A LINGUISTIC CONTRIBUTION TO *GOTO*-LESS PROGRAMMING

We don't know where to GOTO if we don't know where we've COME FROM. This linguistic innovation lives up to all expectations

R. LAWRENCE CLARK*

Nearly six years after publication of Dijkstra's now-famous letter,[1] the subject of GOTO-less programming still stirs considerable controversy. Dijkstra and his supporters claim that the GOTO statement leads to difficulty in debugging, modifying, understanding, and proving programs. GOTO advocates argue that this statement, used correctly, need not lead to problems, and that it provides a natural, straightforward solution to common programming procedures.

Numerous solutions have been advanced in an attempt to resolve this debate. Nevertheless, despite the efforts of some of the foremost computer scientists, the battle continues to rage.

The author has developed a new language construct on which, he believes, both the pro- and the anti-GOTO factions can agree. This construct is called the COME FROM statement. Although usage of the COME FROM statement is independent of the linguistic environment, its use will be illustrated within the FORTRAN language.

Unconditional COME FROM statement

General Form

COME FROM xxxxx

Where: xxxxx is the number of an executable statement in the same program unit.

This statement causes control to be transferred to the next statement (the statement immediately following the COME FROM upon completion of the designated statement.

Example:

```
    10 J = 1
    11 COME FROM 20
    12 WRITE (6,40) J
       STOP
    13 COME FROM 10
    20 J = J+2
    40 FORMAT (I4)
```

Explanation:

In this example, *J* is set to 1 by statement 10. Statement 13 then causes control to be passed to statement 20, which sets *J* to 3. Statement 11 then causes control to be passed to statement 12, which writes the current value of *J*. The STOP statement then terminates the program.

Conditional COME FROM statement

General Form

IF (cond) COME FROM xxxxx
Where: cond is any logical expression.
xxxxx is the number of an executable statement in the same program unit.

This statement causes control to be transferred to the next statement whenever the condition cond is true and the designated statement has just been completed.

Example:

```
    I = 1
    IF (I .LT. 10) COME FROM 50
    I = I+1
 50 WRITE (6,60) I
    STOP
 60 FORMAT (I4)
```

Explanation:

The COME FROM takes effect only while *I* is less than

*The author is indebted to G. F. Groner and N.A. Palley for a valuable discussion which took place in New Haven, Conn.

[1] E. W. Dijkstra, "GOTO Statement Considered Harmful," Letter to the Editor, *Communications of the ACM*, March 1968, pp. 147-148.

Reprinted courtesy of *Datamation Magazine*.

10. Thus when *I* is equal to 10, the program continues past statement 50 and terminates. This is equivalent to the now-obsolete formulations:

```
    I = 1
 30 I = I+1
    WRITE (6,60) I
    IF (I .LT. 10) GO TO 30
    STOP
 60 FORMAT (I4)
```

or

```
    DO 50 I = 2, 10
 50 WRITE (6,60) I
    STOP
 60 FORMAT (I4)
```

Note how much clearer is the intent of the code containing the COME FROM construct.

Computed COME FROM statement

General Form

COME FROM (x1, x2, x3,...,xn), i
Where: Each *x* is the number of an executable statement in the same program unit.
i is an integer variable.

This statement causes control to be transferred to the next statement whenever any of the following conditions holds:

- statement *x*1 has just been executed and *i* is equal to 1
- statement *x*2 has just been executed and *i* is equal to 2
- statement *x*3 has just been executed and *i* is equal to 3

  .
  .
  .
- statement *x*n has just been executed and *i* is equal to *n*

If, when statement *xj* is executed, *i* has any value other than *j*, this statement has no effect.

Example:

```
    DO 200 INDEX = 1,10
 10 X = 1.
 20 X = X*2.
 30 X = X*3.
 40 X = X*4.
 50 X = X*5.
 60 X = X*6.
 70 X = X*7.
 80 X = X*8.
 90 X = X*9.
100 X = X*10.
    COME FROM (10,20,30,40,50,60,70,80,90,100),
    INDEX
    WRITE (6,500). INDEX,X
200 CONTINUE
    STOP
500 FORMAT (I4,2X,F12.0)
```

Explanation:

This program illustrates the power of the computed COME FROM by providing a compact algorithm for computing factorials. On the first iteration (INDEX = 1), as soon as statement 10 has been executed, control passes to the WRITE statement. As a more general case, consider the fifth iteration: *X* is set to 1, and then multiplied by 2., 3., 4. and 5. before control passes to the WRITE statement.

Assign and assigned COME FROM statements

General Form

ASSIGN xxxxx TO m

  .
  .
  .

COME FROM m, (x1, x2, x3,..., xn)
Where: *xxxxx* is the number of an executable statement. It must be one of the numbers *x*1,*x*2,*x*3,..., *xn*.
Each *x* is the number of an executable statement in the same program unit.
m is an integer variable which is assigned one of the statement numbers *x*1,*x*2,*x*3,..., *xn*.

The assigned COME FROM causes control to be transferred to the next statement upon completion of the statement whose number is currently assigned to *m*. This provides a convenient means of passing control to a common point from a variety of points in the program unit. The actual point from which control is to be passed can be selected under program control.

Example:

```
    DO 60 I = 6,32
 20 X = I*6 + 14
    IF (X - 20.) 10, 30, 50
 10 ASSIGN 40 TO JUMP
 30 Y = 2*X**2. - 17.4
    COME FROM JUMP, (40,20,30)
    ASSIGN 30 TO JUMP
    X = X*Y - X**2
 40 ASSIGN 40 TO JUMP
    IF (Y - X) 20, 60, 50
 50 ASSIGN 20 TO JUMP
 60 CONTINUE
```

Explanation:
This example is self-explanatory.

The author feels that the COME FROM will prove an invaluable contribution to the field of computer science. It is confidently predicted that this solution will be implemented in all future programming languages, and will be retrofitted into existing languages. Although it is clear that the COME FROM statement fulfills most of the requirements of the advocates of GOTO-less programming, it remains for the practitioners of automatic programming to evaluate just how much this construct contributes to the development of automatic proofs of program correctness. Having at last put to rest the GOTO controversy, we now may enter the era of the COME FROM conundrum.