報　文

# 決定性有限オートマトン生成系の開発

岐阜職業訓練短期大学校　宮田　利通

## Implementation of Deterministic Finite Automaton Generator

Toshimichi Miyata

要　約　正規表現を入力すると、これを受理する決定性有限オートマトンを出力する生成系を開発した。

生成系は、正規表現を読み込むと、まず、トンプソンの構成法により非決定性有限オートマトンを生成する。次に、非決定性有限オートマトンから部分集合構成法により決定性有限オートマトンを生成する。更に、こうして得られた決定性有限オートマトンの状態数を最小化して、最終的に簡易化した決定性有限オートマトンを生成する。

決定性有限オートマトン生成系は、シミュレーション機能を持っているので、入力された言語が受理可能かどうか確認することができる。

生成系の実行は、教育的配慮からメニューを選択することで、非決定性有限オートマトン、決定性有限オートマトン、シミュレーションを順に行うことができる。

生成系により得られた決定性有限オートマトンは、オートマトン理論、形式言語理論、コンパイラー理論などの教材として利用できるのみならず、字句認識系のプログラム開発のツールとしても利用できる。

## I　はじめに

情報工学を学習するための重要な基礎科目にオートマトン理論や形式言語理論がある。これらの科目は、電気工学における電気磁気学や電気回路理論に相当するものである。さらに、形式言語理論を理解するには、オートマトン理論の知識が不可欠である。それゆえ、職業訓練短期大学校おいて情報工学を教えるうえで、この決定性有限オートマトン生成系は役に立つ。

オートマトンは、アルファベット（日常語における意味ではない）から構成された記号列で表された言語を受理する自動機械の数学モデルである。各種オートマトンのうちで有限オートマトンは、最も基本的なものでありかつ重要である。有限オートマトンは、コンパイラの字句解析、オペレーティング・システムのコマンド解析、テキスト処理におけるパターン照合などに応用されている。このように、理論を理解するためばかりでなく実際にソフトウェアの開発においても利用される。

今回開発した有限オートマトン生成系は、入力として正規表現を読み込み、非決定性有限オートマトンを経て決定性有限オートマトンを出力する。出力された決定性有限オートマトンは、入力された正規表現で表される言語を受理するものの内で、状態数の最小な決定性有限オートマトンである。決定性有限オートマトンが求まれば、これからプログラムを設計することは容易である。

生成系の開発用言語は、Lispを用いた。

論文受付日　1991.2.12

## II　生成系の構造

生成系の入力には正規表現を用いた。言語を定義するための方法として正規表現以外にBNF形式、生成規則などが良く使われる。しかし、機械処理が容易なものとしては、正規表現が最も適している。

生成系は正規表現を読み込むと、構文解析をした後トンプソンの構成法により最初に非決定性有限オートマトンを生成する。正規表現からただちに決定性有限オートマトンを求めるより、非決定性有限オートマトンを求めるほうが簡単である。

次に、トンプソンの構成法により求めた非決定性有限

オートマトンを部分集合構成法を適用して、決定性有限オートマトンを生成した。同一の言語を受理する決定性有限オートマトンは多数存在する。部分集合構成法により得られた決定性有限オートマトンは幾つかある内の一つである。そこで、状態数の最も少ない決定性有限オートマトンを求めた。得られた結果は、遷移状態が明確になるように表示出力した。

図1に開発した決定性有限オートマトンの生成系の構成を示す。

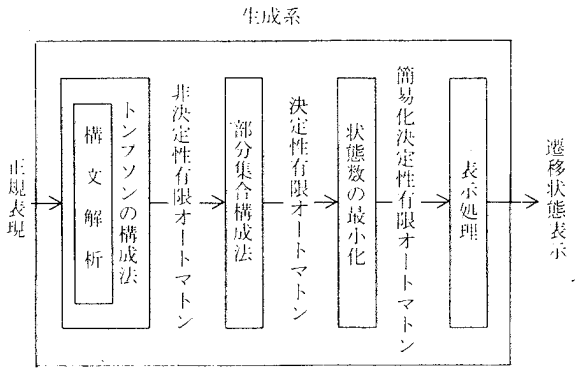


**図1　決定性有限オートマトン生成系の構成**

## Ⅲ　生成系のデータ構造と制御構造

1　正規表現

生成系に入力可能な正規表現は、次の通りである。

(1) 〈@〉 は正規表現であり、空列をからなる言語を表す。

(2) 〈a〉 は正規表現であり、一つの記号からなる言語を表す。

(3) RとSは正規表現であり、それぞれの言語を$L_R$と$L_S$で表せば、合併演算R｜Sは正規表現であり、$L_R \cup L_S$を表す。連接演算R・Sは正規表現であり、$L_R \cdot L_S$を表す。閉包演算　R＊　は正規表現であり、$L_R{}^*$を表す。

各演算子の優先順位は、＊、・、｜の順である。

正規表現の例として、識別子を記述すると、次の通りである。

〈id〉 = 〈letter〉・(〈letter〉｜〈digit〉)＊

生成系は、正規表現を読み込むと演算子の優先順位を利用して逆ポーランド記法へ変換する。変換をしながら次々に非決定性有限オートマトンを生成していく。

2　非決定性有限オートマトン

正規表現から非決定性有限オートマトンへ変換するにはトンプソンの構成法を用いた。

(1) 〈@〉 に対しては、次のS式へ変換した。

(Ni (@ Nf))

(2) 〈a〉 に対しては、次のS式へ変換した。

(Ni (a Nf))

(3) R｜S に対しては、次のS式へ変換した。

(Ni (@ Ri) (@ Si))
(Ri (@ Nf))
(Si (@ Nf))

(4) R・S に対しては、次のS式へ変換した。

(Rf (@ Sf))

(5) R＊ に対しては、次のS式へ変換した。

(Ni (@ Ri) (@ Nf))
(Rf (@ Ri) (@ Nf))

正規表現から非決定性有限オートマトンへの変換処理を図2の行動ダイアグラムに示す。図3は、識別子を表す非決定性有限オートマトンの内部表現とそれを状態遷移図に書き表したものを示す。S式で表される内部表現は、生成系で実際に処理した結果を示す。

決定性有限オートマトンの生成

WHILE トークンがある。

WHEN トークンは、アルファベットである。

アルファベットに対する非決定性有限オートマトンを生成する。

データスタックへ入れる。

WHEN トークンは、演算子である。

WHILE 演算子スタックトップ≠字句 かつ
演算子スタックトップ≠左括弧

WHEN 合併演算子

合併演算に対する非決定性有限オートマトンを生成する。

WHEN 連接演算子

連接演算に対する非決定性有限オートマトンを生成する。

WHEN 閉包演算子

閉包演算に対する非決定性有限オートマトンを生成する。

IF トークン≠右括弧

トークンを演算子スタックへ入れる。

WHILE 演算子スタックに演算子がある。

WHEN 合併演算子

合併演算に対する非決定性有限オートマトンを生成する。

WHEN 連接演算子

連接演算に対する非決定性有限オートマトンを生成する。

WHEN 閉包演算子

閉包演算に対する非決定性有限オートマトンを生成する。

**図２　正規表現から非決定性有限オートマトンへの変換処理に関する行動ダイアグラム**

内部表現

((S10)　　(S4(@ S8))
(S2(@ S9))　　(S7(@ S3)(@ S5))
(S8(@ S7)(@ S10))　　(S5(digit S6))
(S9(@ S7)(@ S10))　　(S3(letter S4))
(S6(@ S8))　　(S4(letter S2)))

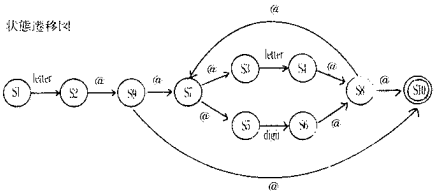


**図３　識別子を表わす非決定性有限オートマトン**

## 3　決定性有限オートマトン

非決定性有限オートマトンは、遷移が入力記号により決定的に決まらないため、機械処理が容易でない。そこで次の条件を満足する有限オートマトンを求めた。

(1)　空列による遷移がない。

(2)　各状態からの遷移は、入力記号に対し、高々一つである。

部分集合構成法

Dに初期値（初期状態S0としたときの ε-CLOSURE(S0) の値）を入れる

WHILE Dの中にまだ印の付いていない状態Dsがある。

Dsを取り出し、それに印を付ける。

FOR すべての入力記号a

Dsの中の状態でaによる遷移先の状態の集合Tを求める。

集合Yは ε-CLOSURE(T) の値とする。

IF YはDの状態の集合にまだ加えられていない。

Yを印の付いていない状態としてDに加える。

入力記号aにより状態Tから状態Yへの遷移を記録する。

状態Dsからのすべての入力記号による遷移を決定性有限オートマトンに加える。

**図４　部分集合構成法の行動ダイアグラム**

```
(de subsetconstruction (S0 Sf)
    (prog   (Ds x D insymbl a T Dn y temp)
    (setq Ds (gensym1 'S))
    (put Ds 'state (epsilonclosure S0))
    (setq D (cons Ds D))
    (while  (setq Ds (getnonmarked D))
        (progn
        (setq temp nil)
        (put Ds 'mark t)
        (setq x (get Ds 'state))
        (setq insymbl !#inputsymbols)
        (while  insymbl
            (progn
            (setq a (car insymbl))
            (setq insymbl (cdr insymbl))
            (cond   ((setq T (ashift x a))
                        (setq y (epsilonclosure T))
                        (cond   ((setq Dn (inD D y)) nil)
                                (t  (setq Dn (gensym1 'S))
                                    (put Dn 'state y)
                                    (setq D (cons Dn D))))
                        (setq temp (cons (list a Dn) temp))))))
    (setq !#DFA (cons (append (list Ds) temp) !#DFA))))
(return (startaccept S0 Sf))))
```

**図５　部分集合構成法のプログラム**

このように遷移が決定性なので、これを決定性有限オートマトンと呼ぶ。決定性有限オートマトンを求めるには、部分集合構成法を用いた。図４に部分集合構成法の行動ダイアグラムを示す。図5にプログラムを示す。なお、図４のε−CLOSURE（T）は、状態の集合Sから空列により遷移する先の状態の集合を表す。

内部表現
((S14 (letter S13) (digit S14))
(S13 (letter S13) (digit S14))
(S12 (letter S13) (digit S14))
(S11 (letter S12)))

状態遷移図

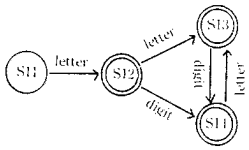


**図6　識別子を表わす決定性有限オートマトン**

図6は、識別子を表す決定性有限オートマトンの内部表現とそれを状態遷移図に書き表したものを示す。

4　状態数の最小化

決定性有限オートマトンの状態を最小化するには、まず受理状態と非受理状態の二つのグループに分割する。それぞれのグループについて、入力記号による遷移を調べる。遷移が自分自身のグループに属するものと別のグループに属するものとに分割する。これにより、同一とみなせる状態をまとめて一つの状態とすることにより、状態数の最小化をした。

```
状態数の最小化
    Πの初期値として最終状態と非最終状態の二つの
    グループの集合を作る。
    WHILE グループに分割ができる。
        FOR Πのすべてのグループ G
            FOR すべての入力記号 a
                IF  aによる遷移先の状態がGに属する。
                    同一グループに入れる。
                ELSE
                    別グループに入れる。
            分割されたグループをΠに加える。
```

**図7　状態数の最小化のダイアグラム**

内部表現
((S11 (letter S12))
(S12 (letter S12) (digit S12)))

状態遷移図

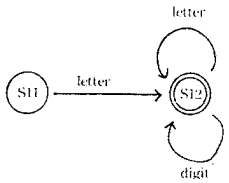


**図8　識別子を表わす簡易化された決定性有限オートマトン**

図7に状態数の最小化の行動ダイアグラムを示す。図8は、識別子を表す簡易化された決定性有限オートマトンの内部表現とそれを状態遷移図に書き表したものを示す。

5　シミュレーション機能

正規表現から生成された決定性有限オートマトンに実際に記号を入力して、正規表現で定義された言語がどのように遷移していくかシミュレーションができる。

```
シミュレーション機能
    Sを初期状態にする。
    入力記号aを読む。
    WHILE 入力がある。
        Sを入力記号aによる遷移先の状態にする
        次の入力記号aを読む。
    IF  Sが受理状態である。
        受理
    ELSE
        拒否
```

**図9　シミュレーション機能の行動ダイアグラム**

図9にシミュレーション機能の行動ダイアグラムを示す。

## Ⅳ　生成系の実行例

決定性有限オートマトン生成系にⅢ－1で定義した識別子を入力して、今回開発した生成系の操作を示す。

生成系を起動すると最初にメニューが表示される。メニューを選択することにより、次の五つの機能を実行できる。

(1)　正規表現から非決定性有限オートマトンへの変換
(2)　非決定性有限オートマトンから決定性有限オートマトンへの変換
(3)　決定性有限オートマトンから簡易化された決定性有限オートマトンへの変換
(4)　決定性有限オートマトンのシミュレーション
(5)　終了

正規表現から簡易化された決定性有限オートマトンへの変換を三段階に分けて求めるようにしたのは、教育的

配慮からである。すなわち、各変換段階が順を追って理解することができる。

図10に生成系の実行時の画面表示を示す。

```
**                                                          **
**            Deterministic Finite Automaton Tool           **
**                                                          **
**                (C) 1990  MIYATA Toshimichi               **
**                                                          **
**************************************************************

                    M E N U

        1.  regular expression   -->   NFA
        2.  NFA                  -->   DFA
        3.  DFA                  -->   reduced DFA
        4.  simulator of DFA
        5.  exit

Your selection? 1
Input a regular expression:
<letter>.(<letter>|<digit>)*
 start: S1          ===  letter          ==>  S2

        S3          ===  letter          ==>  S4

        S5          ===  digit           ==>  S6

        S7          ===  *epsilon*       ==>  S3
                    ===  *epsilon*       ==>  S5

        S4          ===  *epsilon*       ==>  S8

        S6          ===  *epsilon*       ==>  S8

        S9          ===  *epsilon*       ==>  S7
                    ===  *epsilon*       ==>  S10

        S8          ===  *epsilon*       ==>  S7
                    ===  *epsilon*       ==>  S10

        S2          ===  *epsilon*       ==>  S9

accept: S10

                    M E N U

        1.  regular expression   -->   NFA
        2.  NFA                  -->   DFA
        3.  DFA                  -->   reduced DFA
        4.  simulator of DFA
        5.  exit

Your selection? 2
 start: S11
                    ===  letter          ==>  S12

accept: S14         ===  letter          ==>    3
                    ===  digit           ==>  S14
```

```
                    M E N U

        1.  regular expression   -->   NFA
        2.  NFA                  -->   DFA
        3.  DFA                  -->   reduced DFA
        4.  simulator of DFA
        5.  exit

Your selection? 3
 start: S11
                    ===  letter          ==>  S12

accept: S12         ===  letter          ==>  S12
                    ===  digit           ==>  S12

                    M E N U

        1.  regular expression   -->   NFA
        2.  NFA                  -->   DFA
        3.  DFA                  -->   reduced DFA
        4.  simulator of DFA
        5.  exit

Your selection? 4
Input alphabet sequence:
<letter><letter><digit><letter>
Start state is S11.

        S11         === letter           ==> S12
        S12         === letter           ==> S12
        S12         === digit            ==> S12
        S12         === letter           ==> S12

Accepted at the state of S12.

                    M E N U

        1.  regular expression   -->   NFA
        2.  NFA                  -->   DFA
        3.  DFA                  -->   reduced DFA
        4.  simulator of DFA
        5.  exit

Your selection? 5
Value: nil
```

**図10　生成系の実行時画面**

## V　有限オートマトンの応用

正規表現で記述された言語を受理する決定性有限オートマトンを生成系により求め、その結果を実際に字句の認識系のプログラミングに応用してみる。プログラム例は、PL／Iを使い記述した。

例として、Ⅲ－1で定義した識別子を認識するプログラムを示す。まず、生成系により出力表示された結果から図8に示す状態遷移図を作る。これを更に修正して、各状態における認識系の動作を書き加える。すなわち、記号の読み込みと、識別子を求めるための処理を状態遷移図に書き加える。図11に修正決定性有限オートマトンを示す。この図から図12の行動ダイアグラムが得られる。実際のプログラムを図13に示す。

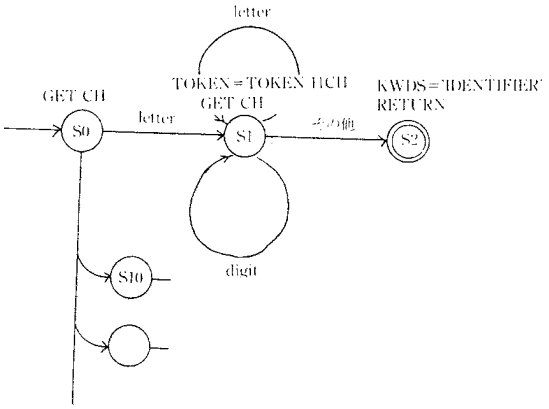


図11　修正決定性有限オートマトン

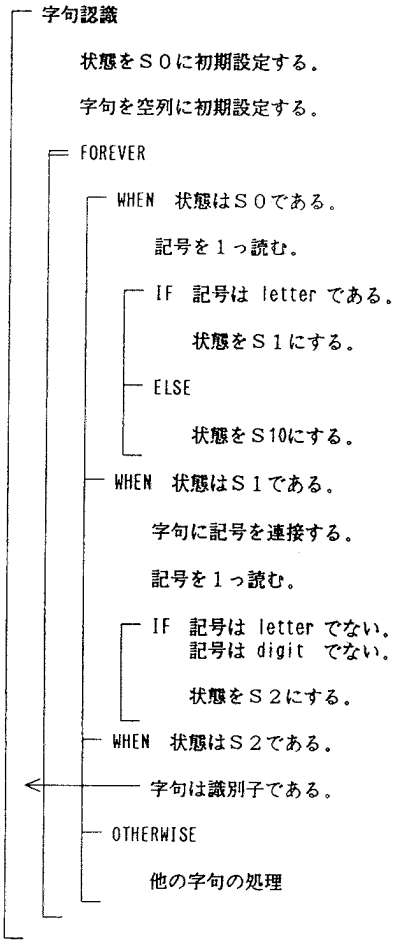


図12　字句認識の行動ダイアグラム

```
            PL/I  V10L20     LEX   OCEDURE OPTIONS(MAIN);

** SOURCE STATEMENT LISTING **

  STMT
     1    LEX:PROCEDURE OPTIONS(MAIN);
     2        DECLARE    (TOKEN,KINDS)  CHARACTER(10) VARYING,
                         TRUE           BIT(1) INITIAL('1'B),
                         SYSIN          FILE INPUT STREAM,
                         SYSPRINT       FILE OUTPUT STREAM;

     3        ON ENDFILE(SYSIN) GO TO OWARI;
     4        DO WHILE(TRUE);
     5          DO;
     6          CALL GET_TOKEN(TOKEN,KINDS);
     7          PUT DATA(TOKEN,KINDS);
     8          END;
     9        END;
    10        OWARI:;

    11    GET_TOKEN:PROCEDURE(TOKEN,KINDS);
    12        DECLARE    (TOKEN,KINDS)  CHARACTER(10) VARYING,
                         STATE          BINARY FIXED(4),
                         CH             CHARACTER(1);

    13        STATE=0;
    14        TOKEN='';
    15        DO WHILE(TRUE);
    16          SELECT(STATE);
    17          WHEN(0)
                  DO;
    18            GET EDIT(CH) (A(1));
    19            IF IS_LETTER(CH) THEN
                    STATE=1;
    20            ELSE
                    STATE=10;
    21            END;
    22          WHEN(1)
                  DO;
    23            TOKEN=TOKEN || CH;
    24            GET EDIT(CH) (A(1));
    25            IF ¬IS_LETTER(CH) & ¬IS_DIGIT(CH) THEN
                    STATE=2;
    26            END;
    27          WHEN(2)
                  DO;
    28            KINDS='IDENTIFIER';
    29            RETURN;
    30            END;
    31          OTHERWISE
                  /* OTHER TOKENS */

  STMT
                  GET EDIT(CH) (A(    ;
                  /* ............
    32          END;
    33        END;
    34    END GET_TOKEN;

    35    IS_LETTER:PROCEDURE(CH) RETURNS(BIT(1));
    36        DECLARE    CH          CHARACTER(1);

    37        SELECT(CH);
    38        WHEN('A','B','C','D','E','F','G','H','I',
                   'J','K','L','M','N','O','P','Q','R',
                   'S','T','U','V','W','X','Y','Z')
                RETURN('1'B);
    39        OTHERWISE
                RETURN('0'B);
    40        END;
    41    END IS_LETTER;

    42    IS_DIGIT:PROCEDURE(CH) RETURNS(BIT(1));
    43        DECLARE    CH          CHARACTER(1);

    44        SELECT(CH);
    45        WHEN('0','1','2','3','4','5','6','7','8','9')
                RETURN('1'B);
    46        OTHERWISE
                RETURN('0'B);
    47        END;
    48    END IS_DIGIT;

    49    END LEX;
```

図13　字句認識のプログラム

## VI　むすび

職業訓練短大の情報工学に関する実技は、応用的なソフトウェア製品を作ることが多い。しかし、これからはコンパイラーやオペレーティング・システムのような基礎的な製品を作ることが重要である。その足掛かりとして、決定性有限オートマトン生成系を開発した。これを教材として利用することで、情報工学分野の職業能力開発に役立つ。

なお、決定性有限オートマトン生成系を職業能力開発のために使用する場合は、ソース・プログラムを無償で提供します。

## 参考文献

(1) A. V. Aho, J. D. Ullman : Principles of compiler Design, Addison-Wesley, 1977 PP. 73-120

(2) A. V. Aho, R. Sethi, J. D. Ullman : Compilers Principles, Techniques, and Tools, Addison - Wesley, 1986 PP.83-146