言語処理系の生成系MYLANGにおける標準活動記号を用いた制御文の翻訳

（昭和57年11月30日　原稿受付）

情報工学教室　藤村啓二
同　山之上卓
同　安在弘幸

Translation of Control Statements Using Standard Action Symbols in the Language Processor Generator MYLANG

by Keiji FUJIMURA
Takashi YAMANOUE
Hiroyuki ANZAI

Abstract

For compilation of control statements such as IF, FOR and WHILE statements in programming languages, this paper presents automata, of which the structures represent the control flows defined by the statements, as intermediate form between the statements and their object codes, and gives a method translating the statements into the automata and a method generating translators, which implement the translation method, by using our language processor generator MYLANG.

Our process translating a given control statement into an automaton is as follows: At first, the statement is transformed into reverse Polish notation of a kind of regular expression which describes the object automaton, and then the notation is transformed into the automaton. The translation process is described by our given notation, attributed regular translation form, which is also regarded as a description of syntax and semantics of statements. MYLANG reads the notation and produces the above type of translator. Several action routines necessary in the notation are standardized and installed in MYLANG, so they are usable by means of only calling their names, i. e. standard action symbols, without defining the routines.

1. まえがき

コンパイラの開発において，算術式の逆ポーランド記法による表現や変換の技法は重要な技術である。しかしながら，IF 文，FOR 文，WHILE 文などの制御に関する逆ポーランド記法による表現や変換の技法は，現在十分に確立されているとは言えない。

一般に，与えられた式（expression）を逆ポーランド記法（で表わされた）式に変換するとき，その与えられた式の演算子や被演算子は変換後の逆ポーランド記法式にそのまま現われる（そして，時には前者で暗示的であった演算子，例えば乗算や連接の演算子，が後者で明示的に現われる）のが普通である。しかし，与えられた式が制御文や宣言文などの場合，このような逆ポーランド記法変換はほとんど不可能か，あるいは意味を持たない場合が多い。これに対して，必要ならば新しい演算子や被演算子を導入して，与えられた式に等価な逆ポーランド記法式に変換する方法が考えられる。これを拡張逆ポーランド記法変換と呼ぶ。

コンパイラがプログラムを翻訳する方式のうち，文を逆ポーランド記法に変換し，次にそれから三つ組や四つ組と呼ばれる中間語を生成するという方式がある。この方式は算術式では極めて自然に行なうことができる。しかし，この方式を制御文や宣言文などの場合に適用するとき，上述した様に逆ポーランド記法変換の拡張が必要となる。Gries[5] は，IF 文と配列の宣言を例に，この種

の拡張逆ポーランド記法変換について述べている。ここで要求されている逆ポーランド記法の形式は，それから中間語としての三つ組や四つ組に変換するのに適していることが必要である。

これに対して，我々のコンパイラが制御文を翻訳する方式では，中間語として新しくオートマトンモデルを採用した。与えられた制御文は，拡張逆ポーランド記法変換された後，その制御文に等価な処理を与えるオートマトンに変換される。この変換の中間の逆ポーランド記法式は，それから変換して得られるオートマトンを記述する式（正規表現の逆ポーランド記法式）の一種である。

本報告では，IF 文，FOR 文，および WHILE 文を例として制御文に対する上述の翻訳処理について述べる。まず，与えられた制御文を上記の種類の逆ポーランド記法式に変換する方法を示し，次に，この逆ポーランド記法式をプッシュダウンスタックとオートマトンの合成の手法を用いて，目的のオートマトンに変換する方法を示す。本報告では，言語処理系の生成システム MYLANG を用いて，上述の方法に基く制御文の翻訳処理システムを生成させる方法を与える。MYLANG は，この翻訳処理に必要な構文と意味の記述（属性付正規翻訳記法：属性付 RTF）を入力して，それが定義する処理系を生成する。我々は，制御文に対する上述の記述に典型的に用いられる活動ルーチンを標準化，すなわちシステム定義し，そのルーチン名を標準活動記号 (Standard Action Symbol) と名付けた。

第 2 章では，本報告で用いるオートマトンの状態図の記法を簡単に説明する。第 3 章と第 4 章では，各標準活動記号の機能を中心に述べる。第 5 章では，簡単なコンパイラの実例を用いて，制御文の記述法や標準活動記号の使用法を説明する。

2. オートマトン状態図の記法

本報告で用いる記法によるオートマトン状態図の例を図-2.1に示す。

状態図は，状態をあらわす節（図-2.2a）と，状態の変化をあらわす矢印（図-2.2b）とから構成される。すなわち，オートマトンが状態 q で入力記号 a を読んだとき状態 p に移るならば，q から p への矢印を書き，それに名札 a を付ける（図-2.2c)。終状態は二重の円で示し（図-2.2d)，初期状態は矢印と円との組合せで示す（図-2.2e)。

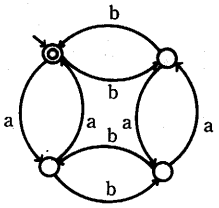


図-2.1 オートマトンの状態図

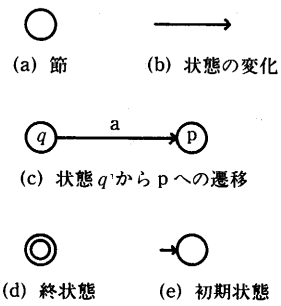


図-2.2 状態図の各要素

初期状態より始まり，終状態で記号を入力し終われば，この入力記号列はオートマトンにより受理されたと言う。また，受理可能な全ての記号列の集合を，そのオートマトンの受理言語と言う。

複数のオートマトンを使うことにより，更に広範囲の文法の定義も可能である。複数のオートマトンを使用した例が図-2.3である。これは算術式を受理するオートマトンである。

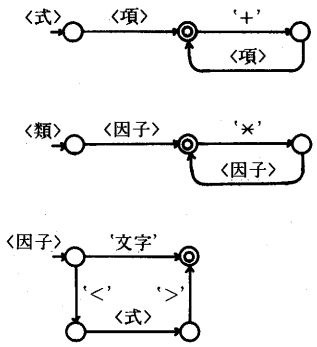


図-2.3 算術式を受理するオートマトン

図-2.3において，＜と＞で囲まれた記号を非終端記号と呼ぶ。非終端記号による遷移は，その非終端記号に対応するオートマトンが挿入されることを示す。

囲まれた記号を終端記号と呼び，入力記号を表わす。

3. 標準活動記号（その1）

本章では，MYLANG に新たに導入した標準活動記号のうち，オートマトンの生成と合成に関連するものについて述べる。

3.1. 標準活動記号の記法

正規翻訳記法 (RTF) において，活動記号は，

|名前（相続属性/合成属性）|

と書かれる。名前は英字で始まる10文字以内の英数字の文字列である。相続属性と合成属性は英字で始まる10文字以内の文字列であり，各々，10個までの記述が許される。なお，相続属性として，数字も許される。

これに対して，標準活動記号は，

|$ 名前（相続属性/合成属性）|

と書かれる。名前と属性の書式は，上述の活動記号と同じである。

一般の活動記号は，動作内容を ACTION 部において記述しなくてはならない（利用者定義）が，標準活動記号の動作内容の記述は不要（システム定義）である。

また，標準活動記号名を一般の活動記号名として使用することは可能である。しかし，この場合は標準活動記号とはならないため，動作内容の記述が必要となる。

3.2. 標準活動記号（その1）

1) $INIT

オートマトン生成システムの初期化を行なう。この活動記号は，他の活動記号の全てに先立って実行されなければならない。

2) $BGN

オートマトンの生成，合成を開始する。

3) $END

オートマトンの生成，合成を終了する。

4) $B, $E

$B により初期状態を，$E により終状態をそれぞれ生成する。すなわち，$B と $E の組合せにより，遷移を1つだけ持つオートマトンが生成される (図-3.1)。このオートマトンの遷移は，RTF の中の $B と $E の間に出現する活動記号によって生成される MYLANG の目的コードにより行なわれる。

|$B|<A>|$E|

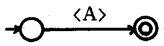


図-3.1 |$B| と |$E| により生成されるオートマトン

5) $OUT

オートマトンの簡約決定性変換を行なう。同時に，オートマトンを MYLANG の目的コードとして，作業ファイルに出力する。

オートマトン生成システムを使用する場合，1)～5)は必ず使わなければならない。

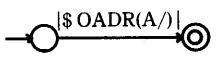


図-3.3 図-3.2 のRTF により生成されるオートマトン

ここで，1)～5)を用いた例を図-3.2に示す。これは，変数に整数を代入する代入文を翻訳するコンパイラを定義した RTF である。この例では，図-3.3に示す様に生成されるオートマトンの遷移は1つである。

```
(*?RTF*)
  <S>         =|$INIT||$BGN|<VARA(/A)>|$B|':''='<INTEGER>|$OADR(A/)||$E|<E>;
  <E>         =';'|$OUT||$END|;
  <VARA(/A)>=<NAME>|VARA(/A)|;
  <NAME>      =|$CLR|<ALPH>|$CON|((<ALPHA>+<NUM>)|$CON|)*;
  <INTEGER>   =|$CLR|<NUM>|$CON|(<NUM>|$CON|)*|$INT|;
  <NUM>       =|$NUM|;
  <ALPHA>     =|$ALPHA|;
(*?END*)
(*?ACTION*)
  (*|VARA(/A)|*)
    BEGIN  A:=LVRECO(NAME)  END;
(*?END*)
```

図-3.2 標準活動記号の使用例1

3.3. 標準活動記号（その2）

本節では，オートマトンの合成に必要な標準活動記号について述べる。

6) $CONC

最後に生成されたオートマトンを，その直前に生成されたオートマトンの後に連結する（図-3.4）。

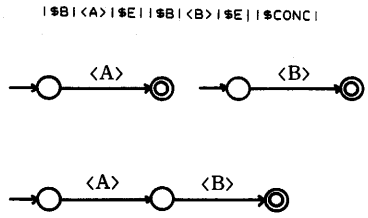


図-3.4 $CONC の使用例

7) $CLOS

この活動記号の直前に生成されたオートマトンを閉包に変換する（図-3.5）。

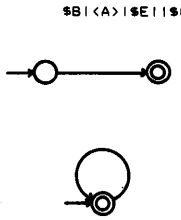


図-3.5 $CLOS の使用例

8) $OPT

この活動記号の直前に生成されたオートマトンを0回，または1回の繰り返しに変換する（図-3.6）。

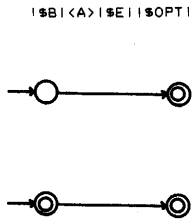


図-3.6 $OPT の使用例

9) $PLUS

この活動記号の直前に生成された2つのオートマトンの初期状態を同一状態にする（図-3.7）。

10) $ELSE

9) によりオートマトンの合成を行なった場合，

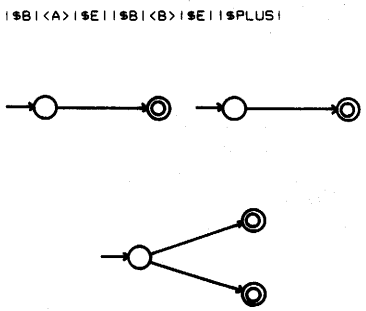


図-3.7 $PLUS の使用例

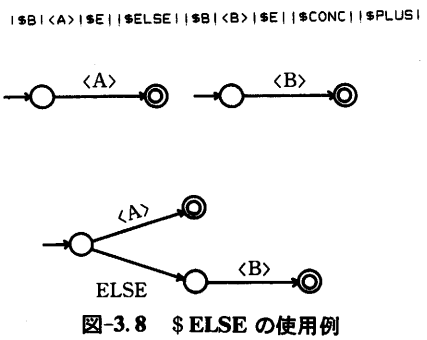


図-3.8 $ELSE の使用例

図-3.7が得られるが，ここでは実行時に <A> と <B> の遷移のうち，どちらが実行可能であるかが調べられる。その際どちらから調べるかは保証されない。しかし，$ELSE を用いることにより，調べる順序を指定することができる。

図-2.7の場合，まず <A> が調べられる。これが実行不能な場合に <B> が調べられる。但し，<B> は必ず実行可能でなければならない。

4. 標準活動記号（その2）

本章では，記号列処理用，および目的コード生成用の標準活動記号について述べる。

4.1. 記号列処理活動記号

1) $ALPHA

入力した文字が英字であるかどうかを調べる。

2) $NUM

入力した文字が数字であるかどうかを調べる。

3) $SYMBOL

入力した文字が特殊文字であるかどうかを調べる。

4) $BLANK

入力した文字が空白であるかどうかを調べる。

5) $CLS

システム変数 NAME を初期化する。

6) $CON

NAME に格納されている 文字列の最後尾に入力した文字を連結し，NAME に返す。但し，連結後の文字列の長さが11文字になった場合，最後尾の文字は捨てられる。

4.2. 目的コード生成活動記号

今まで示してきた活動記号は属性を持っていなかった。しかし，本節に示す活動記号には属性の付加が必要である。

1) $OADR (A/)

汎用スタックから値を取り出し，相続属性 A の示す場所に格納する命令を生成する。

2) $UADR (A/)

相続属性 A の示す場所に格納されている値を，汎用スタックに PUSH する命令を生成する。

3) $PUI (A/)

相続属性 A の値を汎用スタックに PUSH する命令を生成する。

4) $INT

システム変数 NAME に格納されている数字の文字列を整数に変換し，汎用スタックに PUSH する命令を生成する。

5) $COND (X/)

相続属性 X の値に対応する比較演算子（表-1に示す）を目的コードとして生成する。

表-1 $COND のコード表

コード	比較演算子
0	A＝B
1	A＞B
2	A＜B
3	A≧B
4	A≦B
5	A≠B

6) $OPR (S, N/)

① 相続属性 S＝2 の場合

相続属性 N の値に対応する演算子（表-2に示す）を目的コードとして生成する。

② 相続属性 S＝10 の場合

表-2 $OPR のコード表 (S=2)

コード	演　算　子
9	符号反転の単項演算子
11	加法の二項演算子
12	減法の二項演算子
13	乗法の二項演算子
14	除法の二項演算子

(i) 相続属性 N＝0 の場合

出力動作を表わす目的コードを生成する。この目的コードは，汎用スタックの TOP の値により指定された数だけ汎用スタックから値を取り出し，出力する（図-4.1)。出力は，スタックの底により近い値から順に行なわれる。

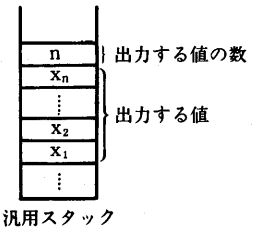


図-4.1 出力命令のスタックの使用

(ii) 相続属性 N＝1 の場合

入力動作を示す目的コードを生成する。この目的コードにより，外部より入力した値が汎用スタックに PUSH される。

5. 標準活動記号の使用例

本章では，例題を用いて，オートマトン生成システムの使用法とオートマトンの合成過程を示す。

標準活動記号の使用例として，簡単なコンパイラの定義例を図-5.1に示す。

5.1. 例題の言語仕様

本節では，図-5.1に示したコンパイラが認識する文の書式について述べる。これらの文が変換されるオートマトンは5.2.で示す。

5.1.1. データ流れに関する文

1) 算術代入文

算術代入文の書式は，PASCAL8000 の整数型の書式に準じる。但し，除算の演算子として '/' を使用する。

```
(*?RTF*)
  <ACTION>       =|$INIT|<AUTOMATA>|$OUT|' '*'.'|$END|;
  <AUTOMATA>     =' '*'(''&'|$BGN|<ASBUN>'&'')'' '*;
  <ABUN>         =' '*(<ASTATE>+ELSE<AEXPR>)' '*';';
  <ASTATE>       =<AFOR>+<AWHILE>+<AIF>+<NONAS>;
  <ASBUN>        =<ABUN>(<ABUN>|$CONC|)*' '*;
  <SUB>          =' '*'('<ASBUN>')'' '*;
  <AEXPR>        =<VARA(/A)>|$B|':''='<STATE>|$OADR(A/)||$E|;
  <STATE>        =<FACTOR>('+'<FACTOR>|$OPR(2,11/)|
                  '-'<FACTOR>|$OPR(2,12/)|)*;
  <FACTOR>       =<ELMT>('*'<ELMT>|$OPR(2,13/)|+'/'<ELMT>|$OPR(2,14/)|)*;
  <ELMT>         ='+'/<ELMTS>+'-'<ELMTS>|$OPR(2,9/)|;
  <ELMTS>        =<TS>+'('<STATE>')';
  <TS>           =<UVAR>+<INTEGER>;
  <AFOR>         ='FOR'<FORINT(/A)>'TO'<FORFIN>'DO'<FORSTATE(A/)>;
  <FORINT(/A)>   =' '*|$B|<VARA(/A)>':''='<STATE>|$OADR(A/)|' '*;
  <FORFIN>       =' '*<STATE>|IPUSH||$E|' '*;
  <FORSTATE(A/)>=' '*<FORCOND(A/)><SUB>|$CONC|<FOREND(A/)><IPOP>;
  <FORCOND(A/)> =|$B||FORCOND||$E|;
  <IPOP>         =|$B||IPOP||$E||$CONC||$CONC|;
  <FOREND(A/)>   =|$B|<PINC(A/)>|$E||$CONC||$CLOS|;
  <PINC(A/)>     ==|$UADR(A/)||$PUI(1/)||$OPR(2,11/)||$OADR(A/)|;
  <AWHILE>       ='WHILE'<ACOND>'DO'<SUB>|$CONC||$CLOS|;
  <AIF>          ='IF'<ACOND>'THEN'<SUB>|$CONC|<ELSESTATE>;
  <ELSESTATE>    ='ELSE'|$ELSE|<SUB>|$CONC||$PLUS|;
  <NONAS>        ='WRITELN''('|$B|' '*<WP>' '*')'|$E|;
  <WP>           =<UVAR>|(A:=1)|(' '*','' '*<UVAR>|(A:=A+1)|)*
                 |$PUI(A/)||$OPR(10,0/)|;
  <ACOND>        =' '*|$B|<STATE><SACOND>|$E|' '*;
  <SACOND>       ='='<STATE>|$COND(0/)|+'>'<ACOND1>+'<'<ACOND2>;
  <ACOND1>       =<STATE>|$COND(1/)|+'='<STATE>|$COND(3/)|;
  <ACOND2>       =<STATE>|$COND(2/)|+'='<STATE>|$COND(4/)|
                  +'>'<STATE>|$COND(5/)|;
  <UVAR>         =<VARA(/A)>|$UADR(A/)|;
  <VARA(/A)>     =<NAME>|VARA(/A)>;
  <NAME>         =|$CLS||$ALPHA||$CON|(<|$ALPHA|+|$NUM|)||$CON|)*;
  <INTEGER>      =|$CLS||$NUM||$CON|(|$NUM||$CON|)*|$INT|;
(*?END*)
(*?ACTION*)
  (*|IPUSH|*)
    BEGIN WRITELN(USRLMD,'(',0) END;
  (*|IPOP|*)
    BEGIN WRITELN(USRLMD,')',0) END;
  (*|FOREND|*)
    BEGIN WRITELN(USRLMD,'=',0) END;
  (*|VARA(/A/)|*)
    BEGIN A:=LVRECO(NAME) END;
(*?END*)
```

図-5.1 標準活動記号の使用例2

2) 出力文

WRITELN (X_1, X_2,…, X_n); n≧1

X_i は変数，または整定数

5.1.2. 制御流れに関する文

1) IF 文

IF ＜比較式＞ THEN (＜文並び＞) ELSE (＜文並び＞);

＜比較式＞の値が真ならば，THEN 以下を実行し，偽ならば，ELSE 以下を実行する。

2) FOR 文

FOR ＜算術代入文＞ TO ＜算術式＞ DO (＜文並び＞);

＜算術代入文＞により，パラメータを初期化する。

＜算術式＞で示される終了値よりもパラメータの値が大きくなるまでパラメータの値を1ずつ増加させながら，DO 以下を実行する。初期値が終了値よりも大きい場合には，DO 以下は実行されない。

3) WHILE 文

WHILE ＜比較式＞ DO (＜文並び＞);

＜比較式＞の値が真である間，DO 以下を実行する。

5.2. 制御文の逆ポーランド記法

ここでは，制御文の逆ポーランド記法について述べる。その準備として，本節で用いる演算子を5.2.1.で説明する。これらの演算子は，オートマトン演算，すなわちオートマトンの合成を行なう演算子である。

5.2.1. オートマトン演算子

1) ・

オートマトンの連結を行なう。

2) ＋

2つのオートマトンの初期状態を同一状態にする。

3) ＊

オートマトンを閉包に変換する。

4) ELSE

条件判定の結果を反転させる。

5.2.2. 制御文の逆ポーランド記法

1) IF 文

((＜比較式＞, ＜文並び＞)・, (ELSE, ＜文並び＞)・)＋

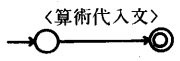
2) FOR 文

(＜FOR 文頭部＞, (((＜終了判定＞, ＜文並び＞)・, ＜パラメータの増加＞)・)＊)・

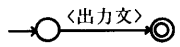
3) WHILE 文

((＜比較式＞, ＜文並び＞)・)＊

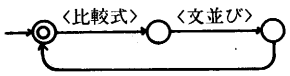
算術代入文，出力文，および各制御文のオートマトン状態図による表現を図-5.2に示す。上記の各式は，同図の対応するオートマトンを記述する正規表現の逆ポーランド記法式である。次節で上述の制御文のオートマトンの合成過程を示す。



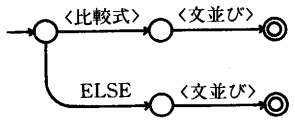
(a) 算術代入文



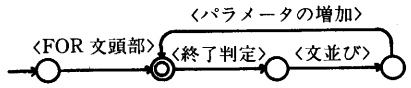
(b) 出力文



(c) WHILE 文



(d) IF 文



(e) FOR 文

図-5.2 逆ポーランド記法式のオートマトン表現

5.3. オートマトンの合成過程

本節では，制御文を例として，オートマトンの合成過程を示す。

5.3.1. IF 文の合成過程

WHILE 文のオートマトン表現への変換を定義する RTF を図-5.3に示す。次にこの RTF からオートマトンが合成される過程を示す。

```
<AIF>        ='IF'<ACOND>'THEN'<SUB>|$CONC|<ELSESTATE>';
<ELSESTATE>  ='ELSE'|$ELSE|<SUB>|$CONC||$PLUS|;
```

図-5.3 IF 文の RIF

① ＜ACOND＞ は比較式の処理を行なう。これは1つのオートマトンとなる (図-3.4a)。

② ＜SUB＞ は文の並びで，これは1つのオートマトンとなり，|$CONC| により ＜ACOND＞ と連結される (図-5.4b)。

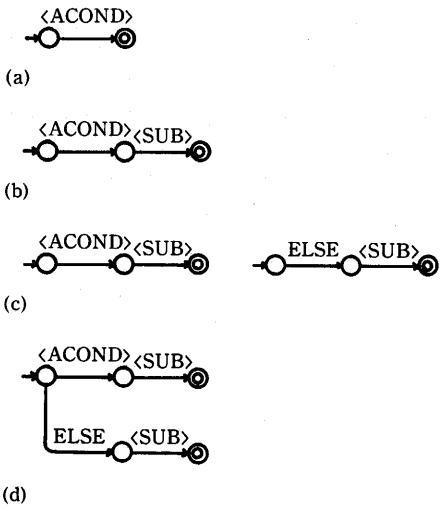


図-5.4 IF 文のオートマトンの合成過程

③ |$ELSE| により，1つのオートマトンが生成される。これは，⟨ACOND⟩ の結果を反転させる遷移を持つ。続く ⟨SUB⟩，|$CONC| により連結が行なわれる（図-5.4c)。

④ |$PLUS| により，IF 文の合成が完了する（図-5.4d)。

5.3.2. FOR 文の合成過程

FOR 文のオートマトン表現への変換を定義する RTF を図-5.5に示す。次にその変換の過程を示す。

① ⟨FORINT (A/)⟩ と ⟨FORFIN⟩ が，1つのオートマトンとなる（図-5.6a)。

② FOR 文の終了判定を行なう |FORCOND| と ⟨SUB⟩ が |$CONC| により連結される（図-5.6b)。

③ パラメータを1だけ増加させる ⟨PINC (A/)⟩ がオートマトンとなり，|$CONC| により ⟨SUB⟩ の後連結される（図-5.6c)。

④ |$CLOS| により，|FORCOND| から ⟨PINC (A/)⟩ までが閉包に変換される（図-5.6d)。

⑤ FOR 文の終了処理を行なう |IPOP| が |$CONC| により，閉包の後に連結される（図-5.6e)。

⑥ |$CONC| により最初に生成されたオートマトンが，閉包の前部に連結され，FOR 文の合成を終了する（図-5.6f)。

```
<AFOR>          ='FOR'<FORINT(/A)>'TO'<FORFIN>'DO'<FORSTATE(A/)>;
<FORINT(/A)>    =' '*|$B|<VARA(/A)>':''='<STATE>|$OADR(A/)|' '*;
<FORFIN>        =' '*<STATE>||PUSH||$E|' '*;
<FORSTATE(A/)>=' '*<FORCOND(A/)><SUB>|$CONC|<FOREND(A/)><IPOP>;
<FORCOND(A/)> =|$B||FORCOND||$E|;
<IPOP>          =|$B|||IPOP||$E||$CONC||$CONC|;
<FOREND(A/)>    =|$B|<PINC(A/)>|$E||$CONC||$CLOS|;
<PINC(A/)>      =|$UADR(A/)||$PUI(1/)||$OPR(2,11/)||$OADR(A/)|;
```

図-5.5 FOR 文の RTF

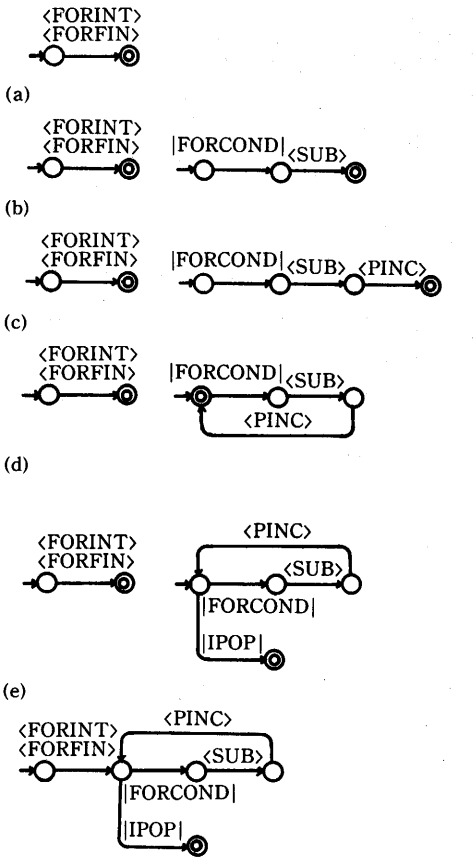


図-5.6 FOR 文の合成過程

```
<AWHILE>        ='WHILE'<ACOND>'DO'<SUB>|$CONC||$CLOS|;
```

図-5.7 WHILE 文の RTF

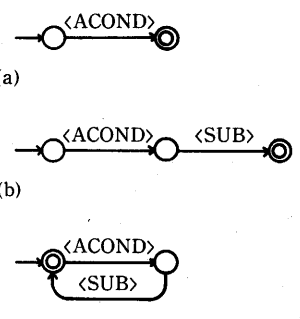


図-5.8 WHILE 文のオートマトンの合成過程

5.3.3. WHILE の合成過程

WHILE 文のオートマトン表現への変換を定義する RTF を図-5.7に示す。次にこの RTF からオートマトンが合成される過程を示す。

① ＜ACOND＞ が1つのオートマトンとなる（図-5.8a)。

② |$CONC| により，＜SUB＞ が ＜ACOND＞ と連結される（図-5.8b)。

③ |$CLOS| により，全体が閉包に変換され，WHILE 文の合成を終了する（図-5.8c)。

6. あとがき

本報告では，IF 文，FOR 文，および WHILE 文の逆ポーランド記法への変換を，プッシュダウンスタックとオートマトンの合成の手法を用いて示し，同時に，標準活動記号を導入した MYLANG を用いて，上述の制御文の逆ポーランド記法変換の実現例を示した。

しかし，IF 文，WHILE 文，FOR 文以外の制御文，すなわち REPEAT 文や GOTO 文などの逆ポーランド記法変換については，まだ未解決であり，現在研究中である。また，宣言文についても未解決である。

今回，標準活動記号を幾つか MYLANG に導入したが，これらだけでは，手続きや関数を持つプログラムを解釈するコンパイラを定義することは困難である。そのため，今回導入した標準活動記号の改良や，新しい標準活動記号の追加が必要である。

参考文献

1) 山之上他：“属性付正規翻訳記法と属性付構文向き翻訳” 九工大研究報告，No. 46, pp. 61–68, (1983)

2) 竹中豊弘：“言語処理系の自動生成に関する研究－MYLANG の開発” 九工大卒業論文，(1983)

3) Hopcroft, Ulman 共著，野崎，木村共訳：“言語理論とオートマトン”，サイエンス社，(1981)

4) Manna, Z: “Program Schema.” In: Aho, A.V. (ed): Currents In the Theory of Computing. Prentice-Hall (1973)

5) Gries,. D 著，牛島訳：“コンパイラ作成の技法”，日本コンピュータ協会 (1978)