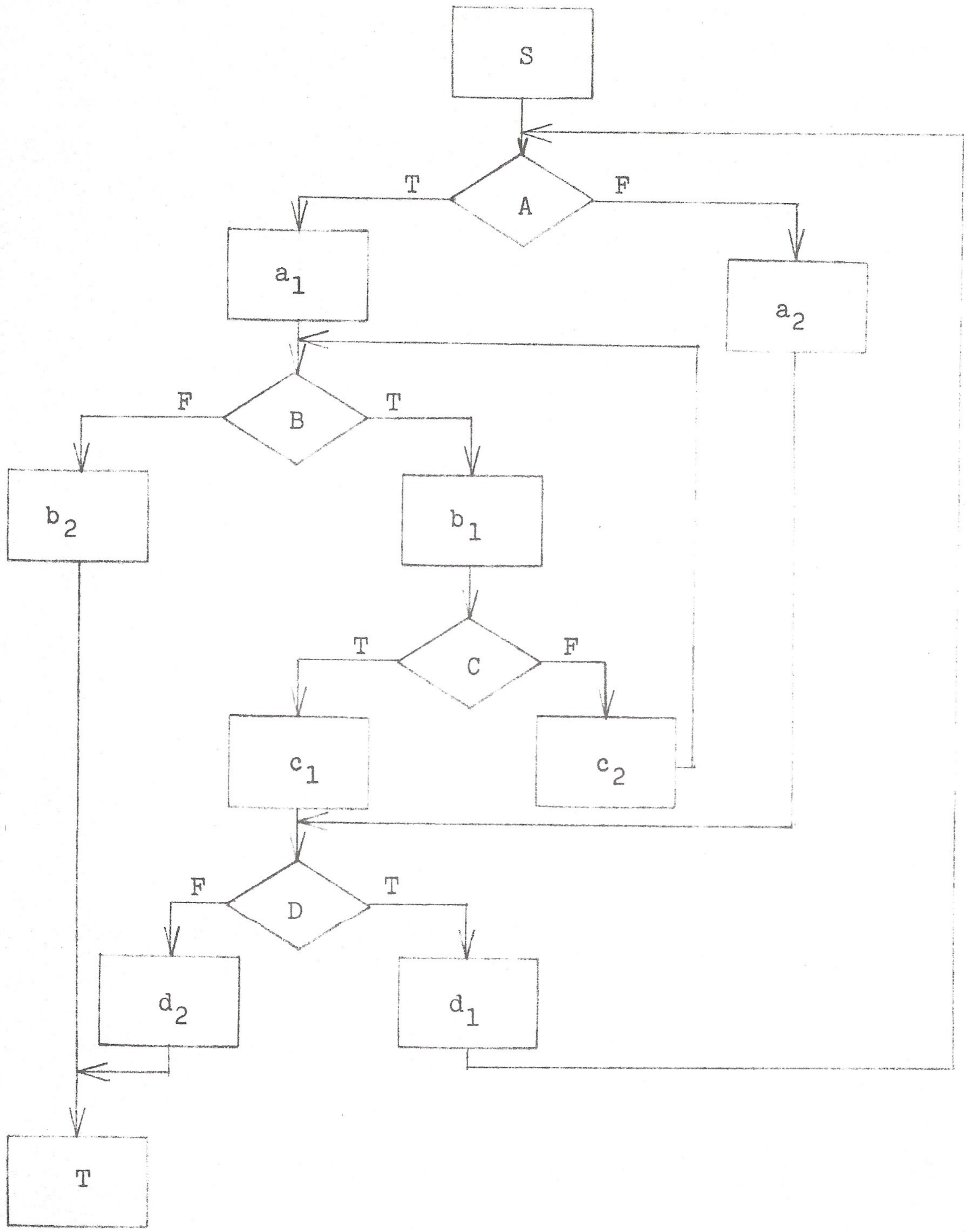

Fig. 5 Flowchart from Peterson, Kasami, and Tokura [6].

Similarly, Ashcroft and Manna [7] have exhibited a flowchart, shown in Figure 6, which cannot be translated into an <u>if</u> and <u>while</u> program. The following <u>while-until</u> program, due to M. Wand and D. Wise, is a translation of this flowchart.

```
if (while (if (while P repeat h until false) then true
                                              else Q)
    repeat h
    until ¬(while (if (while Q repeat g until false) then true
                                                     else P)
       repeat g until true))
then g else h
```

We have introduced the <u>while-until</u> as an additional control structure for structured programming. We have demonstrated cases in which use of the <u>while-until</u> results in more readable programs and allows programmers to program closer to the way they think. Although the <u>while-until</u> statement can be defined in terms of the <u>while</u> or the <u>until</u>, we have shown that the <u>while-until</u> expression is a stronger control structure than either.

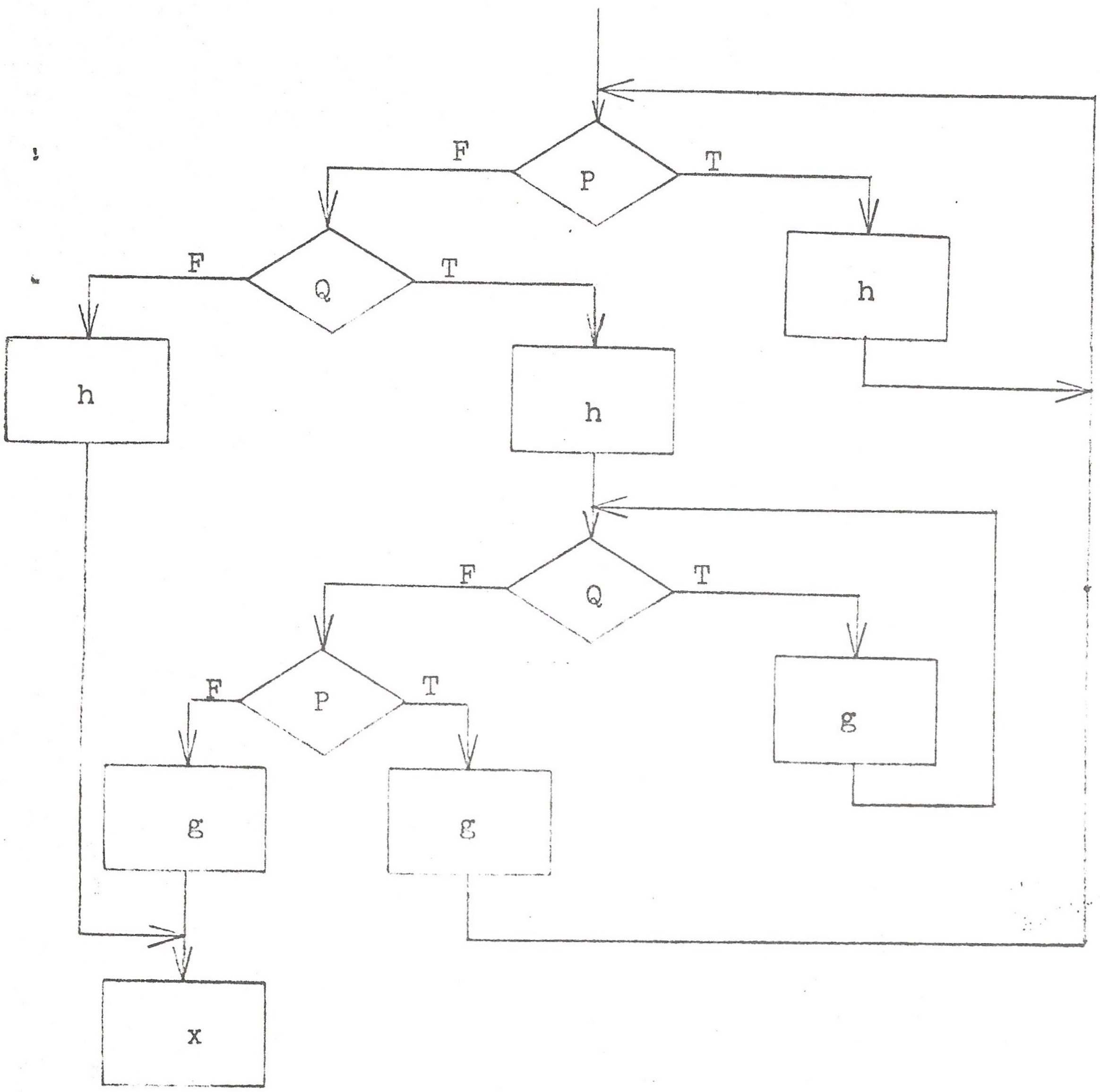

Fig. 6 Flowchart from Ashcroft and Manna [7].

References

1. Dijkstra, E. W., "Notes on structured programming," Structured Programming, Academic Press, London, 1972, pp. 1-82.

2. Friedman, D. P., Shapiro, S. C., "String union as an operation for control structure generation," (in progress).

3. McCarthy, J., et al., LISP 1.5 Programmers Manual, MIT Press, Cambridge, Mass., 1962.

4. van Wijngaarden, A., Mailloux, B. J., Peck, J. F. L., Koster, C. H. A., Report of the Algorithmic Language ALGOL 68, ACM, New York, 1969.

5. Wulf, W. A., Russell, D. B., Habermann, A. N., "BLISS: a language for systems programming," CACM 14, 12 (Dec. 1971), pp. 780-790.

6. Peterson, W. W., Kasami, T., Tokura, N., "On the capabilities of while, repeat and exit statements," CACM 16,8, (Aug. 1973), pp.503-512.

7. Ashcroft, E., Manna, Z., "The translation of 'go to' programs to 'while' programs," Information Processing 71, North-Holland Publishing Co., 1972, pp. 250-255.