表言語とその処理法について*

——汎用2次元言語の提案——

守屋慎次** 平松啓二**

Abstract

The decision tables are almost good method for analyzing and documenting systems, but it is pointed out that they cannot satisfactorily describe the program parts with loops or with actions between conditions. We propose to give the name "Table Language" on the improved decision tables in this paper.

We can perfectly improve these defects of the decision tables by adding the following two functions; one is to allow the program loops by labeling the numbers to the statements for both conditions and actions of decision tables. The other is to define two meanings, "don't care" and "neglect", to avoid the ambiguity brought by labeling the numbers to the statements. These have important meanings that the improved decision tables become one of the programming languages which can describe any algorithms into very simple tabular forms.

We have become convinced that Table Language is an exceedingly good method for not only analyzing and documenting systems but also giving the clear concept of programming language. As a result, it becomes easier to teach it to beginners. Furthermore, by defining the syntaxe of Table Language appropriately, we can obtain the general-purpose language of two dimensional programming which is compatible to the usual programming languages.

1. まえがき

複雑な判定条件 (condition) と，それに対する処理内容 (action) を，2次元的に表現するプログラム技術に，decision table[1] がある．decision table は，そのすぐれた文書性，書きやすさ，論理チェックの容易さなどの利点を持つ反面，その適用範囲が強く制限されるという欠点をも有する，適用範囲の制限というのは，アルゴリズムの中で，condition と condition の間に action を含む部分や，ループを形成する部分には，decision table がうまく適用できないことである．

ここに提案する**表言語 (Table Language)** は，decision table の持つ上記の欠点を完全に除去したものである．それは，decision table の condition と actionに，文番号（または文名）をつけることによってループを許し，condition entry のブランクに，don't care と neglect と呼ぶ二つの意味を持たせることによってambiguity が生ずるのを解決した．したがって，表言語は，形式上 decision table に文番号（または文名）をつけただけにすぎない．しかしながら，このことは，次のような重要な意味を有している．すなわち，表言語が，いくつかの condition と action によって組み合わされた任意のアルゴリズムを，きわめて簡潔な表形式で2次元的に表現できる一種のプログラミング言語を形成していることである．

後に述べるように，表言語は，decision table の機能を完全に含み，かつ，いくつかの decision table を階層構造的に結合させたものとして表現することもできる．したがって，decision table の処理アルゴリズムに多少の変更を加えるだけで，表言語の変換アルゴリズムを作成することが可能である．

表言語は，そのアルゴリズムが2次元的に表現されているため，文書性，書き易さ，論理チェックの容易さなどの特徴を持つだけでなく，初心者に対する教育もきわめて容易である．また，表言語の文法を適切に定めたならば，既存のプログラミング言語との間に

* Table Language and its Conversion to Computer Programs —A Proposal of Two-Dimensional Programming Language — by Shinji MORIYA, and Keiji HIRAMATSU (Tokyo Electrical Engineering College)

** 東京電機大学工学部

compatibility を有する汎用の2次元プログラミング言語ともなりうる.

本論文では以下の点について考察する. **2.** で表言語の一般形と一般則, 表言語と decision table との関係, それに特徴的な例題を述べる. **3.** では表言語の変換アルゴリズム, **4.** で表言語の特徴, **5.** でその適用分野, そして **6.** で Implementation について述べる.

2. 表 言 語

2.1 表言語の一般形

表言語 (Table Language, 以後 **TL** と略記する) によって書かれたプログラムを**表プログラム** (**Table Program**, 以後 **TP** と略記する) と呼ぶことにする. **TP** は, 形式的には decision table の condition と action に文番号 (または文名) を付したものとして表わされる. したがって, **TP** は一般に **Fig. 1** のように表現できる.

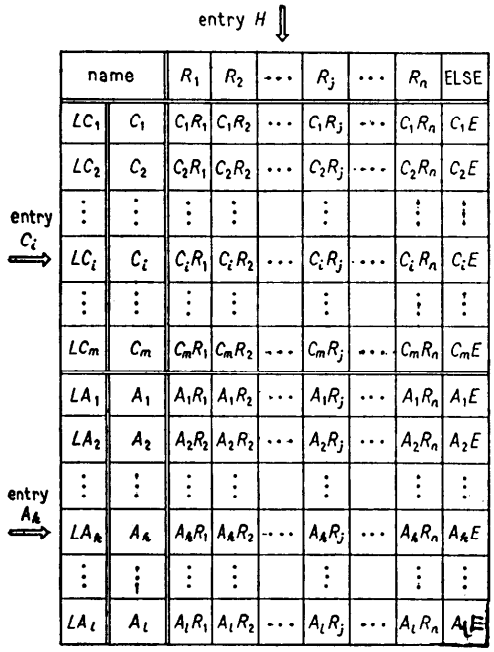

Fig 1 General format of *table program.*

Fig. 1 の **TP** を構成する各要素の意味は次のとおりである.

name: **TP** の名称.

R_j: **TP** の rule 番号. rule 数を n とする. ($j=1, 2, \ldots, n$).

ELSE: 可能な rule のうち R_j に含まれない rule のすべてを表現する一種の rule. ELSE は option とする.

C_i: **TP** の condition. condition 数を m とする. ($i=1, 2, \ldots, m$). **TP** において C_i ($i=1, 2, \ldots, m$) の書かれた部分を **condition stub** と呼ぶ. **TP** の condition stub の記法は, decision table[1] のそれに従う.

LC_i: C_i につけられた文番号 (または文名). **TP** における LC_i ($i=1, 2, \ldots, m$) の書かれた部分を **condition label stub** と呼ぶ.

C_iR_j: condition C_i に対する rule R_i の decision を表わす. また, condition C_i に対する ELSE の decision を C_iE と表わすとき, **TP** における C_iR_j と C_iE ($i=1, 2, \ldots, m$; $j=1, 2, \ldots, n$) の部分を **condition entry** と呼ぶ. **TP** の condition entry の記法は decision table[1] のそれに従う.

C_iE: 何も書かれない.

A_k: **TP** の action. action 数を l とする. **TP** における A_k ($k=1, 2, \ldots, l$) の部分を **action stub** と呼ぶ. **TP** の action stub の記法と意味は, 後述する規則12を除いて, decision table のそれに従う.

LA_k: A_k につけられた文番号 (または文名). **TP** で LA_k ($k=1, 2, \ldots, l$) の書かれた部分を **action lable stub** と呼ぶ.

A_kR_j, A_kE: R_j または ELSE に対する A_k の実行に関する記述を表わす. **TP** における A_kR_j ($k=1, 2, \ldots, l$; $j=1, 2, \ldots, n$) の部分を action entry と呼ぶ. **TP** の action entry の記法と意味は, 後述する規則12を除いて decision table のそれに従う.

entry H: **TP** への入口の一つで, decision table の持つ唯一の入口と同一である.

entry C_i: GO TO 文などによって C_i に制御が移る場合の入口を示す.

entry A_k: GO TO 文などによって A_k に制御が移る場合の入口を示す.

次に, **TL** によるプログラミングの機能を拡大するため, **TL** に次の種類を置く.

(イ) open 型表言語
(ロ) closed 型表言語

さらに

(a) 制限エントリ (Limited Entry) 表言語.
(b) 拡張エントリ (Extended Entry) 表言語.
(c) 混合エントリ (Mixed Entry) 表言語.

上記（イ），（ロ）と（a），（b），（c）の意味と記法は，decision table[1] で用いられる場合と同様であるから，ここでは述べない．**TL** では，使用範囲が拡大されて（イ），（ロ）と（a），（b），（c）を互いに組み合わせて使用できるものとする．

2.2 表言語の一般則

ここでは，**TL** の一般的な規則を述べる．Hollerith constant としてのブランクの，任意個数の string を "blank" と書き，文番号（あるいは文名）を "label" と表現する．また，たとえば，LC_i として文番号(あるいは文名) が書かれていることを "LC_i=label" と表わし，書かれていないことを "LC_i=blank" と表現する．**TP** の他の構成要素についても同様の記法を用いる．

[規則1] condition entry における blank には，**"don't care"** と **"neglect"** の2種類の意味があり，一つの blank が別々の時点で二つの意味を持つことができる．しかし，同時に二つの意味を持つことはできない．

いま，$i=1,2,\ldots,m$; $j=1,2,\ldots,n$ とする．このとき，次の規則を設ける．

[規則2] $(\forall i)(\forall j)(LC_i=\text{blank} \cap C_iR_j=\text{blank} \Rightarrow C_iR_j=\text{don't care})$

[規則3] C_iR_j=don't care のとき，blank である C_iR_j は，C_i の判断に対する論理値(Y または N)のいずれをとってもよい．

entry H が生ずることを "H=entry" と表現する。同様にして，"C_i=entry"，"A_k=entry" を表わす．

[規則4] $C_1=\text{entry} \Leftrightarrow H=\text{entry} \cap LC_1=\text{label}$

[規則5] $(\forall i)(\forall j)(C_i=\text{entry} \cap C_iR_j=\text{blank} \Rightarrow C_iR_j=\text{neglect})$

[規則6] C_iR_j=neglect のとき，(そのような blank C_iR_j の置かれた) rule R_j を選択してはならない．

$p=i+1, i+2,\ldots,m$ とすれば

[規則7] $(\forall i)(\forall j)(\forall p)(C_i=\text{entry} \cap C_iR_j \neq \text{blank} \cap C_pR_j=\text{blank} \Rightarrow C_pR_j=\text{don't care})$

[規則8] $(\forall i)(\forall j)(H=\text{entry} \cap LC_1=\text{blank} \cap C_iR_j=\text{blank} \Rightarrow C_iR_j=\text{don't care})$

次に，ある時点において，論理判断の対象となる condition が $C_1, C_2, \ldots, C_i, \ldots, C_m$ であるとき，これを "$(C_1, C_2, \ldots, C_i, \ldots, C_m)$=decision" のように表現しよう．このとき，次の規則を設ける．

[規則9] $(\forall i)(C_i=\text{entry} \Rightarrow (C_i, C_{i+1}, C_{i+2}, \ldots, C_m)=\text{decision})$

$H=\text{entry} \Rightarrow (C_1, C_2, \ldots, C_m)=\text{decision}$

[規則10] $(C_i, C_{i+1}, \ldots, C_m)$=decision のとき，condition $C_i, C_{i+1}, \ldots, C_m$ の論理値（真または偽）のすべてを満足する C_pR_j $(p=i, i+1, i+2, \ldots, m;\ j=1,2,\ldots n)$ の組み合わせが，

（i） 唯一の j について存在するとき，rule R_j が選択される．

（ii） 複数の j について存在するとき，そのような rule R_j 間には ambiguity が存在する．

（iii） いかなる j についても存在しないとき，rule ELSE が選択される．このとき，ELSE が指定されていなければ error である．

ある時点において実行の対称となる action が $A_1, A_2, \ldots, A_l$ の順であるとき，これを "$(A_1, A_2, \ldots, A_l)$=execution" のように表現しよう．

[規則11] すべての j に対し，R_j または ELSE が選択されると，$A_kR_j \neq \text{blank} \cup A_kE \neq$ blank なる k $(k=1,2,\ldots,l)$ に対して $(A_1, A_2, \ldots, A_k, \ldots, A_l)$=execution となる．

[規則12] すべての k に対し，$A_k=\text{entry} \cap A_kR_j \neq$ blank なる j が唯一存在するとき，または $A_k=\text{entry} \cap A_qE \neq \text{blank}$ のとき，$A_qR_j \neq \text{blank} \cup A_qE \neq \text{blank}$ なる $q(q=k, k+1, k+2, \ldots, l)$ に対して $(A_k, A_{k+1}, \ldots, A_q, \ldots, A_l)$=execution となる．

2.3 表言語の例

次に，いままで述べた点並びに問題点を具体例で示す．

[例 1] 表言語と decision table の関係

Fig. 2 の流れ図について考える．この流れ図を点線で示す二つのブロック (block 1 と block 2) に分け，各ブロックごとに，それぞれ DETAB 1, DET

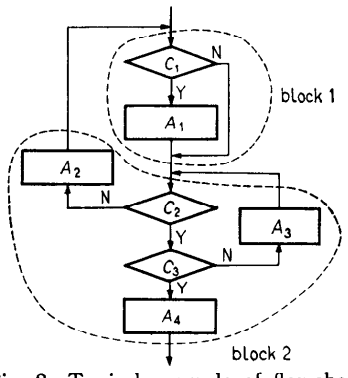

Fig. 2 Typical example of flowchart

DETAB 1	R_1	R_2
C_1	Y	N
A_1	X	
GO TO DETAB 2	X	X

DETAB 2	R_1	R_2	R_3
C_2	Y	Y	N
C_3	Y	N	
A_2			X
A_3		X	
A_4	X		
GO TO DETAB 1			X
GO TO DETAB 2		X	

Fig. 3 Two dicision tables DETAB 1 and DETAB 2 which represent each algorithm of block 1 and block 2 in Fig. 2 seperately

AB 2として decision table を構成し，Fig. 2 の論理を decision table で実現すると **Fig. 3** のようになる．

ループを形成するアルゴリズムを，decision table だけを使用して構成するには，上例で示したように，ループのないブロックごとに，open あるいは closed の decision table を作成し，それらを GO TO 文または CALL 文に相当する文で連結してゆくのが従来の方法[1]である．

いま，condition C_1 と C_2 にそれぞれ文番号1と2をつけ，GO TO DETAB 1 と GO TO DETAB 2 をそれぞれ GO TO 1 と GO TO 2 でおきかえ，次に，DETAB 1 と DETAB 2 を一つの表プログラム (TP 1) にまとめたものを **Fig. 4** に示す．

TP 1		R_1	R_2	R_3	R_4	R_5
1	C_1	Y	N			
2	C_2			Y	Y	N
	C_3			Y	N	
	A_1	X				
	A_2					X
	A_3				X	
	A_4			X		
	GO TO 1					X
	GO TO 2	X	X		X	

Fig. 4 *Table program* TP 1 which contains the function of two decision tables DETAB 1 and DETAB 2 of Fig. 3

TP 1 の condition entry に，decision table の規則を適用すると C_iR_j の blank が don't care となって ambiguity が生じ，実行不可能である．そこで，TP 1 に **2.2** で述べた規則を適用してみる．

まず，H=entry または C_1=entry のとき，規則4と規則5より blank C_1R_3, C_1R_4, C_1R_5 は neglect となり，規則6より rule R_3, R_4, R_5 は H=entry に無関係となる．このとき，規則7より blank C_2R_1, C_2R_2, C_3R_1, C_3R_2 は don't care となり，かつ規則9と規則10から，C_1, C_2, C_3 の満たす論理値（真または偽）の組み合わせを満足する rule を，R_1 と R_2 から選択すればよいことになる．したがって，この場合には，C_1=真なら R_1 を，C_1=偽なら R_2 を選択することになる．

次に，C_2=entry の場合，規則5から C_2R_1, C_2R_2 は neglect となるから，選択可能な rule は規則6から R_3, R_4, R_5 のいずれかになる（H=entry の場合，C_2R_1 と C_2R_2 は don't care であった．このように，一つの brank が場合によって二つの意味を共有することになる．規則1参照）．また規則7より C_3R_5 は don't care であるから，規則9と規則10より，C_2, C_3 の満たす論理値（真または偽）の組み合わせと合致する rule を R_3, R_4, R_5 から選択することになる．

さらに規則11を適用すれば，TP 1 が，Fig. 2 と Fig. 3 が表現するアルゴリズムと全く同じアルゴリズムを表わしていることがわかる．

[例 2] rule 数の減少

TP 1 の rule R_2 は，C_1=偽のとき，GO TO 2 によって直接 C_2=entry となる．そこで，TP 1 を **Fig. 5** の TP 2 のように書きかえてみる．この TP 2 に，**2.2** の規則を適用すると，Fig. 2, Fig. 3, Fig.

TP 2		R_1	R_2	R_3	R_4
1	C_1	Y	N	N	N
2	C_2		Y	Y	N
	C_3		Y	N	
	A_1	X			
	A_2				X
	A_3			X	
	A_4		X		
	GO TO 1				X
	GO TO 2	X		X	

Fig. 5 *Table program* TP 2 which is equivalent to that of Fig. 4

4 の表わすアルゴリズムと同一であることがわかる．このことは，一つのアルゴリズムを，いくつかの異なった形式の **TP** で表現できることを示し，視点を変えれば，プログラマに対し，**TP** の文書性，わかりやすさ，書きやすさなどの融通性を与えることを意味している．一方，**TP** の処理系の立場からみれば，rule 数の減少，すなわち，むだのある **TP** の簡単化の可能性を示している．

［例 3］ dummy の文番号

Fig. 6 に示す流れ図を **TP** で表わすと **Fig. 7** の TP 3 となる．

TP 3 の C_1 につけられた文番号 1 は，dummy である．Fig. 6 の流れ図から TP を作成すると，文番号 1 は不要に思われる．しかし，もし文番号 1 がなければ，規則 2

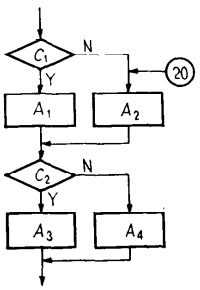

Fig. 6　Another example of flowchart

TP3		R_1	R_2	R_3	R_4
1	C_1	Y	N		
2	C_2			Y	N
	A_1	X			
20	A_2		X		
	A_3			X	
	A_4				X
	GO TO 2	X	X		

Fig. 7　*Table program* TP3 of Fig. 6 which has a dummy statement number "1"

から C_1R_3, C_1R_4 は don't care となって ambiguity を生ずることになる．そこで，規則 4 と規則 5 を逆に利用し，文番号をつけることによって ambiguity からのがれた．この問題に対する解決法は，ほかにもいくつか考えられるが，思想の統一と文書性の見地から，dummy の文番号をつけることにする．

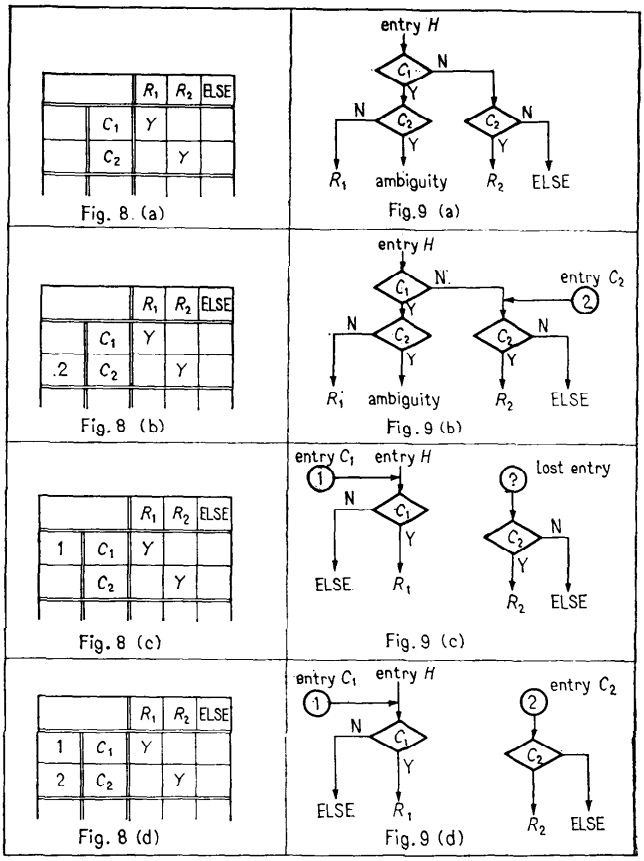

Fig. 8, 9　The effects of labeling statement numbers

また，action A_2 につけられた文番号 20 は，entry A_2 を許すためのものである．A_2=entry のときは，規則12によって (A_2, GO TO 2)=execution となる。

［例 4］ TP の ambiguity と文番号

Fig. 8 と **Fig. 9** は，同一の condition stub と condition entry[6] を持つ **TP** に，文番号をつけた場合とつけない場合の，**TP** の表わす意味を図解したものである．Fig. 8 の (a), (b), (c), (d) は，それぞれ Fig. 9 の (a), (b), (c), (d) の意味を表わしている．

Fig. 8 と Fig. 9 が示すように，同一の condition entry でも，文番号の有無によって意味が大きく異なり，また ambiguity の可能性があったとしても，condition が満たす論理値と entry の種類によっては，必ずしも ambiguity とはならないことが知られる．

2.4 表言語と decision table

decision table の規則は，**2.2** で述べた規則のうち，規則2，規則3，規則7の $i=1$ の場合，規則8，規則9の $H=$entry の場合，規則10の $i=1$ の場合，そして規則11で表現できる．したがって，**2.1**，**2.2** および **2.3** で述べた点を考え合わせれば，

表言語 ⊇ decision table

が成立することは明らかである．

また，**2.2** に述べた規則や Fig. 4, 5, 7, 8 それに，後に述べる Fig. 11, 12 などから，表言語と decision table が，**Fig. 10** に示すような構造関係で表現できることも明らかである．

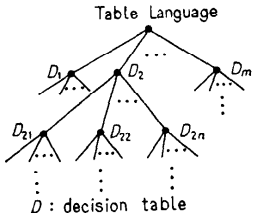

Fig. 10 Structural relations between *Table Language* and decision tables

3. 表言語の処理法

TP は，いくつかの decision table を組み合わせて一つの表形式にまとめたものと考えることもできる．したがって，一つの **TP** を **2.2** の規則に従っていくつかの decision table に分割すれば，これまでに開発されてきた decision table の処理法に，多少の変更を加えるだけで **TP** の処理系を作成できることが想像される．

decision table を computer program に変換する方法には，大別して tree 法[2),3)] mask 法[4)~6)]，2進計算法[7)] の三つがあり，それぞれ実行速度，記憶容量，アルゴリズムの容易さなどに関する特長を持っている．しかしながら，decision table の処理系に要求されるもう一つの主要な機能は，論理の ambiguity の検出である[6),8)]．**2.3** の例4で述べたように，**TP** の持つ ambiguity は，decision table のそれとはいくぶん異なる面を持っているため，この点に関する考慮が必要である．コンパイル時に，ambiguity の可能性が検知されても，論理の流れによっては，必ずしも ambiguity になるとは限らない．したがって，ambiguity に対する最終的な処理は，目的プログラムの実行時にゆだねなければならない．

TP を computer program に変換するアルゴリズムにもいくつかの方法が考えられる．われわれは，実行時における ambiguity 検出の容易さ，アルゴリズムの明解さ，所要記憶容量の少なさなどの見地から，Muthukrishnan と Rajaraman の mask 法[6)] を選び，これに，ごくわずかな手順を追加するだけで，**TP** の変換アルゴリズムを作成した．したがって，説明の都合上，文献 6) の内容と一部重複する部分の生ずるのは避けがたい．

3.1 制限エントリ表言語の変換アルゴリズム

TP のコンパイル時と実行時に分けて説明する．

(1) コンパイル時

m を condition 数，n を rule 数とするとき，$\boldsymbol{T}$ マトリクスと $\boldsymbol{F}$ マトリクスを，

$$\boldsymbol{T}=\begin{pmatrix}\boldsymbol{t}_1\\ \boldsymbol{t}_2\\ \vdots\\ \boldsymbol{t}_i\\ \vdots\\ \boldsymbol{t}_m\end{pmatrix}\quad \boldsymbol{F}=\begin{pmatrix}\boldsymbol{f}_1\\ \boldsymbol{f}_2\\ \vdots\\ \boldsymbol{f}_i\\ \vdots\\ \boldsymbol{f}_m\end{pmatrix}\quad \begin{aligned}\boldsymbol{t}_i&=[t_i{}^1\, t_i{}^2 \ldots t_i{}^j \ldots t_i{}^n]\\ \boldsymbol{f}_i&=[f_i{}^1\, f_i{}^2 \ldots f_i{}^j \ldots f_i{}^n]\end{aligned}$$

とする．各要素 $t_i{}^j$ と $f_i{}^j$ は，すべての i ($i=1, 2, \ldots, m$)，j ($j=1, 2, \ldots, n$) に対し，C_iR_j に書かれた Y, N, blank により，次の値に設定する．すなわち，

$$t_i{}^j=\begin{cases}1 \ldots C_iR_j=Y \cup C_iR_j=\text{blank のとき}\\ 0 \ldots C_iR_j=N \text{ のとき}\end{cases}$$

$$f_i{}^j=\begin{cases}0 \ldots C_iR_j=Y \text{ のとき}\\ 1 \ldots C_iR_j=N \cup C_iR_j=\text{blank のとき}\end{cases}$$

とする．

文献 6) では，この段階で，C_iR_j に書かれた Y, N, blank により，

$$Y\to{}^1_0,\quad N\to{}^0_1,\quad \text{blank}\to{}^1_1$$

のようにコード化したマトリクス M_{2mxn} を作成している．本論文におけるマトリクス $\boldsymbol{T}_{mxn}$ と $\boldsymbol{F}_{mxn}$ は，この M_{2mxn} を2つに分割しただけで，文献 6) と本質的な差は全くない．

(2) 実行時

$C_i=$entry のとき，規則9より $(C_i, C_{i+1}, \ldots, C_m)=$ decision になる．$H=$entry の場合は，特にことわらない限り，以下の議論で $i=1$ とおく．いま，$p=i$, $i+1$, $i+2, \ldots, m$ とおき，マトリクス $\boldsymbol{D}$ を，

$$\boldsymbol{D}=\begin{pmatrix}\boldsymbol{d}_i\\ \boldsymbol{d}_{i+1}\\ \vdots\\ \boldsymbol{d}_p\\ \vdots\\ \boldsymbol{d}_m\end{pmatrix}\quad \boldsymbol{d}_p=[d_p{}^1\, d_p{}^2 \ldots d_p{}^j \ldots d_p{}^n]$$

とする．このとき，マトリクス $\boldsymbol{D}$ の各行ベクトル $\boldsymbol{d}_p$

はすべてのpに対して次のように定義される．すなわち，

$$\boldsymbol{d}_p=\begin{cases}\boldsymbol{t}_p\ldots C_p\text{ の判定が真であるとき．ただし}\\ \quad d_p{}^j=t_p{}^j.\\ \boldsymbol{f}_p\ldots C_p\text{ の判定が偽であるとき．ただし}\\ \quad d_p{}^j=f_p{}^j.\end{cases}$$

$(j=1, 2, \ldots, n)$

とする．次に，論理積を∧記号で表わし，すべてのjに対して$d^j=\overset{m}{\underset{p=i}{\wedge}} d_p{}^j$を要素とする行ベクトル$\boldsymbol{d}=[d^1 d^2\ldots d^j\ldots d^n]$を作る．

本節（2）におけるいままでの議論で，文献 6）と本質的に異なる点は，C_i=entry に対する配慮である．文献 6）では，LC_1=blank∩H=entry の場合だけ扱えばよいから，$i=1$ の場合だけについて述べられている．特に，C_i=entry∩C_iR_j=blank のとき，規則5からC_iR_j=neglect となる，これに対する処理が，文献 6）と最も異なる点で，本論文では次のように解決した．すなわち，H=entry∩LC_1=blank の場合には，行ベクトル$\boldsymbol{r}=[r^1 r^2\ldots r^j\ldots r^n]$を，

$\boldsymbol{r}=\boldsymbol{d}$　ただし $r^j=d^j$

とおく．また，排他的論理和を ⊕ で表わせば，C_i=entry の場合は，

$\boldsymbol{r}=\boldsymbol{d}\wedge(\boldsymbol{t}_i\oplus\boldsymbol{f}_i)$　ただし $r^j=d^j\wedge(t_i{}^j\oplus f_i{}^j)$

とおく．

このとき，

（i）　$r^j=1$ となる唯一の $j(j=1,\ldots,n)$ が存在するとき，R_j が選択された rule である．

（ii）　$r^j=1$ となる $j(j=1,\ldots,n)$ がいくつか存在するとき，そのような R_j 間には ambiguity が存在する．

（iii）　$\sum_{j=1}^{n} r^j=0$ のとき，rule ELSE を選択する．

このアルゴリズムの証明は自明である．

［例］ **Fig. 11** について，各マトリクスやベクトルを作成すると以下のようになる．

		R_1	R_2	R_3
	C_1	Y	N	
2	C_2		Y	N
3	C_3	N	N	Y

Fig. 11　An example of Limited entry *table program*

$$\boldsymbol{T}=\begin{bmatrix}1&0&1\\1&1&0\\0&0&1\end{bmatrix},\quad \boldsymbol{F}=\begin{bmatrix}0&1&1\\1&0&1\\1&1&0\end{bmatrix}$$

ここで，

（イ）　H=entry，C_1=真，C_2=偽，C_3=真のとき

$\boldsymbol{D}=\begin{bmatrix}\boldsymbol{t}_1\\ \boldsymbol{f}_2\\ \boldsymbol{t}_3\end{bmatrix}=\begin{bmatrix}1&0&1\\1&0&1\\0&0&1\end{bmatrix}$，$\boldsymbol{d}=[0\ 0\ 1]$，$\boldsymbol{r}=\boldsymbol{d}=[0\ 0\ 1]$

したがって R_3 が選択される．

（ロ）　C_2=entry，C_2=真，C_2=偽のとき，

$\boldsymbol{D}=\begin{bmatrix}\boldsymbol{t}_2\\ \boldsymbol{f}_3\end{bmatrix}=\begin{bmatrix}1&1&0\\1&1&0\end{bmatrix}$，$\boldsymbol{d}=[1\ 1\ 0]$，

$\boldsymbol{r}=\boldsymbol{d}\wedge(\boldsymbol{t}_2\oplus\boldsymbol{f}_2)=[0\ 1\ 0]$

したがって R_2 が選択される．

（ハ）　C_3=entry，C_3=偽のとき

$\boldsymbol{D}=[\boldsymbol{f}_3]=[1\ 1\ 0]$，$\boldsymbol{d}=[1\ 1\ 0]$

$\boldsymbol{r}=\boldsymbol{d}\wedge(\boldsymbol{t}_3\oplus\boldsymbol{f}_3)=[1\ 1\ 0]$

したがって R_1 と R_2 は ambiguity である．

3.2　混合エントリ表言語の変換アルゴリズム

コンパイル時と実行時に分けて説明する．

（1）　**コンパイル時**

文献 6）と同様に，

$$\boldsymbol{H}=\begin{bmatrix}\boldsymbol{h}_1\\ \boldsymbol{h}_2\\ \vdots\\ \boldsymbol{h}_i\\ \vdots\\ \boldsymbol{h}_m\end{bmatrix}\quad \boldsymbol{h}_i=[h_i{}^1 h_i{}^2\ldots h_i{}^j\ldots h_i{}^n]$$

のようなマトリクス $\boldsymbol{H}$ の i 行 j 列の要素 $h_i{}^j$ を，次のように定める．すなわち，

$$h_i{}^j=\begin{cases}0\cdots C_iR_j\neq\text{blank のとき}\\ 1\cdots C_iR_j=\text{blank のとき}\end{cases}$$

（2）　**実行時**

C_i=entry のとき，$(C_i, C_{i+1}, \ldots, C_m)$=decision となる．$H$=entry の場合は，文献 6）の場合と全く同じであるから，特にことわらない限り，以下の議論で $i=1$ とする．このとき C_i と C_iR_j に与えられた論理判断が実行され，その結果は次のようなマトリクス

$$\boldsymbol{T}=\begin{bmatrix}\boldsymbol{t}_i\\ \boldsymbol{t}_{i+1}\\ \vdots\\ \boldsymbol{t}_p\\ \vdots\\ \boldsymbol{t}_m\end{bmatrix}\quad \boldsymbol{t}_p=[t_p{}^1 t_p{}^2\ldots t_p{}^j\ldots t_p{}^n]$$

の p 行 j 列の要素 $t_p{}^j$ に以下の形式で記憶される．）すなわち，

$$t_p{}^j=\begin{cases}1\cdots\text{condition } C_i \text{ が rule } R_j \text{ の判断を満足するとき.}\\ 0\cdots\text{それ以外の場合 } (C_iR_j=\text{don't care の場合も含める}).\end{cases}$$

とおく．次に，**3.1** と同様なマトリクス $\boldsymbol{D}$ の各行ベクトル $\boldsymbol{d}_p$ を，すべての $j=1, 2, \ldots, n$ に対して以下のように定義する．

すなわち，論理和を∨で表わせば，

$H=\text{entry}\cap LC_1=\text{blank}$ の場合には，

$$\boldsymbol{d}_p=\boldsymbol{h}_p\vee\boldsymbol{t}_p \quad (\text{ただし } d_p{}^j=h_p{}^j\vee t_p{}^j,\ p=i, i+1, \ldots, m)$$

また $C_i=\text{entry}$ の場合には，

$$\begin{cases}\boldsymbol{d}_i=\boldsymbol{t}_i & (\text{ただし, } d_i{}^j=t_i{}^j\\ \boldsymbol{d}_p=\boldsymbol{h}_p\vee\boldsymbol{t}_p & d_p{}^j=h_p{}^j\vee t_p{}^j\\ & p=i+1, i+2, \ldots, m)\end{cases}$$

とおく．文献 6) では，$H=\text{entry}\cap L_1=\text{blank}$ の場合だけである．このとき，行ベクトル $\boldsymbol{r}=[r^1\, r^2 \cdots r^j \cdots r^n]$ を，次のように定める．すなわち，

$$r^j=\bigwedge_{p=i}^{m} d_p{}^j$$

ここで，**3.1** と同様に，

(i) $r^j=1$ となる $j(j=1, \ldots, n)$ がいくつか存在するとき，R^j が選択された rule である．

(ii) $r^j=1$ となる唯一の $j(j=1, \ldots, n)$ が存在するとき，そのような R_j 間には，ambiguity が存在する．

(iii) $\sum_{j=1}^{n} r^j=0$ のとき，rule ELSE を選択する．

このアルゴリズムの証明は自明である．

[例] **Fig. 12** について，各マトリクスやベクトルを作成すると以下のようになる．

		R_1	R_2	R_3
1	CV_1	LT 70		GT 60
2	C_2		Y	N
	CV_3	EQ X	EQ X	

Fig. 12 An example of mixed entry *table program*

$$\boldsymbol{H}=\begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix}$$

ここで

(イ) $C_1=\text{entry}$, $CV_1=80$, $C_2=$偽. $CV_3=5$, $X=5$ のとき

$$\boldsymbol{T}=\begin{bmatrix}0&0&1\\0&0&1\\1&1&0\end{bmatrix},\quad \boldsymbol{D}=\begin{bmatrix}\boldsymbol{t}_1\\ \boldsymbol{h}_2\vee\boldsymbol{t}_2\\ \boldsymbol{h}_3\vee\boldsymbol{t}_3\end{bmatrix}=\begin{bmatrix}0&0&1\\1&0&1\\1&1&1\end{bmatrix},$$

$\boldsymbol{r}=[0\ 0\ 1]$

したがって R_3 が選択される．

(ロ) $C_1=\text{entry}$, $CV_1=65$, $C_2=$偽, $CV_3=5$, $X=5$ のとき

$$\boldsymbol{T}=\begin{bmatrix}1&0&1\\0&0&1\\1&1&0\end{bmatrix},\quad \boldsymbol{D}=\begin{bmatrix}1&0&1\\1&0&1\\1&1&1\end{bmatrix},\quad \boldsymbol{r}=[1\ 0\ 1]$$

したがって R_1 と R_3 は ambiguity である．

(ハ) $C_2=\text{entry}$, $C_2=$真, $CV_3=5$, $X=5$ のとき

$$\boldsymbol{T}=\begin{bmatrix}\boldsymbol{t}_2\\ \boldsymbol{t}_3\end{bmatrix}=\begin{bmatrix}0&1&0\\1&1&0\end{bmatrix}\quad \boldsymbol{D}=\begin{bmatrix}\boldsymbol{t}_2\\ \boldsymbol{h}_3\vee\boldsymbol{t}_3\end{bmatrix}=\begin{bmatrix}0&1&0\\1&1&1\end{bmatrix},$$

$\boldsymbol{r}=[0\ 1\ 0]$

したがって R_2 が選択される．

4. 表言語の特徴

ここで，表言語の長所と短所について述べる．

(a) 短 所

(1) 使用可能 rule 数が制限される．ただし，欄の最大数をラインプリンタ用紙にすれば，それほどの制限にはならない．

(2) 判断文の少ないプログラムや，action の多いプログラムには，あまり適さない．

(3) 先頭の condition には，dummy の文番号（または文名）をつけなければならない場合がある．

(b) 長 所

(1) プログラムの文書性が高い．したがって，わかりやすく，書きやすく，覚えやすい．初心者の教育もきわめて容易と思われる．

(2) プログラムの論理チェックがしやすい．

(3) 既存のプログラム言語との間に compatibility がある．

TL における condition や action を適切に統一して記述すれば，既存のプログラム言語（たとえば FORTRAN, COBOL, ALGOL, PL/I）のいずれにも，そのまま使用可能な汎用のアルゴリズムとして作成することも可能である．

(4) プログラムの論理単位ごとに **TP** を形成し，それらを連結できる．

decision table は，ループを表現する能力がきわ

めてとぼしいため，プログラムの論理単位ごとの分割がむずかしい．

(5) condition と action を含めた **TP** の論理チェックと optimization が，処理系によって，ある程度まで容易に実行できる可能性がある．

(6) decision table の変換アルゴリズムを多少変更するだけで implement ができる．

(7) decision table を完全に含んでいる．

(8) その他，特殊な応用の可能性がある．

5. 表言語の適用分野

計算機が利用されているほとんどすべての分野，たとえば，事務処理，設計計算，人工知能問題，Graphics, System Program の作成などに適用の可能性がある．

6. Implementation

混合エントリ表言語の，実験的な pre-compiler **TFORTRAN** を，NEAC 2230 (東京電機大学電子計算機センター) 用に作成した．文字処理用に作られた機械語のサブルーチンを除き，ほとんど FORTRAN Ⅳ で書かれた **TFORTRAN** は，FORTRAN Ⅳ プログラム内に，**Fig. 13** のように書かれた混合エントリ表プログラムを，FORTRAN Ⅳ プログラムの目的プログラムに変換する．

マトリクス ***T*** と ***D*** は，condition ごとに1語内にパックされている．rule の最大数は ELSE を除いて 39 である (NEAC 2230 の1語 48 ビットには，10進 12 桁の整数が収容できる．$2^{39}<10^{12}-1<2^{40}$)．

TFORTRAN は，closed 型と open 型の表言語を処理する．closed 型を呼ぶには @ DO name と書き，open 型は，それが書かれた部分に組み込まれ，これを参照するには @ GO TO name と書く．これら二つの文は，FORTRAN 文と同レベルで使用できる．なお，name の先頭文字が英字のとき closed 型，数字で始まるとき open 型である．

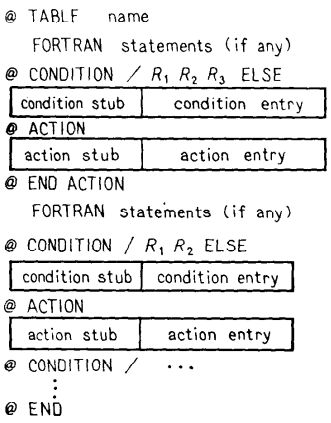

Fig. 13 An example of program sequences of experimental table language system "TFORTRAN"

7. むすび

decision table の condition と action に，文番号 (または文名) をつけることによってプログラム・ループを許し，condition entry のブランクに don't care と neglect なる新しい解釈を導入することにより，2次元の表形式，すなわち表言語によって，任意のアルゴリズムがプログラミングできることを示した．また，表言語が decision table を完全に含み，かつその処理系が，decision table の変換アルゴリズムに多少の変更を加えるだけで，容易に implement できることも示した．本論文で述べた表言語の特徴や可能性のうち，ここに示すことのできなかった件については，別の機会に報告したい．

未筆ながら，日ごろご指導いただく青山学院大学間野浩太郎教授，変換アルゴリズムに関して有益なご教示をいただいた電子技術総合研究所山本和彦氏，ご討論いただいた本学情報工学研究室の諸兄に深謝いたします．

参考文献

1) たとえば Hughes, Shank, Stein.: DECISION TABLES McGRAW-HILL. 1968.
2) Pollack, S. L.: Conversion of limited entry decision tables to computer programs. Comm. ACM 8, 11 (Nov. 1965), p. 677～682
3) Reinwald, L. T., and Soland, R. M.: Conversion of limited entry decision tables to optimal computer programs I: minimum average processing time. J. ACM 13, 7 (July 1966), p. 339～358
4) Kirk, H. W.: Use of decision tables in computer programming. Comm. ACM 8, 1 (Jan, 1965), p. 41～43
5) King, P. J. H.: Conversion of decision tables to computer programs by rule mask techniques. Comm. ACM 9, 11 (Nov. 1966) p. 796～801
6) C. R. Muthukrishnan and V. Rajaraman: On the conversion of decision tables to Computer programs. Comm. ACM 13, 6 (June 1970), p. 347～351
7) Cyril, G. V.: Programming decision tables in FORTRAN, COBOL or ALGOL. Comm. ACM 9, 1 (Jan. 1966), p. 31～35
8) King, P. J. H.: Ambiguity in limited entry decision tables. Comm. ACM 11, 10 (Oct. 1968) p. 680～684

(昭和46年6月30日受付)