A FORTRAN to ALGOL translator

By D. Pullin*

This article gives a brief description of a translator which will accept a program written in FORTRAN II or FORTRAN IV and produce an ALGOL procedure acceptable to any ALGOL compiler.

It is probable that the controversy concerning the respective merits of FORTRAN (IBM, 1961, 1962) and ALGOL (Naur, 1962) will never be resolved. Even to one who is convinced that ALGOL is a superior language, the fact that so many useful routines are already available in FORTRAN is a powerful argument for the latter.

One way of resolving this difficulty is to provide a routine which will translate from FORTRAN source language into ALGOL source language. The latter can then be translated by any normal ALGOL compiler. The translating routine has itself been written in ALGOL, thus enabling future modifications of FORTRAN to be easily incorporated and ensuring that any establishment possessing an ALGOL compiler can make use of the considerable FORTRAN library.

This two-stage technique has several advantages over the provision of a normal FORTRAN compiler. Both the translator and all the routines it translates will be available on any future system as soon as the ALGOL compiler has been written. Existing FORTRAN subroutines and functions can be independently translated and incorporated as procedures in ALGOL programs. The problems of object program optimization can be safely left to the ALGOL compiler, and as the translator is designed to accept routines which are known to be working, the normal debugging facilities may be omitted. FORTRAN programs could, of course, be transcribed into ALGOL manually, but the tedium involved, and the consequent large scope for human errors, render this alternative highly unattractive.

Basic principles

FORTRAN programs are written as a group of subprograms all but one of which are designated as functions or subroutines. The translator reads in each of these as a unit and puts out equivalent ALGOL statements. Each subroutine is translated into a procedure, each function into a type procedure. After the main subprogram has been translated, an extra output occurs, containing the declarations of the global variables and certain initialization statements. The output, together with some standard procedures, may then be compiled by ALGOL. It is also possible for a function or a subroutine to be translated independently and the resulting procedure used in an ALGOL program.

A great deal of the actual translation is easily achieved. For example the arithmetic expression

```
A (I, 3*J) = 6.   *B + DEL*(Q**T + 3·6*X)
```

is simply transcribed as

```
A [I, 3*J] := 6·0*B + DEL*(Q ↑ T + 3·6*X);
```

Since all arrays must appear in some dimension statement there is no difficulty in deciding whether [or (is required at any point.

In the present version of the Elliott ALGOL translator (as in the ECMA subset(Naur, 1963)) labels must begin with a letter. This condition is satisfied by placing an *l* before every FORTRAN label. This cannot produce a clash with another identifier as the FORTRAN character set contains upper-case letters only.

Thus 2173 GOTO 69

becomes *l*2173: **goto** *l*69;

The effects of a FORTRAN IF can always be achieved by two conditional statements as follows:

IF (A(I) — B) 10, 20, 30

if A [I] — B $<$ 0 **then goto** *l*10;

if A[I] — B = 0 **then goto** *l*20 **else goto** *l*30;

For each computed GOTO a **switch** is declared at the head of the ALGOL block and a **goto** inserted into the body.

GOTO I, (9, 11, 15, 13)	may produce
switch s7 := *l*9, *l*11, *l*15, *l*13;	in the head and
goto s7 [I];	in the body

The switch identifiers are simply formed by an s followed by a count.

Assigned GOTOs are slightly more difficult and rather less efficient. They are, however, of less frequent occurrence (they do not appear at all in FORTRAN II) and so this should not be too serious.

ASSIGN 17 TO I

is translated into

```
I := 17;
```

GOTO (17, 23, 2), I

is translated into

* *Elliott Brothers (London) Ltd., Elstree Way, Borehamwood, Herts.*

```
if I = 17 then goto l 17;
if I = 23 then goto l 23;
if I =  2 then goto l  2;
```

Subroutine entries, as for instance

```
CALL PRECOM (SUM, DIFF, 2*J + K)
```

convert into **procedure** calls as

```
PRECOM (SUM, DIFF, 2*J + K);
```

Function calls are identical in the two languages.

A RETURN at the end of a sub-program is omitted entirely in ALGOL, and if it occurs elsewhere it is translated typically into

```
goto d007;
```

and a specially invented label is attached to the end of the ALGOL block.

A dictionary is kept of standard FORTRAN functions which are also standard ALGOL procedures.

```
X = SQRTF (A + SINF (B))
```

is accordingly rendered as

```
X := sqrt (A + sin (B));
```

DO loops vary slightly from **for** statements, so they cannot be simply transcribed. Instead a count of DO loops is kept and a unique label of the form d004 allocated to each. A group of equivalent statements is compiled as shown below

```
      DO 20 K = 3, N, 2
      :
      .
      20 P = Q + R(K)
```

turns into

```
      K := 3;
d004: .
      .
      .
      .
 l20: P := Q + R [K];
      K := K + 2;
      if K ⩽ N then goto d004;
```

If a CONTINUE is encountered it is omitted completely.

Advanced concepts

There are several aspects of the problem which present much greater difficulties, and may, in extreme cases, require manual intervention. Unlike FORTRAN, ALGOL requires that all variables be declared at the beginning of each block. Thus before translation can begin, the sub-program has to be scanned and a list of all variables gathered into groups for declaration. At the same time the DIMENSION statements are used to provide details of the arrays. As ALGOL is much less restrictive than FORTRAN concerning these, no trouble can arise here. Any variables which occur in COMMON statements will be treated as global in the ALGOL program. The first time a variable appears in the COMMON workspace, its name is placed in a dictionary. In a subsequent sub-program the same variable may (by means of COMMON) be used with a different name. The translator will substitute the original name wherever it appears and eventually include it in a global declaration. A similar technique is used for EQUIVALENCE.

The following example may help to clarify this.

```
SUBROUTINE BSUM (A, C, D)
COMMON B, B1, X
B = A*C + D
B1 = A/C − X
RETURN
END
SUBROUTINE QUOT (W)
COMMON AA, BB, CC
BSUM (R, S, T)
CC = BB*R
W = AA/S
RETURN
END
```

The ALGOL version of these statements would be

```
begin own real B, B1, X;
procedure BSUM (A, C, D); real A, C, D;
begin B := A*C + D;
B1 := A/C − X;
end;
procedure QUOT (W); real W;
begin      BSUM (R, S, T);
X := B1*R;
W := B/S;
end;
end;
```

This example is obviously not complete but has been written merely to illustrate a point.

In FORTRAN, variables of different mode and arrays of different dimension structure may be placed together in COMMON. Should this occur, the name substitution described above will not work. It is, however, probable

that this facility will be used to save space, rather than to transmit values. Accordingly the translator will allocate separate space for the offending variables and put out a warning message of the form

INCONSISTENT COMMON A, B

It then examines the previous rejects, if any. On finding a consistent one which has been ejected from the same slot it will put out

COMMONED WITH C

If value transmission is required in the rejected case, appropriate orders may be inserted manually.

Input and output

To convert FORTRAN input/output statements into a form acceptable by Elliott ALGOL a two-stage process has been developed. Two procedures called setformat (a, b) and fortout (y, z) have been written in ALGOL and are added to every translated program. Setformat specifies that input/output device a is to be used, and data transmission is under control of a FORMAT held in location b.

Fortout transmits the value y. z is a Boolean parameter, which specifies whether input or output is required.

In the body of the program an input/output order is replaced by a call of setformat to initialize the process followed by one call of fortout for every item on the list. If the list contains an implied DO loop, it is replaced by a **for** statement.

For example

```
PRINT 7 A, B, C, (D(I) I = 1, 10)
```

might be translated as

```
z := true;
setformat (1, 15);
fortout   (A, z);
fortout   (B, z);
fortout   (C, z);
for I := 1 step 1 until 10 do fortout (D[I], z);
```

The FORMAT statements themselves are put out by the translator between string quotes. Orders are placed at the beginning of the program to re-input these strings and pack them in an array where they may be accessed by fortout. This ensures that FORMAT statements may be read in at run time if desired.

To enable this system to work with a different ALGOL compiler (which will in general employ different input/output procedures) only setformat and fortout need be rewritten.

The translator has been designed to work as far as possible with either FORTRAN II or IV, and will accept most programs written in either language. The machine-dependent features of some compilers naturally cannot be accepted, and the more esoteric attempts to "fool the compiler" will result in an error message appearing during translation or at run time. However, provided no rules of the FORTRAN specification have been sidestepped, a program of any degree of complexity will be translated without hesitation and may be run on any ALGOL compiler.

There follows below a FORTRAN subroutine, and the ALGOL **procedure** which was produced by the translator.

```
      SUBROUTINE SOLVE
      COMMON A, B, C, X1R, X1I, X2R, X2I
      DISC = B**2 - 4. * A * C
      IF(DISC) 50, 60, 70
50    X1R = - B/(2.*A)
      X2R = X1R
      X1I = SQRTF(-DISC)/(2.*A)
      X2I = - X1I
      RETURN
60    X1R = - B/(2.*A)
      X2R = X1R
      X1I = 0.
      X2I = 0.
      RETURN
70    S = SQRTF(DISC)
      X1R = (-B + S)/(2.*A)
      X2R = (-B - S)/(2.*A)
      X1I = 0.
      X2I = 0.
      RETURN
      END
```

```
procedure SOLVE;
begin begin own real DISC, S;
switch ss := l50, l60, l70, d 001;
DISC := B ↑ 2 - 4.␣000*A*C;
if (DISC) < 0 then goto l50;
if (DISC) = 0 then goto l60 else goto l70;
l50 :X1R := - B/(2.␣000*A);
X2R := X1R;
```

```
X1I := sqrt(−DISC)/(2.␣000*A);
X2I := − X1I;
goto d 001;
l60:X1R := − B/(2.␣000*A);
X2R := X1R;
X1I := 0.␣000;
X2I := 0.␣000;
goto d 001;
l70:S := sqrt(DISC);
X1R := (−B + S)/(2.␣000*A);
X2R := (−B − S)/(2.␣000*A);
X1I := 0.␣000;
X2I := 0.␣000;
d 001: end end;
```

References

NAUR, P. (Editor) (1962). "Revised Report on the Algorithmic Language ALGOL 60," *The Computer Journal*, Vol. 5, p. 349.
NAUR, P. (Editor) (1963). "ECMA Subset of ALGOL 60," *Communications of the A.C.M.*, Vol. 6, p. 595.
IBM (1961). Reference Manual 709/7090 FORTRAN Programming System (January 1961).
IBM (1962). 7040/7044 Programming Systems FORTRAN IV Advance Specifications.

Book review: Computers in research

Digital Computers in Research. An Introduction for Behavioral and Social Scientists, by BERT F. GREEN, Jr., 1963; 333 pages. (London: *McGraw-Hill Publishing Company Ltd.*, 83s. 6d.)

In his preface, the author says that this book began as a short report intended as an introduction to the uses of computers in psychology; it appears as a book which includes an introduction to computer programming, together with an account of some research applications of computers in the psychological and social fields.

The introduction to computer programming occupies the first third of the book, and is quite the best of its kind known to the reviewer. It is written simply and clearly in a fresh and engaging style, and is without the usual confusing material which properly belongs to numerical mathematics and the art of numerical calculation. Basic programming is described in terms of a hypothetical computer with a simple one-address code and counter-modifier index registers; algebraic compilation is illustrated by an introduction to FORTRAN; and there is an introduction to linked-list storage and processing as a programming technique of especial value in the behavioral field. The computer is throughout described as a digital information-processing machine with applications to numerical calculation. Psychologists and the like will not only learn the elements of programming from this introduction; they will also come to a just appreciation of phrases like "electronic brain" and "machine intelligence". A final section of the book describing logical components and also elementary Turing-machine theory will particularly interest them in this connection.

The greater part of the book—which is in fact independent of the section on programming—consists of a series of chapters devoted to applications. These include statistical data analysis; the use of computers for generating experimental stimulus-patterns; computer models of psychological processes; man-machine complexes like those used for satellite tracking; and the author's own "Baseball" project for man-machine communication in natural language. These are selected topics, introduced in an informal and persuasive fashion; they are presented as hors-d'œuvres to stimulate the reader's appetite rather than as the solid meat of exhaustive review.

The book is warmly recommended to its intended audience, and indeed to all who are interested in the wider applications of computers. The reviewer would in addition welcome the publication of the introduction to programming as a separate booklet for novice programmers.

W. L. B. NIXON