ソフトウエア工学 93－2
（１９９３．７．８）

データフロー設計とＪＳＰを結びつけたプログラム設計のモデル

林　雄二
北海道情報大学

m-n列変換器はｍ本の入力テープとｎ本の出力テープを持つ有限状態機械であり、それが表す関数をm-n列関数という．本稿ではm-n列関数をプロセスとするデータフローネットワークのモデルを定義し考察した．

m-n列関数には、ＪＳＰにおける構造一致という良い性質があるので、データフローネットワークのプロセスを基本ＪＳＰ法によって設計することができる．したがって、このモデルはデータフロー設計とＪＳＰを結び付けた新たなプログラム設計法を生み出す可能性を持っている．

A Model for Program Design Connect Dataflow Design with JSP

Yuji Hayashi

Hokkaido Information University

The m-n sequential transducer is a finite state machine with m input tapes and n output tapes. The relation defined by a m-n sequential transducer to be a function is called m-n sequential function . In this paper a data flow network model with processes of m-n sequential functions is defined and investigated.

The m-n sequential function has a good property such as structure correspondences in JSP, so that processes in data flow networks are able to be designed using JSP basic method. Therefore there is a probability that this model will produce a new program design method connected data flow design with JSP.

1. まえがき

ソフトウェアの分析，設計を一貫してデータフローに基づいて行うための形式的モデルを考察した。

ソフトウェア設計をデータフローに基づいて進めようとする場合、ループによるアルゴリズムを必要とする段階になると、ＤＦＤにどう描くべきか、はたと行きずまってしまう．変数を自由に使える手続き型言語を想定して、データをデータストア（ファイル）に表してしまえば、あとの実行順序は制御フローに頼らざるを得なくなる．プロセスの設計を詳細化していけば、一旦データストアにしてしまったデータが、データフローとしてプロセスを駆動させるために必要であることが分かってくる．こう考えると、設計の初期段階から最終段階までを一貫して、データストアを作らずに、すべてをデータフローに統一して設計を行うことも有効である．また、データフローをプロセスが処理するタイミングなどの厳密な定義がなされていないことも、データフローのループをＤＦＤ上で容易に作ることができない原因の一つと考えられる．このような意味でもＤＦＤを形式化することが必要である。

ＤＦＤは全体の形態は有向グラフとして形式化できる。本モデルでは、ＤＦＤにおけるプロセスを関数型m-n列変換器とした。このプロセスは入力データフローにデータがすべて到着したときに実行され、プロセス実行後にまとめて出力データフローに結果が送り出されるものである。

関数型m-n列変換器の表す関数は、その入出力データ間にＪＳＰ[7]（Jackson Structured Programming）における構造一致の性質を持っている[8]ので、最終段階のプロセス，すなわち関数型m-n列変換器に対応するプログラムを、ＪＳＰによって容易に設計することができる．したがってこのモデルは、データフロー設計とＪＳＰとを結び付けたプログラム設計法に発展することが期待される．

2. 列関数上のＤＦＤ

以下ではＤＦＤをm-n列関数のプロセスを基礎にして形式化する．なおm-n列関数は複数の記号列を受け取り、有限状態の機械によって変換し、結果を複数の記号列としてもとめるものであり、詳しくは、付録２に示している。以下ではm-n列関数を単に列関数ということにする。

ここで定義する列関数ネットワーク（sequential function network）は、全体の形が有向グラフであり、基本的に有向グラフの各点には列関数が対応し、列関数の間をテープを媒介としてデータが流れていく機構を表したものである．すなわちΣ上の列関数ネットワークΦは、

$$\Phi = (G, \tau, \pi)$$

と表せる．ここにＧは多重辺を持たない有向グラフであり、

$$G = (V, s, d, E)$$

と表せる．Ｖは自然数で表された点の集合とし、ｓ，ｄという特別な点を含むものとする．ｓはこのネットワークへの入力データの源泉を表す点であり、ｄはこのネットワークからの出力データの吸収先を表す点である．Ｅは点の対の集合で、個々の点の対（ｉ，ｊ）は点ｉから点ｊへの有向辺を表す．ｓ，ｄはそれぞれ、有向辺の終点，始点になりえない点である．

τ は各有向辺にテープ数を対応させる関数であり　$\tau : E \rightarrow N$（ただしＮは自然数の集合）．すなわち、有向グラフの各辺をデータフローとしたときの各辺が持つテープ数を表すものである。

$\forall k \in V$　に対し関数it，otを次のように定義し、

$$it(k) = \sum_{(i,k) \in E} \tau((i,k))$$

$$ot(k) = \sum_{(k,j) \in E} \tau((k,j))$$

それぞれ、点ｋの入力テープ数，点ｋの出力テープ数と呼ぶ．

πはs,dを除くGの各点に対し、Σ上の列関数または列関数ネットワークを対応させる関数である．点kに対応する列関数または列関数ネットワークをπ_kで表し、点kのプロセスともいう．ただし各π_kはつぎの条件を満たしているものとする．

π_kが列関数のとき　その入力，出力テープ数はそれぞれit(k)，ot(k)．

π_kが列関数ネットワークΦのとき、その入力，出力テープ数はΦのデータ源泉s，データ吸収点dに対し、ot(s)=it(k),it(d)=ot(k)．

このような列関数ネットワークは、テープ数を辺の数で表せば、例えば図1のようになる．これは、例3.1（後述）の列関数ネットワークを図示したものである．

π_kへの入力データフローは　it（k）本のテープから成る．このテープ上の記号列を始点の点番号i_jの大きさの順に並べて、簡単に（w_1, w_2, ・・・・, w_q）　と記述する．ただし　q＝it（k）である．π_kからの出力データフローも同様に　（z_1, z_2, ・・・・, z_r）　と記述する．ただしr＝ot（k）である．

こうして関数π_kの入力，出力データフローの関係を

$$\pi_k(w_1, w_2, \cdots, w_q)=(z_1, z_2, \cdots, z_r)$$

と表すことができる．各辺の各テープ（すなわちデータフロー）に固有の名前をつけると、列関数ネットワークが表す関数は、以下のように連立された関数定義式で表現することができる．

$$\pi_1(w_1, w_2, \cdots, w_q)=(Z_1, Z_2, \cdots, Z_r)$$
$$\pi_2(w'_1, w'_2, \cdots, w'_{q'})=(Z'_1, Z'_2, \cdots, Z'_{r'})$$
$$\vdots$$
$$\pi_j(w''_1, w''_2, \cdots, w''_{q''})=(Z''_1, Z''_2, \cdots, Z''_{r''})$$

個々の関数は、基本的には、その関数に対する左辺のテープ上のデータがすべて求められた後に計算される．ただしデータループ関数（定義3.1）だけは例外であり、左辺のすべてのテープに対するデータが求められていなくとも計算が実行される．

こうして、データ源泉をsとする列関数ネットワークの(s,k)なるすべての辺にテープ数τ（(s,k)）の初期データ列を与えると、このネットワークへの入力は

$(x_1, x_2, \cdots, x_m)$　　ただし　m＝ot(s)

と表される．

各点に対応する列関数に対し、入力テープにデータが生成された時、列関数の実行を行い出力テープのデータを求める．すなわち、連立の関数定義式に基づいた計算を行って、辺を通してデータを順次後続の列関数に送っていくなら、途中で未定義状態になるような列関数がない限り、(k,d)なるすべての辺にデータ列が得られる．dはデータ吸収点であり、この最終データ列は

$(y_1, y_2, \cdots, y_n)$　　ただし　n=it(d)

と表される．

列関数ネットワークΦの入力データフローと出力データフローのこのような対応関係を、

$\Phi(X_1, X_2, \cdots, X_m)=(Y_1, Y_2, \cdots, Y_n)$

と表し、Φの定義する関数と呼ぶ．

［例２．１］以下のΦは列関数ネットワークの一例である（図1）．

$\Phi=(G, M)$

$G=(V, 1, 5, E)$

$V=\{1,2,3,5\}$

$E=\{(1,2),(1,3),(2,3),(3,5)\}$

$M=(\tau, \pi)$

$\tau=\{(1,2)\rightarrow 1,(1,3)\rightarrow 2,(2,3)\rightarrow 2,(3,5)\rightarrow 1\}$

$\pi: \pi_2=\{(w,(1^m,0^n)) \mid w\in(1\cup 0)^*,$ m,nはそれぞれw内の1,0の個数$\}$

$\pi_3=\{((1^k,0^j),(1^m,0^n),u) \mid$ k=mかつj=nならu='y'　そうでないなら　u='n'$\}$

このΦの定義する関数は、３本のテープから$(w, 1^k, 0^j)$を入力し、1,0の記号列w内の1と0の個数が、それぞれk,jに等しいときは'y'，等しくないときは'n'を出力するものである．□

列関数ネットワークの点（プロセス）が再び列関数ネットワークであるばあい、後者は前者の子ネットワークであるという．子ネットワークの子ネットワーク，などが順次考えられるが、子ネッ

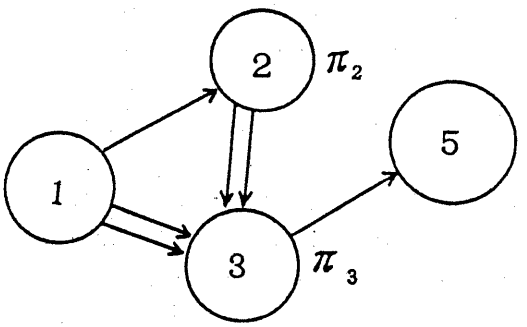

図1　列関数ネットワークの例

トワークも含めてそれらを子孫ネットワークということにする．

閉路なし列関数ネットワークは、有向グラフとしてGに閉路が存在しないものである．つぎの定理は、閉路なし列関数ネットワークの能力の限界を示したものである．

［定理２．１］すべての子孫ネットワークが閉路なしである列関数ネットワークを考える．このような列関数ネットワークの定義する関数のクラスは、合成列関数と同じクラスC_2である（クラスC_2は付録参照）．
証明）省略．□

閉路なし列関数ネットワークの定義する関数は、前述の定理に示されているように、関数の能力としてはかなり限定されている．より広い範囲の問題に対応する列関数ネットワークであるためには、データフローのループが必要となる．しかし、列関数ネットワークに無制限なデータフローのループを認めることは、構造化されていないプログラムがそうであるように、望ましいことではない．そこで、最も基本的なデータフローのループを以下に定義する．

［定義２．１］つぎの連立した関数定義式はデータフローをループ状に結合したものを表し（図２）、データループ関数（data loop function）と呼ぶ．

$f\ ((X_1, X_2, \cdots, X_m), (V_1, V_2, \cdots, V_k)) = ((Y_1, Y_2, \cdots, Y_n), (U_1, U_2, \cdots, U_l))$

$g(U_1, U_2, \cdots, U_l) = (V_1, V_2, \cdots, V_k)$

ただし、関数fはつぎの二つを満たす．

① fは$(X_1, X_2, \cdots, X_m) = (\varepsilon, \varepsilon, \cdots, \varepsilon)$
または$(V_1, V_2, \cdots, V_k) = (\varepsilon, \varepsilon, \cdots, \varepsilon)$
のいずれかでのみ計算がなされる．

② fは$(Y_1, Y_2, \cdots, Y_n) = (\varepsilon, \varepsilon, \cdots, \varepsilon)$
または$(U_1, U_2, \cdots, U_l) = (\varepsilon, \varepsilon, \cdots, \varepsilon)$
のいずれかを満たす出力を出す．

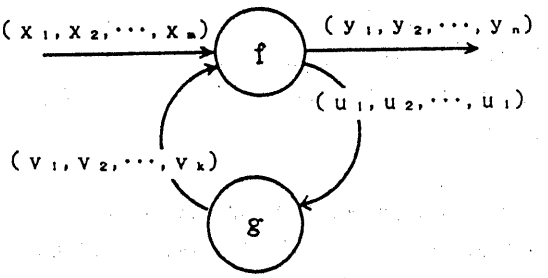

図2　データループ関数

f，gとして、もっとも基本的なものは列関数であり、一般には後述の構造的列関数ネットワークの定義する関数とする．ただしgが空の場合も許すものとし、その場合はl=kで、$(U_1, U_2, \cdots, U_l)$と$(V_1, V_2, \cdots, V_k)$はテープの並びとして同じものになる．これは関数fの出力テープの一部がそのまま関数fの入力テープの一部になっている場合である．□

このデータループ関数の計算は、つぎのように行われる．最初は入力として$(X_1, X_2, \cdots, X_m)$が与えられ、$(V_1, V_2, \cdots, V_k) = (\varepsilon, \varepsilon, \cdots, \varepsilon)$である．そのときの関数fの出力は、$(Y_1, Y_2, \cdots, Y_n) = (\varepsilon, \varepsilon, \cdots, \varepsilon)$ または $(U_1, U_2, \cdots, U_l) = (\varepsilon, \varepsilon, \cdots, \varepsilon)$ である．

もし$(U_1, U_2, \cdots, U_l) \neq (\varepsilon, \varepsilon, \cdots, \varepsilon)$ であれば、関数gの入力データフロー $(U_1, U_2, \cdots, U_l)$が生じたのであるから、関数gが計算され、出力データフローとして$(V_1, V_2, \cdots, V_k)$が流

れる．$(V_1, V_2, \cdots, V_k)$は関数fの入力データフローであるから、再び関数fの計算が実行される．このときは$(X_1, X_2, \cdots, X_m)=(\varepsilon, \varepsilon, \cdots, \varepsilon)$として扱われる．

もし$(Y_1, Y_2, \cdots, Y_n)\neq(\varepsilon, \varepsilon, \cdots, \varepsilon)$であれば $(U_1, U_2, \cdots, U_l)=(\varepsilon, \varepsilon, \cdots, \varepsilon)$であるから、関数gは実行されずに終了する．こうしてこのデータループ関数の最終結果はデータフロー$(Y_1, Y_2, \cdots, Y_n)$に得られることになる．

［定義２．２］　列関数ネットワーク内のデータフローのループとして、上のデータループ関数だけを許したものを、構造的列関数ネットワーク（structured sequential function network）と呼ぶ．□

ループはコントロールの表れであるという見方から、ＤＦＤでは一般に、データフローのループを望ましくないものとしている[2]．しかし、データ駆動の原理で関数を順次計算する場合には、同一関数の繰り返し実行を考えないわけにはいかない．したがって、プログラム設計段階のＤＦＤでは、データフローのループを表現できなければならない．

構造的列関数ネットワークでは、こうして、データフローに基づいたプログラムを、多入力，多出力の関数によって統一的に記述することができる．つぎのような、記述上の約束をしたうえで、構造的列関数ネットワークによってプログラムを設計する例をあげてみよう．

データフロー名は日本語，関数名（プロセス名）は小文字のアルファベットや＿を使った英字名，データの記号列を表すのにu, v, w, x, y, zなど、Σのアルファベットには0, 1などを用いる．最下層のプロセスが列関数になったら、その列関数の機能（入力データフローにどのような処理をして出力データフローに変換するか）を関数定義式の行の下に記述する．これは列変換器の動作規則や状態推移図で表すことができるが、ＪＳＰによる設計が可能な部分である．簡単のために以下の例では

if-then-else-　の構文のみを使って表現することにする．

［例２．２］２進数を下位のbitから逆順に入力し、１進数に変換して出力する．たとえば、入力が011なら6であるから、出力は111111である．この問題に対して、構造的列関数ネットワークの各プロセス，各データフローに固有の名前を付けてそれらの関係を表すと、つぎのようになる．

```
binary_to_unary(２進列)＝(１進列)
  is_rest(２進列, 列残₀, 位値₀, 現１進₀)＝
          (１進列, 列残₁, 位値₁, 現１進₁)
    if(w, ε, ε, ε)then(ε, w, 1, ε) else
    if(ε, x, v, y)then(ε, x, v v, y)else
    if(ε, β, v, y)then(y, ε, ε, ε)endif
                                    endif
                                    endif
  one_bit(列残₁, 位値₁, 現１進₁)＝
          (列残₀, 位値₀, 現１進₀)
    if(1x, v, y)then(x, v, v y)   else
    if(0x, v, y)then(x, v, y)   endif
                                  endif
```

第１段階の設計は、関数binary_to_unaryである．この関数は、２進数の列（２進列）を変換し１進数（１進列）として求めるものである．このbinary_to_unaryは、２進数を1bitづつ調べて１の列に変換することの繰り返しであるから、is_restとone_bitという関数で表されたデータループ関数に詳細化される（図3）．この２つの関数はいづれも列関数である．

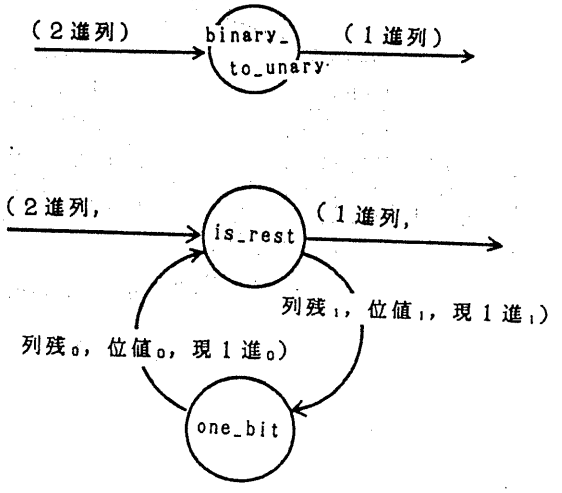

図３　関数binary_to_unary と詳細化

列関数の機能を記述している部分で、if(w, ε, ε, ε)then(ε, w, 1, ε)はif(2進列=w , 列残$_0$=ε, 位置$_0$=ε, 現1進$_0$=ε)then(1進列=ε, 列残$_1$=w, 位置$_1$=1, 現1進$_1$=ε)を意味している.

is_restは入力の2進数に残り（列残$_0$）があるなら、そのbitの位取りの値（1,2,4,8,…）を1の個数で表して（位値$_0$）、1の個数を計算する関数one_bitに送る. 残り（列残$_0$）がないなら、「現1進$_0$」を最終結果として「1進列」に出力し、この計算を終える. 関数one_bitでは、残っている2進数（列残$_1$）の先頭が1か0かによって、対応する位取りの1の個数（位値$_1$）を「現1進$_1$」に付け加えて送り出し（現1進$_0$）先頭bitを除いた残りの2進数列もそのまま出力として送り出す（列残$_0$）.

図4は関数one_bitについて、入出力データフローのデータ構造をジャクソン木で表したものである. この図から、入力(列残$_1$, 位値$_1$, 現1進$_1$)と出力(列残$_0$, 位値$_0$, 現1進$_0$)には構造の上に対応のあることが分かる（図では、対応の一部分のみを曲線で示している）. こうして、関数is_rest, one_bitを基本ＪＳＰ[10]によって設計することができ、設計されたプログラム（疑似コード）を付録1に示してある.

なお、2進列が011すなわち6の場合の実行過程を示しておく.

binary_to_unary(011)
　is_rest(011, ε, ε, ε)=(ε, 011, 1, ε)
　one_bit(011, 1, ε)=(11, 1, ε)
　is_rest(ε, 11, 1, ε)=(ε, 11, 11, ε)
　one_bit(11, 11, ε)=(1, 11, 11)
　is_rest(ε, 1, 11, 11)=(ε, 1, 1111, 11)
　one_bit(1, 1111, 11)=(ε, 1111, 111111)
　is_rest(ε, β, 1111, 111111)
　　=(111111, ε, ε, ε)=(111111) □

3. むすび

列関数ネットワークは、それを実行する実在のシステムを想定したものではないが、シミュレートするシステムを開発することは難しくはない. それは、プロセスの関数機能の記述部分（例2.3のif-then-else-）を、一般のプログラミング言語（Pascal, Cなど）で記述した表現にしておけばよい. その場合、プレコンパイルによって、データ駆動の原理で順次関数を実行していくようなプログラムに変換することで、このモデルを実行することができると考えている. そのようなシステムの下に、本論文で述べた形式化がデータフローの世界でのより一般的な設計法になりうるか、検討をすすめていきたい.

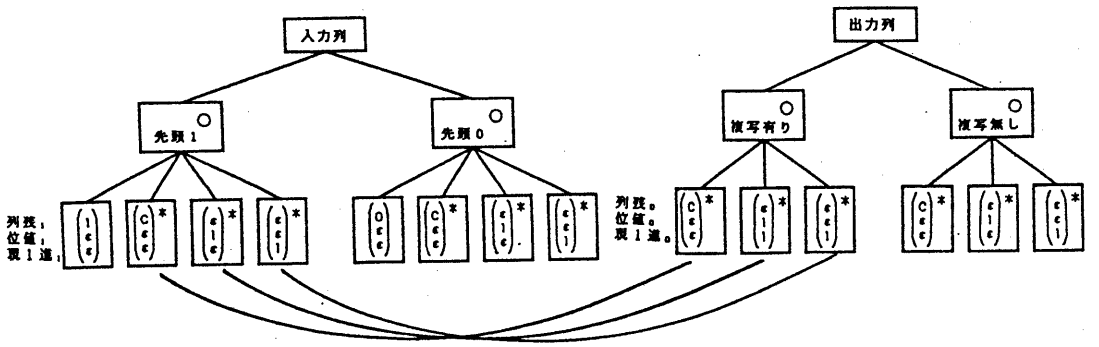

図4　関数one_bitの入出力構造

文献

[1] Alder M.:An Algebra for Data Flow Diagram Process Decomposition, IEEE Trans. Software Engineering, 14, 2, pp. 169-183(1988)

[2] DeMarco T.:"Structured Analysis and System Specification", Prentice-Hall(1979) 邦訳:高梨ほか"構造化分析とシステム仕様", 日経BP社(1986)

[3] Eilenberg S.:"Automata Languages and Machines vol.A", Academic Press(1974)

[4] Elgot C.C and Mezei, J.E:"On Relations Defined by Generalized Finite Automata", IBM J.Res.Devel., 9, 1, pp. 47-68(1965)

[5] Harary F.:"Graph Theory", Addison-Wesley(1969) 邦訳:池田"グラフ理論", 共立出版(1971)

[6] Ibarra O.H:"Characterizations of Transductions Defined by Abstract Families of Transducers", Mathematical Systems Theory, 5, pp. 271-281(1971)

[7] Jackson M.A.:Principles of Program Design, Academic Press(1975) 邦訳:鳥居"構造的プログラム設計の原理", 日本ｺﾝﾋﾟｭｰﾀ協会(1980)

[8] Veen A.H.:"Dataflow Machine Architecture", ACM computing Surveys, vol.18, no.4, pp.365-396(1986)

[9] 林:"ジャクソン法における構造不一致問題の形式化", 信学論(D-I), J76-D-I, 1, pp. 11-18(1993)

[10] Cameron, J.R: JSP and JSD:The Jackson Approach to Software Development, IEEE Computer Society Press(1989)

[11] 島田:"数値計算向きデータフロー計算機", 情報処理, vol.26, no.7, pp.780-786(1985)

[12] Ackerman, W.B.:"Data Flow Languages" IEEE Computer, vol.15, no.2, pp.15-25(1982)

付録1.

データフロー設計による2進1進変換プログラム

このプログラムで、ifやwhileの条件式の部分に記述されている(1, 1, ε)などは、入力命令によって3本のテープから入力した記号が、それぞれ1, 1, εであること、すなわち1, 2番目のテープから入力したものがどちらも1で、3番目のテープからは何も入力しないこと、を表している. また、$T_1, T_2, \cdots$などは、直前に入力した1, 2, ‥番目のテープの記号を意味している.

```
binary_to_unary (2進列) = (1進列)
  is_rest (2進列, 列残0, 位値0, 現1進0) =
          (1進列, 列残1, 位値1, 現1進1)
    if (1, ε, ε, ε) or (0, ε, ε, ε) then
      write(ε, T1, 1, ε)
      while (1, ε, ε, ε) or (0, ε, ε, ε) do
              write(ε, T1, ε, ε) endwhile
    else if (ε, 1, ε, ε) or (ε, 0, ε, ε) then
          write(ε, T2, ε, ε)
          while (ε, 1, ε, ε) or (ε, 0, ε, ε)
            do write(ε, T2, ε, ε) endwhile
          while (ε, ε, 1, ε) do
              write(ε, ε, 11, ε) endwhile
          while (ε, ε, ε, 1) do
              write(ε, ε, ε, 1) endwhile
        else if(ε, β, ε, ε) then
              while (ε, ε, ε, 1) do
               write(1, ε, ε, ε) endwhile
              while (ε, ε, 1, ε) do
               write(ε, ε, ε, ε) endwhile
            endif
      endif
  endif

one_bit (列残1, 位値1, 現1進1) =
        (列残0, 位値0, 現1進0)
  if (1, ε, ε) then
    while (1, ε, ε) or (0, ε, ε) do
          write(T1, ε, ε) endwhile
    while (ε, 1, ε) do
          write(ε, 1, 1) endwhile
    while (ε, ε, 1) do
          write(ε, ε, 1) endwhile
  else if (0, ε, ε) then
        while (1, ε, ε) or (0, ε, ε) do
          write(T1, ε, ε) endwhile
        while (ε, 1, ε) do
```

```
        write(ε, 1, ε)  endwhile
      while (ε, ε, 1)  do
        write(ε, ε, 1)  endwhile
    endif
endif
```

付録2．列変換器と列関数

以下では、Σは任意のアルファベットの有限集合として、あらかじめ与えられたものとする．βは空白を表す特別な記号である。さらに，$\Sigma_\beta = \Sigma \cup \{\beta\}$ とする．空列を表すにはεを用いる．

i番めのテープ上の記号列をw_iとするとき，n本のテープに記録されている記号列の組を，$(w_1, w_2, \cdots, w_i, \cdots, w_n)$と記述する．n本のテープ上の記録

$W=(w_1, w_2, \cdots\cdots, w_n)$，　$Z=(z_1, z_2, \cdots\cdots, z_n)$

に対し，WZは

$$WZ=(w_1, w_2, \cdots, w_n)(z_1, z_2, \cdots, z_n)$$
$$=(w_1z_1, w_2z_2, \cdots\cdots, w_nz_n)$$

を表すものとする．

Σ_β上の記号列wから記号σを除いた（即ち空列εに変換した）記号列を　w/σ　と表す．同様に$W=(w_1, w_2, \cdots\cdots, w_n)$のとき、$W/\sigma$は各テープ上の記号列から$\sigma$を除いたもの$(w_1/\sigma, w_2/\sigma, \cdots\cdots, w_n/\sigma)$を意味する．

［定義1］Σ上の m-n列変換器(sequential transducer)は　(S, Λ, s_0, D)で表される．ただし，Sは状態の有限集合．Dは受理状態集合といい，$D \subset S$．s_0は初期状態といい，$s_0 \in S$．Λは推移規則の有限集合で，

$$\Lambda \subset S \times (\Sigma_\beta{}^*)^m \times (\Sigma^*)^n \times S.$$

m, nをそれぞれ，この列変換器の入力，出力テープ数という．□

m-n列変換器は、m本の入力テープ上の記号列を入力しn本の出力テープ上に出力を行うもの，即ち入力と出力の関係を示す機械である．

［定義2］m-n列変換器$\Omega=(S, \Lambda, s_0, D)$に対し，(m+n)項関係　$T(\Omega)$を以下で定義する．

$w \in T(\Omega)$　　⇔

$\exists k>0$:

$\exists p_i \in S$　（但し i=1, 2, ‥, k+1）:

$\exists u_i \in (\Sigma_\beta{}^*)^m (\Sigma^*)^n$　（但し i=1, 2, ‥, k）:

$p_1=s_0$, $p_{k+1} \in D$, $(p_i, u_i, p_{i+1}) \in \Lambda$　かつ

$w=u_1u_2\cdots u_k/\beta$　□

［定義3］Ωが関数型m-n列変換器　⇔　Ωがm-n列変換器であり，T(Ω)の任意の2つの要素

$(x_1, x_2, \cdots\cdots, x_m, y_1, y_2, \cdots\cdots, y_n)$，$(u_1, u_2, \cdots\cdots, u_m, v_1, v_2, \cdots\cdots, v_n)$　に対し，

$(x_1, x_2, \cdots\cdots, x_m)=(u_1, u_2, \cdots\cdots, u_m)$ なら$(y_1, y_2, \cdots\cdots, y_n)=(v_1, v_2, \cdots\cdots, v_n)$　□

［定義4］Ωが関数型m-n列変換器なら、T（Ω）は（Σ*）mから（Σ*）nへの関数である．この関数をm-n列関数（sequential function）といい、この関数全体をクラスC_1という．□

Fがm-n列関数のとき、$F(x_1, x_2, \cdots, x_m)$の各出力テープの内容をそれぞれ，

$$f_1(x_1, x_2, \cdots, x_m)$$
$$f_2(x_1, x_2, \cdots, x_m)$$
$$\vdots$$
$$\vdots$$
$$f_n(x_1, x_2, \cdots, x_m)$$

と表す．一般に，Fがm入力，n出力の関数，Gがn入力，p出力の関数であるとき，その合成関数

$$G(f_1(x_1, x_2, \cdots, x_m), f_2(x_1, x_2, \cdots, x_m), \cdots, f_n(x_1, x_2, \cdots, x_m))$$

を，簡単に　$G \cdot F(x_1, x_2, \cdots, x_m)$と記す．即ち関数G・Fはm入力，p出力の関数である．

［定義5］m入力，n出力の関数Hが，列関数$G_k, \cdots, G_2, G_1$　の合成　$H=G_k\cdots G_2 \cdot G_1$で表されるとき，m-n合成列関数（synthesis sequential function）とよぶ．m-n合成列関数の全体をクラスC_2という．□