The classification of FORTRAN statements

A. H. J. Sale

Basser Computing Department, University of Sydney, Sydney, Australia

This paper discusses the problem of classifying ANSI FORTRAN statements into types, for example: assignment statements, DO statements, etc. The various problems that arise are explained, with suggested solutions. An algorithm is presented which can type ANSI FORTRAN statements into 36 classes.

(Received June 1969)

The classification of FORTRAN source statements from the lines presented to it is usually one of the first tasks of a FORTRAN compiler, or other FORTRAN source-language processors. The determination of the type of a statement is not completely trivial, although the structure of FORTRAN lends itself to a line-by-line classification, and it is surprising to find that so few algorithms for this purpose have been published.

The classification scheme presented here is built around three phases. Most FORTRAN statements can be identified by an initial keyword, the recognition of which is attempted in the second phase. The purpose of the first phase is to remove those statement types not characterised by keywords and which may therefore confuse the second phase. Examples are comments and assignment statements. The third phase is to make finer distinctions, for example between a logical IF and an arithmetic IF. By the *ad hoc* nature of FORTRAN, this phase consists of very specific tests. This structure has proved capable of accepting wide variations in dialects.

The discussion to follow will assume that syntactically correct ANSI FORTRAN statements are presented to the classifier routine in the form of a 72 character array, which represent columns 1 to 72 of a FORTRAN line. Treatment of one line only means that there are restrictions on statements that may be successfully classified, but is desirable to eliminate fruitless scans over long statements.

It is obvious that comment statements must be isolated first, for columns 2 to 72 of a comment line may contain any FORTRAN characters, and even syntactically correct FORTRAN statements. This is very easy to do: the C in column 1 identifies them.

Secondly, continuation lines must be isolated, for their type is in general indeterminable, being at least partially determined by the initial line that precedes them. According to the American National standard either the character blank or the character zero in column 6 of a line signifies an initial line, all other characters signify a continuation line.

The third type that should be separated is the assignment statement, for there are no FORTRAN reserved words, and it is legal to use variable names such as READ, GOTO2, DO3J, etc. There are several difficulties here which confuse the basic recognition rule of scanning the statement for an equals sign not enclosed in parentheses.

The DO statement poses the first problem, for it too has an equals sign not within any parentheses, and the recognition rules must distinguish between:

```
DO2J=1,5
```

and

```
DO2J=1.5
```

A scan for a comma not in parentheses allows DO statements to be differentiated from assignment statements.

The logical IF also gives rise to problems, for it is really two statements in one: the IF test and an executable statement. Since most of the executable statements are permitted in a logical IF, they should not be permitted to confuse the recognition rule. The natural way therefore to treat this problem is to ensure that the scan is terminated at most one non-blank character after encountering a right parenthesis at level 0. This ensures that the equals sign of an assignment statement is always found, while the statement following a logical IF expression will never be scanned past its first character. A DO statement cannot contain any parentheses.

Hollerith constants, which in ANSI FORTRAN can appear in FORMAT statements, in DATA statements, and in CALL statements (as actual parameters of subroutine calls), also create problems. Hollerith constants may contain any characters acceptable to the system, and consequently may wreck any scan scheme that ignores their existence. The only way to avoid this problem is to recognise a Hollerith constant on meeting it, and then either to skip over the constant, or to terminate the scan. Since Hollerith constants may only follow a left parenthesis (FORMAT and CALL), or a comma (FORMAT, DATA, and CALL), or a slash (FORMAT and DATA), or an asterisk (DATA), occurrences of these characters should initiate a special sequence which looks for the unsigned integer constant followed by an H which characterises such a constant.

There is one insoluble problem on a line-by-line classification basis: statement function definitions, arithmetic assignments, and logical assignments are not distinguished by syntax alone. While some occurrences can possibly be classified, there are cases which cannot be classified without a knowledge of the variable and array declarations, for example:

```
MOD(I,J) = I - (I/J)*J
BOOL = IJK
```

All three types are therefore assigned the same type code.

Practically all statements must pass through the assignment scan, and it should therefore be as efficient as possible. Any conditions which indicate that there is no need to continue the scan further might well be noted and used, thus reducing the scan length. The algorithm suggested is therefore to scan for upper-level equals signs and commas (to identify assignment statements and DO statements, and incidentally assigned GO TOs), while watching for Hollerith constants. The scan may be terminated under the following conditions:

1. On meeting an upper-level comma.
2. On finding a Hollerith constant.
3. On meeting an equals sign in parentheses.
4. One non-blank character after the parenthesis level drops to zero.
5. On reaching the end of a line.

Nearly all other statements, after the comments, continuations, assignments and statement function definitions, and DOs have been removed, can be classified by their initial keywords. The DO can also be classified this way, but the assignment scan produces it as a by-product. This scan should of course be efficient, for there is a fairly large number of keywords to be tested. Probably the fastest and most flexible way to organise this scan is around a linked binary tree. The arrangement of links can be chosen to make common statements fast to classify, and to optimise the overall scan time. Statements which are not classified after this scan are either in error, or do not comply with the recognition rules, and they should be given a 'rogue' type code.

There remain three more problems. The first is that simple GO TOs and assigned GO TOs are not differentiated by their keywords. Separation is most easily done by checking to see if a comma was found in the earlier scan, which identifies an assigned GO TO.

Secondly, it is desirable to give arithmetic and logical IF statements different type codes. This is done by checking the first non-blank character after the closing parenthesis of the IF test, which must be a numeric digit in an arithmetic IF, and cannot be numeric in a logical IF.

The third problem concerns typed functions such as:

REAL FUNCTION RANDOM (A,B)

It is probably better that such statements be classified as FUNCTION statements and not as INTEGER, REAL, etc. All type statements (INTEGER, REAL, DOUBLE PRECISION, COMPLEX and LOGICAL) should therefore be checked for the occurrence of the 8-character word 'FUNCTION', which cannot be confused with any 6-character name.

A statement, to be recognisable from its initial line, must therefore satisfy the following conditions:

1. Comments, continuations and END statements are always recognisable.
2. An assignment statement must have up to and including the equals sign on the initial line.
3. DO statements and assigned GOTOs must have up to and including the first comma on the initial line.
4. Arithmetic IFs must have the first digit of the first statement label on the initial line.
5. All other statements must have the minimum key characters as shown in **Table 1** on the initial line, except for typed function declarations which must have up to and including the word 'FUNCTION' on the initial line. It can be seen that these requirements are far from restrictive, and very seldom will they be the cause of classification failure.

A FORTRAN subroutine is presented which classifies ANSI FORTRAN lines. It has been written almost entirely to comply with the requirements of Basic FORTRAN; the exceptions being the DATA statement and the implication of an A1 format type. Table 1 shows the integer type-codes assigned to the various statement types, which have been grouped into a reasonable order.

Table 1 **Statement type codes**

CODE	DESCRIPTION	KEY CHARACTERS	REMARKS
1	comment	—	C in column 1
2	continuation	—	not blank or 0 in column 6
3	assignment	—	special rules
4	ASSIGN	A	
5	GO TO	GOTO	see 6
6	assigned GO TO	GOTO	separated from 5 by special rules
7	computed GO TO	GOTO(	
8	arithmetic IF	IF	see 9
9	logical IF	IF	separated from 8 by special rules
10	DO	—	special rules
11	CONTINUE	CON	
12	CALL	CA	
13	RETURN	RET	
14	STOP	ST	
15	PAUSE	P	
16	READ	READ	
17	WRITE	W	
18	REWIND	REW	
19	BACKSPACE	BA	
20	ENDFILE	ENDF	
21	FORMAT	FO	
22	INTEGER	IN	need to be further checked for FUNCTION type
23	REAL	REAL	
24	DOUBLE PRECISION	DOU	
25	COMPLEX	COMP	
26	LOGICAL	L	
27	EXTERNAL	EX	
28	DIMENSION	DI	
29	COMMON	COMM	
30	EQUIVALENCE	EQ	
31	DATA	DA	
32	BLOCK DATA	BL	
33	FUNCTION	FU	
34	SUBROUTINE	SU	
35	END	END	
36	rogue type	—	unclassified statements

The algorithm

```
C.....TO CLASSIFY USAS FORTRAN RECORDS INTO 36 CLASSES
      SUBROUTINE CLASS(K,ITYP)
C.....          *****
C.....SPECIFICATION
      DIMENSION K(72)
C*************************************************************************
C*    USAS FORTRAN RECORD CLASSIFIER  -CLASS-                            *
C*                                                                       *
C*    LANGUAGE... .  .                                                   *
C*        USAS FORTRAN (BUT NEARLY ALL USAS BASIC FORTRAN)               *
C*    INPUT... .  .                                                      *
C*        K - AN INTEGER ARRAY CONTAINING 72 ALL CHARACTERS (NOT ALTERED)*
C*    OUTPUT... .  .                                                     *
C*        ITYP - AN INTEGER TYPE CODE FROM 1 TO 36                       *
C*    ERROR EXITS... .  .                                                *
C*        NONE                                                           *
C*    SUBROUTINES REQUIRED... .  .                                       *
C*        KCOMP - A MACHINE DEPENDENT INTEGER FUNCTION THAT ACCEPTS      *
C*        AS ARGUMENTS AN AI CHARACTER AND A HOLLERITH CONSTANT, AND     *
C*        RETURNS 0 IF THEY REPRESENT THE SAME CHARACTER, OTHERWISE 1    *
C*    TYPE CODES... .  .                                                 *
C*          1 COMMENT       2 CONTINUATION 3 ASSIGNMENT   4 ASSIGN     *
C*          5 GO TO         6 ASSI.. 7 T.  7 ..MP, GO TO  8 ARITH. IF  *
C*          9 LOGICAL IF   10 33           11 CONTINUE    12 CALL      *
C*         13 RETURN       14 STOP        15 PAUSE       16 READ      *
C*         17 WRITE        18 REWIND      19 BACKSPACE   20 ENDFILE   *
C*         21 FORMAT       22 INTEGER     23 REAL        24 DOUB. PREC.*
C*         25 COMPLEX      26 LOGICAL     27 EXTERNAL    28 DIMENSION *
C*         29 COMMON       30 EQUIVALENCE 31 DATA        32 BLOCK DATA*
C*         33 SUBROUTINE   34 FUNCTION    35 END         36 *ROGUE*   *
C*************************************************************************
C.....DECLARATIONS
      DIMENSION KALP(46),KSUC(45),KFAL(46),KDEC(10),KF(8)
C.....LENGTH OF ARRAY K
      DATA LENG /72/
C.....ALPHAMERIC CONSTANTS AND MATCH TABLES
C.....THE DECIMAL DIGITS FOR INTEGER RECOGNITION
      DATA KDEC(1),KDEC(2),KDEC(3),KDEC(4),KDEC(5),
     1 AND 1     KDEC(6),KDEC(7),KDEC(8),KDEC(9),KDEC(10)
     2 AND 2     /1HG,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
C.....THE WORD -FUNCTION- FOR MATCHING
      DATA KF(1),KF(2),KF(3),KF(4),KF(5),KF(6),KF(7),KF(8)
     1 AND 1     /1HF,1HU,1HN,1HC,1HT,1HI,1HO,1HN/
C.....CHARACTERS FOR RECOGNITION TESTS
C.....       WHICH ARE...   C BLANK ( ) = H / * $ ,
      DATA KC,KBLNK,KLPAR,KRPAR,KEQ  /1HC,1H ,1H(,1H),1H=/
      DATA KH,KSLSH,KASTK,KZERO,KCMA /1HH,1H/,1H*,1H0,1H,/
C.....CHARACTER ARRAY FOR TREE SCAN
      DATA KALP( 1),KALP( 2),KALP( 3),KALP( 4) /1HI,1HF,1HN,1HG/
      DATA KALP( 5),KALP( 6),KALP( 7),KALP( 8) /1HO,1HT,1HO,1H(/
      DATA KALP( 9),KALP(10),KALP(11),KALP(12) /1HC,1HA,1HO,1HN/
      DATA KALP(13),KALP(14),KALP(15),KALP(16) /1HM,1HM,1HP,1HR/
      DATA KALP(17),KALP(18),KALP(19),KALP(20) /1HE,1HA,1HD,1HL/
      DATA KALP(21),KALP(22),KALP(23),KALP(24) /1HT,1HH,1HE,1HO/
      DATA KALP(25),KALP(26),KALP(27),KALP(28) /1HO,1HO,1H1,1HA/
      DATA KALP(29),KALP(30),KALP(31),KALP(32) /1HO,1HO,1HW,1HS/
      DATA KALP(33),KALP(34),KALP(35),KALP(36) /1HT,1HO,1HE,1HN/
      DATA KALP(37),KALP(38),KALP(39),KALP(40) /1HD,1HF,1HX,1HO/
      DATA KALP(41),KALP(42),KALP(43),KALP(44) /1HB,1HA,1HC,1HA/
      DATA KALP(45),KALP(46)                   /1HL,1HP/
C.....SUCCEED LINK FOR TREE SCAN
      DATA KSUC( 1),KSUC( 2),KSUC( 3),KSUC( 4) /  2, -8,-22,  5/
      DATA KSUC( 5),KSUC( 6),KSUC( 7),KSUC( 8) /  6,  7,  8, -7/
      DATA KSUC( 9),KSUC(10),KSUC(11),KSUC(12) / 10,-12, 12,-11/
      DATA KSUC(13),KSUC(14),KSUC(15),KSUC(16) / 14,-29,-25, 17/
      DATA KSUC(17),KSUC(18),KSUC(19),KSUC(20) / 18, 19,-16,-23/
      DATA KSUC(21),KSUC(22),KSUC(23),KSUC(24) /-13,-18, 24,-21/
      DATA KSUC(25),KSUC(26),KSUC(27),KSUC(28) /-34, 27,-28,-31/
      DATA KSUC(29),KSUC(30),KSUC(31),KSUC(32) / 30,-24,-17, 33/
      DATA KSUC(33),KSUC(34),KSUC(35),KSUC(36) /-14,-33, 36, 37/
      DATA KSUC(37),KSUC(38),KSUC(39),KSUC(40) / 38,-22,-27,-39/
      DATA KSUC(41),KSUC(42),KSUC(43),KSUC(44) / 42,-19,-32, -4/
      DATA KSUC(45),KSUC(46)                   /-26,-15/
C.....FAIL LINK FOR TREE SCAN
      DATA KFAL( 1),KFAL( 2),KFAL( 3),KFAL( 4) /  4,  3,-36,  9/
      DATA KFAL( 5),KFAL( 6),KFAL( 7),KFAL( 8) /-36,-36,-36, -5/
      DATA KFAL( 9),KFAL(10),KFAL(11),KFAL(12) / 16, 11,-36, 13/
      DATA KFAL(13),KFAL(14),KFAL(15),KFAL(16) /-36, 15,-36, 23/
      DATA KFAL(17),KFAL(18),KFAL(19),KFAL(20) /-36, 21, 20,-36/
      DATA KFAL(21),KFAL(22),KFAL(23),KFAL(24) / 22,-36, 26, 25/
      DATA KFAL(25),KFAL(26),KFAL(27),KFAL(28) /-36, 31, 28, 29/
      DATA KFAL(29),KFAL(30),KFAL(31),KFAL(32) /-36,-36, 32, 35/
      DATA KFAL(33),KFAL(34),KFAL(35),KFAL(36) / 34,-36, 41, 30/
      DATA KFAL(37),KFAL(38),KFAL(39),KFAL(40) /-36,-35, 40,-36/
      DATA KFAL(41),KFAL(42),KFAL(43),KFAL(44) / 44, 43,-36, 45/
      DATA KFAL(45),KFAL(46)                   / 46,-36/
C.....
C.....
C.....FIRST CHECK FOR COMMENT
      IF (K( *PR1(1),KC)) 2,1,2
    1 ITYP=1
      GO TO 55
C.....THEN CHECK FOR A CONTINUATION RECORD
    2 IF (K( *PR1(6),KBLNK)) 3,5,3
    3 IF (K(*PR1(6),KZERO)) 4,5,4
    4 ITYP=2
      GO TO 55
C.....INITIALIZE THE LOOP
    5 JSW=0
      ISW=0
      JEQ=0
      JCMA=0
      JHOLL=0
      JSAVE=KBLNK
C.....ASSIGNMENT SCAN LOOP
      DO 26 J=7,LENG
      JCH=K(J)
      IF (KCOMP(JCH,KBLNK)) 6,26,6
C.....ITS NOT BLANK, IS HOLLERITH SWITCH ON ... .  .
    6 IF (JHOLL) 12,12,7
    7 DO 8 L=1,10
      IF (KCOMP(JCH,KDEC(L))) 8,10,8
    8 CONTINUE
C.....FIRST TIME, NO INTEGER MEANS NOT HOLLERITH
      IF (JHOLL-1) 11,11,9
C.....OTHERWISE LOOK FOR THE H
    9 IF (KCOMP(JCH,KH)) 11,32,11
C.....STILL FITS HOLLERITH CONSTANT SYNTAX
   10 JHOLL=JHOLL+1
      GO TO 25
C.....NOT A HOLLERITH CONSTANT, SET SWITCH OFF
   11 JHOLL=0
C.....TEST OTHER CHARACTERS (),=/*
   12 IF (KCOMP(JCH,KLPAR)) 13,22,13
   13 IF (KCOMP(JCH,KRPAR)) 14,16,14
   14 IF (KCOMP(JCH,KCMA )) 15,22,15
   15 IF (KCOMP(JCH,KEQ  )) 16,23,16
   16 IF (KCOMP(JCH,KSLSH)) 17,21,17
   17 IF (KCOMP(JCH,KASTK)) 25,21,25
C.....RIGHT PARENTHESIS FOUND
   18 JSW=JSW-1
      IF (JSW) 19,19,25
C.....SET SWITCH TO ALLOW ONLY ONE MORE NON-BLANK CHARACTER
   19 ISW=1
      GO TO 26
C.....LEFT PARENTHESIS FOUND
   20 JSW=JSW+1
C.....SET HOLLERITH SWITCH FOR ( , / *
   21 JHOLL=1
      GO TO 25
C.....COMMA FOUND, CHECK LEVEL
   22 IF (JSW) 30,30,21
C.....EQUALS SIGN FOUND, CHECK LEVEL
   23 IF (JSW) 24,24,32
   24 JEQ=1
C.....TEST IF TERMINATED BY SWITCH SET
   25 IF (ISW) 26,26,27
C.....END OF ASSIGNMENT SCAN LOOP
   26 CONTINUE
      GO TO 28
C.....SAVE LAST CHARACTER IF TERMINATED EARLY
   27 JSAVE=JCH
C.....................................................................
C.....LEAVE SCAN AND COME HERE IF ... .  .
C.....     NO MORE CHARACTERS
C.....     ONE NON-BLANK CHARACTER AFTER A RIGHT PARENTHESIS
C.....NOT A DO, MIGHT BE ASSIGNMENT
   28 IF (JEQ) 32,32,29
   29 JTYP=3
      GO TO 55
C.....................................................................
C.....LEAVE SCAN AND COME HERE IF ... .  .
C.....     AN UPPER LEVEL COMMA FOUND
C.....MIGHT BE A DO, NOT AN ASSIGNMENT
   30 JCMA=1
      IF (JEQ) 32,32,31
   31 JTYP=10
      GO TO 55
C.....................................................................
C.....LEAVE SCAN AND COME HERE IF ... .  .
C.....     A HOLLERITH CONSTANT FOUND
C.....     AN EQUALS IN PARENTHESES
C.....     FAILURE OF DO AND ASSIGNMENT TESTS
C.....NEITHER A DO NOR AN ASSIGNMENT
C.....ENTER THE KEYWORD CLASSIFICATION
   32 J=1
      ISW=7
   33 JCH=K(ISW)
C.....IF A BLANK, IGNORE, GET THE NEXT
      IF (KCOMP(JCH,KBLNK)) 34,37,34
C.....TEST AGAINST CURRENT TREE CHARACTER
   34 IF (KCOMP(JCH,KALP(J))) 35,36,35
C.....CHARACTER DOES NOT MATCH, TRY THE NEXT IN TREE
   35 J=KFAL(J)
      IF (J) 39,39,34
C.....CHARACTER MATCHES, TRY NEXT IN TREE AND IN RECORD
   36 J=KSUC(J)
      IF (J) 39,39,37
   37 ISW=ISW+1
      IF (ISW-LENG) 33,33,38
C.....IF RUN OUT OF CHARACTERS, FORCE A ROGUE TYPE
   38 JCH=KBLNK
      GO TO 35
C.....................................................................
C.....CLASSIFICATION COMPLETED, FORM TYPE CODE
   39 JTYP=-J
C.....CHECK TO SEE IF MORE TREATMENT NEEDED
      IF (JTYP-5 ) 55,45,40
   40 IF (JTYP-8 ) 55,43,41
   41 IF (JTYP-22) 55,42,42
   42 IF (JTYP-26) 47,47,55
C.....................................................................
C.....LOGICAL IF SEPARATION TEST
   43 DO 44 L=1,10
      IF (KCOMP(JSAVE,KDEC(L))) 44,55,44
   44 CONTINUE
      JTYP=9
      GO TO 55
C.....................................................................
C.....SEPARATE ASSIGNED AND UNCONDITIONAL GOTOS
   45 IF (JCMA) 55,55,46
   46 JTYP=6
      GO TO 55
C.....................................................................
C.....CHECK WHETHER THIS IS A TYPE STATEMENT OR A TYPED FUNCTION
   47 L=1
      GO TO 52
   48 L=L+1
      IF (L-LENG) 49,49,55
   49 IF (KCOMP(K(L),KBLNK))   50,48,50
   50 IF (KCOMP(K(L),KF(ISW))) 51,53,51
   51 IF (ISW-1) 52,48,52
   52 ISW=1
      GO TO 50
   53 ISW=ISW+1
      IF (ISW-8) 48,48,54
   54 JTYP=34
C.....................................................................
C.....ALL RESULTS COME HERE FOR RETURN
   55 ITYP=JTYP
      RETURN
      END
```

References

AMERICAN NATIONAL STANDARDS INSTITUTE (1966). ANSI FORTRAN, X3.9-1966.

AMERICAN NATIONAL STANDARDS INSTITUTE (1966). ANSI Basic FORTRAN, X3.10-1966.

PYLE, I. C. (1964). Implementation of FORTRAN on ATLAS, *Introduction to System Programming*, Ed. P. Wegner, Academic Press, 1964.