中間語としてオートマトンを用いるコンパイラとその生成

藤村啓二、安在弘幸
（九州工業大学情報工学科）

１．まえがき

我々の研究室では、言語処理系の開発ツールとして、古典的オートマトン理論を用いた言語処理系の生成系ＭＹＬＡＮＧを開発した。ここで生成される言語処理系は、その中間モデルとしてオートマトン・モデルを採用している。本稿では、そのような言語処理系が再び原プログラムを中間語としてオートマトン・モデルに変換する方法について述べる。
一般にプログラム言語処理系の仕事は、コンパイラとして、原プログラムを読みこみ、何らかの形の目的プログラムに変換するか、あるいは、インタプリタとして、それを直接に実行するかである。

コンパイラの場合には、その目的プログラムを直接生成することはほとんどなく、何らかの中間表現(通常は中間語と呼ばれる)を(時には何段も)経由する。また、インタプリタの場合にも、普通は何らかの中間語に変換され、それが実行される。

このような中間語として、通常は三つ組や四つ組と呼ばれるデータ構造が使用される。これに対し、本稿では中間語として、次のようなオートマトン・モデルを提案し、ＭＹＬＡＮＧで生成される言語処理系が、与えられた原プログラムをそのようなオートマトン・モデルに変換する方法を示す。即ち、そのオートマトン・モデルは、次のような一種の有限オートマトンである。

1)　データの流れを表す文(代入文、入出力文、手続き文など)を状態遷移枝に付けられた名札とする。
2)　制御の流れの構造を状態遷移の構造とする。

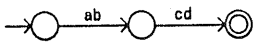
２．オートマトン演算について

本稿で用いるオートマトン演算について簡単に述べる。

２．１　連接

二つのオートマトンを縦列に接続(連接接続と言う)する。例えば、

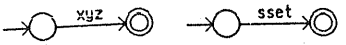
を連接接続すると、



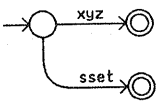
となる。

２．２　並列接続

二つのオートマトンを並列に接続する。例えば、



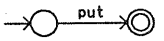
を並列接続すると、



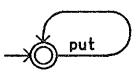
となる。

２．３　閉包

オートマトンを閉包形に変換する。例えば、



を閉包変換すると、



となる。

３．正規翻訳記法

ＭＹＬＡＮＧは、正規翻訳記法(regular translation form:RTF)を入力し、属性付構文指示翻訳系(attributed syntax

directed translator)を出力する一種の言語処理系である。ここで、正規翻訳記法とは、拡張BNFに活動記号を付加したものに次の拡張を行なったもので、

(1) 非終端記号と活動記号に属性が付加できる。
(2) 活動記号の代わりに、その場所に属性の述語、算術代入文、及び入出力文などを直接に記述できる。

この正規翻訳記法(RTF)は、目的の言語処理系が処理するプログラム言語の構文と意味を記述する記法である。ここでは、これについて簡単に説明する。

3.1 属性

RTFにおいて使用される属性は、プログラム言語における引数的な性格を持ち、変数に対応するものを属性生起変数と呼ぶ。属性には、相続属性と合成属性の2種類がある。属性の書式は、次の通りである。

1)非終端記号の場合

<N(i1,……,im/s1,……,sn)>　　(m,n≧0)

2)活動記号の場合

[A(i1,……,im/s1,……,sn)]　　(m,n≧0)

ここで、Nは非終端記号名、Aは活動記号名である。i1,…,imは相続属性生起並び、s1,…,snは合成属性生起並びを表わす。

また、特に属性を持たない場合には、

<N>　　[A]

のように表わす。

3.2 活動記号

活動記号には、次の四つのタイプがある。

typeⅠ　活動ルーチン呼出しを含む活動記号。即ち、伝統的な活動記号。
　例　[getc(name/val)]

typeⅡ　属性として使用されている変数の算術代入文や出力文が直接に書き込まれている活動記号。
　例　[(val1:=val1*val2)]、[(WRITELN(val1,val2)]

typeⅢ　述語(条件式)が書かれている活動記号。
　例　[?(x=y)]

typeⅣ　システム定義の活動ルーチンを呼び出す活動記号。標準活動記号と呼ぶ。
　例　[$PLUS(prt1,ptr2/ptr1)]

typeⅤ　活動ルーチンが直接書き込まれている活動記号。属性は相続属性と合成属性をそれぞれ一つまで書くことが出来る。但し、宣言は記述出来ない。
　例　["ij:=hi*keiji"]、["(hval/) imu:=hval/23"]

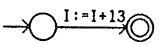
4.プログラムのオートマトンへの変換

ここでは、与えられたプログラムを有限オートマトンに変換する方法について述べる。

状態として、初期状態と最終状態のみを一つずつ持ち、かつ唯一の状態遷移枝を持つような有限オートマトンを単位オートマトンと言う。代入文、入力文、あるいは手続き呼び出し文のようなデータの流れに関する文に対しては、その文による一つの状態遷移のみを持つような単位オートマトンを生成する。これに対して、文の列、条件文や繰り返し文のような制御の流れに関する文については、単位オートマトンより出発し、連接、並列接続、閉包の三つの演算を施すことにより、有限オートマトンの合成を行なっていく。このようなプロセスを繰り返すことにより、一つの手続きとして宣言されたプログラム全体を、その手続き名で呼び出されて実行される一つの有限オートマトンに合成する。

4.1 基本文

基本文は、その文を状態遷移記号として持つ単位オートマトンに変換される。例えば、代入文"I:=I+13;"は、



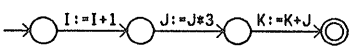
となる。これは、"I:=I+13"が実現可能ならば、次の状態に遷移すると解釈されるが、事実上の無条件遷移と考えてよい。

4.2 文の連接

文の連接は、オートマトンの連接接続に変換される。例えば、

I:=I+1 ; J:=J*3 ; K:=K+J ;

は、



となる。

4.3 制御文

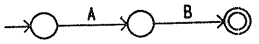
制御文は、条件式や実行文などの構成要素に分解して取り扱う。

4.3.1 IF文

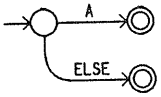
IF文は、オートマトンの並列接続に変換される。

形式Ⅰ　IF A THEN B ;
(A:条件式　B:実行文)

条件式Aは、Aを遷移記号とする単位オートマトンに変換され、それに実行文Bを表すオートマトンを連接接続する。



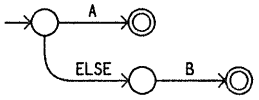
これは、Aの値がtrueならば、Bを実行することを示している。しかし、これでは、Aの値がfalseの場合の遷移先がない。そこでNot Aを表す"ELSE"を遷移記号として持つオートマトンを並列接続する。



これが、IF文(形式Ⅰ)を表わすオートマトンである。

形式Ⅱ　IF A THEN B ELSE C ;
(A:条件式　B,C:実行文)

この形式は、形式Ⅰの拡張形である。Not Aに実行文Cを表わすオートマトンを連接接続すればよい。

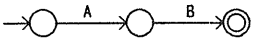


4.3.2 WHILE文

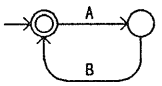
WHILE文は、環状の状態遷移枝を持つ閉包形に変換される。

形式　WHILE A DO B ;
(A:条件式　B:実行文)

まず、条件式Aを表す単位オートマトンと実行文Bを表す単位オートマトンを連接接続する。



次に、これを閉包形に変換する。



これは、Aの値が真である間Bを実行する。

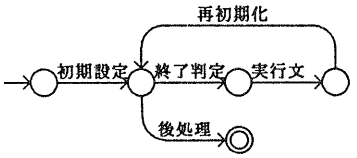
4.3.3 FOR文

FOR文は、WHILE文の変形である。

形式　FOR B TO n DO C ;
(B:初期設定文　C:実行文　n:終了値)

FOR文を表わすオートマトンは、次の５つの状態遷移、すなわち初期設定、終了テスト、実行文、再初期化、および後処理から構成される。

まず、終了テストを表すオートマトン、実行文を表すオートマトン、および再初期化を表すオートマトンを連接接続し、次にこのオートマトンを閉包変換する。次いで初期化を表すオートマトンと変換後のオートマトンを連接接続する。そして最後にそれに後処理を表すオートマトンを連接接続する。

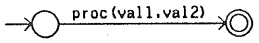


4.4 手続き

手続き本体は、メイン・プログラムとは別のオートマトンに変換される。

4.4.1 手続き呼び出し文

手続き呼び出し文は、手続きの呼出しを状態遷移記号として持つ単位オートマトンに変換される。
例えば、"proc(val1,val2);"は、



と変換される。

5. MYLANGによるコンパイラの生成

前節で示した手順に基づいて、MYLANGにコンパイラを生成させるために、MYLANGに与えるRTFをどのように書くかについて述べる。なお、このコンパイラが最終的に生成するコードは、MYLANGが用いている仮想スタック・マシン用のコードである。
コンパイラが受理する言語の構文は、PASCALに準じる。ただし、"BEGIN"と"END"の代わりにそれぞれ、"(&"と"&)"を使用する。変数の型は、整数型のみとする。type宣言と定数宣言はない。
なお、本節で使用する標準活動記号の機能は付録で述べている。

5.1 単文の翻訳

プログラムの有限オートマトンへの翻訳において、単文の翻訳が基本となる。
最も簡単な文として

```
val:=24;
```

を考える。この文を定義するRTFは、次の通りである。

```
<STATE0> = <NAME>':='<INTEGER>';' ;
```

<NAME>は、英字で始まる英数字列を受理する非終端記号である。<INTEGER>は、数字列を受理する非終端記号である。受理した文字列は、システム変数NAMEに代入される。
これにコード生成を追加すると、次のようになる。

```
<STATE0> = <NAME>["(/adr) adr:=reco(name);"]':='<INTEGER>[$INT][$OADR(adr/)] ;
```

ここで、["(/adr) adr:=reco(name);"]は、<NAME>で受理した変数に割り当てるアドレスを合成属性adrとして返す。ただし、"name"は、<NAME>で受理された変数名を値として持つシステム変数である。

更にオートマトン生成を追加すると、次のようになる。

```
<STATE0(/aptr)> = [$B(/aptr)]<NAME>["(/adr) adr:=reco(name);"]':='<INTEGER>[$INT][$OADR(adr/)][$E] ;
```

次に一般的な算術代入文の翻訳を定義するRTFを示す。

```
<AEXPR(/aptr)> = <VARA(/adr)>[$B(/aptr)]':='<STATE>';'[$OADR(adr/)][$E] ;
<STATE>        = <FACTOR>('+'<FACTOR>[$OPR(2,11/)]
                        +'-'<FACTOR>[$OPR(2,12/)])* ;
<FACTOR>       = <ELEMENT>('*'<ELEMENT>[$OPR(2,13/)]
                         +'/'<ELEMENT>[$OPR(2,14/)])* ;
<ELEMENT>      = '+'/<TERM>+'-'<TERM>[$OPR(2,9/)] ;
<TERM>         = <IDENT>+'('<STATE>')' ;
<IDENT>        = <UVAR>+<INTEGER>[$INT] ;
<UVAR>         = <VARA(/adr)>[$UADR(adr/)] ;
<VARA(/adr)>   = <NAME>["(/adr) adr:=reco(name);"] ;
```

ここで、<VARA(/adr)>は、<NAME>で受理した変数に割り当てられているアドレスを合成属性adrとして返す。

5.2 文の連接

次に連続した文の翻訳について述べる。連続した文をそれぞれ一つの状態遷移を持つ単位オートマトンに変換した後、これらを連接接続する。このような文の連接を定義するRTFは、次の通りである。

```
<CONCEXPR(/aptr1)> = <AEXPR(/aptr1)>(<AEXPR(/aptr2)>[$CONC(aptr1,aptr2/aptr1)])* ;
```

ここで、<AEXPR(/aptr)>は、算術代入文を定義した非終端記号である。

5.3 制御文

制御文で用いる条件式を表すオートマトンを生成するためのRTFは、次の通りである。生成したオートマトンへのポインタは合成属性aptrとして返される。

```
<COND(/aptr)> = [$B(/aptr)]<STATE>('='<STATE>[$COND(0/)]
                                  +'>'<STATE>[$COND(1/)]
                                  +'<'<STATE>[$COND(2/)]
                                  +'<='<STATE>[$COND(3/)]
                                  +'>='<STATE>[$COND(4/)]
                                  +'<>'<STATE>[$COND(5/)]
                                  )[$E] ;
```

5.3.1 IF文

形式I　IF　A　THEN　B ;
(A:条件式　B:実行文)

この形式の文のオートマトンへの変換を定義するRTFは、次の通りである。

```
<IF1(/aptr)> = 'IF'' '' '*<COND(/aptr)>'THEN'' '<SUB(aptr/aptr)>[$ELSE(/eptr)][$PLUS(aptr,eptr/aptr)] ;
<SUB(aptr/aptr)> = ('(&'' '<BEGIN(/bptr)>
                    +ELSE<AEXPR(/bptr)>
                    )[$CONC(aptr,bptr/aptr)] ;
<BEGIN(/bptr)> = <AEXPR(/bptr)>
                 [(e:=0)]([?(e=0)]('&)'[(e:=1)]
                                  +ELSE<AEXPR(/cptr)>[$CONC(bptr,cptr/bptr)]
                                  )
                        )* ;
```

<SUB(aptr/aptr)>は、"THEN"以下の実行文を表すオートマトンを生成し、条件式を表すオートマトンと連接変換する。
<BEGIN(/bptr)>は、実行文が複文である場合の処理を表す。
[$ELSE(/eptr)]は、条件式の値がfalseである場合の処理を表す単位オートマトンを生成する。

形式II　IF　A　THEN　B　ELSE　C ;
(A:条件式　B,C:実行文)
この形式の文のオートマトンへの変換を定義するRTFは、次の通りである。

```
<IF2(/aptr)> = 'IF'' '<COND(/aptr)>'THEN' '<SUB(aptr/aptr)>
               [$ELSE(/eptr)]'ELSE'' '<SUB(eptr/eptr)>
               [$PLUS(aptr,eptr/aptr)] ;
```

この形式では、形式IのRTFに"ELSE"以下の実行文の翻訳とオートマトンの生成を行なうための「'ELSE'' '<SUB(eptr/eptr)>」が追加されている。

5.3.2 WHILE文

WHILE文のオートマトンへの変換を定義するRTFは、次の通りである。

```
<WHILE(/aptr)> = 'WHILE'' '<COND(/aptr)>'DO'' '<SUB(aptr/aptr)>[$CLOS(aptr/aptr)] ;
```

5.3.3 FOR文

FOR文のオートマトンへの変換を定義するRTFは、次の通りである。

```
<FOR(/aptr)> = 'FOR'' '<FORHEAD(/aptr,adr)>' ''DO'' '<FORBODY(adr/bptr)>[$CONC(aptr,bptr/aptr)]
               <FOREND(/eptr)>[$CONC(aptr,eptr/aptr)] ;
```

```
<FORHEAD(/aptr,adr)> = <VARA(/adr)>[$B(/aptr)]':='<STATE>[$OADR(adr/)]
                       ' ''TO'' '<STATE>[$E] ;
<FORBODY(adr/aptr)> = <FORTEST(adr/aptr)><SUB(aptr/aptr)><PINC(adr/bptr)>
                      [$CONC(aptr,bptr/aptr)][$CLOS(aptr/aptr)] ;
<FORTEST(adr/aptr)> = [$B(/aptr)][$DUP][$UADR(adr/)][$COND(3/)][$E] ;
<PINC(adr/aptr)> = [$B(/aptr)][$UADR(adr/)][$OPR(2,16/)][$OADR(adr/)][$E] ;
<FOREND(/aptr)> = [$B(/aptr)][$DROP][$E] ;
```

<FORHEAD(/aptr,adr)>は、パラメータの初期値と終了値の設定を行なうためのオートマトンを生成する。終了値は、その設定後、汎用スタックの最上部にある。

<FORTEST(adr/aptr)>は、終了テストのためのオートマトンを生成する。[$DUP]は、汎用スタックの最上部にある終了値のコピーを汎用スタックに積むためのコードを生成する。

<PINC(adr/aptr)>は、パラメータの値を1だけ増すためのオートマトンを生成する。相続属性adrは、パラメータのアドレスである。[$OPR(2,16/)]は、汎用スタックの最上部の値を1だけ増すためのコードを生成する。

<FOREND(/aptr)>は、FOR文の後処理を行なうオートマトンを生成する。[$DROP]は、汎用スタックの最上部にある値を破棄するためのコードを生成する。

5.4 生成したコンパイラと既存のコンパイラの性能比較

MYLANGにより生成したコンパイラと既存のコンパイラとで性能の比較を行なった。性能比較は、コンパイル速度と生成される目的プログラムの実行速度の比較により行なう。計算機はMELCOM COSMO800 IIIを使用し、既存のコンパイラとしてPASCAL8000コンパイラを使用した。性能の比較に用いたプログラムは、次の通りである。

テスト・プログラム1

```
PROGRAM TEST1 ;
  VAR A,B,C,D,E,F,G,H,I,J:INTEGER ;
      X:INTEGER ;
  BEGIN
    A:=1 ; B:=2 ; C:=3 ;
    D:=4 ; E:=5 ; F:=6 ;
    G:=7 ; H:=8 ; I:=9 ;
    J:=10 ; K:=11 ;
    X:=(A+(B+(C+(D+(E+(F+(G+(H+(I+J)))))))));
  END ;
```

テスト・プログラム2

```
PROGRAM TEST2 ;
  VAR I,J:INTEGER ;
      K:INTEGER ;
  BEGIN
    I:=1 ;
    FOR J:=1 TO 10 DO BEGIN I:=1+1 ; WRITELN(J,I) END ;
    WRITELN(I,J) ;
    IF I=J THEN BEGIN K:=1 ; I:=I*J ; END
          ELSE BEGIN K:=0 ; I:=I+J ; END ;
    WRITE(I,J) ; WRITELN(K)
  END.
```

コンパイル時間の測定結果を表1に、実行速度の測定結果を表2に示す。

表1 コンパイル時間

テスト・プログラム	1	2
PASCALコンパイラ	11.2ms	12.2ms
生成したコンパイラ	76.1ms	70.5ms

表2 実行速度

テスト・プログラム	1	2
PASCALコンパイラ	5.2ms	5.8ms
生成したコンパイラ	9.3ms	10.4ms

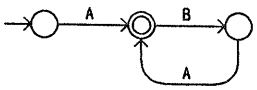
MYLANGが生成したコンパイラは、コンパイル時間がPASCALコンパイラよりも約5倍ほど遅い。この原因としては、MYLANGが生成したコンパイラは、中間語である有限オートマトンをシュミレートすることによって動作しているためと考えられる。また、字句解析フェイズがないことも原因であろう。

6．あとがき

本稿では、原プログラムをコンパイルする際の中間語として有限オートマトンを提案し、その変換について制御文を中心に述べた。本稿で示した以外の代表的な制御文にrepeat文とgoto文がある。
while文はループの先頭で終了判定を行なうのに対し、repeat文はループの最後で終了判定を行なう。今回我々が用いた有限オートマトンの閉包形は、終了判定をループの最初の状態遷移で行なっており、while文と同一の形式である。このため、repeat文をwhile文と同一の手法でオートマトンに変換することは出来ない。その解決策として、現在二つの方法を検討している。

その１つは、２つのオートマトン

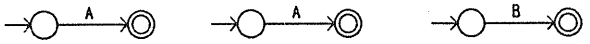
を



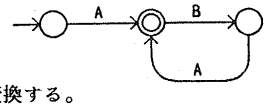
に変換する演算の導入である。この演算のための変換式の証明は、まだ完了していない。

もう１つは、オートマトンのコピーを生成する演算の導入である。まず、２つのオートマトン

のうちの一方のコピーを生成する。



そして、この３つのオートマトンに連接接続、閉包変換の演算を施し、



に合成変換する。

次いで、goto文であるが、この制御文の有限オートマトンへの変換には、ラベル(名札)の処理が必要である。すなわち、オートマトンの状態にラベルをつけ、それへの空系列遷移を与えることになる。これは、まだ解決していない。

参考文献

1) 安在、他:"半線形代数"、第1報～第8報、電子通信学会技術報告AL80-60,61,AL81-4,5,38,39,74(1981)
2) 安在:"言語処理系の生成系ＭＹＬＡＮＧの基礎概念"、昭和59年度特定研究「多元知識情報」Ｃ班研究会報告資料(1984.11)
3) 藤村、山之上、安在:"言語処理系ＭＹＬＡＮＧにおける標準活動記号を用いた制御文の翻訳"、九州工大研究報告(工学)、第48号、昭和59年3月
4) 山之上、安在:"属性付構文指示翻訳系の生成系ＭＹＬＡＮＧ"、情報処理学会論文誌、第26巻、第1号、pp.195-204(1985)
5) 安在:"構文、意味、および知識向き翻訳系の生成について"、情報処理学会、知識工学と人工知能研究会、31-3(1984)

付録　標準活動記号の機能概要

標準活動記号は、利用者が活動ルーチンを定義することなく使用できる活動記号である。標準活動記号の一般形は、次の通りである。

[$標準活動記号名(相続属性並び/合成属性並び)]

1.入力用標準活動記号

受理した文字は、長さ1の文字列としてシステム変数CHに代入される。

1) ALPHA　(属性なし)
入力文字が英字ならば、受理する。'A'+'B'+…+'Y'+'Z'と同等の機能。
2) NUM　(属性なし)
入力文字が数字ならば、受理する。'0'+'1'+…+'8'+'9'と同等の機能。
3) SYMBOL　(属性なし)
入力文字が特殊記号ならば、受理する。但し、"'"(single quotation)と" "(blank)は除く。

2.文字列処理用標準活動記号

システム変数CHとNAMEの内容を用い、結果はNAMEに代入される。但し、NAMEに代入される文字列の最大長は10文字であり、それを越えた場合、11文字目以降は捨てられる。また、文字列内に空白文字を含めることは出来ない。空白文字は、長さ0の文字列として取り扱われる。

1) CLS　(属性なし)
システム変数NAMEを初期化する。
2) CON　(属性なし)
システム変数NAMEの終りにCHの内容を付け加える。
3) NVAR(/val)
システム変数NAMEの内容である数字列を数値に変換する。変換後の数値は、合成属性valに代入される。

4) NSTR(val/)
相続属性valの値を数字列に変換する。変換後の数字列は、システム変数NAMEに代入される。

3.コード生成用標準活動記号
以下の処理を行なうMYLANGコードを生成する。

1) UADR(adr/)
相続属性adrの値が示すアドレスの内容を汎用スタックに積む。
2) OADR(adr/)
汎用スタックから値を取り出し、相続属性adrの値が示すアドレスに格納する。
3) PUI(val/)
相続属性valの値を汎用スタックに積む。
4) INT (属性なし)
[$NVAR(/val)][$PUI(val/)]と同等。
5) DUP (属性なし)
システム・スタックにスタック・トップの値を積む。
6) DROP (属性なし)
システム・スタックのスタック・トップの値を捨てる。
7) COND(opr/)
oprの値に対応する比較演算を行なう。oprと演算の対応は、次の通りである。

opr	演算
0	等しい
1	より大きい
2	より小さい
3	以上
4	以下
5	等しくない

8) OPR(2,calc/)
calcの値に対応する算術演算を行なう。calcと演算の対応は、次の通りである。

calc	演算
9	符合反転
11	加算
12	減算
13	乗算
14	除算
16	プラス1
17	マイナス1

9) END (属性なし)
この標準活動記号は、コードの生成を行なわない。コード生成を終了し、その後処理を行なう。

4.オートマトンの生成・合成用標準活動記号
オートマトンの生成・合成と、それに関連する処理を行なう。

1) BGN(/nptr)
非終端記号により呼び出されるオートマトンのヘッダを生成する。生成したヘッダへのポインタが合成属性nptrとして返される。この時点では、ヘッダとオートマトンは、関係付けられていない。
2) B(/aptr)
遷移を一つ持つオートマトンを生成し、遷移記号の生成を開始する。この時点では、遷移記号は不定である。生成するオートマトンへのポインタが合成属性aptrとなる。
3) E (属性なし)
標準活動記号「[$B(/aptr)]」と対で使用され、遷移記号の生成を終了する。対となる「[$B(/aptr)]」は、最後に呼び出されたものである。「[$B(/aptr)]」が呼び出されてから、「[$E]」が呼び出される間に生成されるMYLANGコードが遷移記号となる。
4) CONC(aptr1,aptr2/aptr3)
相続属性aptr1、aptr2が指し示すオートマトンを連接変換する。変換後のオートマトンへのポインタが合成属性aptr3となる。
5) CLOS(aptr1/aptr2)
相続属性aptr1が指し示すオートマトンを閉包形に変換する。変換後のオートマトンへのポインタが合成属性aptr2となる。
6) PLUS(aptr1,aptr2/aptr3)
相続属性aptr1、aptr2が指し示すオートマトンを並列接続する。変換後のオートマトンへのポインタが合成属性aptr3となる。
7) OPT(aptr1/aptr2)
相続属性aptr1が指し示すオートマトンを繰り返しの上限を一回とする閉包形に変換する。即ち、入力が変換前のオートマトンに受理されるか否かに拘わらず終状態に遷移する形に変換する。変換後のオートマトンへのポインタが、合成属性aptr2となる。
8) ELSE(/aptr)
特別な遷移記号"ELSE"を持つオートマトンを生成する。生成したオートマトンへのポインタが合成属性aptrとなる。この遷移記号を持つ遷移は無条件遷移であるが、分岐点において優先順位が最低になる。
初期状態からの遷移の遷移記号として"ELSE"を持つオートマトンを閉包変換してはならない。他のオートマトンの後ろに連接接続することも許されない。(逆は許される)
9) OUT(aptr,nptr/)
相続属性aptrが指し示すオートマトンを簡約決定性変換した後、相続属性nptrが、指し示すヘッダと関係付ける。このオートマトンが中間コードとなる。
10) NTM(nptr/aptr)
相続属性nptrが指し示すヘッダを持つオートマトンを呼び出す非終端記号を遷移として持つオートマトンを生成する。生成したオートマトンへのポインタが合成属性aptrとなる。

5.その他の標準活動記号

1) START(nptr/)
相続属性nptrが指し示すヘッダを持つオートマトンをメイン・ルーチンとする。
2) GET(nptr,area/)
相続属性nptrが指し示すヘッダを持つオートマトンが必要とするデータ領域を確保する。データ領域の大きさは、相続属性areaにより指定する。areaの値は0以上でなければならない。