SHARE PROGRAM LIBRARY AGENCY

PROGRAM NUMBER

036022

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA
(305) - 284-6257

SHARE PROGRAM LIBRARY SUBMITTAL FORM

SHARE PROGRAM LIBRARY AGENCY
Triangle Universities Computation Center
Post Office Box 12076
Research Triangle Park, North Carolina
27709 USA

Attention: Mr. Joe Ragland

SPLA CONTROL NUMBER:

This form should be completed and submitted with the program package to the SHARE Program Library Agency at the address shown above. Standards and instructions for submitting programs are in the "SHARE Program Library Standards Manual".

(1) Program Number (to be filled in by SPLA) 360D-03.6.022

(2) System Type (machine) ..... IBM S360 or S370

(3) Search Key ..... DECISION TABLE TRANSLATOR BASED ON LIST PROCESSING TECHNIQUES

(4) Programming Language ..... PL/I

(5) Author's Name and Address .....
Dr. Kenneth Conrow
Computing Center
Kansas State University
Manhattan, KS 66502

(6) Direct Inquiries to Name and Address ..... (if different than Author)
same

(7) Title of Program ..... DECTALB, A Decision Table Translator Based on List Processing Techniques

(8) Submitter's Installation Membership Code..... S1099

(9) Submitter's Own Program Identification and Suffix(Optional)... 03.6.003

(10) Primary Subject Code..... 03.6

(11) Operating or Monitor System Required MFT; minor changes only for MVT,VS

(12) New or Revision Code (if revision, show prior Program Number in Item 1).. new

(13) Year Completed..... 1973

(14) Date of Submittal..... Jan. 29, 1974

(15) Documentation (number of original pages submitted)..... 71

(16) Abstract (should contain sufficient information for a reader to determine the value of the program). Listed on the reverse side of this form are subjects which may serve as a guide for a descriptive abstract.

SHARE PROGRAM LIBRARY SUBMITTAL FORM

Subject Guide:

a. Purpose
b. Programming Language used
c. Version and modification level or release number
d. Field of application
e. Type of routine (main program, subroutine, etc.)
f. Specific description of machine requirements

ABSTRACT DECTALB, a DECision Table ALgorithm Based on list processing techniques, is a translator which converts programs or program segments written in decision tables into compilable PL/I coding. The use of a directory vector to control execution enables complete elimination of duplicate coding of stubs, complete freedom of reuse of stubs throughout a DECTALB block, and automatic rearrangement of condition stubs to reduce the overhead of rule selection. The execution time control section is so simple that it adds very little overhead at execution time. The version submitted is the bootstrap which was employed to implement a more complete system. The bootstrap implements the basic features mentioned above but does not incorporate elaborations like processing extended entry decision tables, provision of diagnostics, and acceptance of control options. DECTALB is written in PL/I, was debugged on the Level F compiler under MFT, and has been shown to work using the Optimizing compiler. It will run on systems supporting PL/I Level F. DECTALB is supplied as a PL/I source deck of about 800 statements.

(Please attach additional pages if necessary).....Total pages attached_____

Permission to Publish

"I hereby give the SHARE Program Library Agency permission to reprint, reproduce, and distribute this program."

(17) Signature of Submitter and Date Kenneth Conner, Jan. 24, 1974

(18) Signature of Installation Addressee E. A. Lunger, Jan 24, 1974

The documentation accompanying this submission of DECTALB is in two parts. One part is a paper describing the academically interesting aspects of the design and implementation of DECTAL. The other part is in the nature of a user's guide, detailing the information which the user will require in order to use the program successfully.

The accompanying listing of the source code for DECTALB was made using NEATER, our PL/I Source Statement Reformatter (SHARE 360D-03.6.018; CACM 13(11), 669(1970)). Hence the listing does not correspond in format to the dard images represented on the source tape. The indentation according to logical structure and the splitting of words at the end of line result from NEATER running with PRINT only enabled.

DEFINITIONS

We find that we employ the following terms in describing DECTAL and its mechanisms.

'DECTAL' is used generically to refer both to the bootstrap program, DECTALB, and to the complete version produced with the aid of the bootstrap. A specific reference to DECTALB makes an assertion applicable to the bootstrap, but not to the complete version of DECTAL.

An activity is any passage of PL/I coding accessible via an element of a label vector in the DECTAl output. There are two kinds of activity: task activity, which is activity specified in the stubs of the user's decision tables, and control activity, which is activity superposed by DECTAL in order for it to control the execution of the user's decision tables.

A DECTAL block is a collection of decision tables which are processed as single batch by the DECTAL translator. Complete freedom of cross referencing of statements and accessing of tables exists within a DECTAL block, but no such cross referencing and table accessing can be done to another block, which, after all, is translated separately.

In referencing specific tables or task activities, the notation Axx (actions), Cxx (conditions), or Txx (tables) is employed. In each case, the xx denotes a two digit decimal number which is unique in its environment.

ALGORITHM AND CONTROL MECHANISM

The translation algorithm and the details of the control mechanism at execution time are described in considerable detail elsewhere. To sum it up extremely briefly, a directory vector is used to control flow of execution among task activities and control activities using a subscripted label vector to access each one as required by the input tables and the execution time situation.

INPUT TO DECTAL

Block and Table Level

DECTAL is designed to permit its user to insert DECTAL blocks into normal PL/I source coding whenever it appears convenient to his purposes (except, for reasons given below, that only one DECTAL block can appear in a PL/I PROC or BEGIN block). DECTAL blocks have the same properties as PL/I BEGIN blocks in regard to flow of execution: they are entered by "falling into" them from the top and, when decision table execution is complete, the normal PL/I coding after the DECTAL block will be given control.

Each DECTAL block is delimited by an *DECTAL card at its beginning and an *END card at its end. A DECTAL block consists of one or more decision tables. PL/I statements appearing between the *DECTAL card and the first Txx card are transmitted to the output stream without inspection or modification. Declare statements for the user variables of the decision table block may, for example, appear here. Any executable statements in this environment will

be executed before the DECTAL control mechanism takes over, which occurs at the position of first occurrence of a Txx statement in the block. The first table in a block is given control at execution time at its initializing action statement (if any) or at its first condition statement (if there were no initializing actions). Flow of execution among the decision tables of a block is controlled by translator statements appearing as action stubs. Hence, any table in the block other than the first must be explicitly accessed by translator statements.

Each decision table is delimited at its beginning by a Txx card and at its end either by the Txx card of the next table or by the *END card of the DECTAL block. Each table consists of initializing actions (if any), a rules card (optional), condition statements (if any), and conditioned action statements (if any), generally in that order. Comments may be interspersed among these statements, and there is some freedom of placement of the rules card. A table may consist solely of a series of initializing actions, in which case it is simply an action table.

Initializing actions for a table are entered as Axx stubs in the usual way, but no entries may appear for them. Initializing actions are fully cross-referenceable, just as are all other actions and conditions. The end of a set of initializing actions is recognized by the appearance of the first condition stub. The rules card may appear before, among, or after the initializing action cards, but must appear prior to any condition statement. Any action statements which appear after one or more condition state-

ments are conditioned actions and must have action entries associated with them. All condition statements in a given table must appear in a group uninterrupted except possibly by comment cards.

The rule card (R) gives the user the opportunity to number his rules and specify the presence or absence of an ELSE rule in his table. If no rule card appears for a table, the rules will be numbered sequentially from left to right, beginning with 1. The rule card entries may be one or two digit numbers, separated either by blanks or commas. An ELSE rule, if any, must be the last rule to appear in a table. Its presence is signaled on the R card with an 'E' or the word 'ELSE'.

If an R card (rule card) was given, then the number of condition entries associated with each condition stub must equal the number of rules (other than the ELSE rule) on the rule card. The number of action entries associated with each action stub must equal the number of rules on the rule card (including the ELSE rule). If the ELSE rule was not specified, then the number of entries on both the condition and the action cards must be equal to the number of rules on the rules card. Since DECTALB has no diagnostic capability, the consequences of breaking the rules are unpredictable. If no rule card is given, the number of entries in the first condition statement will be taken to define the number of rules, and the presence or absence of an additional entry in the first action statement will be taken to define the presence or absence of an ELSE rule.

Comment cards (#) may appear at any point in the translator input. They do not interrupt blocks of cards constituting a statement, but are transmitted without inspection to the printed output on a whole-card basis. A series of comment cards must all bear the # signs in column 1. Target language comments which are intended to appear in the compiler input must not contain a # in column 1, as this causes them to be diverted from the compiler input stream. Target language comments imbedded in target language statements are transmitted without change. DECTAL comments and stubs or entries cannot co-occur on a single card, except that comments can intervene between a translator statement and any entries associated with it (see below).

Statement Level Input Format

The overriding consideration is to have a maximum of freedom in the input format, so that little care need be taken in formatting of the decision table statements for input to the processor. The stubs are written in the target language (i.e. PL/I); since the processor transmits the stubs to the output stream without change beyond surrounding them with strings which implement the decision table control mechanism, the free format properties of the target language are maintained in the stubs.

Either column 1 or columns 1-3 are used to identify statements for the processor. All cards except T, A, or C card employ only column 1; T, A, and C card employ columns 1-3. Columns 73-80

are reserved for identification sequence fields. The remaining columns, 2-72 or 4-72, are available for use for input text.

The characters which may appear in column 1, and the meaning of each are:

*: Block delimiter card.

T: Card for table initiation.

A: Card initiating an action statement.

C: Card initiating a condition statement.

} each must be followed by a 2 digit decimal number in cards 2 and 3

#: Translator comment card.

R: Rule card (optional).

blank: A continuation card.

On T, A, and C cards, the two digit decimal number in columns 2 and 3 must uniquely identify that statement in its environment. For example, only one T03 can appear in a given DECTAL block; only one A05 can occur in a given table; and only one C13 can occur in a given table. (Note that this intrinsically limits the number of tables in each DECTAL block, the number of actions in each table, and the number of conditions in each table to 100. More severe limits are imposed by dimensions in certain declare statements of DECTALB; see below.)

In condition and action stubs, the first character of the stub may appear in column 4 or any succeeding column.

The condition stub must contain the target language keyword 'IF'. This requirement enables pre-condition actions. The processor will supply the 'THEN' keyword of the PL/I target language.

If continuation cards are required by a lengthy A or C stub and entry, a blank appearing in column 1 of the following card signals the continuation. The text may be continued from column

2. A change in statement will be recognized only when a non-blank symbol appears in column 1 on a later card.

SPECIFICATION OF TABLE ENTRIES

Table entries are made on A and C cards (or their continuations) following the stubs to which they apply. The table entries are separated from the stubs with the delimiter '::'. Arbitrary blanks may be included before or after the double colon and before, among, or after the entries. Among the condition entries, only the sumbols 'T', 'F', 'Y', 'N', and '-' are acceptable to DECTALB. 'T' and 'Y' equivalently define an entry in a rule which may be employed if this condition is true, and 'F' and 'N' equivalently define an entry in a rule which may be employed if this condition is false. A '-' ('don't care') indicates that this condition is irrelevant to the rule.

Among the action entries, 'X' and '-' are the acceptable entries. An 'X' indicates that the action is to be taken if the rule is selected, and a '-' indicates that the action is not to be taken if the rule is selected.

Note that all entries, including don't care and no action entries, must be explicit (by use of hyphens). Blanks are never interpreted as entries.

No symbols other than X may appear in action entries to specify that a given action stub is to be executed in a rule. If it is desired to modify the order of execution of the action stubs, the modification should be effected by using translator cross

reference statements which reference action stubs within the same table. For example, if the action stubs A,B,C were to be executed by three different rules in the orders A,B,C, B,C,A, and C,A,B, this could be accomplished by inclusion of two cross references in the action stubs as briefly indicated below.

	1	2	3
*C	-	-	X
A	X	-	X
B	X	X	X
C	X	X	-
*A	-	X	-

Translator Statements

Translator statements appear as condition or action stubs to specify that the translator is to take some action relevant to control of the execution flow of the processing of the decision tables. Each translator statement encountered at translate time is interpreted and appropriate entries made in the directory vector to implement the activity specified. Translator statements may provide cross-references to other stubs in the DECTAL block or may cause invocation of another or the same table in the block. Note that if the user wishes to take advantage of the control mechanism's capability of eliminating replicate coding of activity stubs, he must provide cross-references <u>via</u> translator statements, as the translator does not seek to identify identical stubs for itself.

The translator statements supported by DECTALB are:

Axx	*TxxAxx	Axx	*STOP
Cxx	*TxxCxx	Axx	*REGOTO Txx
Axx	*GOTO Txx	Axx	*RECALL Txx
Axx	*CALL Txx	Axx	*REGOAGAIN
Axx	*GOAGAIN		

(The *RETURN function is available in the bootstrap only implicitly as the result of completing the action list of a rule which contains no explicit transfer of control.)

As is self-evident, every translator statement begins with an asterisk. The statements Txx(A|C)xx are cross references. They indicate that the task activity desired here is that specified in the table and task activity statement cross referenced. Cross references must be made to actual task activity, not to other cross references or to other translator statements. Cross references may be either forward or backward in the decision table block. All translator statements must appear on a single card without continuation cards.

*CALL is used to invoke a table with planting of a return link; it is used when return to the next action in the current rule is anticipated. *GOTO is used to invoke a table without planting a return link; when it is used there will be no way to return to the current rule after the invcked table has completed its execution. RE is prefixed if the table is to be entered at the point of its first condition testing (the implication being that the table was previously entered at its initializing actions and that their repetition now is inappropriate). AGAIN is added as a suffix to specify a new invocation of the table in which the translator statement appears. *STOP stops execution of the DECTAL

block and yields control to the statement following the *END delimiter of the DECTAL block.

In translator statements, the * may appear in column 4 or in any succeeding column. Once started, the statements must appear exactly as written, with no extra and no few blanks after the *.

The space between a translator statement and any entries may be employed for comments; it will not be inspected by the translator. It is convenient to use this space for documentation. A cross reference or an invocation of another table may be explained in context by indicating briefly what action that referenced task activity or table implements.

Debugging with DECTALB

Since DECTAL transmits the strings which constituted the user's stubs to the compiler, any PL/I syntax errors in the stubs will be the cause of diagnostics issued by the compiler. Since DECTAL has modified the program organization, some method of linking compiler diagnostic messages back through the DECTAL output to the decision table input is desirable to assist in location and correction of errors at this level. The method provided for this utilizes PL/I comments placed right after the statement label which labels the code transmitted from each decision table stub. The contents of the comment are the Txx (A|C)xx encodement employed in translator statement cross references. Thus, if the compiler gives a diagnostic in statement m, and the comment last appearing before statement m in the DECTAL

output is /*T45C06*/, then the correction must be entered in condition 6 of table 45. The relatively compressed format of DECTAL's output to the compiler should not be permitted to be a detriment to the use of the system: the discovery and correction of errors should all be done at the decision table level (as accessed, when required, via the /*Txx(A|C)xx*/ comments as just explained) and little or no time should be devoted to inspecting the organization of the coding which the compiler is presented with. Since DECTAL's printed output is a carefully formatted version of the inputted decision tables, debugging at the decision table level is greatly facilitated.

The diagnostic capabilities invested in DECTALB are extremely limited. Many diagnostic capabilities had been visualized at the time the bootstrap was being written and comments indicating their nature and/or mechanism appear from time to time in the bootstrap source coding. An ON ERROR unit in the bootstrap will trap up to ten error situations detected at the hardware level which follow from input errors and will cause the inputting to be resumed at the next input card. This device, while not ideally convenient, will serve to locate errors on a particular input card and to permit more than one error to be found in a single DECTALB run.

DECTALB does not accept control options at any level. It assumes that all tables are CLOSED and supplies an implicit *RETURN translator statement to any rule which does not close with

an explicit transfer of control translator statement. All tables are regarded as not-ORDERED by DECTALB. Hence, whatever applicable rule first emerges as a pivot rule will be selected for execution by DECTALB even if another applicable rule appears to its left in the decision table.

Interactions of DECTAL with the PL/I Compiler; Efficiency Considerations

The DECTAL control mechanism employs a number of variables for its purposes. Each of these variables starts with the symbol '#'. The DECTAL user can therefore safely avoid inadvertant double use of variable names by simply avoiding the use of variable names beginning with this symbol. If a user has employed variables beginning with this symbol and has explicitly declared each one, then the compiler will produce a diagnostic to the effect that a variable has been multiply declared, and the user will be informed of the interference between the DECTAL control mechanism and the task activity.

The DECTAL block is the unit within which cross references and table to table transfer of control may be made. The DECTAL control mechanism is therefore created afresh for every DECTAL block entry. The fact that the DECTAL control mechanism has a number of variables declared for its use implies that only one DECTAL block may appear in a given PL/I BEGIN or PROCEDURE block. If more than one appears, replicate declaration of control variables will lead to serious difficulty. Subject to this restriction, DECTAL blocks may be used multiply in a given program.

A certain amount of overhead is derived from the initialization of the values of the label vector which DECTAL employs to control the flow of execution. We attempted to minimize this overhead by employing a static variable for the label vector and a switch which simply indicates whether the label vector has been initialized, in order to avoid re-initialization whenever possible. Nevertheless, it seems clear that DECTAL block entry is a relatively high overhead operation so that the user, if he can conveniently do so, would be well advised to structure a program so that DECTAL blocks are entered infrequently relative to the duration of the entire program.

As a further means of reducing overhead, blocks of target language statements which are always executed together should be grouped together in a single DECTAL statement. Such a unit requires only one label subscript and is referenced by only a single element in the action lists of rules which invoke it. If fragmented, multiple label subscripts and multiple elements in action lists would be required, at a correspondingly greater overhead.

Since utilization of the cross referencing of a stubs in one table from another table is anticipated to be the principal source of reduction of object module size by use of DECTAL, and since includion of all related tables into a single DECTAL block will raise the probability that cross references can be usefully employed, and since, as mentioned above, initialization overhead will be reduced by entering DECTAL blocks infrequently, there are strong

pressures for building large complex blocks. One feels almost apologetic for this in these days of structured programming, but we argue that the extra power and clarity of the decision table mode of programming, and the diagnostic capability which the full version of DECTAL will implement will more than compensate for losses due to the pressures toward building large DECTAL blocks.

When compiling DECTAL output of large decision table blocks, it may be necessary to employ the External Dictionary PL/I compiler option in order to achieve successful compilation of the DECTAL output, and to do the compilation in a large partition to ensure use by the compiler of a sufficiently large text block.

The PL/I level F error message IEM1796I "ASSIGNMENT OF AN ILLEGAL LABEL CONSTANT IN STATEMENT NUMBER xxxx" appears as many times as there are elements in the label vector constructed by DECTAL for execution time control. This message may be safely ignored; correct coding is compiled in spite of its appearance. The condition code is returned with a value of 8 as a result of this diagnostic; subsequent steps in the PL1LFCLG procedure must be permitted to run in spite of this diagnostic. The appearance of this diagnostic was deliberately chosen as the least of various evils. In order to make every 'GO TO # LAB (---);' in the execution control mechanism be executed by in-line coding, the compiler had to be assured that the values assigned to the label vector elements would be known in the current active block. This assurance was given by deceiving the compiler by giving it a

restricted list of local statement label constants in the declare statement for the label vector. (The alternative of giving it a full list of local statement constants is precluded by a list-length limitation imposed by the compiler.) The compiler recognizes that it has been given a value for assignment to a label element different from those in the declare statement and issues the diagnostic. The alternative of initializing the statement label vector directly with LAB(i): prefixed appeared to generate less efficient coding than the alternative chosen.

The PL/I Optimizing Compiler does not generate this error message when it compiles the same DECTAL output.

JOB CONTROL LANGUAGE

Since DECTAL is a preprocessor for PL/I coding (it accepts input in a form which is not acceptable PL/I and generates acceptable PL/I source language statements from it), the usual mode of employment of DECTAL will be one in which its PUNCH output is passed directly to SYSIN of the PL/I compiler. The input stream for the first sample job illustrates this usage.

Input Limitations

DECTAL blocks processed by the bootstrap cannot be so large that either the number of directory vector elements or the number of label vector elements becomes larger than 999. The source of the restriction is the three digit format in the DECTAL PUNCH file which serves as PL/I compiler input and the restriction is reinforced by the initial value of MNADS in statement 11. If the

number of entries in the translate time dictionary will exceed 300, the initial value for MNDICL in statement 11 must be increased.

As submitted, the bootstrap will accept up to 21 rules in a table, and up to 50 stubs in a table. These limits may be expanded by altering the initial values given to MNR (maximum number of rules) and MNS (maximum number of stubs) in statement 9. Changes in these initial values will change the amount of core which DECTALB will require to run. The format of the printed output line makes 42 the practical upper limit for the number of rules in a table.

Resources required for DECTALB

The submitted bootstrap consists of 803 PL/I source statements. It compiled in 0.050 hours on an S360/50 in a 128K partition, using version 5.4 of the F-Level PL/I compiler.

In typical short test runs, DECTALB ran in 68K of core. It transformed decision tables in 1/2 - 1/3 of the time it takes PL/I-Level F to compile the transformed output.

Standard configurations supporting PL/I should support DECTALB without difficulty. A card reader, card punch, line printer, and direct access auxiliary storage suffice to support the system as we have used it.

ACKNOWLEDGEMENTS

The authors are indebted to Dr. Tom L. Gallagher and Mrs. Elizabeth A. Unger for calling their attention to this problem,

and to the Kansas State University computing center for support during its prosecution.

```
//DECTAL EXEC PGM=DECTALD
//STEPLIB  DD  DSN=COWA01.DECTAL,DISP=SHR
//PRINT    DD  SYSOUT=A
//PUNCH DD UNIT=SYSDA,DSN=&&DECOUT,DISP=(NEW,PASS),SPACE=(TRK,(1,2)),  X
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=480)
//SYSPRINT DD  SYSOUT=A
//ERRFILE  DD  SYSOUT=A
//SYSIN    DD  *
 DECTEST:PROC OPTIONS(MAIN);
         DCL CS256 BASED(PT) CHAR(256);
         DCL (BIRTHS,KEEP)(0:255) CHAR(1);
         DCL (HIGHC,LOWC)CHAR(1);
         DCL NAME CHAR(12) INIT('BIRTHS DS 0C');
*DECTAL
TOL
# THIS PRODUCES THE TRANSLATE TABLES FOR THE GAME OF LIFE, SEE
# SCIENTIFIC AMERICAN, OCTOBER, 1971 ET SEQ.
#THE TRANSLATE TABLES WOULD BE USED IN A BAL IMPLEMENTATION OF THE GAME
# BRIEFLY, CELLS SURROUNDED BY 3 OCCUPIED CELLS (COLLECTED AS THREE BITS
# IN A BYTE REPRESENTING THE NEIGHBORHOOD SITUATION) GIVE BIRTH IN THE
# NEXT GENERATION.
#CELLS SURROUNDED BY 3|4 NEIGHBORS KEEP ANY OCCUPANT CURRENTLY PRESENT.
# ALL OTHER CELLS ARE EMPTY IN THE NEXT GENERATION.
A00 UNSPEC(HIGHC)='11111111'B; UNSPEC(LOWC)='00000000'B;
 K,I,N,NP=0;
R 01 02 03 04 05
C00 IF I=8::TT-FT
C01 IF K=3::TF---
C02 IF K=4::FT---
C03 IF K>4::-FTFF
C04 IF K<3::FF--T
A01 BIRTHS(N)=HIGHC:::X----
A02 KEEP(N)=HIGHC:::XX---
A03 BIRTHS(N),KEEP(N)=LOWC:::--X-X
A07 BIRTHS(N)=LOWC:::-X---
A04 N,NP=N+1; IF N=256 THEN GO TO SPT; K,I=0;::XXX-X
A05 NP=NP+NP; I=I+1; IF NP>255 THEN DO; K=K+1; NP=NP-256; END; :: ---X-
A06 *REGOAGAIN :: XXXXX
*END
 SPT:     PT=ADDR(BIRTHS(0));
          OPEN FILE(SYSPNCH) STREAM OUTPUT TITLE('SYSPUNCH');
 SPUT:    PUT FILE(SYSPNCH) EDIT (NAME)(A(80));
          PUT FILE(SYSPNCH) EDIT ((' DC C''',SUBSTR(CS256,I,64),''' DC
 I=1 TO 200 BY 64)) (A(6),A(64),A(10));
              IF SUBSTR(NAME,1,1)='B' THEN
                 DO;
                 PT=ADDR(KEEP(0));
                 NAME='KEEP DS 0C';
                 GO TO SPUT;
                 END;
          END DECTEST;
//CLD       EXEC PL1LFCLD,TIME=(,10),COND=(9,LT),                      X
//              PARM.PL1L='A,X,NST,SKS,SIZE=124K',                     X
//              PARM.GO='SIZE=50000'
//PL1L.SYSIN DD DSN=*.DECTAL.PUNCH,DISP=(OLD,DELETE)
//GO.ERRFILE DD SYSOUT=A
/*
```

Input stream for sample problem (listing of File 4 of submittal tape.)

```
T01
 ID   XREF                 STUB                                                                   ENTRIE
          # THIS PRODUCES THE TRANSLATE TABLES FOR THE GAME OF LIFE, SEE
          # SCIENTIFIC AMERICAN, OCTOBER, 1971 ET SEQ.
          #THE TRANSLATE TABLES WOULD BE USED IN A BAL IMPLEMENTATION OF THE GAME
          # BRIEFLY, CELLS SURROUNDED BY 3 OCCUPIED CELLS (COLLECTED AS THREE BITS
          # IN A BYTE REPRESENTING THE NEIGHBORHOOD SITUATION) GIVE BIRTH IN THE
          # NEXT GENERATION.
          #CELLS SURROUNDED BY 3|4 NEIGHBORS KEEP ANY OCCUPANT CURRENTLY PRESENT.
          # ALL OTHER CELLS ARE EMPTY IN THE NEXT GENERATION.
A00            UNSPEC(HIGHC)='11111111'B; UNSPEC(LOWC)='00000000'B;
           K,I,N,NP=0;
                                                                               01 02 03 04 05
C00              IF I=8                                                    ||  T  T  -  F  T
C01              IF K=3                                                    ||  T  F  -  -  -
C02              IF K=4                                                    ||  F  T  -  -  -
C03              IF K>4                                                    ||  -  F  T  F  F
C04              IF K<3                                                    ||  F  F  -  -  T
           ===========================================================================================
A01              BIRTHS(N)=HIGHC;                                          ||  X  -  -  -  -
A02              KEEP(N)=HIGHC;                                            ||  X  X  -  -  -
A03              BIRTHS(N),KEEP(N)=LOWC;                                   ||  -  -  X  -  X
A07              BIRTHS(N)=LOWC;                                           ||  -  X  -  -  -
A04              N,NP=N+1; IF N=256 THEN GO TO SPT; K,I=0;                 ||  X  X  X  -  X
A05              NP=NP+NP; I=I+1; IF NP>255 THEN DO; K=K+1; NP=NP-256; END; || -  -  -  X  -
A06              *REGOAGAIN                                                ||  X  X  X  X  X
```

Printed DECTALB output from sample problem.

```
DECTEST:PROC OPTIONS(MAIN);
        DCL CS256 BASED(PT) CHAR(256);
        DCL (BIRTHS,KEEP)(0:255) CHAR(1);
        DCL (HIGHC,LOWC)CHAR(1);
        DCL NAME CHAR(12) INIT('BIRTHS DS 0C');
        GO TO #START;
#001:/*T01A00*/   UNSPEC(HIGHC)='11111111'B; UNSPEC(LOWC)='00000000'B;          1
K,I,N,NP=0;GOTO #NEXT;#002:/*T01C00*/   IF I=8   THEN GOTO #TRUE;GOTO #        2
FALSE;#003:/*T01C01*/   IF K=3   THEN GOTO #TRUE;GOTO #FALSE;#004:/*T01         3
C02*/   IF K=4   THEN GOTO #TRUE;GOTO #FALSE;#005:/*T01C03*/   IF K>4           4
 THEN GOTO #TRUE;GOTO #FALSE;#006:/*T01C04*/   IF K<3   THEN GOTO #TRUE         5
;GOTO #FALSE;#007:/*T01A01*/   BIRTHS(N)=HIGHC;  GOTO #NEXT;#008:/*T01A          6
02*/   KEEP(N)=HIGHC;    GOTO #NEXT;#009:/*T01A03*/   BIRTHS(N),KEEP(N)       7
=LOWC;GOTO #NEXT;#010:/*T01A07*/   BIRTHS(N)=LOWC;   GOTO #NEXT;#011:/*         8
T01A04*/   N,NP=N+1; IF N=256 THEN GO TO SPT; K,I=0;  GOTO #NEXT;#012:/         9
*T01A05*/   NP=NP+NP; I=I+1; IF NP>255 THEN DO; K=K+1; NP=NP-256; END;G        10
OTO #NEXT;                                                                     11
#013:                                                                          12
#GOTO:  #CPT=#DIR(#DIR(#CPT))+1;                                               13
        GO TO #LAB(#DIR(#CPT-1));                                              14
#NEXT:  #CPT=#CPT+1;                                                           15
        GO TO #LAB(#DIR(#CPT-1));                                              16
#FALSE: #CPT=#CPT+2;                                                           17
#TRUE:  #NXT=#DIR(#CPT);                                                       18
        #CPT=#DIR(#CPT+1);                                                     19
        GO TO #LAB(#NXT);                                                      20
#START: IF #SWT='1' THEN DO; #SWT='0';                                         21
        #LAB(  1)=#001; #LAB(  2)=#002; #LAB(  3)=#003; #LAB(  4)=#004;        22
        #LAB(  5)=#005; #LAB(  6)=#006; #LAB(  7)=#007; #LAB(  8)=#008;        23
        #LAB(  9)=#009; #LAB( 10)=#010; #LAB( 11)=#011; #LAB( 12)=#012;        24
        #LAB( 13)=#013; #LAB( 14)=#014; END;                                   25
        #CPT=1; GO TO #GOTO;                                                   26
        DCL #SWT CHAR(1) STATIC INIT('1'), #LAB( 14)STATIC LABEL(              27
        #START,#NEXT,#FALSE,#TRUE);                                            28
        DCL (#CPT,#NXT) BIN FIXED STATIC,#LNK CTL BIN FIXED,                   29
        #DIR( 49) BIN FIXED STATIC INIT(                                       30
          2, 20, 21,  8, 11, 13,  3, 10, 11, 13,  3, 11, 13,  3, 13,          31
          3, 11, 13,  3,  1,  2,  5, 26,  5, 46,  9, 12,  6, 30,  9,          32
         17,  3, 34,  4, 38,  4, 42,  0,796,  7,  4,  8,  8,  0,796,          33
          9, 12, 12, 15);                                                      34
#014:   ;                                                                      35
SPT:    PT=ADDR(BIRTHS(0));
        OPEN FILE(SYSPNCH) STREAM OUTPUT TITLE('SYSPUNCH');
SPUT:   PUT FILE(SYSPNCH) EDIT (NAME)(A(80));
        PUT FILE(SYSPNCH) EDIT ((' DC C''',SUBSTR(CS256,I,64),'''' DO
I=1 TO 200 BY 64)) (A(6),A(64),A(10));
            IF SUBSTR(NAME,1,1)='B' THEN
               DO;
               PT=ADDR(KEEP(0));
               NAME='KEEP DS 0C';
               GO TO SPUT;
               END;
        END DECTEST;
```

DECTALB output from sample problem; serves as PL/I source module.

Magnetic Tape Key

Volume prepared using IEBUPDTE
[redacted] TRACK & DENSITY AS ORDERED.
VOL=SER=DECTAL
DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)

File 1 JCL cards invoking PL1LFCL for compiling DECTALB
DSN=JCL1F EBCDIC
Sequence 00000010 and 00000020 in cc 73-80

File 2 PL/I source of DECTALB
DSN=DECTALB EBCDIC
Sequence 00000100 through 0001101C in cc 73-80

File 3 LKED.SYSLMOD card establishing DECTALD load module
DSN=JCL1R EBCDIC
Sequence 00019900 in cc 73-80

File 4 Example deck, including all JCL to invoke DECTALD, process the sample deck, pass the output to PL/I, and execute the compiled program.
DSN=JCL2F EBCDIC
Sequence 00020000 through 00020550 in cc 73-80

A Decision Table Algorithm Based on List-Processing Techniques

Ronald G. Smith*
Kenneth Conrow**
Kansas State University
Manhattan, Kansas 66506

ABSTRACT

A control mechanism utilizing list processing techniques has been devised to facilitate the execution of decision tables. A directory vector controls the flow of execution among condition and action stubs, from one table to another in a block of tables, and through whatever diagnostic modules might be required in a given execution. The choice of stub for execution via the directory vector implies that any stub need be coded only once in the block of decision tables, with all other uses of that stub achieved via reference to the single encodement. This feature enables dramatic reduction in the size of an object module in favorable cases. Translator commands effecting transfer of control among the several tables of a block of decision tables are implemented within the directory vector, and invite more freedom in linking tables together than has been customary. Introduction of a trace feature and various levels of diagnostic assistance are likewise readily accommodated.

KEY WORDS AND PHRASES: decision tables, diagnostic aids, list-processing application, system analysis

CR Categories: 3.50, 4.19, 4.29, 4.49

---

*Computing Center
**Department of Computer Science, to whom inquiries should be addressed.

I. DEFINITIONS AND ORIENTATION

Decision tables provide a clear and compact medium in which to analyze situations according to prescribed complex logical criteria and to express the actions which result from a given decision. We are concerned here with the description of a translator and an execution time control mechanism which will permit coding expressed in decision table format to be compiled by a standard compiler and eventually executed to achieve the programmer's ends.

We will assume that the reader is familiar with the basic vocabulary employed in discussion of decision tables [1,2] so that he can, with the aid of Figure 1, tolerate omission of a review of the basic vocabulary. We will give here a few orienting definitions which go beyond the usual decision table definitions.

The basic control mechanisms utilized by this system could be implemented in any of a variety of computer languages. The language in which the stubs are expressed and whose compiler will serve to convert the translator output into an object module will be called the target language. Our target language is PL/I.

Sections of coding in the decision table mode suitable for processing by the translator will be referred to as decision table blocks. Decision table blocks may be intermixed with normal target language coding and have the same properties as PL/I BEGIN blocks in regard to execution flow. Each block consists of a series of tables, and each table consists of a series of statements. Aspects of the translator's activity may be controlled at the block level,

the table level, or at the individual statement level, so provision is made for control options to be specified at each of these three levels of control heirarchy (see Figure 1). Each table and each action or condition statement is uniquely numbered in its environment with a two digit decimal number denoted as 'xx'.

As the result of the translation process, each table is converted into a binary decision tree and a set of action lists. The binary decision tree is an implementation of the condition entries in the table and its traversal at execution time will result in the choice of some one action list corresponding to the rule selected when a leaf node of the decision tree has been encountered.

A series of action stubs which occur prior to any condition stubs in a table will comprise its initializing actions. Two entry points are defined by the translator for every table. The first entry point is at the first statement after the Txx card; the second entry point is at the first Cxx statement. In the case of a table without initializing actions, the two entry points will coincide.

II. OBJECTIVES AND PROSPECTUS

The principal design objective was to create a decision table translator which would permit the target language programmer to employ decision tables whenever he felt it convenient. The translator was to provide as much flexibility as possible and as much diagnostic assistance [3] as possible. Naturally, we also desired to minimize repetitious coding of activity, and keep the translator

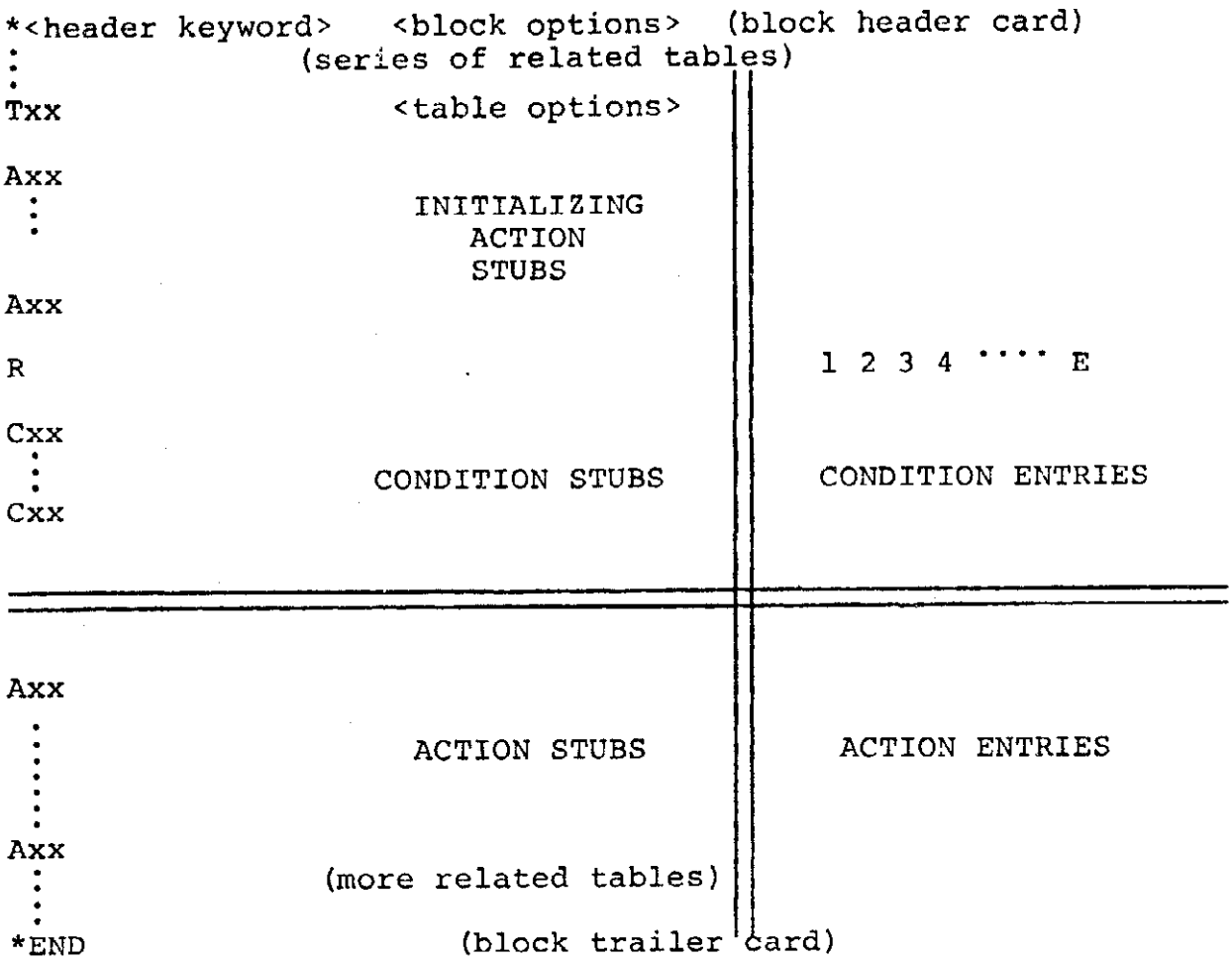

Figure 1. Decision Table Format for the Decision Table Translator

as small and efficient as possible.

In brief, decision table blocks can be inserted in a PL/I source stream at arbitrary locations, and large PL/I source code segments can constitute action or condition statements within a table. The content of the stubs presented in tables to the translator are arbitrary within certain bounds. Loops and testing of lowest-level conditions within Axx statements in a table can be coded directly in the target language. Similarly, actions can be coded in a Cxx statement before 'IF ⟨expression⟩'. These options should provide enough flexibility for most user's needs. Note, however, that decision table blocks cannot be nested, and that the user cannot employ PL/I execution-flow-controlling statements to transfer control between decision table statements because such action would confuse the superposed decision table execution flow control mechanism.

The basic design approach taken was to employ list processing techniques both at execution time and at translation time. These techniques enable achievement of the above goals and invite innovation in providing new diagnostic aids. It could be considered that the list processing techniques employed here are extensions of King's interrupted rule mask techniques[4] among the condition entries and of Coulter's cross-linking of duplicate action strings.[5]

At execution time, a directory vector controls the flow of execution among condition stubs and action stubs. The majority of directory entries are half word binary numbers which serve as subscripts for a vector of labels which uniquely identify every

different stub in the set of input tables. Most of the remaining directory entries are half word binary numbers which are subscripts of entries in the directory vector itself and which permit transfer of control within the directory. As a consequence, every choice of a condition to be tested and every choice of an action to be taken can be made completely independently of every other occasion on which that condition or action might be invoked. Each different stub need therefore be coded exactly once in the object module, thus providing the desired minimization of object coding.

The use of the equivalent of subscripted label vectors in other languages has been suggested previously in decision table processing,[6] though not in connection with a list processing approach.

The advantage of list processing control via the directory vector in enabling incorporation of diagnostic aids is manifested in the diverse cases of a TRACE feature, a variety of possible responses to redundancies and apparent and real ambiguities, and a response to missing ELSE rules.

At translation time, the list processing techniques enable a number of economies and diagnostics. One can visualize the process of deciding among the rules of a decision table as equivalent to the process of tracing a path through a binary decision tree. The truth of some one condition stub is tested to divide the possibilities into two groups. A second condition stub's truth then further subdivides the sub-tree. The repetitive testing of the truth of stubs eventually selects for execution some one rule (in

a table with all rules given and no ambiguities). The decision tree is constructed efficiently at translation time by maintaining the table rules in either a true branch list or a false branch list stack. When abnormalities arise in the contents of a true or false branch list, diagnostics can be issued.

Cross references from one stub to another are quickly handled by binary search of a dictionary list and the resolutions entered into the directory vector; failures of resolution of cross references are easily discovered for diagnostic purposes in a final scan of the dictionary list. Provision of two standard entry points to a table[7] and specification of the starting table in a decision table block are also easily accommodated because of the list processing scheme.

III. THE EXECUTION TIME CONTROL MECHANISM

The flow of control at execution time utilizes a directory vector (DIR) and a control pointer (CPT). The convention is adopted (just as with the program counter in a conventional computer) that CPT always points to the next DIR entry which is to be used. Most entries in the directory vector are either a label subscript or a control pointer value. The label subscripts permit access of activities (either condition testing or execution of an action) which appear at a labeled location designated by the value of the subscript. The CPT entries in DIR permit control to be transferred from place to place within the directory list.

Smooth transition among various kinds of control activities and task activities can be achieved by careful coordination of the directory list elements and of the control section statements. The key feature is the application of a subscripted label to certain

control sections, so that control activity can be initiated by exactly the same mechanism that initiates task activity.

Although additional control sections will be required to support the trace and execution time diagnostic features and some of the basic control sections will need elaboration to provide diagnostic support, the essential simplicity of the execution time control mechanism will be evident.

Five basic control sections exist. The first three effect transition between control activities and are accessed via subscripted labels so that these control activities appear in the directory list exactly like task activities. The other two control sections effect transition between task activities and require no subscripted labels. The five control sections:

1) call a closed table
2) branch to a table
3) return from a closed table
4) respond to completion of a task action, and
5) respond to completion of a task condition test.

Figure 2 should be referenced as these control sections are described as it provides a diagrammatic representation of the context in which the various control sections operate. Although the figure shows a complete table, each control section is operative in only those restricted parts of it where its ministrations are relevant.

Sequential execution of a series of actions is conceptually the simplest. This employs control section 4. The action list

appears sequentially in DIR, so all that is required is augmentation of CPT after each action is completed. After each action is placed: 'GO TO NEXT;', which serves to maintain the action list execution mode. The control section reads:

```
NEXT:  CPT=CPT+1;
       GO TO LAB(DIR(CPT-1));
```

The last action in an action list in an open table is a branch to another table. This involves control section 2. Thus, the DIR entry immediately after the subscript of the last task action is a '2', and the DIR entry after that is the indirect address in DIR of the required table. Control section 4 invokes control section 2 just as it invokes any other action, but control is not returned to it this time. Control section 2 updates CPT to the new situation, and reads simply as follows:

```
LAB(2):  CPT=DIR(DIR(CPT))+1;
         GO TO LAB(DIR(CPT-1));
```

Similarly, if any action in an action list is a call of a closed table, the contents of DIR(CPT-1) after control section 4 has been executed is a '1' and the contents of DIR(CPT) is, as before, the indirect address of the required table. Control section 1 need only make a record of the linkage required for return before continuing with the activity of control section 2 which effects the actual branch. It reads as follows:

```
LAB(1):  {NOL=NOL+1;          }  |  {ALLOCATE LNK;}
         {LNK(NOL)=CPT+1;     }     {LNK=CPT+1;    }
```

(Control section 2 immediately follows)

(In the second alternative LNK is a CONTROLLED variable; successive allocations are stacked automatically by PL/I.)

Return from a closed table is accomplished by inserting at the end of every action list of a closed table a '3'. When control section 3 thereby assumes control, it recovers the link and removes it from the link stack. Since closed tables can only be called from action lists, it is known that continuation in action chain execution mode is appropriate, so control section 3 is followed immediately by control section 4. Control section 3 reads as follows:

```
LAB(3):  {CPT=LNK(NOL);}  |  {CPT=LNK;  }
         {NOL=NOL-1;   }     {FREE LNK;}
```

(Control section 4 immediately follows)

The binary tree which must be searched during task condition testing is included in Figure 2. Every condition test result can be either true or false, and so every one must permit control to branch in either of two ways according to the result of the condition test. This is accomplished in a series of four DIR entries. The left pair of entries will be employed if the preceding test was true, and the right pair will be employed if the preceding test was false. The first entry in each pair is an activity subscript which permits access of the next condition or of the first action of an action list if a leaf node has been reached; the second entry in each pair is a CPT value which points to the next set of four DIR entries if testing must continue or to the address in DIR of the subscript of the second action in the

action list if the first entry was an action. Control section 5 monitors this activity and reads as follows:

```
FALSE:  CPT=CPT+2;
TRUE:   NXT=DIR(CPT);
        CPT=DIR(CPT+1);
        GO TO LAB(NXT);
```

At each subscripted label which contains a condition test, the coding reads as follows:

'IF ⟨expression⟩ THEN GO TO TRUE; GO TO FALSE;',

where the 'IF ⟨expression⟩' was taken directly from the user's condition stub. When return is made to TRUE, the control pointer is already pointing to the correct doublet for execution of the next condition test, so all that occurs is setting up of the control pointer to the next condition quartet and branching to the point where the test will be made. When return is made to FALSE, CPT is pointing to the doublet for the TRUE condition and this must be corrected by stepping it along two positions in the DIR vector. After this is done, the same actions suffice to prepare for the next test.

The transition between task actions and task conditions is normally effected in both directions by letting the first item of the new kind be addressed as if it were another item of the old kind. (An exception will arise when an execution time message is to appear.) Two examples appear in Figure 2. 1) The first action of an action list appropriate to the rule selected appears in the same position in a quartet as would still another condition to be tested. 2) After a series of initiating actions for a table, the last address is simply that of the first condition to be tested.

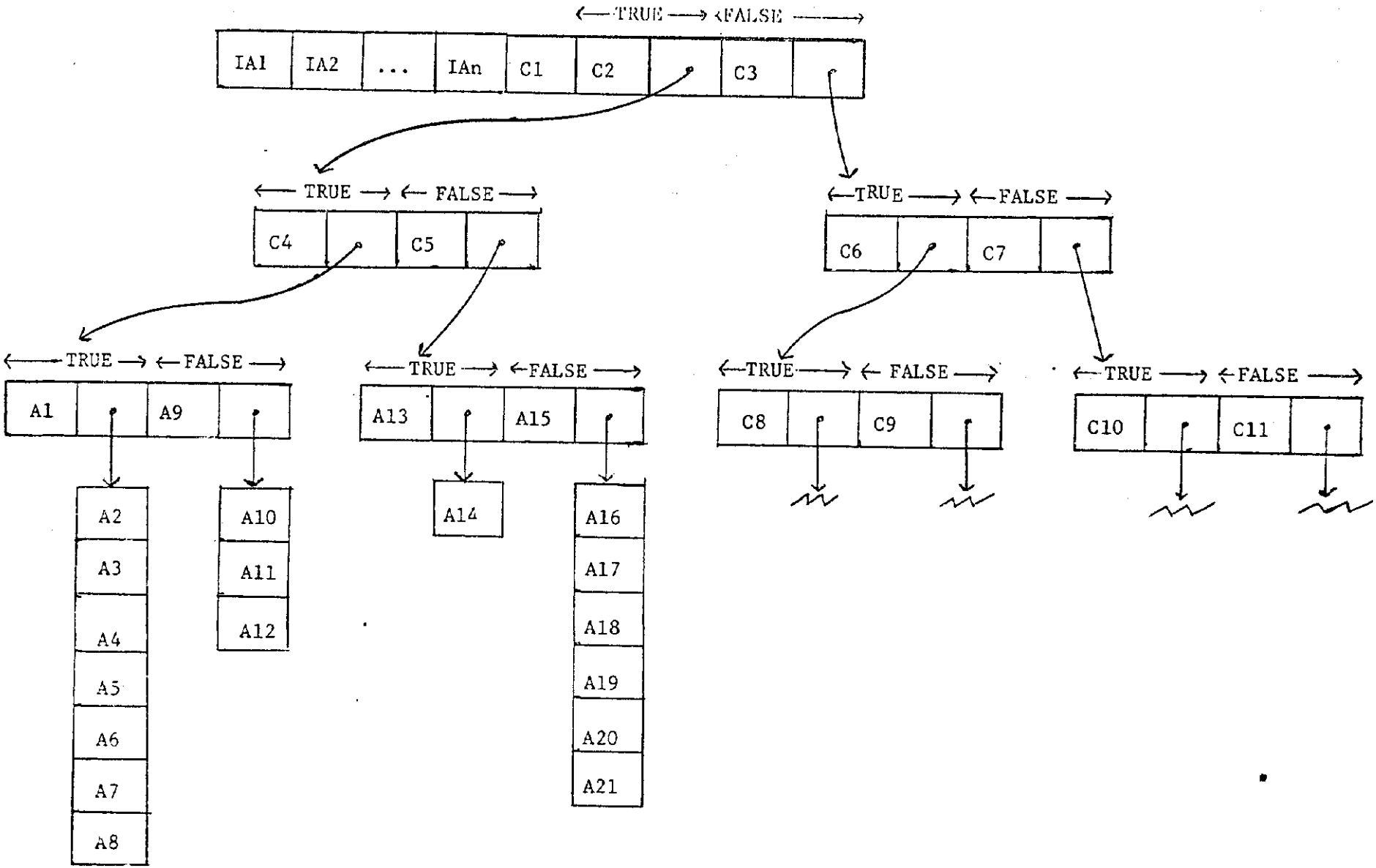

Figure 2. Diagrammatic Representation of the Decision Table Execution Time Control

It is the 'GO TO' statements after every condition and every action which actually effect the transition in mode of operation by returning control to control section 4 or 5 as appropriate.

Normal control activity may be recapitulated. In the action list mode, single entries in DIR are accessed sequentially and the mode is maintained by 'GO TO NEXT;' commands at the end of each action. In the decision tree mode, quartets in DIR are accessed, but only the TRUE doublet or the FALSE doublet is employed, as dictated by the result of the previous test. The decision tree mode is maintained by the 'GO TO TRUE;' and 'GO TO FALSE;' commands after each decision test. Mode change between these modes is effected by encounter of a label subscript labelling the complementary activity, and the 'GO TO' at the end of the activity effects the actual mode change. Control activity having to do with new table access has been arranged to mimic task control activity by providing these control sections with subscripted labels and accessing them in a manner identical to the access of task activity.

The complete normal activity control section follows:

```
CALL:       LAB(1):   NOL=NOL+1;
                      LNK(NOL)=CPT+1;
GOTO:       LAB(2):   CPT=DIR(DIR(CPT))+1;
                      GO TO LAB(DIR(CPT-1));
RETURN:     LAB(3):   CPT=LNK(NOL);
                      NOL=NOL-1;
            NEXT:     CPT=CPT+1;
                      GO TO LAB(DIR(CPT-1));
            FALSE:    CPT=CPT+2;
            TRUE:     NXT=DIR(CPT);
                      CPT=DIR(CPT+1);
                      GO TO LAB(NXT);
```

The extreme simplicity of the coding generated by PL/I from this source coding reinforces the impression that the control overhead at execution time for this list-processing approach to decision table control will be negligible both in point of view of time and space consumed. The PL/I label vector overhead is the most serious source of concern.

It is evident from the foregoing that a given control section may utilize an arbitrary number of DIR entries to accomplish its task. The number it uses are skipped over by proper resetting of the CPT value. This observation suggests that information other than label subscripts or directory vector subscripts could be encoded in the directory vector and skipped over by the invoked control section. It is this technique which will enable implementation of trace and execution time diagnostic features.

IV. TRANSLATION-TIME ACTIVITY: OVERVIEW

The translator consists of four main sections which

1) search for decision table blocks and interpret block options,
2) input decision tables,
3) create the required directory and dictionary entries, and
4) dump the execution time control blocks and directory

The first and fourth activities are performed once for each decision table block. The second and third activities are performed once for each table within the block.

Section 1) merely reproduces the input stream as output until a block header card is encountered. Any block options on this card are interpreted, and following cards are throughput to the output stream until a Txx card is encountered, at which time Section 2) is given control.

Section 2) processes the group of cards which contain one table before control is given to Section 3). A table is delimited by a T card at the beginning and either a new T card or an *END card at the end. This section reads the statement identifying symbols, and causes them to be entered in the dictionary or resolved if they are already in the dictionary. It assigns label subscripts to new statements. It prepares the entry table for Section 3) and will issue diagnostics for irregularities in the input table. Any extended entry statements will be recognized and converted into a series of limited entry statements. All printed and punched output produced specifically for this table is produced by this section. Key information about the statements of the table is prepared for Section 3).

Section 3) creates the decision tree for the table currently being processed. The image of the decision tree is produced in the directory vector with the proper control section references and directory entries to cause proper decisions to be made and the proper action lists to be performed at execution time. Upon completion of these operations, control is returned to Section 2) or Section 4) according as the next card is a T card or an *END card.

Section 4) outputs all the control information and control sections required by the decision table block whose translation is being completed. Only those control sections actually invoked in the block will be included in the produced coding. The dimensioning and the initialization of the directory and of the label vector are done at exactly the size required by the block at hand.

TABLE INPUT - BUILDING THE DICTIONARY

During the inputting of each decision table by Section 2 of the translator, a dictionary must be constructed or referenced which contains the identifiers of the tables and their conditions and actions, and the resolution of them to either a label subscript or a directory subscript. The problem has the following features: the number of entries overall and the number of entries per table is very variable; the entries may be encountered in an arbitrary order (since both forward and backward cross-references are permitted); at any time an identifier which is sought may already be present or may be new and hence need to be added to the dictionary.

One can meet these requirements with two parallel vectors:

1) the vector of coded identifiers, and

2) the vector of their resolution as subscripts.

These vectors are kept arranged in ascending order of the coded identifiers.

For economy of storage, each vector is a vector of half words. The identifier value is calculated from one or the other of the following formulas. If the identifier is that for a table, the value is 2^8Tno-1. (Tno stands for the value of the digits xx in the Txx identifier of the table statement. Cno and Ano have analogous meanings.) If the identifier is one for a condition or an action, the value is 2^8Tno+2^7 (if an action) + its Cno or Ano. The 20100 possible identifiers in a block are thus each given a unique value between -1 and 25571.

Parallel to the identifier vector is the resolution vector. Conditions and actions are resolved to label vector subscripts and table identifiers are resolved to directory subscripts. Every condition and every action will have a subscripted label at execution time which enables each one to be accessed. At that point indicated by the directory subscript for a table appear <u>two</u> label subscripts, the label subscript for the first initializing action and the label subscript for the first condition in the table.

The maintenance of the coded identifiers in order permits a binary search for entries in the identifier vector. If an identifier is present in the identifier vector, the required subscript is transferred. If an identifier is absent from the identifier

vector, the binary search will terminate at the position in the vectors at which the new identifier and subscript must be entered. In order to make room for the new entry, the entries already in the vectors from that point downwards must be moved down one. If users typically number their tables, and the conditions and actions within them, sequentially, and make only backwards references to actions and conditions, then their identifiers will occur in roughly ascending sequence and there should be relatively few occasions for moving large numbers of elements even in very large blocks.

ESTABLISHING THE DECISION TREE

The essence of the approach used to construct the decision tree in Section 3) of the translator is to segregate into two groups all the rules present at each decision node. (The else rule, if any, is kept to one side and ignored during decision tree construction.) Into the true branch list are put all those rules one of which will be obeyed if the condition is true, and into the false branch list are put all those rules one of which will be obeyed if the condition is false. When rules with a "don't-care" entry on the condition arise, they are put into both the true branch list and the false branch list. By successive subdivision of the two branch lists, a leaf node of the decision tree is eventually reached.

The complete decision tree is searched by taking all the possible branches until every leaf node is accessed. This is accomplished by maintaining the successive false branch lists in a false branch list stack. Whenever the processing of successive

true branches is concluded by encountering a leaf node, a false branch list is popped from the stack and processed in its turn. If the false branch list is not at a leaf node, it will generate both a true branch list and a false branch list and the latter will be pushed into the false branch list stack. When the false branch list stack is empty, then the decision tree has been completely searched.

In order to reduce the number of tests of conditions required to establish the applicable rule at execution time, some translation time is expended in dynamically reordering the remaining condition stubs for every decision sub-tree. At each node in the decision tree, that condition stub among those remaining in the active branch list which has the largest number of explicit symbols (T, Y,N, or F, not "don't-care") in the rules still present is chosen for processing next. The freedom of execution time access to condition stubs permits this dynamic reordering to be readily expressed in the directory vector without incurring any expense in repetitious coding of condition stubs. The presence of this alternative mechanism for eliminating replicate coding of decision stubs very much reduces the otherwise critical nature of the choice of order of testing of conditions.[8] Our choice merely ensures early emergence of pivot rules (see below), which is a major factor critical to minimizing execution time.

Naturally, if all conditions of a decision table have been tested when some decision node has been processed, we have arrived at two leaf nodes. If a branch list is empty, a leaf node consisting of the else rule is provided.

Certain cases arise which define leaf nodes prior to testing of all the conditions of the table or to obtaining an empty list. In order to discuss these cases economically, we will define a pivot rule as the leftmost rule in an active branch list in which all the conditions which have not yet been tested are "don't-care." The presence of a pivot rule in the active branch list is important to recognize for one reason or another, so a pivot rule is checked for at each repetition of the tree growing cycle.

In a table designated as ORDERED, a pivot rule should be recognized as soon as it appears in an active branch list because it will be obeyed in preference to rules to the right of it. Since all the rules to the right cannot possibly be applicable in this branch of the decision tree, these rules can be dropped from the active branch list before the next decision node is processed. The decision process in this branch of the tree is expedited by dropping the useless rules. For a table which is not labeled ORDERED, the pivot rule must be promptly recognized because it is the rule whose action list is executed. This choice is made to minimize execution time and translation time: the choice of the first rule whose applicability is established makes it unnecessary to continue testing conditions.

In brief, the condition for early recognition of leaf nodes depends upon the table options. If the table had been specified as ORDERED, then a leaf node is reached when the leftmost rule in the active branch list has become the pivot rule. If the table is

not ORDERED, then a leaf node is reached when there is a pivot rule in the active branch list.

The growth of the decision tree by stepwise division of rules into a true and a false branch list is convenient because of the ease with which diagnostic messages can be produced as a natural and relatively effortless consequence of the normal flow of processing of an error-free input table. Several distinct kinds of errors may need to be flagged at translation time.

If a branch list is empty at any point in the development of the decision tree, it implies that an else rule must be invoked. If no else rule is present, one will be supplied which generates a diagnostic message at execution time and causes an exit from the decision table block since there is no available action list to implement. As has been said, the insertion of a few entries into the directory vector will provide a mechanism by which this execution time diagnostic can be provided. If the missing else situation never arises, then the potential diagnostic is simply never expressed during execution. Translation time messages may be produced whenever a decision tree leaf node apparently requires a missing else rule.

DIAGNOSTIC ASSISTANCE FOR AMBIGUOUS RULES

All rules which coexist in a branch list at a leaf node have been termed ambiguous.[9] Those rules which, because of the logical relationships among the conditions defining them, cannot be simultaneously satisfied are only apparently ambiguous. The only

rules which are really ambiguous are those which can be simultaneously satisfied.[9] For this reason, the problem as to the appropriate diagnostic to produce in case of ambiguous rules is a subtle one which we have sought to answer at various levels in the hope that each user's preference will appear somewhere among the translator's options. The next few paragraphs will describe the range of responses we will make. They are included as illustrations of the design variability accommodated by the basic list processing control device.

The level of diagnostic response will be controlled by the block or table option 'DIAG=n', where n may have the values 1, 2, or 3. When DIAG is 1, no diagnostics are given at either compile time or execution time as a consequence of ambiguous rules. This option will be useful only after the contents of a table have been thoroughly checked, and the user is satisfied that the ambiguities in it are only apparent ones so that he need not have the translator point them out to him in further runs.

When DIAG is 2, diagnostics will be produced at translation time but no execution time diagnostics will be inserted in the directory when ambiguities are encountered. This option will be useful if the programmer wishes his ambiguities flagged so that he can check the cases against his understanding of the problem to see if corrections in the table are warranted. Use of this level of diagnostic assistance uses only compile time, and has no effect on either the size or efficiency of the execution time module.

When DIAG is 3, execution time diagnostics will be created and triplets will be inserted into the directory vector so that an execution time diagnostic will greet the actual occurrence of an ambiguous situation. Since apparent and real ambiguities can not be distinguished at compile time, while real ambiguities can express themselves at execution time, this level of diagnostic assistance is the only way real ambiguities can be discovered. Since, as the following example will demonstrate, provision of execution time diagnostic capabilities requires a further development of the decision tree in order to identify really ambiguous situations, there will be a degradation of performance as well as an increase in the size of the directory vector consequent to the choice of DIAG=3.

The diagnostic assistance at level 3 is useful to automate a check on the programmer's understanding of the logic of his problem. If a user, for example, has a pair of condition entries

T -

- F

in the expectation that the two conditions will never be simultaneously met, i.e., that the ambiguity is only apparent and not real,[9] then the occurrence of T/F at execution time should be flaggable as a check on the programmer's understanding of his problem. The effect of early selection of a leaf node under either an ORDERED or a not-ORDERED regimen will be to produce a two-node decision tree (Fig. 3a). Such a tree cannot detect the occurrence

of the ambiguous situation. Hence when DIAG=3 is specified, the decision tree is more fully developed (Fig. 3b) so that the occurrence of the ambiguous T/F situation can be detected and execution routed through a diagnostic triplet. An execution time diagnostic will be included only in those leaves of the tree which are ambiguous. The leaf nodes which have only a single rule are not ambiguous, and the action lists for these rules will be included in the directory without insertion of a diagnostic triplet. If the user's understanding is correct, the error message will never be expressed; only if his ambiguity is real will he be given a diagnostic at execution time.

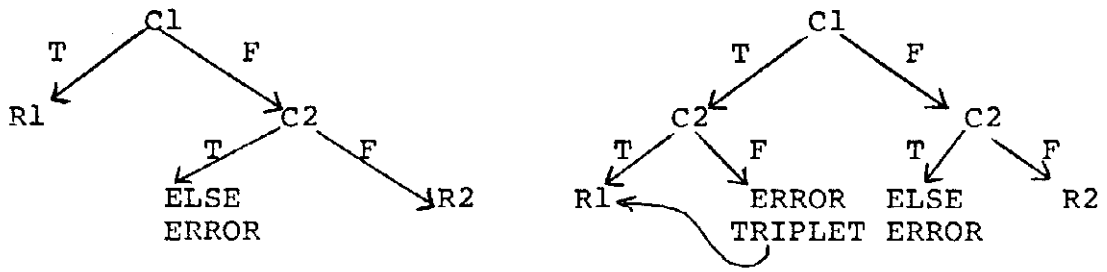

Figure 3. a. Decision tree produced from C_1 T-C_2-F when DIAG=1 or DIAG=2. b. Decision tree produced when DIAG=3.

The rule chosen for execution is selected according to the ORDERED or not-ORDERED option for the table; the level of diagnostic assistance will cause elaboration of the decision tree in such a way that the choice of rule to be executed will not change as diagnostic assistance is requested or refused.

TRANSLATOR STATEMENTS

Translator statements are statements which specify that the translator is to take some action relevant to control of the execu-

cution flow of the processing of the decision tables.[11] Each translator statement encountered at translate time is interpreted and appropriate entries made in the directory vector to implement the action specified. Translator statements may provide cross-references to stubs in another table in the block or may cause invocation of another or the same table in the block. Note that if the user wishes to take advantage of the control mechanism's capability of eliminating replicate coding of activity stubs, he must provide cross-references via translator statements, as the translator does not seek to identify identical stubs for itself.

The allowable translator statements are:

```
Axx *TxxAxx          Axx *STOP
Cxx *TxxCxx          Axx *RETURN
Axx *GOTO Txx        Axx *REGOTO Txx
Axx *CALL Txx        Axx *RECALL Txx
Axx *GOAGAIN         Axx *REGOAGAIN
```

Each translator statement is preceded by an asterisk. The interpretation associated with the card Axx *TxxAxx is that the action appropriate here is identical with the action in the table and action specified in the cross reference. Cross references must be to executable statements (or testable conditions), and not to cross references or other translator statements. *CALL may be employed to invoke a closed table. *RETURN will cause control to return to the action list after the most recent incompleted *CALL. *GOTO may be employed to invoke an open table.

RE is prefixed if the table is to be entered at the point of starting its condition tests. AGAIN is added as a suffix to specify a new invocation of the table in which the translator statement appears. *STOP causes control to be transferred to the target language coding immediately after the *END card delimiting the decision table block.

V. ADVANCED FEATURES

Condition stubs and action stubs may be arbitrarily complex, and may even include such peculiarities as precondition actions (target language assignments in a decision table Cxx statement) and preaction conditions (target language IF ⟨expression⟩ THEN ⟨statement⟩; ELSE ⟨statement⟩; in a decision table Axx statement). The principal limitations to use of such features is that they must be self-contained (i.e. make no transfers of control to statement labels outside themselves) and self-sufficient (i.e. they must neither refer to nor establish values of variables of other conditions and actions, since each is done independently in an order possibly unique to each rule). The ability to use quite complicated passages of target language coding as a single action in a decision table is a real advantage. The complicated action may be used elsewhere in the program by the usual cross referencing mechanism, with the consequent ease of invocation which usually applies only to procedures in PL/I.

The usual convention among users of decision tables is to consider that closed tables may only be called by and may, in turn, only call other closed tables. If the action list of a rule is

completed in a closed table, then return is automatically made to the invoking rule.

This convention is followed by this system, and users who wish to observe it are free to do so. However, the following usage, which may at times prove convenient, is permitted. Since the planting or failure to plant of a return linkage is triggered by use of *CALL or *GOTO, respectively, one can create a set of tables, one of which is accessed by a *CALL statement, which intercommunicate by means of *GOTO statements. Return to the point of invocation can be arranged by explicit use of a *RETURN statement or by implicit return by completing an action list in a table marked CLOSED. This usage permits considerably more freedom in table construction without incurring the liability of a large stack of link-back information.

This increased freedom in program construction using interconnected decision tables carries with it a requirement for increased programmer responsibility. If the program structure is such that there are circuits with unequal numbers of *CALL and *RETURN statements, then the potential exists for blundering into an incorrect number of entries in the stack of return links. Correct use of the extended program construction facility requires that a flow chart showing gross program structure at the table level be susceptible to a division by a number of contour lines, each of which is crossed in one direction by a *CALL statement and in the opposite direction by a *RETURN statement or its equivalent. Note that this requirement does not exclude con-

struction of recursive tables or groups of tables which invoke one another recursively.

VI. IMPLEMENTATION STRATEGY

The ideas described in this paper have been implemented in two stages. The first stage was construction of a modularized bootstrap capable of translating correct limited entry decision tables into PL/I coding without any diagnostic capability. Although most modules were skeletal, certain central modules concerned with building the decision tree and constructing the directory vector were so written that they would serve in these capacities in the ultimate system. This bootstrap was then employed to bring up the whole system expressed in decision tables. The use of decision tables processed by the bootstrap to implement the final system played the double role of providing test data for the system and of increasing our productivity as programmers during the final stages of system development.

At this writing our experience with the system has been with the decision tables which implement it and with a few contrived examples of very limited scope. The system appears to be quite satisfactory. The design offers the possibility of including additional features beyond those suggested here, and so deserves consideration in any new decision table implementation.

1. HUGHES, M. L., Shank, R. M., and Stein, E. S., Decision Tables, MDI Publications (Management Development Institute Division of Information Industries, Inc.), Wayne, Pennsylvania (1968).

2. SILBERG, B., (ed.), Special Issue on Decision Tables, SIGPLAN Notices, 6, #8 (1971).

3. KING, P. J. H., Ambiguity in Limited Entry Decision Tables, Comm. ACM, 11, (10), 680 (1968).

4. KING, P. J. H., Conversion of Decision Tables to Computer Programs by Rule Mask Techniques, Comm, Acm, 9, (11), 796 (1966).

5. COULTER, K. J., PET (Pre-processed of Encoded Tables), SHARE Program Library, 360D-03.2.004 (1967).

6. VEINOTT, C. G., Programming Decision Tables in FORTRAN, COBOL, or ALGOL, Comm. ACM, 9, (1), 31 (1966).

7. CHAPIN, N., Abstract 16, 712 Comp. Revs., 10, (5), 233 (1969).

8. POLIACK, S. L., Conversion of Limited-Entry Decision Tables to Computer Programs, Comm. ACM, 8, (11), 677, (1965).

9. MUTHUKRISHNAN, C. R., and Rajarman, V., On the Conversion of Decision Tables to Computer Programs, Comm. ACM, 13, (6), 347 (1970).

10. PRESS, L. I., Conversion of Decision Tables to Computer Programs, Comm. ACM, 8, (6), 385 (1965).

11. CHAPIN, N., Parsing of Decision Tables, Comm, ACM, 10, (8), 507 (1967).

```
DECTAL8: PROC OPTIONS(MAIN):                                          1    1
/* FILE DECLARATIONS          */                                     1    2
      DCL                                                           1    2
/* INPUT FILE                 */ SYSIN FILE RECORD INPUT EXT,        1    2
/* PRINT FILE                 */ PRINT FILE PRINT OUTPUT EXT,        1    2
/* PUNCH FILE                 */ PUNCH FILE RECORD OUTPUT EXT;       1    2
/* INPUT RECORD               */                                     1    3
      DCL                        1 CARDIN,                          1    3
/* COLUMN ONE                 */   2 BLANK CHAR(1),                 1    3
/* INPUT TEXT                 */   2 TEXT CHAR(71),                 1    3
/* SEQUENCE NUMBER            */   2 SQ CHAR(8),                    1    3
/* STATEMENT ID FIELD         */ ID CHAR(3) DEF CARDIN,              1    3
/* LETTER PART OF ID          */ IDL CHAR(1) DEF CARDIN,             1    3
                                 CARDINSTR CHAR(80) DEF CARDIN,     1    3
/* INDEX OF ACTIVE CHAR       */ INX BIN FIXED,                      1    3
/* LENGTH OF ACTIVE SUBSTR    */ INL BIN FIXED,                      1    3
/* CHARACTER 2 BASED          */ CHARB2 CHAR(2) BASED(CHARB2PT),     1    3
/* ONE CHAR SUBSTRINGS        */ CHAR1N(71) CHAR(1) DEF CARDIN.TEXT; 1    3
/**/                                                                1    4
/* PUNCH RECORD               */                                     1    4
      DCL                        1 CARDOUT,                         1    4
/* COLUMN ONE                 */   2 BLANK CHAR(1),                 1    4
/* OUTPUT TEXT                */   2 TEXT CHAR(71),                 1    4
/* SEQUENCE NUMBER            */   2 SQ PIC 'ZZZZZZZ9',             1    4
/* INDEX OF ACTIVE CHAR       */ OUTX BIN FIXED,                     1    4
/* LENGTH OF ACTIVE SUBSTR    */ OUTL BIN FIXED,                     1    4
                                 SEQNO BIN FIXED(15) INIT(0),       1    4
/* CARDOUT STRING             */ CARDOUTSTR CHAR(80) DEF CARDOUT;    1    4
/**/                                                                1    5
/* PRINT RECORD               */                                     1    5
      DCL                        1 LINEOUT,                         1    5
/* STUB ID                    */   2 SID CHAR(3),                   1    5
                                   2 FA CHAR(2),                    1    5
/* CROSS REFERENCE            */   2 XREF CHAR(6),                  1    5
                                   2 FB CHAR(2),                    1    5
/* # SIGN OR BLANK            */   2 #SIGN CHAR(1),                 1    5
/* STUB                       */   2 STUB CHAR(71),                 1    5
/* CONSTANT='  :: '           */   2 DCOLON CHAR(5),                1    5
/* ENTRY                      */   2 ENTRY(42) CHAR(1),             1    5
/* LINEOUT STRING             */ LINEOUTSTR CHAR(132) DEF LINEOUT;   1    5
/* DICTIONARY LIST            */                                     1    6
      DCL                        1 DICL(0:MNDICL+32) CTL,           1    6
/* ID NUMBER CODE             */   2 IDC BIN FIXED,                 1    6
/* RESOLUTION FOR ID CODE     */   2 RES BIN FIXED,                 1    6
/* ACTUAL NO IN DIC LIST      */ NDICL BIN FIXED,                    1    6
/* TABLE CODE BASE            */ TCBASE BIN FIXED,                   1    6
/* TRIAL ID CODE              */ TIDC BIN FIXED,                     1    6
/* TRIAL ID CODE OFFSET       */ TIDCOFF BIN FIXED,                  1    6
/* DIC LIST SUBSCRIPT         */ DICLS BIN FIXED:                    1    6
      DCL                        CSS(4) BIN FIXED:                  1    7
/* 1: CALL CONTROL SECT       */                                     1    8
/* 2: GO TO CONTROL SECT      */                                     1    8
/* 3: RETURN CONTROL SECT     */                                     1    8
/* 4: END STMT LABEL SUBS     */                                     1    8
      DCL                        DIR(MNADS) BIN FIXED CTL,          1    8
/* NEXT AVAILABLE DIR SUBS    */ NADS BIN FIXED,                     1    8
/* NEXT AVAILABLE LABEL SUB   */ NAS BIN FIXED;                      1    8
/* CURRENT TABLE INFO         */                                     1    9
```

```
        DCL                                                          1    9
/* ENTRY TABLE                */ ET(MNS,MNR) CHAR(1) CTL,           1    9
/* ENTRY TABLE ROW ELEMENTS*/ ETELEM(100) CHAR(1) BASED(ETELEMPT),  1    9
/* ENTRY TABLE ROW            */ ETROW CHAR(100) BASED(ETROWPT),    1    9
/* ENTRY TABLE ROW PRIME      */ ETROWP CHAR(100) BASED(ETROWPPT),  1    9
/* RULE NUMBERS               */ RULENO(MNR) CHAR(2) CTL,           1    9
/* ELSE SWITCH                */ ELSE CHAR(1),                      1    9
/* STUB LABEL SUBSCRIPTS      */ SS(MNS) BIN FIXED CTL,             1    9
/* MAX NO RULES               */ MNR BIN FIXED INIT(21),            1    9
/*MAX NO STUBS                */ MNS BIN FIXED INIT(50),            1    9
/* ACTUAL NO STUBS            */ NS BIN FIXED,                      1    9
/* ACTUAL NO RULES            */ NR BIN FIXED,                      1    9
/* ACTUAL NO CONDITIONS       */ NC BIN FIXED,                      1    9
/* ACTUAL NO INIT ACTIONS     */ NIA BIN FIXED,                     1    9
/* ACTUAL NO ACTIONS          */ NA BIN FIXED,                      1    9
/* FIRST ACTION LABEL SUB     */ FAS(MNR) BIN FIXED CTL,            1    9
/* SECOND ACTION DIR SUB      */ SADS(MNR) BIN FIXED CTL;           1    9
/**/                                                                 1   10
/**/    DCL                                                          1   10
/* NEXT PROCEDURE TO CALL     */ NPROC BIN FIXED;                   1   10
        DCL MNDICL BIN FIXED INIT(300),                              1   11
        MNADS BIN FIXED INIT(1000);                                  1   11
        DCL PROCLAB(5) LABEL;                                        1   12
        OPEN FILE(PRINT)OUTPUT LINESIZE(132);                        1   13
        ALLOCATE ET,SS,DICL,DIR,RULENO,FAS,SADS;                     1   14
        NPROC=1;                                                     1   15
NEXT:   GO TO PROCLAB(NPROC);                                        1   16
PROCLAB(4):                                                          1   17
        CALL EXOUT;                                                  1   17
PROCLAB(1):                                                          1   18
        CALL DECOPT;                                                 1   18
        GO TO NEXT;                                                  1   19
PROCLAB(3):                                                          1   20
        CALL PARSE;                                                  1   20
PROCLAB(2):                                                          1   21
        CALL INPUT;                                                  1   21
        GO TO NEXT;                                                  1   22
PROCLAB(5):                                                          1   23
        FREE ET,SS,DICL,DIR,RULENO,FAS,SADS;                         1   23
IACSS:      PROC(ICODE);                                             2   24
               IF CSS(ICODE)=0 THEN                                  3   25
                  DO;                                                4   26
                  CSS(ICODE)=NAS;                                    4   27
                  NAS=NAS+1;                                         4   28
                  END;                                               4   29
            RETURN(CSS(ICODE));                                      2   30
            END IACSS;                                               2   31
```

```
DECOPT:     PROC;                                                    2  32
            ON ENDFILE(SYSIN)                                        2  33
            GO TO TERMINATE;                                         2  34
/*LOOK FOR DECTAL OPTION CARD*/                                      2  35
LOOKOPT:    READ FILE(SYSIN)INTO(CARDIN);                            2  35
               IF SUBSTR(CARDINSTR,1,8)¬='*DECTAL ' THEN             3  36
                  DO;                                                4  37
                  WRITE FILE(PUNCH)FROM(CARDIN);                     4  38
                  GO TO LOOKOPT;                                     4  39
                  END;                                               4  40
            PUT PAGE FILE(PRINT);                                    2  41
            LINEOUTSTR=' ';                                          2  42
            REVERT ENDFILE(SYSIN);                                   2  43
/*NO OPTIONS ARE SUPPORTED BY THE BOOTSTRAP*/                        2  44
            READ FILE(SYSIN)INTO(CARDIN);                            2  44
LOOKT:         IF IDL='T' THEN                                       3  45
                  DO;                                                4  46
                  CSS=0;                                             4  47
                  DIR(1)=2;                                          4  48
                  NADS=2;                                            4  49
                  NAS=1;                                             4  50
                  NDICL=0;                                           4  51
                  CARDOUTSTR='          GO TO #START;';              4  52
                  WRITE FILE(PUNCH)FROM(CARDOUT);                    4  53
                  NPROC=2;                                           4  54
                  RETURN;                                            4  55
                  END;                                               4  56
            WRITE FILE(PUNCH)FROM(CARDIN);                           2  57
            SUBSTR(LINEOUTSTR,14,72)=SUBSTR(CARDINSTR,1,72);         2  57
            PUT FILE(PRINT)SKIP EDIT(LINEOUTSTR)(A);                 2  58
            READ FILE(SYSIN)INTO(CARDIN);                            2  59
            GO TO LOOKT;                                             2  60
TERMINATE:  NPROC=5;                                                 2  61
            END DECOPT;                                              2  62
```

```
INPUT:      PROC;                                                          2  63
/* INPUT SECTION VARIABLES */                                             2  64
            DCL                                                           2  64
   /* TEMPORARY NO OF RULES    */ TNR BIN FIXED,                         2  64
   /* ID CODE BREAKER          */ 1 CDBKR BASED(CDBKRPT),                2  64
                                    2 TDOPE,                             2  64
                                      3 T CHAR(1),                       2  64
                                      3 TNO PIC'99',                     2  64
                                    2 ADOPE,                             2  64
                                      3 AORC CHAR(1),                    2  64
                                      3 SNO PIC'99',                     2  64
   /* ID CODE BREAKER STRING   */ CDBKRSTR CHAR(6) BASED(CDBKRPT),       2  64
   /* RE-ENTRY OFFSET          */ REOFF BIN FIXED,                       2  64
   /* WORK STRING              */ WORKSTR CHAR(71),                      2  64
   /* LABEL SUPSCRIPT          */ SUBPIC PIC'999',                       2  64
   /* VALUE OF NS FOR FIRST    */ STOP(0:MNR) BIN FIXED,                 2  64
   /* TRANSLATOR GOTO OR STOP */                                         2  64
   /* NO RULE CARD SWITCH      */ NRCSW CHAR(1),                         2  64
   /* WORK STRING              */ TEXTP CHAR(71),                        2  64
   /* GO CODE                  */ GOCD BIN FIXED,                        2  64
   /* OPEN OR CLOSED              OORC CHAR(1),          */             2  64
   /* TABLE ID                 */ TID CHAR(3),                           2  64
   /*STATEMENT ID              */ STID CHAR(3),                          2  64
   /* NUMBER PART OF ID        */ IDN PIC'99' DEF STID POS(2),           2  64
   /*STATEMENT ID LETTER       */ STIDL CHAR(1) DEF STID;                2  64
            DCL NERRS INIT(0);                                           2  65
            ON ERROR                                                     2  66
               BEGIN;                                                    3  67
               NERRS=NERRS+1;                                            3  68
                  IF NERRS>10 THEN                                       4  69
                     DO;                                                 5  70
                     NPROC=5;                                            5  71
                     GOTO SRET;                                          5  72
                     END;                                                5  73
               PUT FILE(SYSPRINT)EDIT(' OCCURRED AT ',TID,STID)(A);      3  74
               GO TO RET41;                                              3  75
               END;                                                      3  76
/*T40: */      IF IDL¬='T' THEN                                          3  77
                  DO;                                                    4  78
                  NPROC=4;                                               4  79
SRET:             RETURN;                                                4  80
                  END;                                                   4  81
               ELSE                                                      3  82
               GO TO T41;                                                3  82
T41:        NIA,NC,NA,NR=0;                                              2  83
            STOP=0;                                                      2  84
            NS=1;                                                        2  85
            CARDOUTSTR=' ';                                              2  86
            LINEOUTSTR=' ';                                              2  87
            OUTX=2;                                                      2  88
            CALL T46;                                                    2  89
RET41:         IF IDL='A' THEN                                           3  90
                  IF NC=0 THEN                                           4  91
                     DO;                                                 5  92
                     CALL T38;                                           5  93
                     NIA=NIA+1;                                          5  94
T41R02P:             TIDOFF=128;                                         5  95
T41R02PP:            CALL T49;                                           5  96
```

```
                  NS=NS+1;                                          5   97
                  GO TO RET41;                                      5   98
                  END;                                              5   99
               ELSE                                                 4  100
                  DO;                                               5  100
                  CALL T38;                                         5  101
                  NA=NA+1;                                          5  102
                  GO TO T41R02P;                                    5  103
                  END;                                              5  104
            ELSE                                                    3  105
               IF IDL='C' THEN                                      4  105
                  DO;                                               5  106
                  CALL T38;                                         5  107
                  TIDCOFF=0;                                        5  108
                  NC=NC+1;                                          5  109
                  GO TO T41R02PP;                                   5  110
                  END;                                              5  111
               ELSE                                                 4  112
                  IF IDL='R' THEN                                   5  112
                     DO;                                            6  113
                     CALL T43;                                      6  114
                     GO TO RET41;                                   6  115
                     END;                                           6  116
                  ELSE                                              5  117
/* A BLANK IDL ==> UNEXPECTED CONTINUATION CARD*/                   6  117
                     IF IDL='T' THEN                                6  117
                        IF NS=1 THEN                                7  118
                           IF NC=0 THEN                             8  119
                           GO TO T41;                               8  120
                           ELSE                                     8  121
                           GO TO T41R08;                            8  121
                        ELSE                                        7  122
SIFNC:                     IF NC=0 THEN                             8  122
                              DO;                                   9  123
                              CALL T59;                             9  124
                              NPROC=2;                              9  125
                              RETURN;                               9  126
                              END;                                  9  127
                           ELSE                                     8  128
T41R08:                       DO;                                   9  128
                              CALL T59;                             9  129
                              NPROC=3;                              9  130
                              RETURN;                               9  131
                              END;                                  9  132
                     ELSE                                           6  133
                     GO TO SIFNC;                                   6  133
T38:           PROC;                                                3  134
/*T38A: */        IF STIDL=IDL THEN                                 4  135
                  RETURN;                                           4  136
                  ELSE                                              4  137
                     IF STIDL='A'&IDL='C'&NC=0 THEN                 5  137
                     DO;                                            6  138
/*A20*/                 IF NR>0 THEN                                7  139
                        DO;                                         8  140
                        IEO=MIN(3,FLOOR(42/(NR+1)));                8  141
                           IF IEO¬=3 THEN                           9  142
                              DO I=1 TO NR;                        10  143
         IF SUBSTR(RULENO(I),1,1)¬=' ' THEN                        11  144
```

```
            DO;                                                          12  145
               DO J=1 TO NR;                                             13  146
               LINEOUT.ENTRY(J*IEO)=SUBSTR(RULENO(J),1,1);               13  147
               SUBSTR(RULENO(J),1,1)=' ';                                13  148
               END;                                                      13  149
            PUT FILE(PRINT)SKIP EDIT(LINEOUT)(A);                        12  150
            GO TO SDOI;                                                  12  151
            END;                                                         12  152
                                    END;                                 10  153
SDOI:                           DO I=NR TO 1 BY-1;                        9  154
                                CHARB2PT=ADDR(LINEOUT.ENTRY(I*IEO-1));    9  155
                                CHARB2=RULENO(I);                         9  156
                                END;                                      9  157
                                IF UNSPEC(ELSE)THEN                       9  158
                                LINEOUT.ENTRY(IEO*(NR+1))='E';            9  159
                             END;                                         8  160
                          ELSE                                            7  161
                             DO;                                          8  161
                             NR=42;                                       8  162
                             UNSPEC(NRCSW),UNSPEC(ELSE)='00000001'B;      8  163
                             END;                                         8  164
                       PUT FILE(PRINT)SKIP EDIT(LINEOUT)(A);              6  165
                       LINEOUTSTR=' ';                                    6  166
                       RETURN;                                            6  167
                       END;                                               6  168
                    ELSE                                                  5  169
                       IF STIDL='C'&IDL='A' THEN                          6  169
                          DO;                                             7  170
                          PUT FILE(PRINT)SKIP EDIT((118)'=')(X(14),A(11   7  171
                          8));                                            7  171
                          RETURN;                                         7  172
                          END;                                            7  173
/* ELSE CLAUSE HERE FOR INCORRECT STATEMENT TYPE&DUMP CARD TO OUTPUT*/     3  174
            END T38;                                                      3  174
T42:        PROC;                                                         3  175
T42A:       READ FILE(SYSIN)INTO(CARDIN);                                 3  176
               IF IDL='#' THEN                                            4  177
                  DO;                                                     5  178
                  LINEOUT.#SIGN='#';                                      5  179
                  LINEOUT.STUB=CARDIN.TEXT;                               5  180
                  PUT FILE(PRINT)SKIP EDIT(LINEOUT)(A);                   5  181
                  LINEOUTSTR=' ';                                         5  182
                  GO TO T42A;                                             5  183
                  END;                                                    5  184
            END T42;                                                      3  185
T43:        PROC;                                                         3  186
            CARDIN.TEXT=TRANSLATE(CARDIN.TEXT,' ',',');                   3  187
RET43:      INX=VERIFY(CARDIN.TEXT,' ');                                  3  188
               IF INX=0 THEN                                              4  189
               CALL T42;                                                  4  190
               ELSE                                                       4  191
                  IF CHARIN(INX)>='0' THEN                                5  191
                     DO;                                                  6  192
                     NR=NR+1;                                             6  193
                     INL=INDEX(SUBSTR(CARDIN.TEXT,INX),' ')+INX-3;        6  194
                     RULENO(NR)=SUBSTR(CARDIN.TEXT,INL,2);                6  195
                     SUBSTR(CARDIN.TEXT,INL,2)=' ';                       6  196
                     GO TO RET43;                                         6  197
```

```
                        END;                                                6  198
                     ELSE                                                   5  199
                        DO;                                                 6  199
                        UNSPEC(ELSE)='00000001'B;                           6  200
                        RULENO(NR+1)=' E';                                  6  201
                        CALL T42;                                           6  202
                        END;                                                6  203
                  END T43;                                                  3  204
T46:              PROC;                                                     3  205
                  UNSPEC(NRCSW),UNSPEC(ELSE)='00000000'B;                   3  206
                  STID,LINEOUT.SID,TID=ID;                                  3  207
                  TCBASE=256*IDN;                                           3  208
                  TIDC=TCBASE-1;                                            3  209
                  ID=' ';                                                   3  210
                  STIDL='A';                                                3  211
                  CALL DICLUP;                                              3  212
/*          II=VERIFY(CARDIN.TEXT,' ');                                     4  213
                IF SUBSTR(CARDIN.TEXT,II,6)='CLOSED' THEN                   4  213
                OORC='C';                                                   4  213
                ELSE                                                        4  213
                OORC='O';             */                                    4  213
                   IF DICL.RES(DICLS)=0 THEN                                4  213
                      DO;                                                   5  214
                      DIR(NADS+1),DIR(NADS)=0;                              5  215
                      DICL.RES(DICLS)=-NADS;                                5  216
                      NADS=NADS+2;                                          5  217
                      GO TO T46R01P;                                        5  218
                      END;                                                  5  219
                   ELSE                                                     4  220
                      DO;                                                   5  220
/*                    IF DICL.RES(DICLS)>0 THEN                          T  5  221
HIS TABLE ID HAS BEEN USED ALREADY*/                                        5  221
T46R01P:              LINEOUT.STUB=CARDIN.TEXT;                             5  221
                      PUT FILE(PRINT)PAGE EDIT(LINEOUT)(A);                 5  222
                      LINEOUTSTR=' ';                                       5  223
                      PUT FILE(PRINT)SKIP EDIT('ID','XREF','STUB','ENTRIE   5  224
                      S')(COL(2),A,COL(7),A,COL(30),A,COL(100),A);          5  224
                      CALL T42;                                             5  225
                      END;                                                  5  226
                  END T46;                                                  3  227
T49:              PROC;                                                     3  228
                  GOCD=0;                                                   3  229
                  STID,LINEOUT.SID=ID;                                      3  230
                  ID=' ';                                                   3  231
                  INX=VERIFY(CARDIN.TEXT,' ');                              3  232
                     IF CHARIN(INX)='*' THEN                                4  233
                        DO;                                                 5  234
                        INX=INX+1;                                          5  235
                        GO TO T52;                                          5  236
                        END;                                                5  237
/*T50:*/          TIDC=TCBASE+IDN+TIDCOFF;                                  3  238
                  CALL DICLUP;                                              3  239
                     IF DICL.RES(DICLS)=0 THEN                              4  240
                        DO;                                                 5  241
                        SS(NS),SUBPIC,DICL.RES(DICLS)=NAS;                  5  242
                        NAS=NAS+1;                                          5  243
                        END;                                                5  244
                     ELSE                                                   4  245
```

```
/*                    IF DICL.RES(DICLS)>0 THEN                          4  245
THIS STATEMENT ID HAS BEEN USED ALREADY*/                                4  245
                   SUBPIC,SS(NS),DICL.RES(DICLS)=ABS(DICL.RES(DICLS));   4  245
                GO TO T56;                                               3  246
T52:            REOFF=0:                                                 3  247
T52A:           II=VERIFY(SUBSTR(CARDIN.TEXT,INX),' ');                  3  248
                   IF II>0 THEN                                          4  249
                   INX=INX+II-1:                                         4  250
                   IF CHARIN(INX)='T' THEN                               4  251
                      DO:                                                5  252
                      CDBKRPT=ADDR(CHARIN(INX));                         5  253
                      GO TO T54;                                         5  254
                      END:                                               5  255
                   ELSE                                                  4  256
                      IF SUBSTR(CARDIN.TEXT,INX,2)='RE' THEN             5  256
                         DO:                                             6  257
                         INX=INX+2;                                      6  258
                         REOFF=1;                                        6  259
                         GO TO T52A;                                     6  260
                         END;                                            6  261
                      ELSE                                               5  262
                         IF SUBSTR(CARDIN.TEXT,INX,4)='STOP' THEN        6  262
                            DO;                                          7  263
                            SS(NS)=IACSS(100B);                          7  264
                            GOCD=3;                                      7  265
                            GO TO T55;                                   7  266
                            END;                                         7  267
                         ELSE                                            6  268
                            IF SUBSTR(CARDIN.TEXT,INX,7)='GOAGAIN' THEN  7  268
                               DO;                                       8  269
                               TIDC=TCBASE-1;                            8  270
                               GOCD=2;                                   8  271
                               CALL DICLUP;                              8  272
                               SS(NS)=IACSS(108);                        8  273
                               SS(NS+1)=ABS(DICL.RES(DICLS))+REOFF;      8  274
                               NS=NS+1;                                  8  275
                                  IF NC=0 THEN                           9  276
                                  NIA=NIA+1;                             9  277
                                  ELSE                                   9  278
                                  NA=NA+1;                               9  278
                               GO TO T55;                                8  279
                               END;                                      8  280
                            ELSE                                         7  281
                               IF SUBSTR(CARDIN.TEXT,INX,5)='GOTO ' THEN 8  281
                                  DO;                                    9  282
                                  CDBKRPT=ADDR(CHARIN(INX+5));           9  283
                                  TIDC=CDBKR.TNO*256-1;                  9  284
                                  GOCD=2;                                9  285
                                  CALL DICLUP;                           9  286
                                  SS(NS)=IACSS(10B);                     9  287
T52R05P:                          NS=NS+1;                               9  288
                                     IF NC=0 THEN                       10  289
                                     NIA=NIA+1;                         10  290
                                     ELSE                               10  291
                                     NA=NA+1;                           10  291
                                  CALL T53;                              9  292
                                  GO TO T55;                             9  293
                                  END;                                   9  294
```

```
                        ELSE                                                8  295
                            IF SUBSTR(CARDIN.TEXT,INX,5)='CALL ' TH         9  295
                            EN                                              9  295
                                DO;                                        10  296
                                CDBKRPT=ADDR(CHARIN(INX+5));               10  297
                                TIDC=CDBKR.TNO*256-1;                      10  298
                                GOCD=1;                                    10  299
                                CALL DICLUP;                               10  300
                                SS(NS)=IACSS(1B);                          10  301
                                GO TO T52RUSP;                             10  302
                                END;                                       10  303
/* ELSE FLUSH ERRONEOUS CARDS */                                           3  304
T54:                                                                       3  304
/*      IF CDBKR.AORC¬=STIDL THEN                                 ERRONE   3  304
OUS CROSS REFERENCE   */                                                   3  304
            TIDC=256*CDBKR.TNO+TIDCOFF+CDBKR.SNO;                          3  304
            CALL DICLUP;                                                   3  305
               IF DICL.RES(DICLS)=0 THEN                                   4  306
                  DO;                                                      5  307
                  DICL.RES(DICLS)=-NAS;                                    5  308
                  SS(NS)=NAS;                                              5  309
                  NAS=NAS+1;                                               5  310
                  LINEOUT.XREF=CDBKRSTR;                                   5  311
                  GO TO T55;                                               5  312
                  END;                                                     5  313
               ELSE                                                        4  314
                  DO;                                                      5  314
                  SS(NS)=ABS(DICL.RES(DICLS));                             5  315
                  LINEOUT.XREF=CDBKRSTR;                                   5  316
                  GO TO T55;                                               5  317
                  END;                                                     5  318
T55:        II=INDEX(CARDIN.TEXT,'::');                                    3  319
               IF II=0 THEN                                                4  320
                  DO;                                                      5  321
                  LINEOUT.STUB=CARDIN.TEXT;                                5  322
                  PUT FILE(PRINT)SKIP EDIT(LINEOUT)(A);                    5  323
                  LINEOUTSTR=' ';                                          5  324
                  CALL T42;                                                5  325
                  GO TO T58;                                               5  326
                  END;                                                     5  327
               ELSE                                                        4  328
/*              IF NC=0 THEN PREMATURE ACTION ENTRIES*/                    5  328
                  DO;                                                      5  328
                  TNR=1;                                                   5  329
                  INX=II+2;                                                5  330
                  LINEOUT.STUB=CARDIN.TEXT;                                5  331
                  SUBSTR(LINEOUT.STUB,II)=' ';                             5  332
                  GO TO T64;                                               5  333
                  END;                                                     5  334
T56:        SUBSTR(WORKSTR,1,15)='#    :/*       */';                      3  335
            OUTL=15;                                                       3  336
            SUBSTR(WORKSTR,2,3)=SUBPIC;                                    3  337
            SUBSTR(WORKSTR,8,3)=TID;                                       3  338
            SUBSTR(WORKSTR,11,3)=LINEOUT.SID;                              3  339
            CALL T69;                                                      3  340
            CALL T39;                                                      3  341
RET56:      II=INDEX(TEXTP,'::');                                          3  342
               IF II=0 THEN                                                4  343
```

```
                  DO;                                                    5  344
                  LINEOUT.STUB=CARDIN.TEXT;                              5  345
                  PUT FILE(PRINT)SKIP EDIT(LINEOUT)(A);                  5  346
                  LINEOUTSTR=' ';                                        5  347
                  WORKSTR=CARDIN.TEXT;                                   5  348
                     DO OUTL=67 TO 2 BY-5 WHILE(SUBSTR(WORKSTR,OUTL,5    6  349
                     )=' ');                                             6  349
                     END;                                                6  350
                  OUTL=OUTL+4;                                           5  351
                  CALL T69;                                              5  352
                  CALL T42;                                              5  353
                  GO TO T57;                                             5  354
                  END;                                                   5  355
               ELSE                                                      4  356
/*             IF NC=0 THEN PREMATURE ACTION ENTRIES*/                   5  356
                  DO;                                                    5  356
                  LINEOUT.STUB=CARDIN.TEXT;                              5  357
                  SUBSTR(LINEOUT.STUB,II)=' ';                           5  358
                  WORKSTR=CARDIN.TEXT;                                   5  359
                  SUBSTR(WORKSTR,II)=' ';                                5  360
                     DO OUTL=67 TO 2 BY-5 WHILE(SUBSTR(WORKSTR,OUTL,5    6  361
                     )=' ');                                             6  361
                     END;                                                6  362
                  OUTL=OUTL+4;                                           5  363
                  CALL T69;                                              5  364
                     IF STIOL='A' THEN                                   6  365
                        DO;                                              7  366
                        SUBSTR(WORKSTR,1,11)='GOTO #NEXT:';              7  367
                        OUTL=11;                                         7  368
                        END;                                             7  369
                     ELSE                                                6  370
                        DO;                                              7  370
                        SUBSTR(WORKSTR,1,29)=' THEN GOTO #TRUE;GOTO #    7  371
                        FALSE;';                                         7  371
                        OUTL=29;                                         7  372
                        END;                                             7  373
                  CALL T69;                                              5  374
                  TNR=1;                                                 5  375
                  INX=II+2;                                              5  376
                  GOTO T64;                                              5  377
                  END;                                                   5  378
T57:           IF IOL=' ' THEN                                           4  379
                  DO;                                                    5  380
                  CALL T39;                                              5  381
                  GO TO RET56;                                           5  382
                  END;                                                   5  383
               ELSE                                                      4  384
                  DO;                                                    5  384
/*                   IF NC>0 THEN ENTRIES ARE MSSSING */                 6  385
                     IF STIOL='A' THEN                                   6  385
                        DO;                                              7  386
                        SUBSTR(WORKSTR,1,11)='GOTO #NEXT:';              7  387
                        OUTL=11;                                         7  388
                        END;                                             7  389
                     ELSE                                                6  390
                        DO;                                              7  390
                        SUBSTR(WORKSTR,1,29)=' THEN GOTO #TRUE;GOTO #    7  391
                        FALSE;';                                         7  391
```

```
                        OUTL=29;                                          7  392
                        END;                                              7  393
                    CALL T69;                                             5  394
                    RETURN;                                               5  395
                    END;                                                  5  396
T58:              IF IDL=' ' THEN                                         4  397
                  GO TO T55;                                              4  398
                  ELSE                                                    4  399
                    DO;                                                   5  399
/*                    IF NC>0 THEN ENTRIES ARE MSSING */                  6  400
                        IF GOOD>=2 THEN                                   6  400
                        STOP(0)=NS;                                       6  401
                    RETURN;                                               5  402
                    END;                                                  5  403
T64:            K=(UNSPEC(ELSE)='00000001'B)&(STIDL='A');                 3  404
T64A:           II=VERIFY(SUBSTR(CARDIN.TEXT,INX),' ');                   3  405
                  IF II>0 THEN                                            4  406
                    DO;                                                   5  407
                    INX=INX+II-1;                                         5  408
                    ET(NS,TNR)=CHARIN(INX);                               5  409
                    TNR=TNR+1;                                            5  410
                    INX=INX+1;                                            5  411
                    GO TO T64A;                                           5  412
                    END;                                                  5  413
                  ELSE                                                    4  414
                    IF TNR<=NR+K THEN                                     5  414
                      DO;                                                 6  415
                      INX=1;                                              6  416
                      CALL T42;                                           6  417
                      GO TO T64;                                          6  418
                      END;                                                6  419
                    ELSE                                                  5  420
                    GO TO T66;                                            5  420
T66:                                                                      3  421
/*A00*/         ETROWPT=ADDR(ET(NS,1));                                   3  421
                  IF UNSPEC(NRCSW)THEN                                    4  422
                    IF NC=1 THEN                                          5  423
                      DO;                                                 6  424
                      NR=TNR-1;                                           6  425
                      IFQ=MIN(3,FLOOR(42/(NR+1)));                        6  426
                      END;                                                6  427
                    ELSE                                                  5  428
                      IF NA=1 THEN                                        6  428
                        DO;                                               7  429
                        UNSPEC(NRCSW)='0'B;                               7  430
                          IF TNR=NR+1 THEN                                8  431
                            DO;                                           9  432
                            UNSPEC(ELSE)='0'B;                            9  433
                            K=0;                                          9  434
                            END;                                          9  435
                        END;                                              7  436
/*      IF STIDL='C' THEN  TEST FOR VALID ENTRIES AND DIAGNOSE   BAD ONES 3  437
 */             SUBSTR(ETROW,1,NR+BIN(UNSPEC(ELSE)))=TRANSLATE(SUBSTR(ETR 3  437
                ROW,1,NR+BIN(UNSPEC(ELSE))),'TF','YN');                   3  437
                  IF GOOD>0 THEN                                          4  438
                    IF GOOD<3 THEN                                        5  439
                      DO;                                                 6  440
                      ETROWPPT=ADDR(ET(NS-1,1));                          6  441
```

```
                        SUBSTR(CTROWP,1,NR+BIN(UNSPEC(ELSE)))=SUBSTR(ETR    6   442
                        OW,1,NR+BIN(UNSPEC(ELSE))):                          6   442
                        END;                                                 6   443
/* TEST TNR VS. NR+K+1 TO DIAGNOSE INCORRECT NUMBER OF ENTRIES */            3   444
                  LINEOUT.DCOLON='  || ';                                    3   444
/*A02*/               DO TNR=1 TO NR+K;                                      4   445
                     ENTRY(IEO*TNR)=ET(NS,TNR);                              4   446
                        IF GUCD>1 THEN                                       5   447
                           IF ET(NS,TNR)='X' THEN                            6   448
                              IF STOP(TNR)=0 THEN                            7   449
                              STOP(TNR)=NS;                                  7   450
                     END;                                                    4   451
                  PUT FILE(PRINT)SKIP EDIT(LINEOUT)(A);                      3   452
                  LINEOUTSTR=' ';                                            3   453
                  CALL T42;                                                  3   454
                  RETURN;                                                    3   455
T53:                 PROC;                                                   4   456
                        IF DICL.RES(DICLS)=0 THEN                            5   457
                           DO;                                               6   458
                           SS(NS)=NADS+REOFF;                                6   459
                           DIR(NADS+REOFF)=-1;                               6   460
                           DIR(NADS-REOFF+1)=0;                              6   461
                           DICL.RES(DICLS)=-NADS;                            6   462
                           NADS=NADS+2;                                      6   463
                           END;                                              6   464
                        ELSE                                                 5   465
                           DO;                                               6   465
                           SS(NS)=ABS(DICL.RES(DICLS))+REOFF;                6   466
                              IF DICL.RES(DICLS)<0 THEN                      7   467
                              DIR(SS(NS))=-1;                                7   468
/*                        ELSE    BAD TABLE ENTRY POINT*/                    6   469
                           END;                                              6   469
                     END T53;                                                4   470
T69:                 PROC;                                                   4   471
                        IF OUTX+OUTL-1>72 THEN                               5   472
                           DO;                                               6   473
                           SUBSTR(CARDOUTSTR,OUTX,73-OUTX)=SUBSTR(WORKSTR,1  6   474
                           ,73-OUTX);                                        6   474
                           SEQNO,CARDOUT.SO=SEQNO+1;                         6   475
                           WRITE FILE(PUNCH)FROM(CARDOUT);                   6   476
                           SUBSTR(CARDOUTSTR,2,OUTL-73+OUTX)=SUBSTR(WORKSTR  6   477
                           ,74-OUTX,OUTL-73+OUTX);                           6   477
                           OUTX=OUTL-71+OUTX;                                6   478
                           END;                                              6   479
                        ELSE                                                 5   480
                           DO;                                               6   480
                           SUBSTR(CARDOUTSTR,OUTX,OUTL)=SUBSTR(WORKSTR,1,OU  6   481
                           TL);                                              6   481
                           OUTX=OUTX+OUTL;                                   6   482
                           END;                                              6   483
                     END T69;                                                4   484
T39:                 PROC;                                                   4   485
                     TEXTP=CARDIN.TEXT;                                      4   486
                     II=INDEX(TEXTP,'''');                                   4   487
                     JJ=INDEX(TEXTP,'/*');                                   4   488
RET39:                  IF II=0 THEN                                         5   489
                           IF JJ=0 THEN                                      6   490
                           RETURN;                                           6   491
```

```
                        ELSE                                                    6  492
T39R04:                     DO;                                                 7  492
                            KK=INDEX(SUBSTR(TEXTP,JJ),'*/');                    7  493
                            SUBSTR(TEXTP,JJ,KK+1)=' ';                          7  494
                            JJ=INDEX(TEXTP,'/*');                               7  495
                            GO TO RET39;                                        7  496
                            END;                                                7  497
                        ELSE                                                    5  498
                            IF JJ=0 THEN                                        6  498
                            DO;                                                 7  499
T39R05:                     KK=INDEX(SUBSTR(TEXTP,II+1),'''');                  7  500
                            SUBSTR(TEXTP,II,KK+1)=' ';                          7  501
                            II=INDEX(TEXTP,'''');                               7  502
                            GO TO RET39;                                        7  503
                            END;                                                7  504
                            ELSE                                                6  505
                            IF II>JJ THEN                                       7  505
                            GO TO T39R04;                                       7  506
                            ELSE                                                7  507
                            GO TO T39R05;                                       7  507
                    END T39:                                                    4  508
                END T49:                                                        3  509
T59:            PROC;                                                           3  510
                SUBSTR(CARDOUTSTR,OUTX)=' ';                                    3  511
                SECND,CARDOUT.SQ=SECND+1;                                       3  512
                WRITE FILE(PUNCH)FROM(CARDOUT);                                 3  513
                CARDOUTSTR=' ';                                                 3  514
                OUTX=2;                                                         3  515
/*A01*/         FAS=0;                                                          3  516
                    DO I=1 TO NR+BIN(UNSPEC(ELSE));                             4  517
                        DO J=NIA+NC+1 TO NS-1;                                  5  518
/*FIRST,CONSTRUCT ACTION LISTS FOR RULES IN DIR*/                               6  519
                        IF RT(J,I)='X' THEN                                     6  519
/*                      IF J>STOP(I)THEN                                      7  520
                          IF STOP(I)>0 THEN INACCESSIBLE ACTION */              7  520
                            IF FAS(I)=0 THEN                                    7  520
                                DO;                                             8  521
                                FAS(I)=-SS(J);                                  8  522
                                SADS(I)=NADS;                                   8  523
                                END;                                            8  524
                            ELSE                                                7  525
                                DO;                                             8  525
                                DIR(NADS)=SS(J);                                8  526
                                NADS=NADS+1;                                    8  527
                                END;                                            8  528
                        END;                                                    5  529
/*PROVIDE ANY REQUIRED DIAGNOSTICS AND RETURN LINKS*/                           5  530
/* THE BOOTSTRAP AUTOMATICALLY SUPPLIES *RETURN TO EVERY RULE NOT               5  530
CONCLUDING WITH A TRANSFER OF CONTROL*/                                         5  530
                        IF STOP(I)<=0 THEN                                      5  530
/*                      IF ODRC='01' THEN MISSING TRANSFER OF CONTROL */       6  531
                            IF FAS(I)=0 THEN                                    6  531
                                DO;                                             7  532
                                FAS(I)=-IACSS(11B);                             7  533
                                SADS(I)=NADS;                                   7  534
                                END;                                            7  535
                            ELSE                                                6  536
                                DO;                                             7  536
```

```
                        DIR(NADS)=IACSS(11B);                                   7  537
                        NADS=NADS+1;                                            7  538
                        END;                                                    7  539
                    END;                                                        4  540
/*PROVIDE TABLE LINKAGE FOR MAIN ENTRY*/                                        3  541
                TIDC=TCBASE-1;                                                  3  541
                CALL DICLUP;                                                    3  542
                DICL.RES(DICLS)=ABS(DICL.RES(DICLS));                           3  543
                DIR(DICL.RES(DICLS))=NADS;                                      3  544
/*TRANSFER INITIALIZING ACTION LIST TO DIR*/                                    4  545
                    DO I=0 TO NIA;                                              4  545
                    DIR(NADS+I)=SS(I+1);                                        4  546
                    END;                                                        4  547
                NADS=NADS+NIA;                                                  3  548
                    IF NC=0 THEN                                                4  549
/*PROVIDE DIAGNOSTICS OR RETURN LINK FOR ACTION TABLE*/                         5  550
                      IF NS-1¬=STOP(0)THEN                                      5  550
/*                    IF STOP(0)>1 THEN INACCESSIBLE ACTIONS */                 6  551
/*                    ELSE                 */                                 6  551
/*                       IF OORC='0' THEN MISSING TRANSFER OF CONTROL  */       6  551
                        DO;                                                     6  551
                        DIR(NADS)=IACSS(11B);                                   6  552
                        NADS=NADS+1;                                            6  553
                        END;                                                    6  554
/*PROVIDE TABLE LINKAGE FOR RE-ENTRY POINT*/                                    3  555
/*              IF NC=0 THEN                                                    3  555
                  IF DIR(DICL.RES(DICLS)+1)=-1 THEN                          I  3  555
NVALID FORWARD REFERENCE SOMEPLACE  */                                          3  555
                DIR(DICL.RES(DICLS)+1)=NADS;                                    3  555
                END T59;                                                        3  556
```

```
DICLUP:        PROC:                                                    3  557
               DCL                                                      3  558
      /* NEW LOCATION            */ NLOC CHAR(128) BASED(NLOCPT),       3  558
      /* OLD LOCATION            */ OLOC CHAR(128) BASED(OLOCPT);       3  558
               DCL (HIGH,LOW) BIN FIXED;                                3  559
/* LOOK FOR SDI */                                                     3  560
               LOW=1;                                                   3  560
               HIGH=NDICL;                                              3  561
GUESS:         DICLS=(LOW+HIGH)/2;                                      3  562
                  IF DICL.IDC(DICLS)<TIDC THEN                          4  563
/* TOO LOW */        DO;                                               5  564
                     LOW=DICLS+1;                                       5  565
                        IF LOW<=HIGH THEN                               6  566
                        GO TO GUESS;                                    6  567
                     GO TO NOTF;                                        5  568
                     END;                                               5  569
                  IF DICL.IDC(DICLS)>TIDC THEN                          4  570
/* TOO HIGH */       DO;                                               5  571
                     HIGH=DICLS-1;                                      5  572
                        IF LOW<=HIGH THEN                               6  573
                        GO TO GUESS;                                    6  574
                     GO TO NOTF;                                        5  575
                     END;                                               5  576
               RETURN;                                                  3  577
NOTF:          DICLS=LOW;                                               3  578
/* IF NDICL >=MNDICL THEN ERROR */                                      4  579
                  IF DICLS<=NDICL THEN                                  4  579
                     DO;                                                5  580
                        DO I=DICLS TO NDICL BY 32;                      6  581
                        END;                                            6  582
                     I=I-32;                                            5  583
                        DO J=I TO DICLS BY-32;                          6  584
                        OLOCPT=ADDR(DICL(J));                           6  585
                        NLOCPT=ADDR(DICL(J+1));                         6  586
                        NLOC=OLOC;                                      6  587
                        END;                                            6  588
                     END;                                               5  589
               DICL.IDC(DICLS)=TIDC;                                    3  590
               DICL.RES(DICLS)=0;                                       3  591
               NDICL=NDICL+1;                                           3  592
               END DICLUP;                                              3  593
            END INPUT;                                                  2  594
```

```
PARSE:      PROC;                                                          2  595
/* DYNAMIC ORDER OF          */                                           2  596
/* CONDITION STUBS           */                                           2  596
            DCL                    DOCS(NC) BIN FIXED,                   2  596
   /* NO OF ACTIVE LEVEL      */ NAL BIN FIXED,                          2  596
   /* COND BRANCH VALUES      */ CDB(NIA+1:NIA+NC) CHAR(1);              2  596
/**/                                                                       2  597
/* TRUE BRANCH LIST          */                                           2  597
            DCL                    1 TRUE,                               2  597
   /* ACTUAL NO IN TBL        */    2 NTBL BIN FIXED,                     2  597
   /* NO IN TBL FOR ERRORS    */    2 MTBL BIN FIXED,                     2  597
   /* TRUE BRANCH LIST        */    2 TBL(NR) BIN FIXED;                  2  597
/**/                                                                       2  598
/* FALSE BRANCH LIST STACK */                                             2  598
            DCL                    1 FALSES(NC),                         2  598
   /* ACTUAL NO IN FBLS(NC)   */    2 NFBLS BIN FIXED,                    2  598
   /* NO IN FBLS FOR ERRORS   */    2 MFBLS BIN FIXED,                    2  598
   /* FALSE BRANCH LIST STACK */    2 FBLS(NR) BIN FIXED,                 2  598
   /* FALSE DIR SUB STACK     */ FDSS(NC) BIN FIXED,                     2  598
   /* ONE ELEMENT IN STACK    */ 1 FALSE BASED(FBLPT),                   2  598
   /* ACTUAL NO IN FBL        */    2 NFBL BIN FIXED,                     2  598
   /* NO IN FBL FOR ERRORS    */    2 MFBL BIN FIXED,                     2  598
   /* FALSE BRANCH LIST       */    2 FBL(100) BIN FIXED;                 2  598
/**/                                                                       2  599
/* ACTIVE BRANCH LIST        */                                           2  599
            DCL                    1 ACTIVE BASED(ABLPT),                2  599
   /* ACTUAL NO IN ABL        */    2 NABL BIN FIXED,                     2  599
   /* NO IN ABL FOR ERRORS    */    2 MABL BIN FIXED,                     2  599
   /* ACTIVE BRANCH LIST      */    2 ABL(100) BIN FIXED;                 2  599
            DCL                                                           2  600
   /* LEAF SWITCH             */ LEAF CHAR(1);                           2  600
               DO I=NIA+1 TO NIA+NC;                                      3  601
/*INITIALIZE CONDITION BRANCH VALUES AND DOCS*/                           3  602
               CDB(I)='-';                                                3  602
               DOCS(I-NIA)=I;                                             3  603
               END;                                                       3  604
               DO I=1 TO NR;                                              3  605
/*INITIALIZE TRUE BRANCH LIST FOR STARTER*/                               3  606
               TBL(I)=I;                                                  3  606
               END;                                                       3  607
            NTBL=NR;                                                      2  608
            NAL=1;                                                        2  609
LTRUE:      ABLPT=ADDR(TRUE);                                             2  610
/*EXPLORE A TRUE BRANCH OF THE DECISION TREE UNTIL A LEAF NODE IS MET*/   2  611
            CALL REDOCS;                                                  2  611
            DIR(NADS)=SS(DOCS(NAL));                                      2  612
               IF NAL=1 THEN                                              3  613
               NADS=NADS+1;                                               3  614
               ELSE                                                       3  615
               NADS,DIR(NADS+1)=NADS+4;                                   3  615
BRANCH:     FBLPT=ADDR(FALSES(NAL));                                      2  616
            FDSS(NAL)=NADS+2;                                             2  617
            CALL BRANCHP;                                                 2  618
            CDB(DOCS(NAL))='T';                                           2  619
            NLDS=NADS;                                                    2  620
            ABLPT=ADDR(TRUE);                                             2  621
            GO TO TLEAF;                                                  2  622
FLEAF:      CDB(DOCS(NAL))='F';                                           2  623
```

```
/*TRUE BRANCH EXHAUSTED AT THIS LEVEL:EXPLORE FALSE BRANCH*/          2  624
          NLDS=FDSS(NAL);                                           2  624
          ABLPT=FBLPT;                                              2  625
TLEAF:       IF NABL=0 THEN                                         3  626
                DO;                                                 4  627
/*IF ¬ELSE THEN ERROR */                                            4  628
                ABL(1)=NR+1;                                        4  628
                GO TO ISLEAF;                                       4  629
                END;                                                4  630
          CALL LEAFREC;                                             2  631
             IF LEAF='1' THEN                                       3  632
                DO;                                                 4  633
ISLEAF:                                                             4  634
/*    IF NABL>1 THEN RESPOND TO APPARENT AMBIGUITY*/                4  634
/*PLANT LINKS APPROPRIATE TO PRESENCE AT LEAF NODE*/                4  634
                FAS(ABL(1))=ABS(FAS(ABL(1)));                       4  634
                DIR(NLDS)=FAS(ABL(1));                              4  635
                DIR(NLDS+1)=SADS(ABL(1));                           4  636
                   IF CDB(DOCS(NAL))='T' THEN                       5  637
                      DO;                                           6  638
                      NADS=NADS+4;                                  6  639
                      GO TO FLEAF;                                  6  640
                      END;                                          6  641
                   ELSE                                             5  642
                      DO;                                           6  642
DECNAL:               CDB(DOCS(NAL))='-';                           6  643
                      NAL=NAL-1;                                    6  644
                         IF NAL=0 THEN                              7  645
/*SCAN FOR RULES NOT EMPLOYED BY CHECKING FOR NEGATIVE FAS ENTRIES*/ 7  646
                         RETURN;                                    7  646
                         IF CDB(DOCS(NAL))='F' THEN                 7  647
/*LOOP THROUGH ALL EXISTING LEVELS OF COMPLETED FALSE BRANCH SUBTREES*/ 7  648
                         GO TO DECNAL;                              7  648
                      FBLPT=ADDR(FALSES(NAL));                      6  649
                      GO TO FLEAF;                                  6  650
                      END;                                          6  651
                END;                                                4  652
          NAL=NAL+1;                                                2  653
             IF CDB(DOCS(NAL-1))='T' THEN                           3  654
             GO TO LTRUE;                                           3  655
/*PROCESS FALSE SUBTREE AT CURRENT LEVEL*/                          2  656
          ABLPT=ADDR(FALSES(NAL-1));                                2  656
          CALL REDOCS;                                              2  657
          DIR(FDSS(NAL-1))=SS(DOCS(NAL));                           2  658
          DIR(FDSS(NAL-1)+1)=NADS;                                  2  659
          GO TO BRANCH;                                             2  660
```

```
BRANCHR:      PROC;                                                    3  661
/* BRANCHR PROCEDURE */                                               3  662
/* BRANCHR DETERMINES WHICH RULES IN THE ACTIVE BRANCH LIST (ABL) */  3  662
/* BELONG IN THE TRUE BRANCH LIST (TBL) AND WHICH BELONG IN THE */    3  662
/* FALSE BRANCH LIST (FBL) BASED ON THEIR CONDITION ENTRY FOR */      3  662
/* CONDITION DOCS(NAL). */                                            3  662
/* SAVE NABL VALUE */                                                3  662
              NABLS=NABL;                                             3  662
/* INITIALIZE NTBL AND NFBL; ONE OVERWRITES NABL, HENCE NABLS*/       3  663
              NTBL,NFBL=0;                                            3  663
              ETELEMPT=ADDR(ET(DOCS(NAL),1));                         3  664
/* ADD EACH RULE IN ABL TO THE APPROPRIATE BRANCH LIST(S) */          4  665
                 DO I=1 TO NABLS;                                     4  665
                 K=ABL(I);                                            4  666
                    IF ETELEM(K)='T' THEN                             5  667
/* ADD RULE TO TBL */  DO;                                           6  668
                       NTBL=NTBL+1;                                   6  669
                       TBL(NTBL)=K;                                   6  670
                       END;                                           6  671
                    ELSE                                              5  672
                       IF ETELEM(K)='F' THEN                          6  672
/* ADD RULE TO FBL */     DO;                                        7  673
                          NFBL=NFBL+1;                                7  674
                          FBL(NFBL)=K;                                7  675
                          END;                                        7  676
                       ELSE                                           6  677
/* FOR DON'T CARE ENTRIES ADD THE RULE TO BOTH TBL AND FBL */         7  677
                          DO;                                         7  677
                          NTBL=NTBL+1;                                7  678
                          NFBL=NFBL+1;                                7  679
                          TBL(NTBL),FBL(NFBL)=K;                      7  680
                          END;                                        7  681
                 END;                                                 4  682
              END BRANCHR;                                            3  683
```

```
REDOCS:        PROC;                                                    3  684
/* REDOCS PROCEDURE */                                                  3  685
/* REDOCS REORDERS THE DYNAMIC ORDER OF CONDITION STUBS (DOCS) */       3  685
/* BY INTERCHANGING DOCS(NAL) WITH DOCS(DOCSS) WHERE DOCS(DOCSS) */     3  685
/* IS THE UNTESTED CONDITION WHICH HAS THE FEWEST DON'T CARE */         3  685
/* ENTRIES IN THE RULES IN THE ACTIVE BRANCH LIST (ABL) */              3  685
               DCL (NDCE,SNDCE,DOCSS) BIN FIXED;                        3  685
/* IF THERE IS ONLY ONE UNTESTED CONDITION THEN NO INTERCHANGE */       4  686
/* IS NECESSARY */                                                      4  686
                  IF NAL>=NC THEN                                       4  686
                  RETURN;                                               4  687
/* DETERMINE SNDCE AND THE CORRESPONDING DOCSS */                       4  688
/* CALCULATE NDCE FOR EACH UNTESTED CONDITION */                        4  688
                  DO I=NAL TO NC;                                       4  688
                  ETELEMPT=ADDR(ET(DOCS(I),1));                         4  689
                  NDCE=0;                                               4  690
/* CALCULATE NDCE FOR CONDITION DOCS(I) */                              5  691
                     DO K=1 TO NABL;                                    5  691
                        IF ETELEM(ABL(K))='-' THEN                      6  692
                        NDCE=NDCE+1;                                    6  693
                     END;                                               5  694
                     IF I=NAL THEN                                      5  695
/* ASSIGN INITIAL VALUES TO SNDCE AND DOCSS */                          6  696
                        DO;                                             6  696
                        SNDCE=NDCE;                                     6  697
                        DOCSS=NAL;                                      6  698
                        GO TO ZERODCE;                                  6  699
                        END;                                            6  700
                     ELSE                                               5  701
                        IF NDCE<SNDCE THEN                              6  701
/* ASSIGN NEW VALUES TO SNDCE AND DOCSS */                              7  702
                           DO;                                          7  702
                           SNDCE=NDCE;                                  7  703
                           DOCSS=I;                                     7  704
ZERODCE:                      IF SNDCE=0 THEN                          8  705
/* NO CONDITION MAY HAVE LESS THAN ZERO DON'T CARE ENTRIES */           8  706
                              GO TO INTCHG;                             8  706
                           END;                                         7  707
                  END;                                                  4  708
INTCHG:           IF NAL¬=DOCSS THEN                                    4  709
/* INTERCHANGE DOCS(NAL) AND DOCS(DOCSS) */                             5  710
                     DO;                                                5  710
                     I=DOCS(NAL);                                       5  711
                     DOCS(NAL)=DOCS(DOCSS);                             5  712
                     DOCS(DOCSS)=I;                                     5  713
                     END;                                               5  714
               END REDOCS;                                              3  715
```

```
LEAFREC:      PROC;                                                    3  716
/* LEAFREC PROCEDURE */                                               4  717
/* LEAFREC DETERMINES IF THE ACTIVE BRANCH IS A LEAF OF THE */        4  717
/* DECISION TREE AND SETS THE LEAF SWITCH. */                         4  717
/* IT IS ASSUMED THAT AT LEAST ONE RULE REMAINS IN THE ABL. */        4  717
/* IF ALL CONDITIONS HAVE BEEN TESTED THEN THIS IS NECESSARILY */     4  717
/* A LEAF */     IF NAL>=NC THEN                                      4  717
                    DO;                                             5  718
                    NPIV=1;                                         5  719
                    GO TO TESTAMB;                                  5  720
                    END;                                            5  721
/* DETERMINE IF ANY OF THE RULES IN ABL CONTAIN DON'T CARE ENTRIES */ 4  722
/* FOR EACH UNTESTED CONDITION */                                     4  722
                 DO NPIV=1 TO NABL;                                 4  722
                 K=ABL(NPIV);                                       4  723
                    DO J=NAL+1 TO NC;                               5  724
                       IF ET(DOCS(J),K)¬='-' THEN                   6  725
                       GO TO NXTRULE;                               6  726
                    END;                                            5  727
/* RULE K CONTAINS DON'T CARE ENTRIES FOR EACH UNTESTED CONDITION */  5  728
                    IF NPIV>1 THEN                                  5  728
                       DO;                                          5  729
                       ABL(1)=ABL(NPIV);                            6  730
                       NABL=1;                                      6  731
                       END;                                         6  732
                 GO TO TESTAMB;                                     4  733
NXTRULE:         END;                                               4  734
/* NO RULE CONTAINS DON'T CARE ENTRIES FOR EACH UNTESTED CONDITION */ 3  735
SLEAF:        LEAF='0';                                             3  735
              RETURN;                                               3  736
TESTAMB:      LEAF='1';                                             3  737
/* THE RULES IN THE ACTIVE BRANCH LIST ARE AT A LEAF OF THE */       3  738
/* DECISION TREE */                                                  3  738
              END LEAFREC;                                          3  738
           END PARSE;                                               2  739
```

```
EXOUT:     PROC;                                                          2  740
/* EXECUTION CODE          */                                            2  741
           DCL                        EXCODE(22) CHAR(80) STATIC INIT(    2  741
    ' #    :   ALLOCATE #LNK;',                                          2  741
    '          #LNK=#CPT+1;',                                            2  741
    ' #    :',                                                           2  741
    ' #GOTO:   #CPT=#DIR(#DIR(#CPT))+1;',                                2  741
    '          GO TO #LAB(#DIR(#CPT-1));',                               2  741
    ' #    :   #CPT=#LNK;',                                              2  741
    '          FREE #LNK;',                                              2  741
    ' #NEXT:   #CPT=#CPT+1;',                                            2  741
    '          GO TO #LAB(#DIR(#CPT-1));',                               2  741
    ' #FALSE:  #CPT=#CPT+2;',                                            2  741
    ' #TRUE:   #NXT=#DIR(#CPT);',                                        2  741
    '          #CPT=#DIR(#CPT+1);',                                      2  741
    '          GO TO #LAB(#NXT);',                                       2  741
    ' #START:  IF #SWT=''1'' THEN DO; #SWT=''0'';',                      2  741
    '          #LAB(    )=#    ; #LAB(    )=#    ; #LAB(    )=#    ; #LAB(    )=  2  741
#    ;',                                                                 2  741
    '          #CPT=1; GO TO #GOTO;',                                    2  741
    '          DCL #SWT CHAR(1) STATIC INIT(''1''), #LAB(    )STATIC LABEL  2  741
(',',      #START,#NEXT,#FALSE,#TRUE);',                                 2  741
    '          DCL (#CPT,#NXT) BIN FIXED STATIC,#LNK CTL BIN FIXED,',    2  741
    '          #DIR(    ) BIN FIXED STATIC INIT(',                       2  741
    '             ,   ,   ,   ,   ,   ,   ,   ,   ,   ,   ,   ,   ,   ,  2  741
 ,',                                                                     2  741
    ' #    :   ;');                                                      2  741
           DCL EXREC CHAR(80) BASED(EXRECPT);                            2  742
           DCL EXREC2 CHAR(80) STATIC;                                   2  743
           DCL 1 LA DEF EXREC2,                                          2  744
           2 FA CHAR(9),                                                 2  744
           2 STMT(4),                                                    2  744
           3 FB CHAR(5),                                                 2  744
           3 SUB PIC'ZZ9',                                               2  744
           3 FC CHAR(3),                                                 2  744
           3 LAB PIC'999',                                               2  744
           3 FD CHAR(2),                                                 2  744
           2 FE PIC'ZZZZZZ9';                                            2  744
           DCL 1 DI DEF EXREC2,                                          2  745
           2 FA CHAR(9),                                                 2  745
           2 INIT(15),                                                   2  745
           3 SUB PIC'ZZ9',                                               2  745
           3 FB CHAR(1),                                                 2  745
           2 FC CHAR(11);                                                2  745
           DCL LABPICP PIC'999';                                         2  746
           DCL SUBPIC2 PIC'99';                                          2  747
           DCL CSX(4) BIN FIXED STATIC INIT(1,3,6,8);                    2  748
           DCL OCX(5) BIN FIXED STATIC INIT(8,15,16,21,22);              2  749
           DCL 1 LABPIC BASED(EXRECPT),                                  2  750
           2 FA CHAR(2),                                                 2  750
           2 SUB PIC'999',                                               2  750
           2 FB CHAR(68),                                                2  750
           2 SQ PIC'ZZZZZZ9';                                            2  750
           DCL SUBPIC PIC'ZZ9';                                          2  751
           I=IACSS(103);                                                 2  752
/*MERELY GUARANTEEING THAT CSS(2) HAS BEEN ASSIGNED*/                    2  753
           SUBPIC=NADS-1;                                                2  753
           SUBSTR(EXCODE(20),15,3)=SUBPIC;                               2  754
```

```
            LABPICP=IACSS(100B);                                    2  755
            SUBSTR(EXCODE(22),3,3)=LABPICP;                         2  756
            SUBPIC=NAS-1;                                           2  757
            SUBSTR(EXCODE(17),50,3)=SUBPIC;                         2  758
/* PUNCH THE CONTROL SECTIONS NEEDED */                            3  759
               DO I=1 TO 3;                                         3  759
                  IF CSS(I)¬=0 THEN                                 4  760
                     DO J=CSX(I)TO CSX(I+1)-1;                      5  761
                     EXRECPT=ADDR(EXCODE(J));                       5  762
                        IF J=CSX(I)THEN                             6  763
                        LABPIC.SUB=CSS(I);                          6  764
                     SEQNO,LABPIC.SQ=SEQNO+1;                       5  765
                     WRITE FILE(PUNCH)FROM(EXREC);                  5  766
                     END;                                           5  767
               END;                                                 3  768
/* PUNCH OTHER CONTROL STATEMENTS */                               3  769
               DO I=OCX(1)TO OCX(2)-1;                              3  769
               EXRECPT=ADDR(EXCODE(I));                             3  770
               SEQNO,LABPIC.SQ=SEQNO+1;                             3  771
               WRITE FILE(PUNCH)FROM(EXREC);                        3  772
               END;                                                 3  773
/* LABEL ASSIGNMENT STATEMENTS */                                  2  774
            EXREC2=EXCODE(OCX(2));                                  2  774
               DO I=1 TO NAS BY 4;                                  3  775
                  DO J=1 TO 4;                                      4  776
                  LA.STMT.SUB(J),LA.STMT.LAB(J)=I+J-1;              4  777
                  END;                                              4  778
                  IF I+4>NAS THEN                                   4  779
                  SUBSTR(EXREC2,10+(NAS-I)*16)='END;';              4  780
               LA.FE,SEQNO=SEQNO+1;                                 3  781
               WRITE FILE(PUNCH)FROM(EXREC2);                       3  782
               END;                                                 3  783
               DO I=OCX(3)TO OCX(4)-1;                              3  784
               EXRECPT=ADDR(EXCODE(I));                             3  785
               SEQNO,LABPIC.SQ=SEQNO+1;                             3  786
               WRITE FILE(PUNCH)FROM(EXREC);                        3  787
               END;                                                 3  788
/* DIRECTORY INITIALIZATION STATEMENT */                           2  789
            EXREC2=EXCODE(OCX(4));                                  2  789
               DO I=1 TO NADS-1 BY 15;                              3  790
                  DO J=1 TO 15;                                     4  791
                  DI.INIT.SUB(J)=DIR(I+J-1);                        4  792
                  END;                                              4  793
                  IF I+15>=NADS THEN                                4  794
                  SUBSTR(EXREC2,9+(NADS-I)*4)=');';                 4  795
               LA.FE,SEQNO=SEQNO+1;                                 3  796
               WRITE FILE(PUNCH)FROM(EXREC2);                       3  797
               END;                                                 3  798
            EXRECPT=ADDR(EXCODE(OCX(5)));                           2  799
            SEQNO,LABPIC.SQ=SEQNO+1;                                2  800
            WRITE FILE(PUNCH)FROM(EXREC);                           2  801
            END EXOUT;                                              2  802
         END DECTAL0;                                               1  803
```