PRELIMINARY APPROACH
TO
TABULAR PROGRAMMING

Earl O. Althoff
Eastman Kodak
Rochester, N.Y.

October, 1960

POINTS ABOUT PRELIMINARY APPROACH TO TABULAR PROGRAMMING

1. For each element used, prepare a 15-digit title to use in the English text and a four-digit abbreviation to use in formulae. The four-digit abbreviation either starts with a letter or is numbered sequentially 0001, 0002, 0003, ...

2. Do not strain to over-abbreviate. For example, CT01, CT02, ... can be used to stand for control totals of various types. It is usually best to give mnemonic abbreviations only for the hundred or less most used elements.

3. Data sets can be listed on a data element sheet if desired. For example, the data set "Target Date" abbreviated TRGT consists of the data elements:

   "Target Month" abbreviated TMON
   "Target Day" abbreviated TDAY
   "Target Year" abbreviated TYR

   In the above, four entries are made, one for each data element and one for the data set.

4. The definition should be clear and unambiguous, but above all must be complete. Differentiate clearly between similar data elements.

5. Prepare a data file for each set of data (not going directly to a report). Do not consider the machine in your preparation. As an example consider a tape with records of Type A followed by several records of Type B; prepare two data files, A and B, since having these on the same tape is pure machine method.

6. For each data set or element listed, record a reference to set number and page number of the data element sheets. Thus, 03-01 refers to data element set 03, page 01. Record both the title and the abbreviation. Record the length for that file. A given element can require four digits on one data file and six on another.

7. Give each data file a letter designation A, B, C, .., record whether input or output. In the case of an updated data file, assign two letters, say A for input and B for output.

8. Obtain a Data Processing Spacing Chart for all report lines (messages as well as fancy reports). Label each report A, B, etc. Use a second letter for each different type of line. Thus, Report A may have lines AA, AB, AC, ...

9. Tables must give all logic except how to start and how to stop. All the statements which follow must be accomplished completely — no exceptions can be permitted.

10. The table is divided into conditions and actions. On the left, one gives the English statement of the condition or action, and on the right, one records the precise formula fully and completely. Thus, on the right;

    A-STAT = B-STAT clearly shows that the condition is <u>true</u> if and only if the STAT element of data File A equals the STAT element of data File B.

    C-0001 > 0 shows the condition is true only if the data element 0001 of data File C is greater than the constant 0.

11. The formula for a condition can include any connectors desired to complete "a single condition". Examples are:

    F-TAX = 10 or 15
    F-TAX = 10 and G-TAX = 15

12. The actions can be varied also. In general, one records data movement or arithmetic actions first, then all data file advance actions, then all table transfer actions.

13. Typical data movement actions are D → E (ASG#, PROG, and TTO1) meaning move from data File D to data File E the data elements ASG#, PROG, and TTO1. In case of one move, D-ASG# → E-ASG#. Others are D-PDHR add to E-YRHR, etc.

14. When data is posted to report lines, increment is used as: D-ASG# → AB14 meaning post data element ASG# of data File D to position 14 (right-hand increment) of line B of Report A.

15. Another action may be to do an action or actions from other tables. Thus, Action 2 of Table 01-A5 can simply be "Do actions 3 and 6 of Table 03-B7.

16. Another action may follow the actions for a data rule from another table in its entirety; if so, simply transfer to Rule XX of Table XX-XX.

17. The advance data files actions are abbreviated GIV X for input and TAK X for output. In some cases posting a data element to a control total is included as: C-AMT add to TOT1; TAK C. When an advance action is given, the next action calling on that file from <u>any</u> table will be from the next record.

18. Tables are numbered NN-XN where NN denotes the project area; NN runs from 01, 02, ,., X is a letter denoting a sub-project and runs from A to Z, while the rightmost N is 1 to 9 and denotes table within sub-project. The last action for any data rule is always a transfer to some table. (Do not transfer to a rule within a table — leave this to the programmer.)

19. On any given table, possible entries opposite condition are Y, N, and -. Y = Yes, N = No, and - means "does not apply".

    The matrix | Y | Y | N |
               | Y | N | N | would indicate an analyst omission since the combination | N |
                                                                                       | Y |
    is not specified. One must specify enough data rules to account for every combination of the conditions, whether <u>possible</u> or <u>impossible</u> (is it really impossible???)

    The - is used primarily in two cases:

    A. If condition 1 is A-STAT = 0 and condition 2 is A-STAT = 1, then a | Y | - | entry
                                                                          | - | Y |
    would show that, if A-STAT = 0, we don't need to test for A-STAT = 1 and vice versa.

    B. The - may be used to indicate a plain transfer to another table, when the only alternative would be to over-run the 6 X 10 matrix. Example:

Condition 1	Y	Y	Y	Y	Y	Y	Y	Y	N
Condition 2	Y	Y	Y	N	Y	N	N	N	-
Condition 3	Y	Y	N	Y	N	Y	N	N	-
Condition 4	Y	N	Y	Y	N	N	Y	N	-

The data rule on the right simply transfers to another table where the NO's for Condition 1 are spelled out.

20. In summary, the preliminary approach is designed to obtain from a job analyst actions to follow for every combination of conditions. The conditions and actions are not to be vague — but must be 100 percent precise to every data element involved. There is no thought given in the preliminary approach to automating any of the steps: tables → programs. Only a person with two - three years active programming and computer systems experience can prepare tables containing many subtle traps which develop only in automatic E.D.P. systems; for the next year or so, it is expected that these people will return expanded tables (with these subtle points included) to the job analyst and will, in addition, write programs in KodaKoder.

EOAlthoff:rds
October 11, 1960

DATA PROCESSING SERVICE

ELEMENTS DEFINITION

SET # 01 PAGE #01

Job No.

Name: Earl O. Althoff

Project: D.P.S. Billing

15-DIGIT TITLE One Letter ASGN 4-DIGIT ABBREV. ASGL

DEFINITION: A letter code used to differentiate between several types of perpetual assignments. Refer to the back of a D.P.S. Time-Reporting Sheet for full details.

15-DIGIT TITLE Assignment No. 4-DIGIT ABBREV. ASG#

DEFINITION: A four-digit number given in sequence to non-perpetual assignments as they occur. The number has no structure of any sort.

15-DIGIT TITLE Billing Number 4-DIGIT ABBREV. BIL#

DEFINITION: A five-digit number assigned by the D.P.S. Accountant to each account or sub-account which D.P.S. bills. It is structured as desired to yield a meaningful report order.

15-DIGIT TITLE PROG-SYSTEM NO. (DATA SET) 4-DIGIT ABBREV. UJ#

DEFINITION: A uniform job number which serves a variety of purposes. It is organized primarily r computer run and program within computer run. (See Chapter 27 - Self-Teach.) Consists of elements MFC, RUN#, and PRGL.

15-DIGIT TITLE Project Title 4-DIGIT ABBREV. TITL

DEFINITION A 45-character title given to each project having a four-digit assignment number.

15-DIGIT TITLE Project Type Code 4-DIGIT ABBREV. TYPE

DEFINITION: A two-character code enabling us to group a project by new programs (N), changes (C), or revision (R). The units position is 1 for a business project, 6 for a program research project.

15-DIGIT TITLE Major Fctn. Code 4-DIGIT ABBREV. MFC

DEFINITION: A two-digit code used by D.P.S. to roughly distinguish between basic major project functions such as Merchandise Billing, Paper Finishing Scheduling, etc. It is the first two digits of Prog-System No.

[illegible]-DIGIT TITLE Target Date 4-DIGIT ABBREV. TRGT

DEFINITION: The date by which an assignment should be completed to the point that production results are obtainable. Six digits as 011560.

DATA PROCESSING SERVICE

ELEMENTS DEFINITION

SET # 01 PAGE # 02

Job No.:

Name: Earl Althoff

Project: D.P.S. Billing

15-DIGIT TITLE Programmer 4-DIGIT ABBREV. PROG

DEFINITION: An official ten-digit name assigned to each individual of the Programming and Methods Staff

15-DIGIT TITLE Registration # 4-DIGIT ABBREV. REG#

DEFINITION: A six-digit number given to each employee of Kodak Rochester. The first three digits indicate department and the last three are sequentially given by various rules.

15-DIGIT TITLE Prog-Syst.Descr. 4-DIGIT ABBREV. DESC

DEFINITION: Refers to a 29-digit alphanumeric title or description given to each specific program or computer systems sub-assignment.

15-DIGIT TITLE EST. Man Months 4-DIGIT ABBREV. ESTM

DEFINITION: Refers to a time estimate given for each program in an assignment. The time is ven in four digits (one decimal place).

15-DIGIT TITLE Due Date for V 4-DIGIT ABBREV. DUEV

DEFINITION: A date given for each program to be ready for system volume testing. Six digits as 011560 or 12B161.

15-DIGIT TITLE Department 4-DIGIT ABBREV. DEPT

DEFINITION: A four-character alphanumeric abbreviation of the department a programmer belongs to. Examples are DPS, MSDD, A&O, DC.

15-DIGIT TITLE Computer Run # 4-DIGIT ABBREV. RUN#

DEFINITION: The third and fourth digit of Prog-System No.. Delineates the programs consititutin a scheduled computer run.

[illegible]-DIGIT TITLE Program Letter 4-DIGIT ABBREV. PRGL

DEFINITION: The fifth digit of Prog-System No. Letters from A to Z are given to programs of a given computer run.

DATA PROCESSING SERVICE

DATA FILE LAYOUT

Job No.:

Project: D.P.S. Billing

FILE DESCRIPTION Assignment Master

Data File: A in B out

Name: Earl Althoff

	TITLE	REF.	ABBREV.	LENGTH	For Programmer Use Only REC #	INCR.
1.	Assignment No.	01-01	ASG#	4		
2.	Billing Number	01-01	BIL#	5		
3.	Proj. Ldr. Name	01-05	PLDR	10		
4.	Project Title	01-01	TITL	45		
5.	Proj. Type Code	01-01	TYPE	2		
6.	Major FCTN Code	01-01	MFC	2		
7.	Target Date	01-01	TRGT	6		
8.	Bill-out Code:	01-01	BILC	1		
9.	Completion Code	01-03	CMPL	1		
10.						
11.						
12.						
13.						
14.						
15.						
16.						
17.						
18.						
19.						
20.						
21.						
22.						
23.						
24.						
25.						
26.						
27.						
28.						
29.						
30.						
31.						
33.						
34.						
35.						

DATA PROCESSING SERVICE

DATA FILE LAYOUT

Job No.:

Project: D.P.S. Billing

FILE DESCRIPTION Current Time Records

Data File: C in

Name: Earl Althoff

	TITLE	REF.	ABBREV.	LENGTH	For Programmer Use Only REC #	INCR.
1.	Assignment No.	01-01	ASG#	4		
2.	Prog-System No.	01-01	UJ#	5		
3.	Programmer	01-02	PROG	10		
4.	Department	01-02	DEPT	4		
5.	Progress Code	01-03	STAT	1		
6.	Est. Date for V	01-03	ESTV	6		
7.	Hrs. This Period	01-03	HRTP	4		
8.	CPU MIN V-Test	01-05	VTST	4		
9.	K-10S This Pd.	01-04	K10P	2		
10.	K-20S This Pd.	01-04	K20P	2		
11.	K-30S This Pd.	01-03	K30P	2		
12.	K-40S This Pd.	01-04	K40P	2		
13.	K-50S This Pd.	01-04	K50P	2		
14.	ASGL can be M and N only	01-01	ASGL	1		
15.						
16.						
17.						
18.						
19.						
20.						
21.						
22.						
23.						
24.						
25.						
26.						
27.						
28.						
29.						
30.						
31.						
33.						
34.						
35.						

DATA PROCESSING SERVICE

DATA FILE LAYOUT

Job No.:

Project: D.P.S. Billing

FILE DESCRIPTION Program System Master

Data File: K in L out

Name: Earl Althoff

	TITLE	REF.	ABBREV.	LENGTH	For Programmer Use Only REC #	INCR.
1.	Assignment No.	01-01	ASG#	4		
2.	Prog-System No.	01-01	UJ#	5		
3.	Programmer	01-02	PROG	10		
4.	Prog-System Description	01-02	DESC	29		
5.	Est. Man Months	01-02	ESTM	4		
6.	Due Date for V	01-02	DUEV	6		
7.	Department	01-02	DEPT	4		
8.	Progress Code	01-03	STAT	1		
9.	Est. Date for V	01-03	ESTV	6		
10.	HRS to Date	01-03	HRTD	5		
11.	Bill-out Code	01-03	BILC	1		
12.	V-TEST To Date	01-05	VTTD	6		
13.	K-10S To Date	01-04	K10S	3		
14.	K-20S To Date	01-05	K20S	3		
	K-30S to Date	01-04	K30S	3		
16.	K-40S To Date	01-04	K40S	3		
17.	K-50S To Date	01-04	K50S	3		
18.	ASGL will be M and N only	01-01	ASGL	1		
19.						
20.						
21.						
22.						
23.						
24.						
25.						
26.						
27.						
28.						
29.						
30.						
31.						
33.						
34.						
35.						

DATA PROCESSING SERVICE
TABLE LAYOUT

Job. No.:
Name: Earl Althoff
Project: D.P.S. Billing
Table No: 01-A1

TITLE Update Assignment Master

CONDITIONS	FREQUENCY	0	10	1	10	0	200					CONDITION ABBREVIATIONS
	RULE NO.	1	2	3	4	5	6	7	8	9	10	
1. Is there a new assignment master?		Y	Y	N	N	N	N					D-ASG# < A-ASG#
2. Is there a change to the assignment master?		-	-	Y	Y	Y	N					D-ASG# = A-ASG#
3. Is the change record a deletion?		Y	N	Y	-	-	-					D-CMPL = 2
4. Is the change record a completion notification?		-	-	-	Y	-	-					D-CMPL = 1
5. Are we posting changes to a completed assignment?		-	-	-	-	Y	-					D-CMPL = A
6.												
ACTIONS												
1. Move entire master record to serve as base.					Y		Y					A → B
2. Post change assignment no. to updated master.			Y	Y		Y						D-ASG# → B-ASG#
3. Post corresponding parts of change to master.			Y			Y						BIL#, PLDR, TITL, TYPE, MFC, TRGT, BILC D → B
4. Set-up and write delete error message.		Y										D-ASG# → AA-39 Write AA
5. Post completion code to master			Y		Y	Y						D-CMPL (Numeric portion) → B-CMPL
6. Set-up and write deleting message				Y								D-ASG# → AB-18 Write AB
7. Advance controls to next change		Y	Y	Y	Y	Y						GIV D
8. Advance controls to next input master				Y	Y	Y	Y					GIV A
9. Transfer to Table 01-A1		Y										TR 01-A1
10. Transfer to Table 01-A3				Y								TR 01-A3
Transfer to Table 01-A2			Y		Y	Y						TR 01-A2
12. Transfer to Table 01-A4							Y					TR 01-A4
13.												

DATA PROCESSING SERVICE
TABLE LAYOUT

Job. No.:
Name: Earl Althoff
Project: D.P.S. Billing
Table No: 01-A2

TITLE Second Change Check

CONDITIONS	FREQUENCY	0	0	0	10							CONDITION ABBREVIATIO
	RULE NO.	1	2	3	4	5	6	7	8	9	10	
1. Is there a change to the updated master?		Y	Y	Y	N							D-ASG# = B-ASG#
2. Is this change a deletion?		Y	-	-	-							D-CMPL = 2
3. Is this change a completion notification?		-	Y	-	-							D-CMPL = 1
4. Are we posting changes to a completed assignment?		-	-	Y	-							D-CMPL = A
5.												
6.												
ACTIONS												
1. Follow action of Rule 3 Table 01-A1		Y										TR Rule 3 - 01A1
2. Post completion code to master.			Y									D-CMPL (numeric part) → B-CMPL
3. Advance controls to next change.			Y	Y								GIV D
4. Follow actions 3 and 5 of Table 01-A1				Y								Duplicate 3 and 5 of 01-A1
5. Transfer to Table 01-A2			Y	Y								TR 01-A2
6. Transfer to Table 01-A4					Y							TR 01-A4
7.												
8.												
9.												
10.												
11.												
12.												
13.												

DATA PROCESSING SERVICE
TABLE LAYOUT

Job. No.:
Name: Earl Althoff
Project: D.P.S. Billing
Table No: 01-B1

TITLE Update Prog-System Master

CONDITIONS	FREQUENCY	0	100	10	1	10	1000	300				CONDITION ABBREVIATIONS
	RULE NO.	1	2	3	4	5	6	7	8	9	10	
1. Is there a new Prog-System Master?		Y	Y	Y	N	N	N	N				E-FLD1 < K-FLD1
2. Is the new master for this assignment?		Y	Y	N	-	-	-	-				E-ASG# = B-ASG#
3. Is this a delete?		Y	N	-	-	-	-	-				E-DPGM = 1
4. Is the next master for this assignment?		-	-	-	Y	Y	Y	N				K-ASG# = B-ASG#
5. Is there a change for the master?		-	-	-	Y	Y	N	N				E-FLD1 = K-FLD1
6. Is the change a delete?		-	-	-	Y	N	N	N				E-DPGM = 1
ACTIONS												
1. Move entire master to serve as base.					Y	Y	Y					K → L
2. Post corresponding change fields to master.			Y			Y						E→L (ASG#, UJ#, PROG, DESC, ESTM, DUEV, DEPT, BILC
3. Post start-up constants to master.			Y									Blanks to STAT, ESTV Zero to HRTD, VTTD, K10S, K20S, K30S, K40S, K50S
4. Set-up and write delete error message.		Y										E-ASG# →AF39 E-UJ# → AF48 Write AF
5. Set-up and write deleting message.					Y							E-ASG# →AG19 E-UJ# → AG29
6. Advance control to next change record.		Y	Y		Y	Y						GIV E
7. Advance control to next master record.					Y	Y	Y					GIV K
8. Transfer to this Table 01-B1		Y										TR 01-B1
9. TRansfer to Table 01-B3			Y			Y						TR 01-B3
10. TRansfer to Table 01-C1				Y				Y				TR 01-C1
TRansfer to Table 01-B2					Y							TR 01-B2
12. TRansfer to Table 01-B4							Y					TR 01-B4
13.												

DATA PROCESSING SERVICE
TABLE LAYOUT

Job. No.:
Name: Earl Althoff
Project: D.P.S. Billing
Table No: 01-B2

TITLE Finish deleting on Prog-Syst. Delete.

CONDITIONS	FREQUENCY RULE NO.	1	2	3	4	5	6	7	8	9	10	CONDITION ABBREVIATION
1. Are there any more Prog-Syst. changes?		Y	N	N								E-FLD1 = L-FLD-1
2. Is there a current time record?		-	Y	N								C-FLD1 = L-FLD1
3.												
4.												
5.												
6.												
ACTIONS												
1. Set-up and write message AC.		Y										E-UJ#->AC30 Write AC
2. Set corresponding to delete message.			Y									Action 6 of table 01-A3
3. Set rest of corresponding and write delete message.			Y									Action 7 of table 01-A3
4. Advance controls to next Prog-Syst. Change.		Y										GIV E
5. Advance Controls to next time record.			Y									GIV K
6. TRansfer to Table 01-B2		Y										TR 01-B2
7. TRansfer to Table 01-B1			Y	Y								TR 01-B1
8.												
9.												
10.												
11.												
12.												
13.												

DATA PROCESSING SERVICE

ELEMENTS DEFINITION

SET # PAGE #

Job No.:

Name:

Project:

15-DIGIT TITLE 4-DIGIT ABBREV.

DEFINITION:

15-DIGIT TITLE 4-DIGIT ABBREV.

DEFINITION:

15-DIGIT TITLE 4-DIGIT ABBREV.

DEFINITION:

15-DIGIT TITLE 4-DIGIT ABBREV.

DEFINITION:

15-DIGIT TITLE 4-DIGIT ABBREV.

DEFINITION:

15-DIGIT TITLE 4-DIGIT ABBREV.

DEFINITION:

15-DIGIT TITLE 4-DIGIT ABBREV.

DEFINITION:

15-DIGIT TITLE 4-DIGIT ABBREV.

DEFINITION:

D A T A P R O C E S S I N G S E R V I C E

DATA FILE LAYOUT

Job No.: _____

Project: _____

FILE DESCRIPTION _____

Data File: _____

Name: _____

	TITLE	REF.	ABBREV.	LENGTH	For Programmer Use Only REC #	INCR.
1.						
2.						
3.						
4.						
5.						
6.						
7.						
8.						
9.						
10.						
11.						
12.						
13.						
[illegible].						
15.						
16.						
17.						
18.						
19.						
20.						
21.						
22.						
23.						
24.						
25.						
26.						
27.						
28.						
29.						
30.						
31.						
[illegible].						
33.						
34.						
35.						

WORK PROBLEMS

Limited Entry Tables

The following rules should be utilized in considering the work problems for limited entry tables. The six common table elements are used in displaying the structure of the sample table and the verbal description of the permissable entries for each element.

Table Header							
Condition Stub			Condition Entry				
Action Stub			Action Entry				
Entry Header							

TABLE HEADER - The name or number of the Table; used for identification and reference purposes. Example: RE-ORDER DETERMINATION, DEDUCTIONS, 004, R26.

CONDITION STUB - A relational expression which, through evaluation, can be determined to be true or false. The following relational operators are permitted:

$$> \quad \geq$$
$$< \quad \leq$$
$$= \quad \neq$$

Logical connectives, such as "AND", "OR", "NOT" are not permitted.

Below is the recommended general form for the relational expression

[Element - 1] [Relational Operator] [Element - 2]

where Element -1 and Element -2 may take one of the following forms:

START ?, MORE RECORDS ?.

Data Name

Literal (enclosed in quotes - "_____", except for numbers)

Arithmetic Expression

Examples of relational expressions: Quantity-on-order Quantity-on-hand, FICA < 144.00, SENTINEL = 99999, MARITAL STATUS ≠ "SINGLE".

CONDITION ENTRY - There are only four possibilities permitted in this portion of the table:

Y (for True)

N (for False)

BLANK OR — (not pertinent)

ALL OTHERS OR A/O (for all others)

All alternatives must be covered, the 'all others' entry should be the last entry (farthest to the right, since columns are assumed to be considered from left to right until a set of conditions is met).

ACTION STUB - The actions to be executed are displayed in this section. The following operators are available for use in the sample problem:

SET... =	READ
OPEN	WRITE
CLOSE	STOP
DO	

Operands may be either data names, literals, or arithmetic expressions (to be used with the SET = operator). Examples: OPEN DETAIL FILE, SET NET = GROSS * .85, READ MASTER RECORD.

ACTION ENTRY - Only two possibilities exist for the action entry:

X (for execute)

blank (for do <u>not</u> execute)

Assume that the actions are executed in the order written for a particular column (if more than a single action exists for a particular condition or set of conditions).

ENTRY HEADER - For the sample problems the entry header is used for specification of the next table. This portion of the table can be considered an extention of the action stub and action entry. At the bottom of the action stub the - GO TO table-name - appears; an "X" in the proper row will indicate which columns are affected. If more than one "next table" is appropriate, more than one "go to" must appear (see example below).

EXAMPLE:

TABLE: FICA CALCULATION					
YTD GROSS ≤ 4800.00	Y	Y	N	Y	A/O
SALARY > 0.00	Y	N	Y	Y	
SEX = "MALE"	Y	—	Y	N	
SET FICA = .04 * SALARY	X				
SET FICA = .02 * SALARY			X	X	
SET FICA = 0.00		X			X
SET YTD FICA = YTD FICA					
+ FICA					X
GO TO TABLE 2	X	X	X	X	
GO TO TABLE 6					X

COLUMN 1 -
IF YTD GROSS ≤ 4800.00 AND SALARY > 0.00
AND SEX = "MALE" THEN SET FICA = .04 * SALARY,
GO TO TABLE 2.

COLUMN 2 -
IF YTD GROSS ≤ 4800.00 AND SALARY NOT > 0.00
THEN SET FICA = 0.00, GO TO TABLE 2.

.....

COLUMN 5 -
OTHERWISE SET YTD FICA = YTD FICA + FICA,
GO TO TABLE 6.

Problem 1 - Selected Stockholders Report

Prepare a limited entry decision table given the following information and rules.

From a file of stockholder records we wish to extract the records of stockholders other than individuals and the records of individuals who hold more than 100,000 shares. With this information produce a listing of the common shares held by these stockholders, the name of each stockholder, the type of stockholder (decoded) and the total number of stockholders and shares owned for this report.

Given - For each stockholder we have:

1. Stockholder name
2. Stockholder type (individual - 01, trust - 02, bank - 03, broker -04)
3. Number of shares owned.

Obtain - List for each selected stockholder:

1. Stockholder name
2. Stockholder type (decoded)
3. Number of shares owned.
4. A summary of the total number of stockholders and total number of shares for this report.

Problem 1 Solution

Type of Stockholder = 01	Y					
Type of Stockholder = 02		Y				
Type of Stockholder = 03			Y			
Type of Stockholder = 04				Y		
Number of shares>100,000	Y					
More stockholders?					N	
Write stockholder name	X	X	X	X		
Write "Individual"	X					
Write "Trust"		X				
Write "Bank"			X			
Write "Broker"				X		
Write number of shares	X	X	X	X		
SET Stockholder count = stockholder count + 1	X	X	X	X		
SET Total shares = Total shares + number of shares	X	X	X	X		
Write stockholder count					X	
Write total shares					X	

Problem 2 - Posting Operation

Prepare a limited entry decision table given the following information and rules.

Input - Two files

A. A master file is in sequence by identification number (I.D.) Each I.D. number has an associated on hand (O.H.) amount.

B. A detail transaction file is in sequence by identification number (I.D.). Each I.D. number has associated types of transactions - receipt, issue, recount-and their amounts, sequenced respectively. There can be multiple receipts and issues, but only a single recount.

Output - One file

A. The new master should contain I.D. number and on hand amount.

Actions:

1. If there is no activity write the new master from the old master.

2. Provide for computing the new on hand amount. (receipt = +, issues = -, recount = replace)

3. Provide for start and end of job.

4. Provide for an error routine in the case of a transaction occuring for which there is no master.

5. There are no additions or deletions to the master file.

Solution to Problem 2

TABLE : POSTING OPERATION

	1	2	3	4	5	6	7	8	9
START ?	Y	N	N	N	N	N	N	Y	N
OM I.D. < TransI.D.		Y							
OM I.D. = Trans I.D.			Y	Y	Y		Y		
OM I.D. > Trans I.D.						Y			
Type of trans = receipt			Y						N
Type of trans = issue				Y					N
Type of trans = recount					Y				N
End of trans file							Y	Y	
End of master file	N	N	N	N	N	N	Y	Y	
Open all files	X								
Close all files							X		
Set OH = OH + Trans Am't			X						
Set OH = OH - Trans Am't				X					
Set OH = Trans Am't					X				
Do Error Routine						X		X	X
Write NM from OM		X							
Read OM	X	X							
Read Trans	X		X	X	X	X			X
STOP							X	X	

TABSOL
A FUNDAMENTAL CONCEPT FOR SYSTEMS-ORIENTED LANGUAGES

T. F. Kavanagh

Manufacturing Services
General Electric Company
New York, New York

Summary

Lack of efficient methods for thinking-through and recording the logic of complex information systems has been a major obstacle to the effective use of computers in manufacturing businesses. To supply this need, this paper introduces and describes "decision structure tables," the essential element in TABSOL, a tabular systems-oriented language developed in the General Electric Company. Decision structure tables can be used to describe complicated, multi-variable, multi-result decision systems. Various approaches to the automatic computer solution of structure tables are presented. Some benefits which have been observed in applying this language concept are also discussed. Decision structure tables appear broadly applicable in information systems design.

In addition, they are of interest because they revise many earlier notions on problem formulation and systems analysis technique. Decision structure tables will be an available feature in GECOM, General Electric's new General Compiler, which will be first implemented on the GE 225.

Introduction

Progress in computers can be broadly divided into two categories. First there is the work that essentially accepts computers for what they are, and directs its energies toward further refinement of the original hardware, and operating technique. Research to improve recording density on magnetic tape would certainly fit this description. In the second category are the efforts to advance by developing new areas of application. This latter work is directed toward generalizing the concepts and hardware, so that they apply to an ever-increasing span of problems and situations. Obviously, both groups are vital; but it was this second stimulus -- the desire to expand the area of economic application -- which motivated the research reported in this paper. While the earliest beginnings can be traced as far back as June, 1955, the primary research effort started in November, 1957, under the title of the Integrated Systems Project. Leadership was assigned to Production Control Service, a component in General Electric's Manufacturing Services. The basic purpose of the Project was to probe the potential for automating the flow of information and material in an integrated business system.

Then, as now, computers were making significant contributions in many areas. Unfortunately, one of these areas was not, as some would have it, in the operation and control of manufacturing businesses. Important advances were made in specific applications such as order processing payroll, and inventory record-keeping; but these represented only a small percentage of the total information processing and decision-making in even the smallest manufacturing firm. Still these early successes were very important. They developed confidence in computer performance and reliability; but even more, they encouraged systems engineers and procedures personnel to continue computer applications research. Similarly, management, under growing foreign and domestic competition, rising costs, and a seeming explosion in paperwork requirements, saw intuitively -- or perhaps hopefully -- that computers offered a possible approach to improved productivity, lower costs and sharply reduced cycle times. It was in this environment that the Integrated Systems Project began a comprehensive study of the decision-making and the information and material processing required to transform customer orders into finished products -- a major part of the total business system for a manufacturing firm.

The Decision-Making Problem

Once underway, it was soon apparent

that there was an enormous amount of decision-making required to operate a business. Indeed, the number and complexity of these decisions is perhaps the most widely underestimated and misunderstood characteristic of industrial information systems today. Tens-of-thousands of elementary decisions are made in the typical manufacturing business each working day. All are necessary to guide and control the many functional activities required to design products, purchase raw material, manufacture parts, assemble products, ship and bill orders, and so on. The typical factory is a veritable beehive of decisions and decision-makers; for example:

- "What size fuses shall we use on this order for XYZ Company?"-- a product engineer's decision.
- "What is the time standard for winding this armature coil?"- a manufacturing engineer's decision.
- "What test voltages shall be applied?"-- a quality control planner's decision.
- "What should be the delivery promise on this customer's order?"-- a production control planner's decision.
- "How much will this model cost."-- an accountant's decision.

This list of elementary day-to-day decisions could be expanded to cover all business activities. If this were done, the list would cover hundreds of sheets of paper before each activity listed all the decisions for which it was responsible. Moreover, some of these decisions are repeated many times each day for various sets of conditions. In the end result, one cannot help but be impressed with the multiplicity of these detailed choices and selections. But more importantly, making these decisions costs money, in many cases more money than the direct labor required to make the product. In addition, business performance is greatly affected by the speed and accuracy with which this decision-making is carried out.

Composing a detailed list of these elementary business decisions is more than an academic exercise. For one thing, such an analysis of an actual operating business will demonstrate conclusively that these elementary decisions are handled quite rationally (which is somewhat contrary to popular opinion.) One must be careful not to be misled by quick, superficial explanations which gloss over fundamental reasoning. In our present-day manual systems which emphasize files of quick answers, the logic behind the decision is often left unrecorded. As a result it is easy to lose contact with their rational nature, and frequently we tend to feel these decisions are substantially more intuitive than is actually the case. At times, some persistent as well as penetrating analysis (often through extensive interviewing of the operating personnel presently on the job) is required to uncover the true parameters and relationships on which operating decisions are really based. This arduous work is more than justified, for it establishes a sound conceptual foundation for automation, and hence the practical application of the concepts and techniques developed in this paper. Thus, once it is established that these operating decisions are rational, it should follow that they can be structured in a consistent logical framework. Such a structure is presented in this paper.

Operating vs. Planning Decisions

At this point let us define terminology a little more precisely, and stress that we are speaking about the detailed, elementary decisions required to "operate" a business as opposed to "planning" one. First, a decision in its simplest form consists of selecting one unique alternative from an allowed set of possible actions. Operating decisions are defined in the context of this paper as selecting the appropriate course of action in accordance with given problem conditions to operate the business successfully. Operating decisions may be assumed to be made under "conditions of certainty." The solution for a specific set of problem conditions will always be the same. Under these premises, the action or outcome decided on can always be predicted. In a pragmatic sense, the decision-making process may be classed as "causal"; that is, B may be said to follow from A. For example, an engineer's decision to install fuses might follow from a customer's requirement for independent circuit protection.

The relevant factors or parameters affecting the decision can also be determined. The relationship values are known. For example, in most homes, the current carrying capacity of the house wiring is the only parameter value one needs to know to select an appropriate fuse. In an industrial application, however, the values of at least three additional parameters are usually required: voltage, time and type of fuse mounting. The strategy and the alternate outcomes are known; that is, the per-

missible fuses are known. To continue the illustration, the fuse selection may be limited to those carried in the stockroom; otherwise the bounds of the operating decision system are exceeded and the decision-maker would appeal to a higher authority.

To approach the analysis of operating decisions from another viewpoint, it might be compared to a linear programming problem, and as will become evident, a linear programming solution might be considered as somewhat of a mathematical bound for the class of decision-making systems under discussion.

These operating decisions are quite apart from the planning decisions of a business. The "planning", "administrative", or "policy" decisions in a business are basically those prior commitments which permitted all the assumptions about operating decision systems in the preceding paragraphs (i.e. certainty, causality, known relationships, etc.) Some examples of planning decisions are:

- "Shall fuses, circuit breakers, or both be used on the product line?"-- a product engineer's planning decision.
- "Should this group of parts be made on the screw machine or from die castings?"-- a manufacturing engineer's planning decision.
- "Should this component be inspected before or after the milling operation?"-- a quality control planning decision.
- "What rule shall be used to determine the correct order quantity?"-- a production control planner's decision.
- "What is an appropriate cost-of-money?"-- an accountant's planning decision.

These are typical planning decisions made in designing an operating decision system. To make the distinction clear, consider the design engineer who is motivated by cost considerations to put fuses on the economy part of the product line, while specifying circuit breakers on more deluxe models. Or consider the production control planner who selects one of the common square root formulas for determining all order quantities. Once he puts this decision rule in the operating system, order quantities for every part will be determined using this square root formula with specific values for cost, lead time, usage shelf life, etc., appropriate to the specific item being ordered. Assuming the operating decision system is automatic, and this is the intention, the production control planner need not make any order quantity determinations himself. Rather he will be watching the measures of operating system performance (inventory level, number of shortages, ordering costs, etc.) to see how well his decision rule is working. Incidentally, it's worth noting that the production control systems designer will be using a "cost-of-money" figure supplied by accountants and an annual requirements figure projected by salesmen. Of course, the objective of this fundamental decision analysis is to suggest a conceptual scheme which will permit automating all the routine operating decision-making required to direct a business, thus permitting the engineers, planners, and other technical advisors, to concentrate on doing a better job in design.

Specifying Decision Systems

But great difficulties still remain. As already pointed out, operating decision systems are invariably large and complex, containing multi-variable, multi-result decision problems with sequence of solution difficulties thrown in on the side. One serious problem which arises quickly is the actual development of the decision logic itself. Numerous techniques have been proposed ranging from precise, legalistic verbal statements to complex mathematical equations. Among these however, it appears that matrix-type displays and flow charts are the most common. The matrix-type displays appear under a variety of names: collation charts, tabulated drawings, standard time data sheets, etc. For example, engineers have frequently used collation charts to show direct relationships between end-product catalog numbers and component identification numbers. Typically, however, collation charts are a tabulation of past decisions rather than a description of the logic used to derive them. Matrix-type displays often suffer from redundancy and frequently become large and unwieldy as operating tools. Similarly, they make no allowance to sequential decision-making.

Flow charts handle this sequence problem very nicely. This graphic method describes a decision system by the extensive use of symbols for "mapping" the various operations. A variety of flow chart techniques are used in factory methods and office procedures work.

They are particularly effective in relatively straightforward, sequential decision chains but run into difficulty when describing multi-variable, multi-result decision processes. As an illustration, flow charts have been used extensively to document the detailed logic of computer programs; but some harried computer programming supervisors still maintain that the best way to transfer program knowledge is to reprogram the job. The difficulty of interpreting someone else's flow charts is certainly one of the major trials in today's computer technology.

In addition to these more popular tools numerous other diagramming or charting techniques have been useful in limited problem areas. However, the basic problem remained: there was really no effective, uniform method for thinking about and specifying decision systems as complex as those required to operate a business. To help solve this problem, the Integrated Systems Project developed a new technique which combines key characteristics of earlier methods and adds some new features of its own. This new technique is called the decision structure table. The balance of this paper will describe what decision structure tables are, how they work, and the results of their use in General Electric.

Structure Table Fundamentals

Structure tables provide a standard method for unambiguously describing complex, multi-variable, multi-result decision systems. Thus, each structure table becomes a precise statement of both the logical and quantitative relationships supporting that particular elementary decision. It is written by the functional specialist in terms of the criteria or parameters affecting the decision and the various outcomes which may result.

A structure table consists of a rectangular array of terms, or blocks, which is further subdivided into four quadrants, as shown in Figure 1. The vertical double line separates the decision logic on the left from the result functions or actions which appear on the right. The horizontal double line separates the structure table column headings or parameters above from the table values recorded in the horizontal rows below. Thus, the upper left quadrant becomes decision logic column headings, and is used to record, on a one per column basis, the names of the parameters (P_{0j}) effecting the decisions. The lower left quadrant records test values (p_{ij}) on a one per row basis, which the decision parameter identified in the column heading may have in a given problem situation. The upper right hand quadrant records the names of result functions or actions to be performed (R_{0j}) as a result of making the decision, once again on a one per column basis. Similarly the lower right quadrant shows the specific result values (r_{ij}) which pertain, directly opposite the appropriate set of decision parameter values. Thus, one horizontal row completely and independently describes all the values for one decision situation.

There is, of course, no limit to the number of columns (decision parameters and result functions) in any given structure table. Even the degenerate case where the number of decision parameters goes to zero is permissible. Also there is no limit on the number of decision situations (rows). Thus, the dimensions (columns by rows) of any specific structure table are completely flexible, and are a natural outgrowth of the specific decision being described. A series of these structure tables taken in combination is said to describe a decision system.

Rather than become further involved in abstract notation, let's consider some actual illustrations to develop an insight into the nature of structure tables. For example, the oversimplified illustrative structure table in Figure 2 states that an elementary decision on transportation from New York to Boston in the afternoon is (according to the person who developed the decision logic) a function of three decision parameters: Weather, Plane Space, and Hotel Room. Weather has only two value states, Fair or Foul; Place Space is either OK or Sorry; and Hotel Room can be Open or Filled. In terms of results, Plane or Train are the only permissible means of Transportation. Following the illustrative problem, we see by inspection that the solution appears in the second row. Therefore, Train is the correct value for Transportation, Other Instructions are Cancel Plane, and this is the End of the decision problem.

The intent of this simple structure table is to provide a general solution to this particular decision situation, and if the problem of afternoon trips to Boston ever arises (and one assumes that it frequently does), then an operating decision can quickly be made by supplying the current value of Weather, Plane Space, and Hotel Room, and, of course, solving the structure table. Solving a structure table consists of examining the specific values assigned the decision parameters in the problem statement and comparing or "testing" these values against

the sets of decision parameter values recorded in the structure table rows. Testing proceeds column by column from the first decision parameter to the last (left to right) and thence row by row (top to bottom). If all tests in a row are satisfied, then the solution is said to be in that row and the correct result values appear in the same horizontal row directly opposite to the right of the double line. When a test is not satisfied, the next condition row is examined.

When a particular structure table has been solved, it is often necessary to make more decisions. To specify what decision is to be made next, the last result column of the structure table may be assigned as a director to provide a link to the next structure table. Notice the last row in the illustrative structure table which specifies that for any value of Weather, with no Plane Space, and no Hotel Room, the decision-maker is directed to solve the next structure table, Transportation, New York-Boston, a.m. -- which is another structure table describing how to select a means of transportation in the morning.

In a similar fashion, the systems designer would use a whole system of structure tables to describe a more realistic operating decision problem. He completely controls the contents of each table, as well as its position in the sequence of total problem solution. He may decide to skip tables, or, if desired, he may resolve tables to achieve the effect of iteration. In any event, the entire system of tables, just as each individual structure table, will be solved using specific decision parameter values appearing in the problem statement. In other words, solving a set of structure tables consists essentially in re-applying the systems designer's operating decision logic.

Having completed this quick and very simplified introduction to structure tables, let us now return to consider each structure table element in greater detail. This will provide a deeper insight into the power of the structure table technique, as well as a better understanding of how they are used to describe operating decision systems. The illustrations are drawn from actual operating decision problems.

Structure Table Tests

Comparisons or tests between problem parameter values (pv) and decision parameter test values (tv) need not be simple identities, such as those used in the previous illustration. Actually the problem parameter values may be compared to the decision test values in any one of the following ways in any structure table block:

EQ	pv = tv	problem value is equal to test value.
GR	pv > tv	problem value is greater than test value.
LS	pv < tv	problem value is less than test value.
NEQ	pv ≠ tv	problem value is not equal to test value.
GREQ	pv ≥ tv	problem value is greater than or equal to test value.
LSEQ	pv ≤ tv	problem value is less than or equal to test value.

This broad selection of test types (or relational operators as they are known technically) greatly increases the power of individual structure tables and sharply reduces size. It permits testing limits or ranges of values rather than only discrete numbers. In Figure 3, TABLE 1000 uses several difference test types to bracket continuous and discontinuous intervals. Also note in Figure 3, that the relational operator may be placed in the test block immediately preceding the test value, or in the column heading immediately following the decision parameter name. When this latter notation is used, the relational operator in the column heading applies to all test values appearing immediately below.

Test values are not limited to specific numbers on alphanumeric constants (indicated by quotation marks); a test block may also refer to the contents of any name. In this case of course, the current contents of that named field are compared to the problem parameter value in accordance with the test type. For example, TABLE 1005 in Figure 3 tests the current value of INSUL~TEMP against MAX~TEMP to make certain that insulation temperature ratings are satisfactory.

In addition to these simple comparisons it is also possible to formulate compound structure table blocks involving two decision parameters or test values using a relational or logical operator.

The following logical operators may be used:

OR	tv_1 OR tv_2	first test value or the second test value.

AND	pv_1 AND pv_2	first problem value and second problem value.
NOT	tv_1 NOT tv_2	first test value and not second test value.

Also the truth or falseness of a compound decision parameter or test value statement can be tested with the symbols:

T	true
F	false

Lastly, any arithmetic expression may be used in place of a parameter name, and complicated blocks involving several names and operators are also permitted. Although in this latter case, it is worth noting that the language capability far surpasses any requirements experienced to date in formulating operating decision systems.

In writing structure tables, the situation often arises where, except for one or two special situations, one course of action is adequate for all input values. The concept of an "all other" row was introduced to avoid enumerating all possible logical combinations of the decision parameter values. The "all other" concept can be verbalized as follows: "if no solution has been found in the table thus far, the solution is in this last row regardless of the problem values." While this greatly reduces table size, it also implies that the problem was stated correctly and does indeed lie within the boundaries of the decision system. The related concept of "all" which appears in the Transportation: New York-Boston, p.m. can be similarly verbalized: "regardless of the problem value proceed to the next column." It was introduced so that a given table need not contain all permissible states of any given decision parameter and also to handle the case where a test in a given column had no significance. In all the above situations the appropriate structure table blocks are left blank signifying no test.

Structure Table Results

Similarly structure table results are not limited to assigning alphabetic constants or numeric values to the result functions or actions named in column headings to the right of the double line. Actually there are four result functions:

"ASSIGN" - which is implied when a named field appears as a result function. This indicates that the result value appearing in (or named by) the solution row is to be assigned or placed in the field named in the column heading.

"CALCULATE" - which is implied by the use of an equal sign after a name appearing as a result value. This indicates that the results of the formula evaluation named in the structure table block should be assigned to the field named as the result function in the column heading. Actually this is not the only way to perform calculations as any arithmetic expression may be used as a result value.

PERFORM - which performs the data processing or arithmetic operations referred to in the label appearing in the result value block. When this is completed, the next result function is executed.

GO - links the structure table to the label appearing in the result value block. There is no implied return in a GO function.

Most of these result functions are illustrated in Figure 3 and Figure 4. In Figure 4, for example, TABLE 2000 assigns the alphabetic constant "FLAT-STRIP" to ASSEMBLE. In the first and third result columns, arithmetic expressions appear as result values. In TABLE 2005 the implied CALCULATE is used for formula evaluation. TABLE 2005 also uses the PERFORM function to solve TABLE 2008 or carry out some other data processing operations depending on the particular solution row. TABLE 2005 is linked by the GO operation to TABLE 2010, 2015, 2020.

TABLE 1005 in Figure 3 shows an interesting use of the GO function. After the winding has been specified in TABLE 1000, assumedly on a lowest cost basis, the product engineer evidently wants to check the insulation temperature rating with the maximum expected operating temperature. If the insulation temperature rating should turn out to be greater everything is fine and the decision-maker proceeds to TABLE 1007. If not, first TYPE-N and then TYPE-T insulation are specified to

supercede TYPE-F, thus getting progressively higher insulation temperature ratings by redirecting the structure table to solve itself.

Frequently, a result function or action will not have a value for all rows. This is common when several result functions are determined by the same structure table. In this situation the phrase "not exist" has been used in verbalizing and the structure table block is left blank.

The use of formulas as structure table results can greatly reduce the size of the table. As an illustration, suppose that a given result function has twenty-six values (10, 12, 14 16, ...60). Ostensibly, the structure table to select the appropriate result value would have twenty-six rows. This decision could be reduced to one row by calculating the result value as some function of the decision parameter as shown in Figure 6. Obviously, all result relationships are not so conveniently proportional but a surprising number of result functions can be described with simple linear and exponential expressions. The curve fitting problem can be greatly simplified by using structure table rows to break the curve into convenient intervals that can be represented by such simple mathematical expressions.

Preambles and Postscripts

Each structure table is preceded by a heading which identifies the table by number and indicates its dimensions in terms of decision parameter columns, result function or action columns, and value rows. Tables may be numbered from TABLE 1 to TABLE 9999999 and allowance is made for up to 999 decision parameter or result functions. Provision is also made for 999 condition rows.

Following the heading is a NOTE which may contain any combination of alphabetic or numeric characters. The NOTE may be used to give the structure table an English name and provide a verbal description of the decision being made. Subsequent to this any labels naming expressions or arithmetic calculations referred to by "CALCULATE" or PERFORM operators in the body of the structure table may be defined. For example, note the definition of TIME ~ 1 and TIME ~ 2 in TABLE 2005 of Figure 4. The structure table proper follows BEGIN.

If no solution row is found in the structure table proper, or if the structure table has executed all results or taken all actions without reaching a GO function then control is passed to the area directly below the structure table. Here are recorded any special instructions pertaining to that particular decision. Of particular note is the situation where no solution row has been found. Such a failure is regarded as an "error." In certain types of decision systems, this may be exactly what the systems designer intended. However, error conditions most often indicate a failure of the decision logic to cope with a certain combination of input values. The systems designer should set up to notify himself whenever such an error occurs by designing an error routine which will provide him with a source language printout identifying the table that failed and the problem being solved at the time. With this problem printout and the source language structure tables, the systems designer has all the data he needs to trouble shoot the system in his own terminology. Thus, each structure table should be followed by the statement: IF NOT SOLVED GO_____. In this way any structure table failures will always be uncovered. Frequently, the situation arises, as mentioned earlier, that regardless of the solution row, the next structure table solved is the same. In this case the statement: GO_____. may be written after or below the preceding error statement, to serve as a universal link to the next structure table.

The areas immediately preceding and succeeding the structure table proper may also be used for input-output, data movement, and other data processing operations.

The Dictionary

The precise name and definition of each decision parameter and result function are recorded in a "dictionary." This dictionary becomes an important planning document in the systems engineer's work for it provides the basic vocabulary for communicating throughout the entire decision system. The dictionary should note a parameter's minimum and maximum values, as well as describe how it behaves. If the parameter is non-numeric in nature, the dictionary should record and define its permissible states. Significantly, the systems engineer formulates both the structure table and the dictionary using his own professional terminology.

The dictionary will also prove useful in compiling and editing structure tables for computer solution. It also follows that problems presented to the resulting operating decision system must also be stated in precisely the same terms as the structure tables. To those as yet uninitiated to the perversity of computers, this may seem a simple matter; unfortunately,

it is not. Interestingly however, one of the more promising application areas for structure tables appears to be in stating the logic for compilers and edit programs.

Summary

The foregoing description of decision structure tables is not meant to be a fully definitive language specification. The intention is to introduce the reader to the decision structure table concept and to discuss their characteristics in sufficient detail to provide the reader with enough understanding to evaluate their inherent flexibility and application potential. Many additional features are available which aid in formulating concise, complete decision structure table systems and also to facilitate input-output operations, but the reader will find that the fundamentals already described are adequate for structuring most operating decision logic.

Automatic Solution of Structure Table Systems

Decision structure tables have proven to be an excellent method for analyzing or formulating the logic of complex industrial information systems, but after taking such great care to precisely record each elementary decision in this highly structured format, it is only natural to speculate on the possibility of solving structure tables automatically with an electronic computer. Before plunging into the computer world, however, it is worth noting that some systems engineers have had very favorable experience using structure tables on a manual basis -- especially as a problem analysis technique, and also in limited applications in manual clerical systems.

Numerous methods for solving structure tables automatically suggest themselves. First, the tables could be coded by hand. Such an approach would use structure tables as a direct substitute for flow charts. Actually this really isn't as bad as it initially sounds. Many benefits would accrue from making this precise readable format the standard method for stating decision logic. It also offers the possibility that a series of macro-instructions could be developed, thereby permitting untrained personnel to code tables without detailed knowledge of computers or programming. However, this approach suffers some distinct disadvantages in comparison with the other alternatives outlined below.

Second, a generalized interpretive program could be written to solve any structure table. This offers the possibility of using a translator to work directly from keypunched structure tables without any manual detail coding. This approach makes economical use of memory since the basic programming to solve any table appears only once and the structure table itself offers a compact statement of decision logic. This reduces the amount of reading time required to bring the problem logic into the computer. File maintenance via recompiling structure table tapes also appears quick and simple. However, interpretive programs usually run more slowly; and this implies some penalty in total machine running time.

A third approach would be to create a structure table program generator in which an object computer program would be generated from the source structure tables. This approach would provide faster computer running times for maximum efficiency. A generator program would probably require more complicated coding than an interpretive translator. In addition, the generated object program would not be as concise as the structure tables themselves. However, where computer running time is of paramount concern, this approach has considerable appeal.

Because of the available time and money, all the early efforts of the Integrated Systems Project toward automatic structure table solution were essentially interpretive. It is interesting that a simple, yet adequate, tabular systems-oriented language could be provided in this way for somewhat less than a man year's effort. Similarly work to date in the area of formula calculations indicates that a comprehensive system of mathematical notation like that required for scientific work is probably not necessary in many operating business decision systems. Initial efforts on the IBM 702 were followed with experimental TABSOL languages for the IBM 305, IBM 650 and the IBM 704. These applications to different computers represented more than simple extrapolations to different pieces of hardware. In each an effort was made to expand capabilities of the language. In addition, the peculiarities of the equipment were explored, since one great concern was to free the user from a programming system usable on one and only one computer. As you might suspect, this wasn't always completely possible on the smaller computers, lacking tape or core memories. Nevertheless, the most recent Manufacturing Service effort on the IBM 650 produced a language with named fields, indexing, a two-address arithmetic, completely generalized structure table formats, and considering the alphabetic restrictions of the

equipment, remarkably flexible output formats.

Although these experimental languages proved quite adequate, one could not help but look toward the tremendous power of one of the more conventional languages. For one thing, the prospects for structure table application in other problem areas brightened, and it seemed reasonable that this power would be desirable in future work. Further our own tabular systems language development had brought us to the point of direct competition with the major language efforts already underway. Here General Electric's Computer Department entered on the scene. The Computer Department was developing a new concept in compiler building for use with General Electric computers. The first version of this new General Compiler, called GECOM, will be available to GE 225 users in May, 1961. It is designed primarily around COBOL, with some of the basic elements of ALGOL, and is now to contain all of TABSOL. To state the results of joining TABSOL with GECOM simply, it places the power of a full-fledged language at the command of every structure table block. Within General Electric, we obviously have a very high regard for the contribution of decision structure tables in information systems design. Significantly, the same committees who developed COBOL are now actively investigating tabular systems-oriented languages as the language of the future. By drawing on the CODASYL work and utilizing the extensive research and development experience already available within General Electric, the Computer Department expects that GECOM will provide users with a system compatible with both the present-day common business language, COBOL, and also the tabular systems-oriented language, TABSOL. Incidentally, the decision structure tables appearing in Figures 3, 4 and 5 are written in conformance with GECOM specifications.

Applications of Structure Tables

As somewhat implied in the illustrations a substantial amount of experience has been gained in applying structure tables to a wide variety of operating decision-making problems over the past three years. But perhaps the most interesting experience, at least from the researcher's point of view, was the very research work which spawned decision structure tables themselves. Earlier, it was mentioned that the Integrated Systems Project undertook a careful study of the essential information and material processing required to directly transform customer orders into finished products. For example, the product must be engineered prior to shipment, but the payroll, though revered by all of us can well be done at some other time, out of the main flow of events. Using this rough rule of thumb, the following activities were studied (Figure 7): order editing, product engineering, drafting, manufacturing methods, and time standards, quality control, cost accounting, and production control. These activities account for a fairly substantial portion of the business system. Normally, they would include 100% of the direct labor and 100% of the direct material as well as about 50% of the overhead. All the production inventory investment lies within the scope of this system and obviously most of the plant and equipment investment. Fortunately, the inputs and outputs to this system are simple and well-defined: the customer order comes in and the finished product goes out. With this in mind, it was possible to treat all activities within these bounds as one integrated, goal-oriented operating decision system and develop decision structure tables accordingly. Working with a small product section in one of the Company's Operating Components, a significant portion of the functional decision logic was successfully structured. Further the resulting structure tables were directly incorporated into a computer-automated operating decision system which transformed customer orders for a wide variety of finished products directly into factory operator instructions and punched paper tape to instruct a numerically programmed machine tool. This prototype system was demonstrated to General Electric management in November, 1958. Starting at the beginning, (Figure 8) the computer system edited the customer order and using the product engineer's design structure tables, developed the product's component characteristics and dimensional details. These in turn were used in the manufacturing engineer's operation structure tables to develop manufacturing methods and determine time standards. And so the flow of information cascaded down through the various business functions computing the quality control procedures, the product costs and the manufacturing schedules; eventually issuing shop paperwork and machine program tapes.

Since the completion of this work further research and development of the structure table concept was conducted in a variety of functional areas for different kinds of businesses in General Electric: defense, industrial apparatus, and consumer-type products. In addition, structure tables have been used in entirely different applications such as compilers. They also appear to hold great promise in complex computer simulation programs.

Benefits of Structure Tables

As a result of these efforts, we have come to believe that the decision structure table is a fundamental language concept which is broadly applicable to many classes of information processing and decision-making problems. They offer many benefits in learning, analyzing, formulating and recording the decision logic:

1. Structure tables force a logical, step-by-step analysis of the decision. First the parameters affecting the decision must be specified; then suitable results must be formulated. The nature of the structure table array is such that it forces consideration of all logical alternatives, even though all need not appear in the final table. Similarly, the precise structure table format highlights illogical statements. This simplifies manual checking of decision logic. The decision logic emphasizes causal relationships and constantly directs attention to the reasons why results are different. Personal design preferences can be resolved and intelligent standardization can be fostered.

   This is no mean capability. Indeed, it was very instructive to witness the development of methods and time standards logic in parallel with the development of the engineering logic during the initial Integrated Systems Project study. Through analysis of the decision structure tables written by the various functional specialists, everyone was able to achieve an insight into the product and the business rarely obtained in so short a period of time. The facts of life in product design, factory methods, and standardization were brought into the open very rapidly.

2. Structure tables are easily understood by humans regardless of their functional background. This does not imply that anyone can design or create new structure tables to describe a particular decision-making activity; but it does mean that the average person, with the aid of a dictionary, can readily understand someone else's structure tables. Thus, structure tables form an excellent basis for communication between functional specialists and systems engineers. Structure tables also go a long way toward solving the difficult systems documentation problem.

3. Structure table format is so simple and straightforward that engineers, planners, and other functional specialists can write structure tables for their own decision-making problems with very little training and practically no knowledge of computers or programming. Given a few ground rules, regarding formats and dictionaries, the structure tables written by these functional people can be keypunched and used directly in operating decision systems without ever being seen by a computer programmer. This cuts computer application costs as well as cycle times.

4. Structure table errors are reported at the source language level, thus permitting the functional specialist to debug without a knowledge of computer coding.

5. Structure tables solved automatically in an electronic computer offer levels of accuracy unequalled in manual systems. Note, however, that any such mechanistic systems lose that tremendous ability of humans to compensate for errors or discrepancies.

6. Structure tables are easy to maintain. Instead of changing all the precalculated answers in all the files, it is often only necessary to change a single value in a single table. For example, when changing the material specified for a component part under current file reference systems, it would be necessary to extract, modify and refile all drawings and parts lists calling for any variation of the component part. Using structure tables, it would only be necessary to alter those structure tables which specified the component material.

Summary

In closing, we recommend that the reader demonstrate the effectiveness of decision

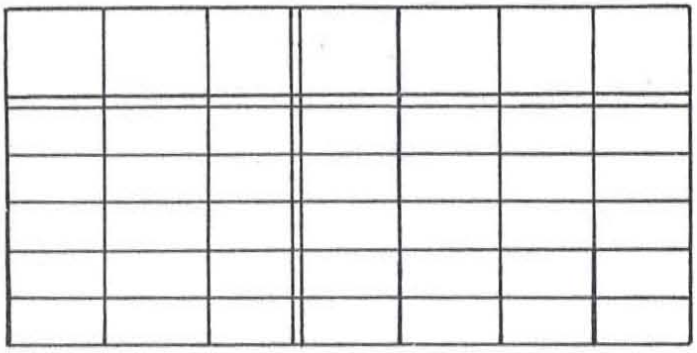
structure tables to himself by "structuring" a few simple decisions. For example, write a structure table which will enable your wife to decide how to pack your suitcase of any business trip. Perhaps a simple business decision such as those mentioned earlier would provide a more instructive example. The first structure tables are usually difficult to write, because most of us do not, as a general rule, probe deeply into the logic supporting our decisions. However, once this mental obstacle is overcome, "structuring" facility develops rapidly. If the reader will take the time to "structure" a few decisions and actually experience the deeper insight and clarity which this technique provides, then decision structure tables need no apologist, they will speak for themselves.

Acknowledgement

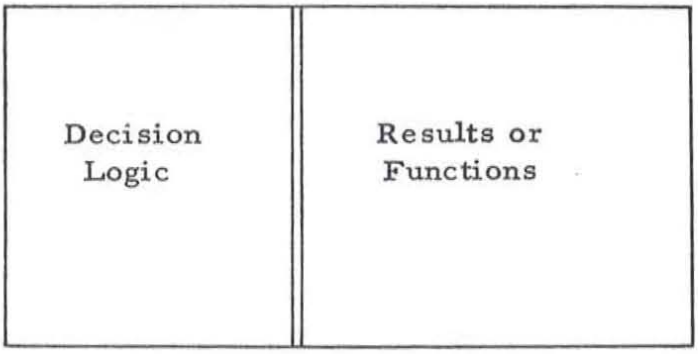
In contrast to most technical papers which essentially document only the work of the author, this discussion reports on the efforts of over seventy-five General Electric men and women. In particular, credit is due Mr. Burton Grad, who though no longer with General Electric, was a principal originator of the decision structure table concept. Mr. Malcolm C. Boggs, Mr. Daniel F. Langenwalter, Mr. Herbert W. Nidenberg, and Mr. Theodore E. Schultz representing Service Components and personnel from some fifteen different Operating Components within General Electric have contributed toward bringing these ideas to their present state of development and application. Acknowledgement is also due Mr. Charles Katz of General Electric's Computer Department who was instrumental in joining TABSOL and GECOM.

Decision Structure Table

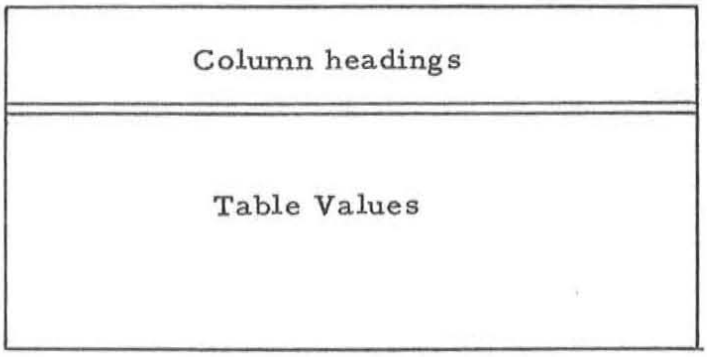
...a rectangular array of terms, or blocks...



...vertical double line...



...horizontal double line...



...structure table values...

P_{01}	P_{02}	P_{03}	R_{01}	R_{02}	R_{03}	R_{04}
P_{11}	P_{12}	P_{13}	r_{11}	r_{12}	r_{13}	r_{14}
P_{21}	P_{22}	P_{23}	r_{21}	r_{22}	r_{23}	r_{24}
P_{31}	P_{32}	P_{33}	r_{31}	r_{32}	r_{33}	r_{34}
P_{41}	P_{42}	P_{43}	r_{41}	r_{42}	r_{43}	r_{44}

Figure 1

Problem Statement: Select Transportation, New York - Boston, p.m.

Weather: Foul

Plane Space: OK

Hotel Room: Open

Decision Structure Table: Transportation, New York - Boston, p.m.

Weather	Plane Space	Hotel Room	Trans-portation	Other In-structions	Next Decision
Fair	OK	Open	Plane		End
Foul	OK	Open	Train	Cancel Plane	End
	Sorry	Open	Train		End
	OK	Filled		Cancel Plane	NY-Bost. a.m.
	Sorry	Filled			NY-Bost. a.m.

Solution:

If the value of Weather is Foul, and

the value of Plane Space is OK, and

the value of Hotel Room is Open,

Then

the value of Transportation is Train, and

the value of Other Instructions is Cancel Plane, and

the value of Next Decision is End.

Figure 2

TABLE 1000. DIMENSION C4 A5 R10.
NOTE TABLE FOR DETERMINING DETAIL VARIABLE PART CHARACTERISTICS FOR A LINE OF SENSING COILS IN ACCORDANCE WITH CUSTOMER END PRODUCT SPECIFICATIONS.
BEGIN.

SERVICE EQ	UNITS EQ	VALUE	VALUE	TURNS	DIA	RESIST	INSUL	INSUL ~TEMP
"DC"	"MAMP"	GR 180	LS 450	0.3/I	.001	2.6*TURNS	"TYPE-F"	150
:	:	:	:	:	:	:	:	:
"DC"	"MVLT"	GREQ 45	LSEQ 150	26	.008	1.84	"TYPE-F"	150
"DC"	"MVLT"	GR 150	LSEQ 330	13	.002	0.46	"TYPE-F"	150
"DC"	"VOLT"	GREQ 0.9	LSEQ 300	60	.002	39.0	"TYPE-F"	150
"DC"	"VOLT"	GR 300	LSEQ 1100	120	.002	137.0	"TYPE-F"	150
:	:	:	:	:	:	:	:	:
"AC"	"WATT"			230	.002	150.0	"TYPE-N"	200

IF NOT SOLVED GO ERROR~COIL.
MOVE "COPPER" TO MATERIAL.
GO TABLE 1005.
END TABLE 1000.

TABLE 1005. DIMENSION C2 A3 R3.
NOTE TABLE TO MAKE CERTAIN THAT INSULATION TEMPERATURE RATING EXCEEDS MAXIMUM OPERATING TEMPERATURE.
BEGIN.

MAX~TEMP	INSUL	INSUL	INSUL~TEMP	GO
LSEQ INSUL~TEMP				TABLE 1007
GR INSUL~TEMP	"TYPE-F"	"TYPE-N"	200	TABLE 1005
GR INSUL~TEMP	"TYPE-N"	"TYPE-T"	250	TABLE 1005

IF NOT SOLVED GO ERROR~COIL.
END TABLE 1005.

Figure 3

TABLE 2000. DIMENSION C3 A3 R4.
NOTE TABLE TO SPECIFY VARIABLE FACTORY OPERATION CHARACTERISTICS FOR THE INITIAL SENSING COIL WINDING FROM PART CHARACTERISTICS.
BEGIN.

SUPPORT~TYPE EQ	MATERIAL EQ	TURNS	START~W	ASSEMBLE	FINISH~W
"TABED-HOLE"	"COPPER"		TURNS		
"FLAT-STRIP"	"COPPER"	LS 100	2	"FLAT-STRIP"	TURNS-2
"FLAT-STRIP"	"COPPER"	GREQ 100	TURNS/2	"FLAT-STRIP"	TURNS/2
"FLAT-STRIP"	"ALUMNM"		TURNS	"2 FLT-STRP"	

IF NOT SOLVED GO ERROR~COIL. GO TABLE 2005.
END TABLE 2000.

TABLE 2005. DIMENSION C2 A3 R3.
NOTE TABLE TO CALCULATE TIME STANDARD FOR PREVIOUS OPERATION.
TIME~1 = 125*DIA*TURNS.
TIME~2 = 1000*DIA/SQRT (TURNS).
BEGIN

TURNS	TURNS	TIME	PERFORM	GO
LS 15		TURNS + 0.88	SETUP	TABLE 2010
GREQ 15	LS 100	TIME~1 =	SETUP	TABLE 2015
GREQ 100		TIME~2 =	TABLE 2008	TABLE 2020

IF NOT SOLVED GO ERROR~COIL.
GO TABLE 2005.

Figure 4

TABLE 1010. DIMENSION C2 A1 R3.
NOTE COIL QUANTITY DETERMINATION.
BEGIN.

SERVICE EQ	UNITS NEQ "WATTS"	COIL~QUAN
"AC"		0
"DC" OR "AC"	T	QUAN
"DC"	F	2*QUAN

IF NOT SOLVED GO ERROR~COIL. GO TABLE 1100.
END TABLE 1010.

TABLE 1500. DIMENSION C4 A3 R10.
NOTE COIL LOAD DATA AND CYCLE TIMES.
BEGIN.

SERVICE EQ	UNIT EQ	ACY EQ	INSP EQ	NORM~CYCLE	MIN~CYCLE	COIL~LOAD
"AC"	"AMPS" OR "MAMP"	1	"COML"	15	11	QUAN
"AC"	"WATT"	1	"COML"	15	11	2.2* QUAN
:	:	:	:	:	:	:
"DC"	"AMPS" OR "MAMP"	2	"COML"	15	9	0.9* QUAN
"DC"	"VOLT" OR "MVLT"	2	"COML"	15	9	0.9* QUAN
"DC"	"AMPS" OR "MAMP"	1	"GOVT"	20	16	1.4* QUAN

IF NOT SOLVED GO ERROR~COIL.
MIN~DATE = TODAY + MIN~CYCLE.
NORM~DATE = TODAY + NORM~CYCLE.
GO TABLE 1510.
END TABLE 1500.

Figure 5

TABLE 1510. DIMENSION C2 A2 R3.
NOTE COIL PROMISE DATE DETERMINATION.
BEGIN.

COIL~LOAD LSEQ	CUST DATE	PROMISE	GO
CUM~CAP (NORM~DATE)	GREQ NORM~DATE	CUST~DATE	NORM~LOAD
CUM~CAP (MIN~DATE)	GREQ MIN~DATE	CUST~ DATE	RUSH~LOAD
CUM~CAP (CUST~DATE)		CUST~DATE	EMER~LOAD

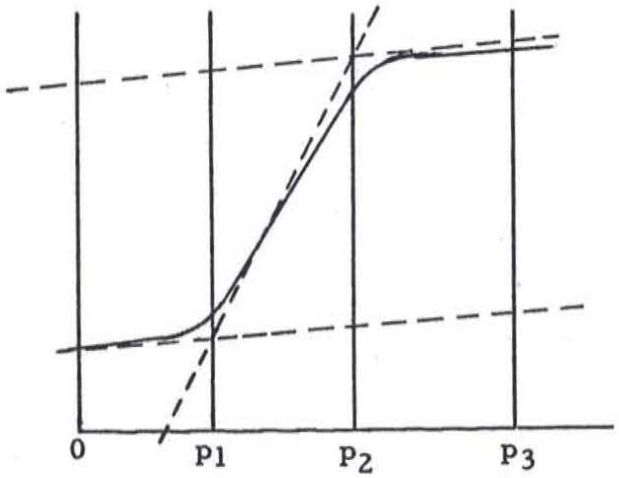
IF NOT SOLVED GO OVERLOAD.
END TABLE 1510.

Figure 5a

P	R
0	10
1	12
2	14
3	16
.	..
.	..
.	..
.	..
25	60

P	P	R
0	25	(2*p) + 10

.... The use of formulas as structure table results can greatly reduce structure table size, as shown by the simple straight line expression above. Structure tables may also be used to partition complicated curves into convenient segments as shown below. ....



P	P	R
0	P_1	lx + a
P_1	P_2	mx + b
P_2	P_3	nx + c

Figure 6

PRESENT MAIN LINE SYSTEM

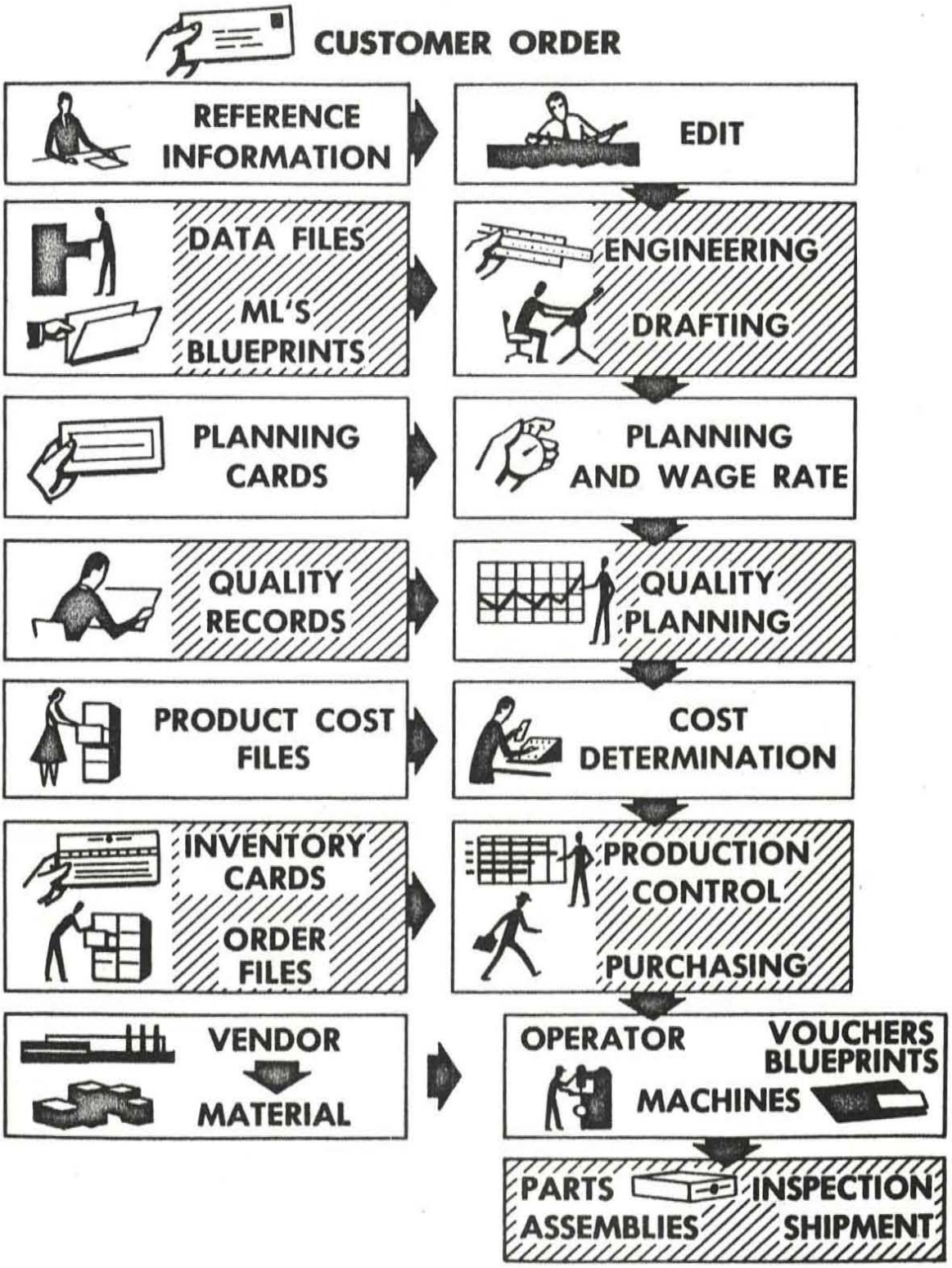


Figure 7

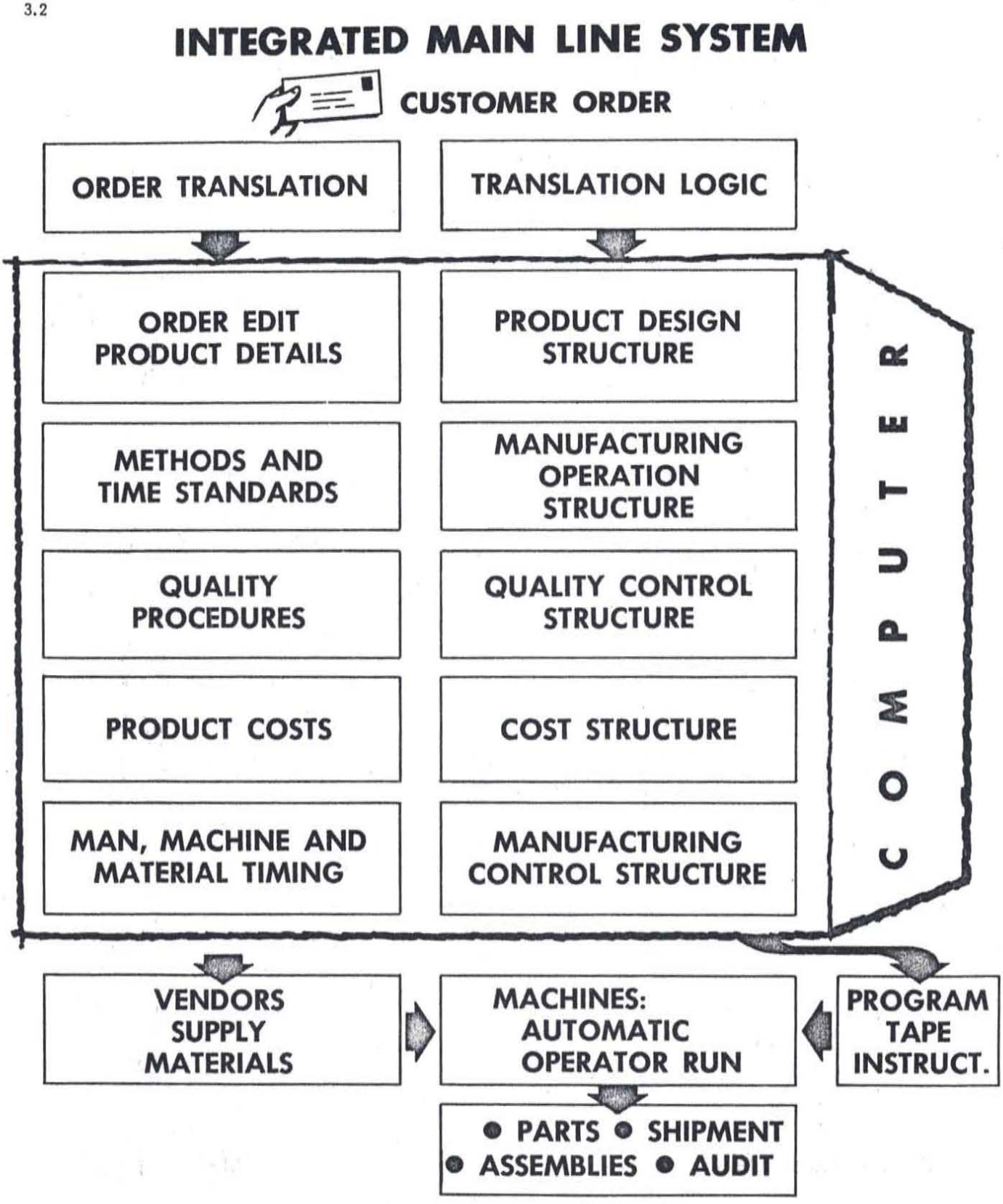

INTEGRATED MAIN LINE SYSTEM
CUSTOMER ORDER
ORDER TRANSLATION
TRANSLATION LOGIC
ORDER EDIT
PRODUCT DETAILS
PRODUCT DESIGN
STRUCTURE
METHODS AND
TIME STANDARDS
MANUFACTURING
OPERATION
STRUCTURE
QUALITY
PROCEDURES
QUALITY CONTROL
STRUCTURE
PRODUCT COSTS
COST STRUCTURE
MAN, MACHINE AND
MATERIAL TIMING
MANUFACTURING
CONTROL STRUCTURE
COMPUTER
VENDORS
SUPPLY
MATERIALS
MACHINES:
AUTOMATIC
OPERATOR RUN
PROGRAM
TAPE
INSTRUCT.
● PARTS ● SHIPMENT
● ASSEMBLIES ● AUDIT

Figure 8

PRELIMINARY REFERENCE MANUAL

TABSOL-225 --- A Tabular Systems Oriented Language

for the

GE225 Information Processing System

Computer Department
Applications Section
Programming Research and Development
Phoenix, Arizona

December, 1960

This document is a draft of a Preliminary Reference Manual and a language specification for integrating decision tables with the General Compiler. The information contained herein assumes a basic knowledge of computers and electronic data processing applications. Therefore, the manual should be used not as a text book but rather to augment already realized skills. Minor changes in language specification may occur during the implementation period of the compiler. Any changes that are made will be reflected in future and more final versions of this manual or in supporting material issued during the interim of implementation.

D. Klick
Programming Research

I. INTRODUCTION

Early automatic coding systems, such as assembly programs, employed mnemonic abbreviations in place of the computer's numerical instruction code and symbolic addresses in place of actual memory addresses. In reality the assembly program language was a set of synthetic computer instructions. Although these systems greatly simplified programming, the programmer was still plagued with the many details dictated by the computer.

Automatic coding languages of today are on the threshold of relieving the programmer of these details. The structure of these new languages are very much like English. By using a combination of English words and phrases to form sentences, the programmer now needs only to "describe" a procedure for the computer to follow. This procedure together with a description of the data is then given to a special computer program for processing. This special program, commonly called a compiler, translate the English problem description and generates a program of computer instructions.

Such a compiler is provided for the GE-225. Its General Compiler evolved from two noteworthy language efforts - the Common Business Oriented Language (COBOL) and the Algorithmic Language (ALGOL). Both languages were developed by voluntary committees of computer manufacturers and users and reflect the recent trend toward "common" compiler languages.

The language presently available with the General Compiler is based primarily on COBOL, since COBOL satisfied the needs of a broad spectrum of data processing applications. To accommodate the demands of more technical

applications, Boolean expressions, floating point arithmetic, and the ability to express equations were incorporated into the format of COBOL. Therefore, one may say that the present version of the General Compiler can accept programs written in one, two, or in a combination of two language forms.

Those programmers familiar with COBOL recognize that it is well suited for creating and processing data files. ALGOL, on the other hand, provides an excellent means for expressing complex mathematical relationships. Recent investigations by the Integrated Systems Project of General Electric's Manufacturing Services uncovered an area of applications which require neither extensive data file processing nor profound mathematics but rather an unwieldy number of sequential decisions.

To cope effectively with these decisions the ISP team devised a tabular language. The purpose of this language was to depict, by means of tables, the relationships of logical decisions. The new language was appropriately termed TABSOL for Tabular Systems Oriented Language. Since its creation TABSOL has been used by many departments of General Electric to analyze and solve problems in product engineering, manufacturing methods, cost accounting, and production control. The application of decision tables is continually growing. Recent studies show that they provide a concise method for supporting the logic of other data processing applications. For example, decision tables may be used to specify the transfer of control associated with the values of one or more fields, to control the printing of detail and summary lines of a report, or to interrogate the sort keys in a multi-file system At

the Computer Department we have found decision tables a valuable tool in designing and implementing the General Compiler.

Decision tables represent a third language for the General Compiler. They may be used by themselves or in conjunction with the features of the compiler language. The specifications outlined in this manual pertain mainly to the table entries and imply and require a knowledge of the General Compiler. Therefore, this manual should be used as a supplement to the GE-225 General Compiler Manual, CPB-123 (5.5M10-60).

II. DECISION TABLE FORMAT

The format of a decision table is given in Fig. 1. In concept a table is an array of blocks divided into four quadrants by a pair of double lines. The vertical double line separates the decisions or "conditions" on the left from the "actions" on the right. The horizontal double line isolates variables from associated operands which will appear in the blocks and rows below. A condition then is a relation between a variable appearing in a primary block and an operand appearing in a corresponding secondary block. For example, we may write AGE in primary block 1 and EQ 26 in secondary block 1. In doing this, we are stating a condition. Verbally, we are asking "if age equals 26". An action, on the other hand, is a statement of what is to be done. By writing AGE in a primary action block and 26 in its associated secondary block, we are stating that "the value 26 is to be assigned to age".

It is interesting to note, at this point, the English interpretation given to the vertical lines. The left-most line may be thought of as representing the word IF. Those lines to the left of the vertical double line may be taken to mean AND; the vertical double line itself the word THEN. Since actions are sequential entities, the lines separating them may be interpreted as semicolons and the right-most line, which actually terminates the actions, as a period. With this in mind, each secondary row becomes an English sentence. For example, each row now reads:

> "IF condition-1 is satisfied AND condition-2 is satisfied AND . . . AND condition-k is satisfied THEN perform action-1; action-2; . . .; action m."

If any condition within a row is not satisfied, the next row is evaluated

DECISION TABLE FORMAT

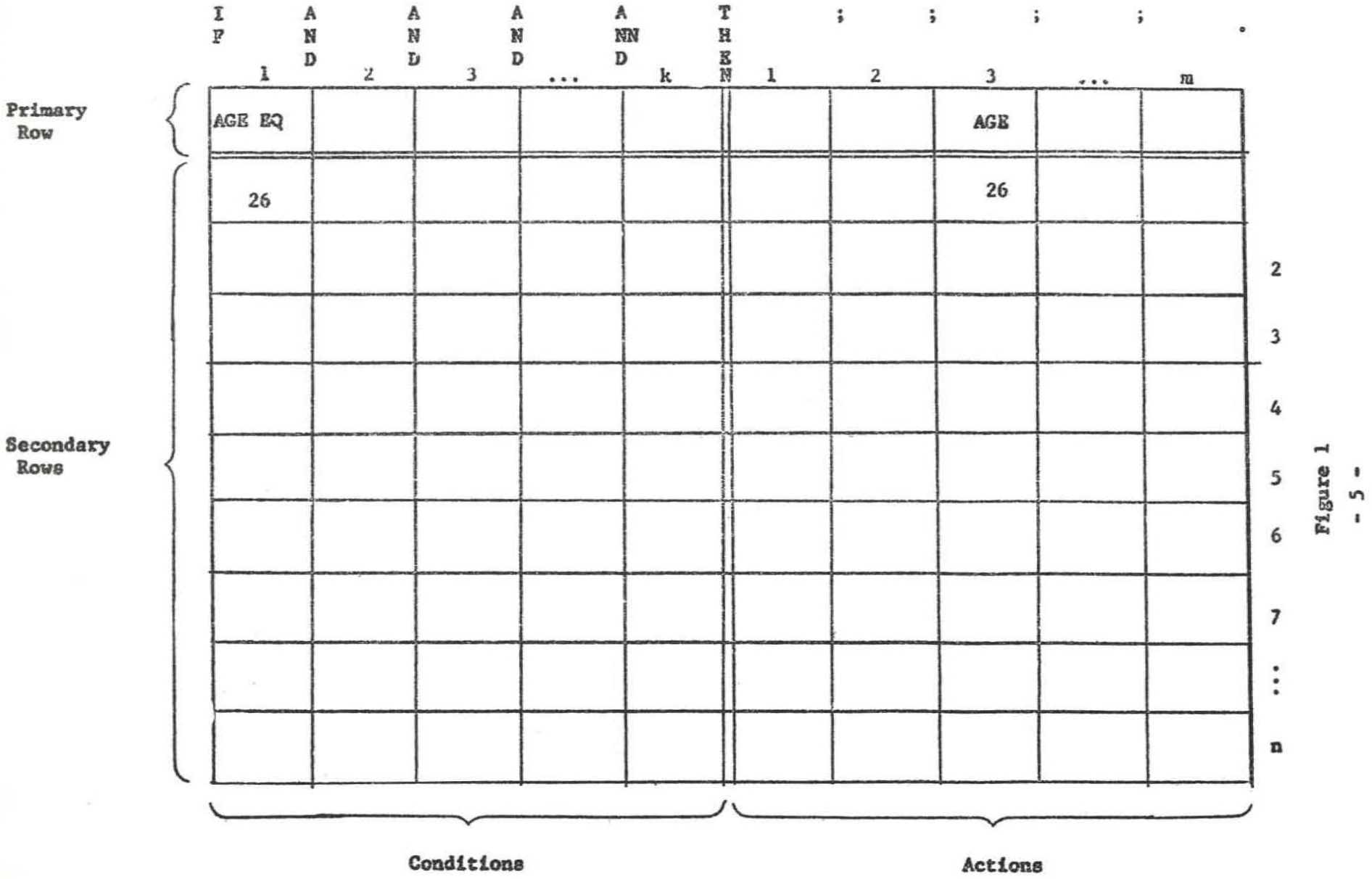


Figure 1

and so on until all the rows are depleted. When this happens the table is said to have "no solution". The table is considered "solved" when all the conditions of a row are satisfied and their associated actions performed.

Before considering the conventions used to formulate conditions and actions, an example may help develop insight into the nature of decision tables and the manner in which they may be used with the General Compiler. In this example (Fig.2) we are searching a master employee file (recorded on magnetic tape) to determine the number of male employees who fall into the following job categories.

Job Level	Years Experience		Title
6		2	Programmer
7		3	Programmer or Analyst
8	More than	3	Analyst
9	More than	4	Analyst or Manager
10	More than	4	Manager

For each employee we find having any of these qualifications, we are to write his department number, name, title, level, and experience on the computer's typewriter. At the end of the run the totals for each of the categories are to be also put on the typewriter.

The core of this problem is the decisions that must be made on the information stored in the records of the master file. These decisions are conveniently expressed above in narrative form. With only minor alteration this form becomes the program statement of our problem. The table and sentences are punched into 80-column cards exactly as they appear in Fig.2. When this is done they may be given directly to the compiler for processing.

As illustrated in our example, General Compiler sentences may be used to support the logic of the table. These sentences accomplish the following:

GENERAL ELECTRIC

GENERAL COMPILER
SENTENCE FORM

PROGRAM: Sample Decision Table
DATE:
PROGRAMMER:
COMPUTER:
PAGE: OF

SEQUENCE NUMBER	
5	PROCEDURE DIVISION.
10	OPEN INPUT MASTER~FILE.
15	GET~RECORD. READ MASTER~FILE RECORD IF END FILE GO TO END~RUN.
20	IF FEMALE GO TO GET~RECORD.
25	EXPERIENCE = 60 - YR~EMPLOYED + PREV~EXP.
30	TABLE EXAMPLE. 3 CONDITIONS 2 ACTIONS 5 ROWS.
35	LEVEL EQ EXPERIENCE I GO TO
40	6 EQ 2 PROGRAMMER 1 TYPE~OUT
45	7 EQ 3 PROGRAMMER OR ANALYST 2 "
50	8 GR 3 ANALYST 3 "
55	9 GR 4 ANALYST OR MANAGER 4 "
60	10 GR 4 MANAGER 5 "
65	GO TO GET~RECORD.
70	TYPE~OUT. WRITE DEPARTMENT NAME TITLE LEVEL EXPERIENCE ON TYPEWRITER.
75	TOTAL(I) = TOTAL(I) + 1.
80	GO TO GET~RECORD.
85	END~RUN. CLOSE MASTER~FILE.
90	WRITE TOTAL(1) TOTAL(2) TOTAL(3) TOTAL(4) TOTAL(5) ON TYPEWRITER.
95	STOP "END RUN".

Computer Department, Phoenix, Arizona CA-13 (10-60)

Figure 2

OPEN --- Declares that the MASTER-FILE is input and since the file is recorded on magnetic tape, validates the tape labels.

READ --- Delivers the next record from the MASTER-FILE and tests for an end-of-file sentinel. When this sentinel is detected, sequential program execution is interupted and control passes to the portion of the program labeled END-RUN.

IF --- Eliminates those data records which contain information about female employees. The word FEMALE (also PROGRAMMER, ANALYST, and MANAGER used in the table) represents a special kind of condition and will be explained later in the manual.

EXPERIENCE = --- Calculates the employees total experience and assigns the value to the field named EXPERIENCE.

The word TABLE informs the compiler that it must process a decision table; EXAMPLE is a name or label which was given to the table. The size of the table is stated next by giving the number of conditions, actions, and rows contained in the table. This information is used only by the compiler and is not executed by the compiled program.

Table execution begins at row 1 (sequence number 40). Using our narrative definition of a table, row 1 is interpreted as follows:

> "IF the job LEVEL field equals (EQ) 6 AND the EXPERIENCE field equals (EQ) 2 years AND the employee's title is PROGRAMMER THEN assign the value 1 to the subscript I; GO TO the part of the program having the label TYPE-OUT."

If one of these condtions cannot be satisfied, row 2 is evaluated starting again with the left-most condition. Sequential execution of the rows continues until either all conditions in a given row are satisfied or

all rows are exhausted. When the latter situation occurs, the sentence immediately following the table is executed. Proceeding from here the sentences in our example accomplish the following:

GO --- Interrupts sequential program execution and passes control to the part of the program labeled GET-RECORD.

WRITE--- Writes the current contents of the DEPARTMENT, NAME, TITLE, LEVEL, and EXPERIENCE fields on the computer's typewriter.

CLOSE--- Rewinds the MASTER-FILE and performs the file's closing conventions.

STOP --- Terminates processing and writes the words END RUN on the typewriter.

By General Compiler standards this example represents relatively simple conditions and actions. In formulating these entries, the programmer may take full advantage of the compiler's capabilities. The remaining sections of this manual are devoted to defining the conventions and manner in which conditions and actions may be formed and entered in tables.

III. BASIC CONCEPTS

Since decision tables are used in conjunction with the General Compiler language, we must first look at the foundations of this language before considering the counterparts that may appear in a table. The compiler's language, like most natural languages, is a body of words and a set of conventions for combining these words to express meanings. Its structure or "syntax" closely resembles the rules of English grammar, and its body of words may be appropriately termed a "vocabulary". The purpose of this section is to show how words are formed and how they may be used to express a desired meaning.

Characters

The basic units of our language are the characters used to form words and symbols. The character set includes the letters of the alphabet (A, B, C, ... Z), the numerals (0, 1, 2, ..., 9), and the special characters shown in Fig. 3. Special characters are presented in more detail as they are encountered in the manual.

Words

The words of a typical General Compiler program fall into one of two categories: the vocabulary of the compiler and the vocabulary used by the programmer. The programmer's vocabulary will consist mostly of arbitrary names given to his data and sections of his program. The compiler's vocabulary, on the other hand, is predetermined and explicitly defined in this manual. Since the compiler, by nature of its designers, is a mistrusting mechanism, the programmer must define the words he uses too. This is done, not by writing a manual, but instead by merely

SPECIAL CHARACTERS

Character	Meaning	Card Code
Δ	Space or blank	Space
.	Period - Decimal point	12-3-8
,	Comma	0-3-8
"	Quotation Mark	3-8
~	Hyphen	5-8
(	Left Parenthesis	0-5-8
)	Right Parenthesis	0-6-8
+	Addition	12
-	Subtraction - Minus Sign	11
*	Multiplication	11-4-8
/	Division	0-1
=	Assignment	6-8
|	Vertical Table Line	12-4-8

Figure 3

filling out a data description form. Once these "data names" are defined, they may be filed either on 80-column punched cards or on magnetic tape and used over and over again. The data description file then is a "dictionary" since it contains the definitions of the words used by the programmer. Furthermore, this dictionary may be revised without redefining all of its entries. This is accomplished by a special service routine which accepts corrections, insertions, and deletions as long as they are written on the compiler's data description form.

Our two categories of words may be illustrated by the following sentence taken from the program example given in Fig. 2.

GET-RECORD. READ MASTER-FILE RECORD IF END FILE GO TO END-RUN.

Here, the words READ, RECORD, IF, END, FILE, GO, and TO belong to the vocabulary of the compiler; whereas, the words GET-RECORD, MASTER-FILE, and END-RUN belong to the programmer's vocabulary. The compiler will assume that MASTER-FILE is a data name due to the word's position in the sentence. It will then search the data description to verify its assumption and to determine the characteristics depicted by this word. Not finding a match in the data description results in an error message typed on the computer's typewriter. The words GET-RECORD and END-RUN will be interpreted as sentence names due to their position in the program. Once again, the compiler will attempt to verify its findings by checking each transfer to make certain that they lead to properly defined sentence names. The consequence of an undefined sentence name is likewise an error message on the computer's typewriter. The compatability checks mentioned here are only two of many which the compiler performs to insure unquestionable results in the programs which creates.

Formation of Names

As previously mentioned, data names are words representing data (files, records, fields, elements, constants, arrays of values, etc.) and are arbitrarily assigned by the programmer. They are formed from the following characters.

Letters	A, B, C, ..., Z
Numerals	0, 1, 2, ..., 9
Hyphen	-

To avoid error messages and possible re-compilation, the programmer should choose data names that

1. Do not exceed 12 characters,
2. Do contain at least one letter,
3. Do not begin or end with a hyphen.

To insure a properly defined program, all data names should be recorded and their characteristic data described on the compiler's data description form. The programmer also should be careful not to use the compiler's vocabulary as data names.

In addition to data names, the programmer is free to name sentences, tables, and other "procedures" in his program. With one exception these names are formed like data names. Since procedure names are judged from their position in the program, they may be formed from only the numerals, 0 through 9.

Constants

The values associated with data names generally change during the actual running of a compiled program. It is for this reason that they

are sometimes called "variables". A constant, as opposed to a variable, is a specific value and does not change within the scope of a program. Constants may be one of two kinds: a literal, or a named constant.

A literal is a value itself rather than a name given to a value. Literals may be numerical, alphabetic, or alphanumeric - i.e., composed from the character set of the computer. All non-numeric literals should be enclosed in quotation marks (") to avoid having the compiler confuse them with data names. The conventions for forming literals are the following:

1. Non-numeric literals are limited to 30 characters, excluding the quotation marks.
2. A numeric literal not enclosed in quotation marks is assumed to be a number. Numbers may contain not more than one decimal point and a minus sign. Unsigned numbers are considered positive. Excluding decimal points and minus signs, numbers must not exceed 11 decimal digits.

3. Numbers may be treated as floating point by writing them as a power of ten - i.e., a number or decimal fraction followed by a power of ten exponent. For example, the number 230100 might be written as 2.301E5 which is equivalent to 2.301 multiplied by 10^5. The exponent part, indicated by the letter E, may contain a minus sign to show a negative exponent. The value range of an exponent is limited to $\pm$ 75. Excluding the decimal point, the minus sign, and

the letter E, the fractional part of a power of ten number must not exceed nine decimal digits. To distinguish data names from floating point numbers, data names should not be formed from only the numerals and the letter E.

4. An alphanumeric literal may not contain an embedded quotation mark since the enclosing quotation marks are used to determine the size and content of the literal.

A named constant is a constant which has been given a name. Named constants are defined by means of the data description and may include any character belonging to the character set of the computer, including the quotation mark. Like literals named constants may be numeric, alphabetic, or alphanumeric. They are unlike literals in that they may be any length.

Subscripts

Subscripts provide a convenient method to reference individual values contained in a list or in an array of values. The variable, I, employed in the decision table of Fig. 2 is a subscript used just for this purpose. Since five totals are to be accumulated, one name was assigned to all five, namely, the data name TOTAL. Whenever reference was made to a particular total, the data name TOTAL was followed by the subscript I. This is illustrated in the expression

TOTAL (I) = TOTAL (I) + 1.

and the sentence which prints all five totals on the typewriter. From this example, it follows that subscripts, like data, may be given names. In fact the same rules that govern forming data names apply to naming subscripts.

Since subscripting is a positional notation, the range of any subscript is limited to the values 1, 2, 3, . . ., n (where n is the maximum number of values in a list). This does not mean that subscripts are limited only to integers. If a subscript is not defined as integer by means of the data division, the compiler will automatically provide coding to truncate its value to an integer. Furthermore, subscripts are not restricted to a single variable name. Arithmetic expressions may also be used as subscript. For example,

```
RATE (P+1)
K ((K-3)*P**3)
A (J)
```

are legitimate forms of subscripts.

Up until now, only one-dimensional subscripting was considered. Values in multi-dimensioned arrays may also be referenced by subscripts. For example, an array in which values are ordered

A_{11} A_{12} A_{13} A_{14} A_{15}

A_{21} A_{22} A_{23} A_{24} A_{25}

A_{31} A_{32} A_{33} A_{34} A_{35}

A_{41} A_{42} A_{43} A_{44} A_{45}

A_{51} A_{52} A_{53} A_{54} A_{55}

might be subscripted as A (J,K), where K is the columnar subscript and J the row. To refer to value A_{35}, J would have to equal 3 and K equal 5.

Preceeding examples show that subscripts are enclosed in parenthesis and separated by commas. This notation permits the compiler to distinguish subscripts from other elements in the language.

Truth-Values

There is a class of variables which, through either usage or definition, may assume only the numerals 1 or 0. The value 1 is said to be their true state and the alue 0 their false state. The words END FILE of the READ sentence in Fig. 2 is such a variable. When the OPEN sentence is executed, END FILE is set to its false state and remains so set until the end-file condition is encountered. At this time, it is set to its true state.

Variables having truth-values are termed "True-False" variables. END FILE is a convenience provided by the compiler; the programmer may also formulate his own true-false variables by merely listing them under the heading TRUE-FALSE in the data division. They may be named according to the rules given for data names.

Arithmetic Expressions

Arithmetic expressions are rules for computing numerical values. They are formed from variables, numbers, functions, and symbols representing addition, subtraction, multiplication, division, and exponentiation. For example, in the expression

PREM-HRS * 2.50 + OT-HRS * 3.75

PREM-HRS and OT-HRS are variables; 2.50 and 3.75 numbers; and + and * symbols for addition and multiplication. If PREM-HRS were 40 and OT-HRS were 4, the expression becomes 40 * 2.50 + 4 * 3.75 and after performing the arithmetic, reduces to the value 115.00. To save this value, a programmer might write

GROSS-PAY = PREM-HRS * 2.50 + OT-HRS * 3.75.

The presence of the = symbol tells the compiler to assign 115.00 to the variable GROSS-PAY. When expressions are written in this form, they are called "assignment statements".

The arithmetic permitted in an expression is stated by the following symbols:

Symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

In addition to arithmetic, the following mathematical functions may be used.

Symbol	Function
SIN	Sine
COS	Cosine
ATAN	Arctangent
SQRT	Square Root
EXP	Exponential
LOG	Common Logarithm
LN	Natural Logarithm
ABS	Absolute Value

Arithmetic expressions are evaluated from left to right according to the following priority:

1. Exponentiation and Functions
2. Multiplication and Division
3. Addition and Subtraction

Parentheses may be used to establish a precedence other than the one above. When they are used, the evaluation is performed from the innermost to the outermost pair but still from left to right within a given pair.

Relational Expressions

A relational expression is a statement of magnitude between two values. For example, FICA GR 144.00 is a comparision between the variable FICA and 144.00. The symbol GR stands for the relation "greater than". Other relations may be stipulated by

Symbol	Relation
EQ	Equal to
GR	Greater than
LS	Less than
NEQ	Not equal to
NGR	Not greater than
NLS	Not less than

To have meaning, relational expressions must be stated as conditions. The expression FICA GR 144.00 tells us nothing. However, when it is written as

IF FICA GR 144.00, GO TO ADJUST-PAY

we know immediately what is intended. By definition then, relational expressions are conditions and when evaluated always give a truth-value.

Relational expressions may be explicitly stated or implied. FICA GR 144.00 is an explicit statement of magnitude. In the program example of Fig. 2, implied relations were stated by the words FEMALE, PROGRAMER, ANALYST, and MANAGER. An implied expression is formed by giving a name to a value, a range of values, or to a series of values and ranges. Once the name and its values are defined in the data division, it may be used to mean its associated values. Implied relations are termed "condition-names" since a name was given to a condition, i.e., a value, of a variable. The

variable from which the value is taken is called a "conditional variable". Therefore, writing PROGRAMMER (fig.2) in a decision table block is the same as writing an expression which will compare the TITLE field with the value associated with the title, programmer.

Logical Expressions

Logical expressions provide a convenient method for obtaining truth-values. They are formed by combining true-false variables and relational expressions with the logical operators AND, OR, and NOT. The expression (Fig.2)

PROGRAMMER OR ANALYST

is a logical expression which is true when an employee's TITLE field indicates that he is either a programmer or an analyst.

The rules governing the evaluation of logical expressions may be expressed as follows:

p	F	F	T	T
	F	T	F	T
NOT p	T	T	F	F
p AND q	F	F	F	T
p OR q	F	T	T	T

where p and q are a combination of true-false variables, relational expressions, or logical expressions.

Logical expressions are evaluated from left to right with the logical operator AND having precedence over the OR. Parentheses may be used for grouping or establishing a precedence of evaluation other than the one mentioned previously. When they are used, the evaluation proceeds from left to right from the innermost pair to the outermost pair.

IV TABLE ENTRIES

The previous section outlined the elements of the General Compiler language and briefly showed how they might be used. In the introduction, it was mentioned that these same elements may be employed within the blocks of decision tables. The purpose of this section is to show how this may be done.

Formation of Conditions

By definition, a condition is a relation between a primary block entry and some corresponding secondary block entry. A condition, like a relational expression, may be either true or false. True conditions are said to be "satisfied" and false conditions "not satisfied". From this definition, a condition may be either a relational expression, a logical expression, or a true-false variable since these are the only elements that yield a truth-value.

The formats noted below show how these expressions may be split between primary and secondary blocks to form conditions. In these examples, the word "operand" stand for either a variable (data name or subscripted data name), a constant (literal or named constant), or an arithmetic expression. The word "relation" signifies one of the relational operators - EQ, GR, LS, NEQ, NGR, or NLS. Since arithmetic expressions may be operands of relational expressions and relational expressions as operands of logical expressions, it necessarily follows that arithmetic expressions may appear in logical expressions.

Format	Example
Operand-1 Relation	LEVEL EQ
Operand-2	10

Unit	Example
Operand-1 Relation Operand-2	EXPERIENCE GR 4
Operand-1 Relation Operand-2 OR Operand-3	TOTAL (I) NLS PT(1) OR PT(2) or PT(3)
Operand-1 Relation-1 Operand$_2$ OR Relation-2 Operand-3 ...	(X+Y) ** 3 GR P+1 OR LS Q(I)
No Entry Condition-name	 PROGRAMMER
NOT Condition-name	NOT FEMALE
No Entry True-False Variable	 REQ-1
NOT True-False Variable	NOT END INVENTORY FILE
No Entry Logical Expression	 PROGRAMMER OR ANALYST
NOT Logical Expression	NOT X GR Y OR X LS (Z+1)

Formation of Actions

Actions are statements of the things to be done when all the conditions of a row are satisfied. The scope of an action may be one of three kinds: implied assignment, procedural, or input-output. The only action presented so far was assignment. The other two are extensions of General Compiler sentences and will be mentioned here only briefly. The compiler manual should be consulted for a more detailed presentation.

1. Value Assignment. Value assignment is an implied function between associated,primary and secondary block entries. By placing a data name in a primary block and some number in a secondary block, for example, I and 1 of Fig. 2, the compiler automatically produces coding to assign the numer to the data name. In the case of our example, 1 is assigned to the subscript I. Other examples of value assignment are given below. In these formats the word variable implies either a data name or a subscripted data name and the word constant either a literal or a named constant.

Format	Example
Variable	I
Constant	1
Constant	"COPPER"
Variable	MATERIAL
Variable	ALPHA (I,J,K)
Arithmetic Expression	SIN THETA + (X/P)**2

Format	Example
Arithmetic Expression	PI * R**2
Variable	AREA#1
True-False Variable	SWITCH#7
Truth-Value 1 or 0	1
Truth-Value 1 or 0	0
True-False Variable	BETA#REQ

2. Procedural actions. Procedural actions provide the means for interrupting the normal execution sequence of a table. Any of the following compiler verbs may be used for this purpose.

GO TO
PERFORM
STOP

The GO verb stipulates an unconditional transfer to a specified part of the table or program. Its destination may be a sentence name, table name, or the row number of a particular table The format of the GO entry is as follows:

Format	Example
GO TO	GO TO
Sentence Name	TYPE#OUT
GO TO	GO TO
Table Name	TABLE 23
GO TO	GO TO
Row of Table	ROW 7 TABLE BETA

The other form of a procedural control is the PERFORM verb. The PERFORM specifies a transfer to some destination, the execution of a table or a set of sentences at that destination, and a return to the action block following the PERFORM. The sentences or tables acted upon are by definition a "closed procedure" - i.e., they have a single entrance point and a defined exit point. Conventions for writing closed procedures are given in the next section. Legitimate forms of the PERFORM action are

Format	Example
PERFORM Sentence Name	PERFORM GROSS PAY
PERFORM Table Name	PERFORM ERROR TABLE

The STOP verb may also be used as an action. It may be placed in either a primary or secondary block. When it is used, no other action may appear with it in the same action column. The STOP terminates processing temporarily or permanently according to what action is taken at the computer's console.

3. Input-Output Actions. Input and output actions are compiler verbs that control the flow of data to and from the computer. They read, write, and validate tape labels of data files assigned to peripheral input-output devices. When data files are referred to from an action block, they must be defined according to the environment and data division specifications listed in the General Compiler manual. The formats of input-output actions are illustrated by the following:

Format	Example
READ File Name	READ MASTER~FILE
OPEN INPUT or OUTPUT File Name	OPEN INPUT MASTER~FILE
CLOSE File Name	CLOSE MASTER~FILE
File Name READ, CLOSE, or OPEN verbs	MASTER~FILE READ
WRITE Record Name	WRITE DETAIL~LINE
Record Name WRITE	TRANSACTION WRITE

The Skip and Repeat Operators

The skip operator makes it possible to show that a condition or action is not to take part in the evalution of a row. This is done by placing a hyphen (~) in the concerned condition or action block. The compiler then will skip this block and proceed to the next.

The repeat operator is a shorthand method to indicate that a condition or action in the block above is repeated. This is shown by entering a ditto mark (") in the block below the one that is to be repeated. This notation was used with the GO TO action in the sample table of Fig. 2.

V THE TABLE AS A PROGRAM

Up until now, only components of tables were presented. It was learned in Section II that General Compiler sentences could be used to support the conditions and actions of tables, and the preceeding section mentioned tables as closed procedures. This sect on relates these topics to tables and tables to compiler programs.

Block Conventions for Writing Expressions

1. Words, abbreviations, and symbols of the compiler's vocabulary should not be used as names. They may be combined with other characters to form names.

2. The words in an expression should be separated by at least one space. More than one space is permitted. The space separator is optional if the words are bound by

+ - * / ** ~ ( ) " = , |

3. Subscripts should be enclosed in parentheses. They may be written adjacent to (without a space separator) or apart (with space separators) from their associated data names. Individual subscripts in a list of subscripts should be separated by commas.

4. When two arithmetic expressions appear side by side as in a series, they should be separated by commas.

5. All columns of a table should be bound by the vertical table line, | (12-4-8 punch).

6. The skip and repeat symbols, ~ and ", should be the only entry , other than spaces, in a block.

Conventions for Placing a Table in a Program

1. Tables are written on the General Compiler Sentence Form.
2. A table is preceeded by the word TABLE. Naming tables is optional. When a table is given a name, the name may preceed or follow the word TABLE. The word

   TABLE,
   name TABLE, or
   TABLE name

   should be followed by a period.
3. The table's size is given next and should be placed on the same line as the table's name. The size may be written in one of two ways:

   kkk CONDITIONS mmm ACTIONS nnn ROWS.

   or

   (kkk, mmm, nnn).

   Both forms are terminated by a period. The order of writing the number of conditions, actions, and rows is optional in the first case since each can be identified. However, order is important in the second form since the compiler interprets the first number enclosed in parentheses as the number of conditions, the second as actions, and the third as rows. Conditions, actions, and rows are numbered sequentially beginning with 1. Row 1 is the first secondary row; the primary row is not counted in the row count.
4. General Compiler sentences should not be placed between the word TABLE and the primary row of the tables.
5. The double vertical lines that separates conditions from actions may be represented by one or two 12-4-5 punches.

6. The size of each block may vary from column to column and row to row.

7. The only limit on the size of a table is row width. Since the compiler prints a listing of compilation, the recommended row width is 120 characters including card sequence number. Maximum row width is 1200 characters.

   Since the table form is an image of an 80-column punched card, a hyphen (~) is placed in column 7 of the form to show that a row is contained on more than one card. In this case, no table column may be split across cards. Each card is to contain a sequence number to insure proper card order. When rows exceed one card, the sequence number of the first card is only printed. Sequence numbers of succeeding cards are stripped out. The row is then printed as a multiple of 120 characters with an integral number of table columns per 120 characters.

8. Expressions too long or complex to be written in blocks may be written after the table's name and size and be executed from the table by means of the PERFORM verb. In addition to expressions, any General Compiler sentence may be used and executed in this manner. To indicate the start of the table the word BEGIN is to follow the list of expressions and sentences. This format may be illustrated by the following:

```
TABLE name.  kkk CONDITIONS mmm ACTIONS nnn ROWS.

... General Compiler Sentences and Expressions - May be
    executed only from the confines of the table.

BEGIN
```

DECISION TABLE

Closed Procedures

Fig. 4 outlines the format of a closed procedure. By definition a closed procedure may be acted on only by the PERFORM verb. It contains one entrance point and one exit point. In fig. 4 these are indicated by the words BEGIN and END TABLE name. BEGIN and END also act as sentence names and may be referred to from within the procedure body.

Expressions too long to be placed in the blocks of a table may be written in the procedure head and executed from the procedure body by means of the PERFORM verb. As such, they must be given names. In addition to expressions any General Compiler sentence may be written in the head and executed accordingly.

The procedure body contains the table. As shown in Fig. 4 compiler sentences may preceed and follow the table. Execution is sequential starting with the sentence or table after the word BEGIN and proceeds until the exit END TABLE is reached. It is at this point that control is reverted to the PERFORM verb which originally referenced the procedure. Any unconditional transfer from within the procedure to the outside is undefined. However, PERFORM verbs in the body may reference other closed procedures.

Closed procedures should be written apart from the main program.

DECISION TABLE AS A CLOSED PROCEDURE

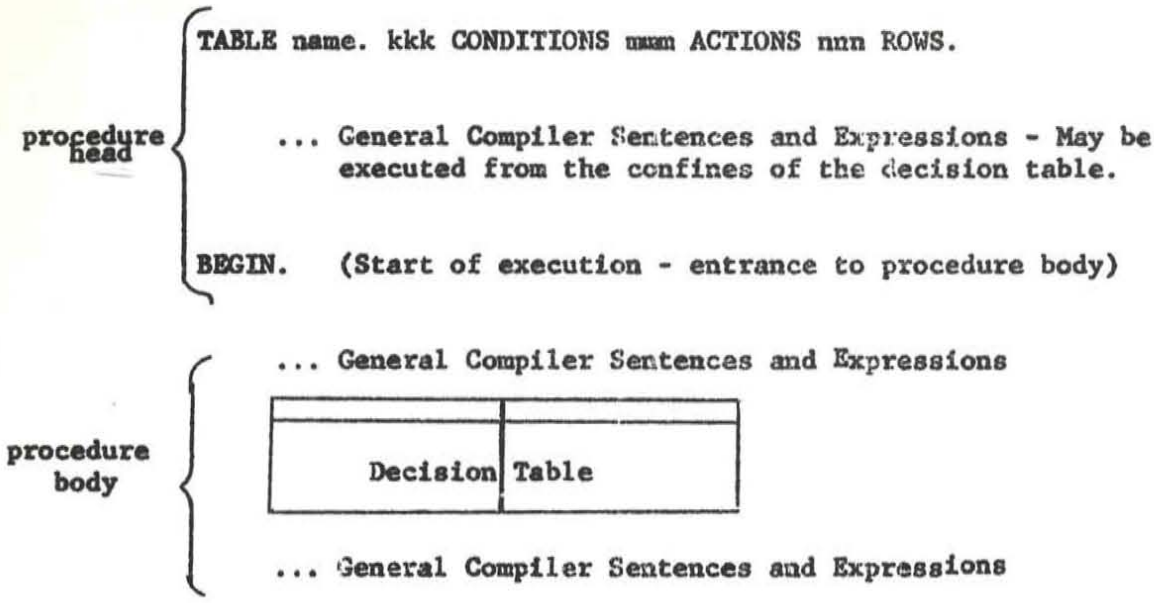


Fig. 4

CODASYL SYSTEMS COMMITTEE

TRANSFORMATION LOGIC

December, 1960

Burton Grad
IBM

TRANSFORMATION LOGIC

The following report proposes a structure for detailed analysis and formulation of the Transformation Logic used in applying Tabular Form to a precise systems language. It suggests major study topics and then divides the Table construction area into a series of specific subjects. Each of these subjects then has possible solutions described, certain problems stated and indicates the related portions of previous reports. It is our hope that this framework will provide a basis for intensive future work and for relating our work with that of the Language Structures Committee.

We feel that there are three major study topics for work in Tabular techniques:

I Table Construction and Data Description
II Applications
III Language Implementation

At the present we are not particularly concerned with Language Implementation (III), but have been concentrating on Table Construction (I) so that appropriate Applications Studies (II) may be carried out.

In subdividing the Table Construction part of the first topic we believe there are three key subjects:

A. Inside - Box Considerations
B. Within - Column Logic
C. General Table Considerations

We will discuss each of these subjects in turn indicating specific areas of work, appropriate solutions, etc.

Subject (A) Inside - Box Considerations:

(1) Operators:

We have identified certain general element operators:

(a) quantitative (+, -, *, /, power, absolute, sine, square root, etc.). These operators may require 1, 2 or more quantitative factors as input and result in a quantitative value.

(b) true-false (Andalso ($\wedge$), Andor ($\vee$), Or ($\nleftrightarrow$), Not ($\neg$)). These operators may require 1 or 2 true-false factors as input and result in a true-false value.

(c) string (Concatenate, Insert, Replace, Substring, etc.). These operators may require 1, 2 or more string factors as input and result in a string value.

(d) mixed (Transform, Convert, Count, etc.) These operators typically require either 1 string factor or else 1 quantitative factor as input and then result in one quantitative factor or else one string factor... the opposite of what was available to start.

(e) multiple (not yet explored) These operators would use single or multiple input factors to establish multiple result values.

There are also a number of Relational Element Operators:

(f) special (Exist, Defined, Non-existent, Undefined). These operators establish whether a particular element has an established value or whether the value assigned has defined meaning in terms of the Data Element Description. The input is typically a single factor and the result is a true-false value.

(g) quantitative relational (Less than, Greater than, Equal, and their negatives). These operators require 2 quantitative factors as input and result in a true-false value.

(i) string relational (Identical, Not Identical, Lower, Higher). These operators require 2 string factors as input and result in a true-false value.

There are Set manipulative operators:

(j) general (arrange, extract, join, etc.). These operators require 1, 2 or more sets as input and result in a new set.

(k) complex (explode, update, post, etc.). These operators change values of elements as well as change set membership. 1 or more sets and/or elements are required as input as either a set or an element may result.

(l) Set relational (Equivalent, Identical, Subset and their negatives). These operators require 2 sets as inputs and result in a true-false value. Specifically they will examine these two sets for identical values, identical order, element name similarity, etc. It may be necessary to differentiate the operators for unordered sets from those for ordered sets. This point requires further study.

There are other operators which may be of particular significance:

(m) Change value or set membership (Assign, Copy, Communicate, Receive, Transmit, etc).
These provide for the specific association of a particular value with an element name. It may be possible to change the value or else it may be regarded as a "permanent" value assignment.

(n) Definition (Define)
This operator provides for a substitution of some other factor for an element or set name whenever it is referred to.

(o) Sequence control (Goto, Interrupt, Stop, Perform, Come from, Prior rule, Start, etc).
These are inherently "procedural" operators which do not themselves change data element values, set memberships or establish conditional logic; however, their importance lies in their ability to break up a highly complex description into an understandable group of simpler descriptions. They provide a convenient way to subdivide system logic and a shorthand for indicating conditional repetition or order of action execution.

The operators in each category will be suitably extended and arrangements will be made so that new or specialized operators may be defined in terms of the basic operators predefined by the Language Specification. It is expected that each operator definition in the language specification will explicitly denote the representation of the operator (words, symbols, abbreviations), the type and number of input factors required and/or permitted. The type of result will be indicated and also any parameterization allowed. In other words each operator will have a full definition sheet with suitable examples of its use.

(2) Factors:

We have identified four value classes which factor value can assume.

(a) quantitative -- arithmetically manipulatable values regardless of number base, radix point or graphics. (See B. Grad, July 11, Section C, pg. 7).

(b) true-false -- boolean values. (See B. Grad, July 11, Section C, pg. 7 and 8).

(c) string -- any ordered symbols other than true-false or quantitative. If a quantitative symbol (e.g. 1, 7) is used in a string it has a different meaning from that same symbol used in a quantity. (See B. Grad, July 11, Section C, pg. 8).

(d) set -- a collection of values each of which may be of any of the three types. A set does not of itself have a value in the sense that it cannot be directly manipulated by quantitative, true-false or string operators. The values of the set elements may be associated with element names but do not have to be. There is a reference order for a set though this has nothing to do with actual physical sequence.

There are three basic ways of referring to a particular value: by a Literal, Name or Expression. The meaning and use of these terms is described in B. Grad, July 11, sect. C, pages 8 and 9. The definitions given may be extended to incorporate Set Literals, Set Names and also Set Expressions. Literals must be easily differentiated from Names or Expressions. This needs further exploration.

Rules for the formation of Expressions using various operators should be stated under the appropriate value type category. In addition each of the value type definitions and rules for using Literals and forming Names should be more clearly spelled out. Furthermore the Set-Element Description should provide a convenient means for establishing value type and also specific value to be associated with a particular Name. The development of this Description Sheet should supply effective definitions for many of the terms used under Factors.

There are other types of Factors which need to be considered. For instance we may find it convenient to name or otherwise identify tables, rows, columns, particular conditions or particular actions. The ability to name subroutines or functions, equipments, or various associated physical objects (e.g. machine tools, personnel, locations) may have a strong impact on the communicability of any systems language. The comments in ALGOL-60 in reference to Labels and the COBOL, Commercial Translator, FACT and Flowmatic discussions of names may prove of value to us. Certainly we must explore set names, name qualification of elements and even the possibility of using jargon names.

For example is it possible that we can communicate at the jargon level with data processing machines -- can a machine (intuitively) understand the difference between a "report" and a"graph"?

(3) Conditions:

Certain Boxes will only be able to accept Conditions as their content. These will be called Condition Boxes. A Condition is defined as a properly constructed group of operators and factors which can be determined to be either satisfied or unsatisfied, i.e., whose condition value can be determined. The comments in COBOL, April 1960 page V-2 (except for the last paragraph) are appropriate. Evaluating a Condition does not change the value of any factor involved in such an evaluation.

A Condition Box consists of three things: Condition Operators, Factors, and Condition syntax or structure.

There are only certain Condition Operators. We identify particularly the various relational operators, (f, g, h, i, l).

One Factor is worthy of special mention; this is the Condition Name which represents a Conditional Expression. This Name is used as though the Expression were substituted for the Name. The Name representation itself cannot be described in a Condition Box; it must be done with a Define Operator either in a Define Box or in the Data Description.

The internal structure of a Condition Box permits a great deal of experimentation. It may, however, be desirable initially to limit the variations in the interest of simplicity and clarity. It will certainly be possible later on to add further sophistications to the Condition Box Structure. The following comments and examples are intended to provide only a fundamental structure.

An evaluated Condition will be either satisfied or unsatisfied. If a particular Condition Box statement is met or is "Not Pertinent" to a Decision Rule then the Condition Box is said to be satisfied. If the statement is not met or if undefined values are related to defined values then the Condition Box is said to be unsatisfied.

A Condition Box statement must always be totally satisfied or else it is unsatisfied. A simple Condition is one which consists of a single Condition Operator and the appropriate number of factors. Simple Conditions can only be compounded within a Box by the proper use of connectives between legitimate simple Conditions. Implied repetition of any factor or operator is <u>not</u> permitted within a Box. True-false factors do not receive special treatment within a Condition Box.

The entire issue of Connectives is quite up in the air. They are not the same as true-false operators, nor do they exhibit the same properties. However the English language equivalents for the true-false operators happen to be essentially the same as the primary Connectives which we would like to use. One solution would be to restrict the true-false operators to symbolic representation (e.g. $\wedge, \vee$, etc.) and reserve the English words for Connectives; an alternative would be to use special punctuation symbols like comma, semi-colon, etc. to represent the Connectives and keep the English words for true-false operators. This will obviously require further work before any firm proposal can be made.

The various artifices suggested in B. Grad, July 11, Section C, page 10 should be ignored in terms of Inside-Box Condition construction. These problems will be discussed under General Table Considerations. Reference is made in this context to D. Nelson, August 17.

Examples of valid Simple Conditions are:

Y = Y

4 / 2 = 4

PAY. CODE = 6

MARRIED (e.g. MARITAL. STATUS = 3)

X $\leq$ (A / 4)

3 > 2

(A andor X) = Y

((A / X) / 3) = PAY. CODE

DATE = 123160

NAME not = "GEORGE"

(MONTH concatenate DAY) higher than 0228

"WHITE" is subset of SNOW

FILE. A is equivalent to FILE. C

(4) Actions:

Action Boxes serve to change values. There are apparently two major types of special Action Operators: Assign and Communicate. Assign is described in B. Grad, July 11, Section C, page 11. It is executable and provides that a Name will retain the assigned value until it is changed by a new Assign. The sequence of Assign Boxes may be pertinent to the use of the values. Assign also permits Actions of the form:

Assign J as J / 1, where a Name is given a value as a function of a previously assigned value for that same Name. Assign provides for quantitative, true-false or string value manipulation as well as the establishment of appropriate set values.

The question is raised as to whether there should be a special Assign operator for modifying factors inside a Box. There are many pros and cons to this issue which should be explored in depth before making even a tentative decision.

The second type of special Action Operator is Communicate. One solution to this is described by Mal Smith, July 1960 where he suggests the use of Receive and Transmit as the particular Operators in this class.

Action Names would be used to represent functions or tables. It is probably desirable that we permit parameterization of Action Names. This could also be interpreted so as to permit a function which generates multiple values rather than just one value. However, opening this loophole would automatically indicate resolution of the question as to whether a single Action Box can be used to establish the value of more than one factor Name.

The construction of Actions seems quite straightforward. The rules for using Action Operators are such that we consider that a value will be established for a Name by evaluating an expression, named function or other factor. Basic questions have to do with whether to allow multiple value assignment within an Action Box. This also leads to consideration of what connectives to use to indicate independent versus non-independent value assignments. For instance, we might specify that a series of Actions separated by a semi-colon must be executed in the order specified while those separated by periods may be in any sequence. This entire issue of independence and dependence will be discussed in the Between-Box portion of this paper. However, under the category of Action Boxes we need to resolve the question of permitting compound Actions.

Examples of valid Simple Actions are:

assign PAY. CODE as 336611

assign WEEK. PAY as HOURS * RATE

assign MARRIED as MARITAL. STATUS identical 1

assign LINE. A as SET. B

receive PAY. DATA from TIME. CARD

transmit CHECK. INFO to PAY. CHECK

(5) Definitions:

Definition Boxes serve to associate a Name with a value generating factor. The basic Operator is DEFINE which is not executable per se. Rather it imputes a substitution. Whenever a Defined Name is used in an Action it will have the Definition substituted for it. This can also be viewed from the standpoint that a Defined Name will be evaluated in terms of the underlying or "root" values each time it is used. Suppose we have the following Boxes:

define K as P / [7]

assign R as K * R

The Define Statement would indicate that K is not to be evaluated except when it is used; and when it is evaluated it is in terms of the then current value of P. In a Define statement the same Name cannot appear in both the "subject" and "predicate". A definition is persistent throughout a system description, but the value is not. In contrast, an Assignment results in a particular value for a Name which can only be modified by another Assignment. Definitions can be nested so that P might in turn be defined in terms of Q, and so on. The point at which a definition appears is of no importance. It is always treated as though it occurred at the very beginning. Definitions should occur only in Unconditional Tables since otherwise there would be the possibility of their being overlooked in a particular solution path. The problem of compound define statements does not seem to arise except in conjunction with multiple synonyms.

(6) Sequence Control:

The final area of Inside - Box discussion is concerned with expressing sequence control. This may be concerned with column selection, Table selection, etc. Reference is made to M. K. Hawes letter August 16, 1960 in which she speaks of Goto, Perform, Halt and Stop. In this class should also be considered Prior Rule. A Sequence Control statement does not change any values, but it is of critical significance to the effective description of the logic of a system.

To support Sequence Control it is necessary to be able to name certain control points in the transformation logic description. In Tabular form this requires identifying a Table Name (numeric, mnemonic or descriptive) or even a particular entry point in a Table; it also may require explicit designation of individual Boxes (probably through a column-row numeric code).

The simplest Sequence Control Operator is Goto, which states unequivocally that logic control should next proceed to the designated named location. It can be used as a short-hand way of indicating the repetition of certain Conditions. This same effect can be produced by the use of a Prior Rule designation. Implicitly, this subsumes (or repeats) all previous conditions preceding that particular branch. This is a highly important convenience because without it we would have to go to very large, highly qualified Tables; with this sequence control ability we can break a complex logic into a series of smaller problems. It is of course essential to the systems planner that he keep careful track of all logical conditions indicated by the chain of Goto's or Prior Rules. This in itself can be a complex problem and may also require the use of Tabular form to maintain logical understanding and control.

Two other Sequence Control Operators are quite simple; these are Halt and Stop. In the first case we provide for a planned interruption for manual intervention--maybe to add a new factor or make a non-formalized decision. This would provide for operational control as in a Man-Machine Simulation of a system. Stop concludes the process and indicates that the transformation logic has been fully defined.

The fourth Sequence Control Operator is Perform. It is used to represent the idea of "Goto and Return". There is a serious question as to the need for this device in a systems planner's Transformation Logic. Basically it is a device for cascading levels of Tables. In other words, in a Box we can indicate by a Perform statement that a whole set of Conditions and Actions are to be carried out. This then is a convenience device designed

to avoid complex elaboration of a particular path within a certain Table or else a means for "repeating" a standard subroutine in many Tables. These are the two major concepts -- (1) an in - line part of the Transformation Logic which could have been connected by a Goto the indicated sub-area and then a Goto back to the main - line and (2) a sub-routine (parameterized or not) which could only be handled by the main - line Table having preset a return instruction before using the Goto. There is, however, an alternative made available by the combined use of the Define and Assign Operators. The Subroutine can be Named and appropriately Defined. Then through a multiple Assign (with or without parameterization) the subroutine can be executed with provision built-in for automatic continuation of the main-line table. With this alternative there is a reasonable likelihood that the Perform Operator will not be needed at all.

(7) Summary

Inside-Box Criteria require an understanding of the various types of boxes which the following have been identified: Condition, Action, Definition and Sequence Control. Each of these boxes may contain a suitable statement consisting of appropriate Operators and Factors as required for that type of box. Operators are a means of transforming Factors: Sets and Elements. A Factor may consist of Literal, Name or Expression. Careful consideration will have to be given to each of these identified topics and adequate definitions of each specific item will be needed.

Subject (B) Within-Column Considerations:

Certainly one of the most potent reasons for using a tabular form for recording transformation logic is the ability to relate one box to another visually; it permits almost automatic implication of certain "noise" words that are required in normal sentence construction. Further, the basic value of the table form comes from the ability to readily associate conditions with actions and to compare alternative sets of conditions or alternative courses of action. So while there are some advances in the Inside-Box concepts mentioned under Section A, we begin to see the basic advantages of table form as we examine Within-Column possibilities.

Essentially, the table form permits ready visual aggregation of conditions or actions and visual relationship between groups of conditions and groups of actions. In every example seen to date the evaluation of a group of conditions (in a single row or column) directly indicates whether or not to carry out a group of actions (in that same row or column).

For convenience, we will talk about a decision rule being expressed in a column rather than a row, though of course the two forms are equivalent. To organize the further discussion on this topic we will sub-divide the subject as follows: (1) Among Condition Boxes; (2) Among Action Boxes; (3) Between Conditions and Actions; (4) Among Conditions, Actions, Definitions and Sequence Control.

(1) Among Condition Boxes

Each Condition Box can be individually evaluated and each will be satisfied or not satisfied as a result of this evaluation. We can therefore relate each box to the whole and indicate whether the whole group of conditions is satisfied. In the simplest case we speak of independent, required Condition Boxes. Independent means that no condition is a function of another; more simply, evaluating one condition has no effect on any other condition. Required refers to the fact that satisfaction of all Conditions is necessary; this can be thought of as "if C_1 and also C_2 and also C_3", where all three Conditions must be satisfied to satisfy the whole.

We can make this more complex by permitting other logical connectives between Conditions like "Andor", or "Or". We could also develop "best 2 out of 3" or "at least 3 satisfied" rules. In the simplest case (if...and also...) the conditions can be examined in any order without influencing whether the group of conditions will be satisfied. In more complex cases this is no longer true and the exact sequence of testing could affect the decision. Although it is logically and technically possible to handle these more complex cases it is recommended that initial consideration be given solely to independent, required Conditions and that only after this is clarified should we be concerned with more advanced approaches. There is a sound reason for this recommendation: The basic power of table form rests on visual relationships; Complex patterns are not visually easy to follow or conceive and hence may well destroy the original reason for going to tabular form. Because of the common need to handle "exclusive or" it should be noted that its incorporation in the Inside-Box concept would provide for the necessary flexibility; e.g., (Inside a Condition Box) MARITAL STATUS is MARRIED or MARITAL STATUS is HEAD.OF.FAMILY. It may even be desirable to offer a short-hand notation for this one special case if it occurs frequently enough; e.g., MARITAL. STATUS is MARRIED or is HEAD OR FAMILY.

(2) Among Action Boxes

Implicit in handling a group of actions is the connective "and then". There must often be a stated sequence of action performance since it is likely that a particular action may affect the value of a factor which is used in a subsequent action. The simplest case is achieved when written sequence (top to bottom) is maintained. Given any series of actions it is clear that they can be written in proper sequence from top to bottom. Questions about explicit sequence indication arise because of attempts at redundancy elimination or for row association. These will be discussed under General Table Considerations.

The problems which arise among actions are concerned not with indication of implicit sequence, but rather with those actions which need not be done in order, but are automatically sequenced in a column approach. This same objection is valid for flow charts, narrative languages and machine-oriented languages. The question can be stated: how can we use sequential language and still indicate that two or more items need not follow each other. No solution is suggested here though certain possibilities are explored briefly. We could have a rule that between action boxes there was no sequential relation except as indicated by common use of a certain factor, in which case written sequence would hold. We would have to explore whether action to action (necessary) sequence can always be logically determined; i.e., the need for sequence, no what the proper sequence is. We should also investigate the usefulness of explicit sequence indication. Another approach is to provide a precedence matrix or precedence chart as shown by Barankin's work on Precedence Matrices and Salveson's paper on Assembly Line Balancing. A third approach might be to use some special line weight or symbol to indicate that the preceding actions must be performed prior to carrying out the following action(s).

It is recommended at this time that we only deal with tables with the following property: all actions in a column are to be performed if any actions in that column are to be performed.

One other interesting idea is the possibility of compacting actions by allowing within an action box a complete loop statement like assign X as X / I for I from 1 to 10 by 1. This would allow a summarization to be explicitly indicated within a single box and would avoid certain types of Goto and condition testing.

(3) Between Conditions and Actions

In virtually all illustrative tables the connection between conditions and actions has been simply, "if...then...". If the stated conditions are satisfied, then execute then actions specified. This may well be the standard and most significant need for tables, but at least one other logical possibility should be explored: "if...then do not do...". This would provide for editing and error correction since as long as various conditions are satisfied, no special action need be taken.

One recommendation is that present studies concentrate on tables where satisfying the various conditions in column will always result in that column's actions being carried out. We should also try to work with "cause and effect" relationships or with functional relationships. Reliance upon incidental or happenstance relations is highly suspect in that it may make table acceptance and maintenance unnecessarily difficult.

As far as deciding whether to execute a group of actions, the Among-Conditions control determines if the Condition-column is satisfied. The knowledge of whether a Condition-column is satisfied determines whether the corresponding action-column is to be executed.

An interesting development of Condition-Action relations can be observed through the following formula.

let N = Total number of conditions in a particular column

and S = number of satisfied conditions in that column after evaluation.

then there seems to be 5 possible situations

(1) if $S = N$ then...

(2) if $S \geq 1$ then...

(3) if $N - S \geq 1$ then...

(4) if $N - S = r$ then...

(5) if $p \leq (N - S) \leq r$ then...for $p \neq r$

These can be reformulated as follows:

General Case:

if $p \leqslant (N - S) \leqslant r$ for $p \leqslant r \geqslant 0$

(1) $p = r = 0$

$0 \leqslant N - S \leqslant 0$

(2) $p = 0$; $r = N - 1$

$0 \leqslant N - S \leqslant N - 1$

(3) $p = 1$; $r = N$

$1 \leqslant N - S \leqslant N$

(4) $p = r > 1$

$r \leqslant N - S \leqslant r$

(5) $p \neq r$; $0 \leqslant p \leqslant N$; $1 < r \leqslant N$

$p \leqslant N - S \leqslant r$

What this reformulation means is that for each table (or even for each column in a table) we could specify a "p" and an "r" and this would fully describe the logic for carrying out the actions in that column.

(4) Among Conditions, Actions, Definitions and Sequence Control.

The first point we might recognize is the usefulness of an "unconditional" column. This says that the actions, definitions, etc. in that column are to be carried out always. This can be done in a one column Table (the "degenerate" form) or by its use as the "last" column in a table (an "all others" column) or in a table with multiple sets of actions permitted it could be any column (a "must" column).

In general Goto should occur after all other actions have been specified, hence it may be desirable to require Goto operations in the last row of a Table. Prior Rule operations should occur as the first row in a Table. The question of sequence control will be discussed more fully under Subject (C) General Ta ble Considerations. We only want to note that a column will need to be able to distinguish at least these basic relationships: must come directly from; must come from one of these; must previously have been considered (and of course the go to complements).

Burton Grad
December 29, 1960

WORK PROBLEMS

Extended Entry Tables

TABLE HEADER	
CONDITION STUB	CONDITION ENTRY
ACTION STUB	ACTION ENTRY
ENTRY HEADER	

TABLE HEADER - The name of the table appears here.

CONDITION STUB - A data name and a relational operator may appear in this section of the table.

data-name $>$ data-name $<$

data-name $\geq$ data-name $\leq$

data-name $\neq$

The absence of a special symbol implies equality (=).

CONDITION ENTRY - A data name, literal (enclosed in quotes, "____") or an arithmetic expression may appear in the conditional entry. Each of these entries is horizontally associated with the condition stub. Thus, a complete relational expression evolves, e.g., QUANTITY ON ORDER $\geq$ QUANTITY ON HAND.

The last column may be used to cover all other cases if all alternatives are not covered by the other conditional expressions. The column is marked "A/O".

ACTION STUB - A name or an operator is an acceptable entry in this section. The following operators are permitted in the action stub.

OPEN	READ
CLOSE	WRITE
DO	

As will be seen below, the operands are placed in the action entry.

When the replace operation (SET =) is desired, only the data name need be entered in the stub.

ACTION ENTRY - Names, literals and arithmetic expressions may appear in the action entry. As with the conditional entries, horizontal association is assumed. A single operator is permitted: the + symbol before a data name or literal will be interpreted as the increment operator. The value represented by the entry will be added to the value represented by the name in the stub.

ENTRY HEADER - The entry header spans the action stub and the action entry. The last row of the former must contain a GO TO operator; the last row of each column of the latter must contain a table name or number indicating the next table to be considered.

EXAMPLE:

TABLE: _____				
LOCAL TIME (EST)	≥ 2:00	≥ 1:00	≥ 3:00	≥ 1:00
	≤ 12:59	≤ 2:00	≤ 12:59	≤ 4:00
DESTINATION TIME ZONE	CST	CST	MST	PST
DESTINATION TIME	Local Time -1:00	Local Time +11:00	Local Time - 2:00	Local Time +9:00
DESTINATION MERIDIAN	Original		Original	
GO TO	End	change meridian	End	change meridian

Problem 3

Prepare appropriate extended entry decision table(s) according to the following rules:

Given:

A. Field Names

Employee name
Department number
Hourly rate
Hours worked
Deduction code (A, B, C, D)
Sex (MALE, FEMALE)

Obtain:

A. Select all males who satisfy the following conditions.

1. They must work in Department 47.
2. Weekly hours not over 40.
3. Must have a deduction code "B" or code "D".

B. Select all females that satisfy the following conditions:

1. They must work in department 48, 49 or 50.
2. Weekly hours not over 40.
3. Must have a deduction under code "C" or hourly rate must be more than $2.50.

C. If section A is satisfied - do routine 1.
If section B is satisfied - do routine 2.
If neither A nor B are satisfied - do routine 3.

Problem 3 Solution

Sex	"Male"	"Male"	"Female"	"Female"	"Female"	"Female"	"Female"	"Female"	All/ others
Depart-ment	47	47	48	48	49	49	50	50	
Weekly Hours	$\leq$ 40	$\leq$ 40	$\leq$ 40	$\leq$ 40	$\leq$ 40	$\leq$40	$\leq$40	$\leq$40	
Deduction Code	"B"	"D"	"C"	—	"C"	—	"C"	—	
Hourly Rate	—	—	—	> 2.50	—	> 2.50	—	> 2.50	
Go to	Rou-tine 1	Routine 1	Routine 2	Routine 2	Routine 2	Routine 2	Routine 2	Routine 2	Routine 3

Problem 3 solution

Table 1

Weekly Hours	≤ 40	≤ 40	all/ others
Sex	"Male"	"Female"	↓
Go To	Table 2	Table 4	Routine 3

Table 4

Deduction Code	≠ "C"	all/ others
Hourly rate	≤ 2.50	↓
Go To	Routine 3	Table 5

Table 2

Depart-ment	47	47	all/ others
Deduction Code	"B"	"D"	↓
Go To	Routine 1	Routine 1	Routine 3

Table 5

Department	48	49	50	all/ others
Go To	Routine 2	Routine 2	Routine 2	Routine 3

Problem 4

Develop extended entry decision tables according to the following information and rules.

A. Classification of capital gains and losses. -The phrase "short-term" applies to gains and losses from the sale or exchange of capital assets held for 6 months or less; the phrase "long-term" applies to capital assets held for more than 6 months.

Treatment of capital gains and losses - Short-term capital gains and losses will be merged to obtain the net short-term capital gain or loss. Long-term capital gains and losses (taken into account at 100 percent) will be merged to obtain the net long-term capital gain or loss.

1. Given: Purchase date
   Sales date
   Net sales price
   Net cost

2. Obtain: Total long-term result
   Total short-term result
   Type of long-term result (gain, loss)
   Type of short-term result (gain, loss)
   Net result.

Problem 4 solution

Table 10

—	—	
Elapsed Time	Sales Date - Purchase Date	
Result	Net sales price - net cost	
Go to	Table 11	

Table 11

Elapsed time	> 6 months	≤ 6 months
Total long-term result	+ result	—
Total Short-term result	—	+ result
Net result	+ result	+ result
Go to	Table 12	Table 12

Table 12

Next Item	Exists	Does not exist
Read	Next Item	—
Go to	Table 10	Table 13

Table 13

Total long-term result	≥ 0	< 0
Type of long-term-result	"Gain"	"Loss"
Go to	Table 14	Table 14

Table 14

short-term result	≥ 0	< 0
type of total short-term result	"Gain"	"Loss"
Go to	Income Calculator	Income Calculator

Problem 5

Prepare an extended entry decision table according to the following information and rules.

A. If the net short-term capital gain exceeds the net long-term capital loss, 100 percent of such excess shall be included in income. If the net long-term capital gain exceeds the net short-term capital loss, 50 percent of the amount of such excess is included in income.

If the sum of all the capital losses exceeds the sum of all the capital gains (all such gains and losses to be taken into account at 100 percent), then such capital losses shall be allowed as a deduction only to the extent of current year capital gains plus $1,000. The excess of total losses over the allowable losses is called "capital loss carryover". Except as noted above 50 percent of all long-term gains and 100 percent of all short-term gains are to be included as income.

1. Given: Total long-term result
   Total short-term result
   Type of long-term result (gain, loss)
   Type of short-term result (gain, loss)
   Net result

2. Obtain: Income
   Loss Carryover.

Problem 5 solution

	Type of long-term result	"Gain"	"Loss"	—	—	"Gain"
	Type of short-term result	"Loss"	"Gain"	—	—	"Gain"
	Net result	—	—	≤ 0	—	—
	Net result	> 0	> 0	≥ -1000	< -1000	—
	Income	.5 Net result	Net result	Net result	-1000	(.5 Total long-term result) + Total short-term result
	Loss carryover	—	—	—	Net result + 1000	—
	Go to	End routine	End routine	End routine	End routine	End routine

EXAMPLES OF TABLES

W. L. Myers
Eastman Kodak
Rochester, N.Y.

September, 1960

Examples of Tables

1. Simple Mutually Exclusive Conditions and Actions:

This type of information does not seem to be aided by the Table System. However, it does tend to pull together all of the possible conditions and possible actions in a formal fashion so that any future variations can be analyzed and explained easily.

The pattern of "Y's" on such a Table usually appear as follows:

	Rules		
Condition	01	02	03
A	Y		
B		Y	
C			Y
Action			
1	Y		
2		Y	
3			Y

2. Several Actions in Various Combinations:

The next most frequently used Table is the one showing several actions to be taken in various combinations based on mutually exclusive conditions.

This type of Table might appear as follows:

	Rules		
Condition	01	02	03
A	Y		
B		Y	
C			Y
Action			
1	Y		Y
2		Y	
3	Y	Y	
4		Y	
5			Y

3. Conditions and Actions Not Mutually Exclusive:

The major value of the Table is for describing series of conditions and actions which are not mutually exclusive. Statements describing these situations in English are quite lengthy, and are difficult to analyze for completeness of the effect of a proposed change.

The Table for this type may appear as follows:

	Rules							
Condition	01	02	03	04	05	06	07	08
A	Y	Y	Y	N	N	N	Y	N
B	Y	Y	N	Y	N	Y	N	N
C	Y	N	Y	Y	Y	N	N	N
Action	01	02	03	04	05	06	07	08
1	Y							
2		Y	Y				Y	
3		Y		Y		Y		
4			Y	Y	Y			
5						Y	Y	
6								Y

The use of the Tables can best be understood through practice with a series of problems.

Problem # 1

Merge two tapes (or decks of cards) which are in order by Stock No. in ascending order. There may be more than one item for each Stock No. on either tape. Call the two input tapes # 1 and # 2, and the output tape # 3. It is helpful to first write the Actions at the bottom of the Table, and then fill in the Conditions. The two Actions desired are:

1. Output the record from tape #1 and read another from tape #1
2. Output the record from tape #2 and read another from tape #2

Problem # 1 (cont.)

Only one test is necessary:

"A" Is the Stock No. of tape #1 lower than the Stock No. of tape #2

The Table will appear as follows:

	Rule	
Condition	01	02
A	Y	N
Action		
1	Y	
2		Y

This appears simple, but does not provide for the end of the job procedure so it is actually incomplete.

Problem # 2

Merge three tapes. This Table will have three Actions.

1. Output the record from tape #1 and read tape #1 again
2. Output the record from tape #2 and read tape #2 again
3. Output the record from tape #3 and read tape #3 again

There will be several tests necessary:

"A" Is tape #1 lower than tape #2

"B" Is tape #1 lower than tape #3

"C" Is tape #2 lower than tape #3

The Table would appear as follows:

	Rules			
Condition	01	02	03	Formula
A	Y	N	—	1 < 2
B	Y	—	N	1 < 3
C	—	Y	N	2 < 3
Action				
1	Y			
2		Y		
3			Y	

Problem # 2 (cont.)

It is helpful to check to be sure all combinations have been provided for. Three Conditions can occur with eight variations as follows:

	Variations							
Condition	1	2	3	4	5	6	7	8
A	Y	Y	Y	N	N	N	Y	N
B	Y	Y	N	Y	N	Y	N	N
C	Y	N	Y	Y	Y	N	N	N
Included In								
A	Y	Y	*	N	N	*	—	—
B	Y	Y		—	—		N	N
C	—	—		Y	Y		N	N
Rule No. in Table	01	01		02	02		03	03
Numeric Ex.								
Tape #1	4	4		5	6		5	6
Tape #2	5	6		4	4		6	5
Tape #3	6	5		6	5		4	4

* Not included in first Table.

As can be seen, two situations are not covered – Variations 3 and 6. Let us examine them.

Variation 3 (YNY) would mean:

Tape #1 is lower than Tape #2, and Tape #1 is higher than Tape #3, and Tape #2 is lower than Tape # 3. This condition is impossible.

Variation 6 (NYN) is similar and means:

Tape #1 is higher than Tape #2, Tape #1 is lower than Tape #3, and Tape #2 is higher than Tape #3.

The Table can help a programmer see the interplay of the different tests. For example: If "A" is "Y" (yes) then it is best to test "B" next since if it is also "Y" you have an answer. However, if "A" is

Problem # 2 (cont.)

"N" (no) then make test "C" next since if it is "Y" you will have an answer. Likewise, if you organize so "A" is used for the longest tape, "B" for the next longest, etc. the number of tests can be kept to a minimum.

Problem # 3

A classic problem is that of posting a requisition to an account, or making a partial delivery and creating a Back Order for the balance. The Table has two sections. One section is concerned with matching the Stock Number of the requisition with the Stock Number on the tape, and the other section with posting the requisition. The Actions are:

1. Read in next Stock Record
2. Transfer to Posting Routine
3. Non-match, write error message
4. Read in next Requisition
5. Reduce Stock balance by amount of Requisition
6. Reduce Stock balance to zero, create Back Order for balance
7. Return Requisition marked "No Stock"

The Conditions are as follows:

"A" Stock Record Number is lower than Requisition Stock Number

"B" Stock Record Number is equal to the Requisition Stock Number

"C" Quantity in Stock Record is not zero

"D" Quantity in Stock Record is greater than Requisition Quantity

Problem # 3 (cont.)

	Rule					
Condition	01	02	03	04	05	06
A	Y	N	N			
B	N	Y	N			
C				Y	Y	N
D				Y	N	N
Action						
1	Y					
2		Y				
3			Y			
4			Y	Y2	Y2	Y2
5				Y1		
6					Y1	
7						Y1

Again it is easy to see whether or not all conditions have been provided for. In the first pair of Conditions a rule for the simultaneous ~~occurance~~ occurrence of "Y" for both "A" and "B" has been omitted since obviously a number cannot be both lower than and equal to another. A rule for Conditions C=N and D=Y is also omitted. Again a quantity cannot be both zero and larger than the Requisition Quantity.

The numerals in the Action Section indicate the desired sequence of Actions.

WLMyers:ekw

Data Processing Service

Table Layout

Title _____

Job No. _____
Project
Program _____
TABLE _____
Date _____

Conditions	Frequency / Rule No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Formula
Actions																		

Formula Data					Remarks
Symbol	Field Name	Record No.	Incr.	Length	