

ダブル配列による有限状態機械の記憶アルゴリズム

1 S-7

入口 浩一 青江 順一
徳島大学

1. まえがき

有限状態機械(finite state machine)のふるまいは非常にシンプルであり、コンパイラの語い解析、文献検索、スペリング検査、テキスト編集などの単語のマッチングとか、順序回路、パーサの有限制御部など数多くのモデルとして応用されている⁽¹⁻⁵⁾。有限状態機械をインプリメントする場合、状態遷移に関する情報(goto関数と呼ぶ)を時間的かつ空間的に如何に効率的よく記憶検索するかが重要な課題となってくる。

青江ら^{(1),(2)}は、パターンマッチングマシンのgoto関数をダブル配列(double-array)により実現することを提案した。ダブル配列の遷移アクセス時間は、 $O(1)$ となるので、非常に高速である。しかし、青江らの議論した状態遷移表はパターンマッチングマシンのものに限定されていたので、一般の有限状態機械の遷移表には適応できなかった。従って、本稿の目的は、ダブル配列法の適用可能な範囲を一般の有限状態機械の状態遷移表に拡張することにある。

2. ダブル配列によるパターンマッチングマシン

キーワード(keyword)と呼ばれるストリングを要素とする有限集合を K とおく。

K に対する有限状態機械 M を次のように表す。

$$M = (S, I, g, s_n, F)$$

ここで、 S は状態の有限集合、 I は入力記号の有限集合、 s_n は初期状態、 g は状態遷移関数で goto 関数と呼び、 F は出力状態の有限集合である。関数 g は $S \times I$ から $S \cup \{fail\}$ への写像を定義し、状態 s から s' への遷移が入力記号 a に対して定義されていれば、 $g(s, a) = s'$ と書き、定義されていなければ、 $g(s, a) = fail$ と書く。状態番号は正の整数値で表されており、番号 1 は初期状態番号を表す。

また、fail をデータ構造での値と密接にするために番号 0 を使用し、 $g(s, a) = 0$ は $g(s, a) = fail$ を意味する。以後、状態 s に入る遷移の数とそこから出る遷移の数をそれぞれ $indegree(s)$, $outdegree(s)$ と表す。

マシン M をパターンマッチングマシンと仮定し、入力ストリング x を $a_1 a_2 \dots a_n$; $n \geq 1$ とするとき、ダブル配列法によるマッチングアルゴリズム 1 を図 1 に示す。

Algorithm 1. Matching Algorithm.
Input: A string $x = a_1 a_2 \dots a_n$ and a machine M
with the double-array for K .
Output: If $x \in K$, then the output is TRUE, otherwise FALSE.
Method:
begin
(1-1) $s := 1$; $j := 0$;
(1-2) **repeat**
 $j := j + 1$;
 $t := a_j + BASE[s]$;
 if $CHECK[t] \neq s$ **then** return(FALSE) **else** $s := t$;
(1-3) **until** $j = n$;
(1-4) **if** ($s \in F$) **then** return(TRUE)
 else return(FALSE)
end.

図1 ダブル配列によるマシン M マッチングアルゴリズム。
Fig. 1 A Matching algorithm of the machine M by the double-array.

ダブル配列法では、goto 関数を二つの 1 次元配列 BASE, CHECK で表現する。この方法では、まず状態 s と入力記号 a_j に対する $g(s, a_j)$ を計算するために $t = a_j + BASE[s]$ なる値 t をまず決定する。ここで、記号 a_j は内部表現値(numerical value)、すなわち 1 以上の整数値で取り扱われているものとする。次に、 $CHECK[t]$ が状態番号 s と一致すれば $g(s, a_j)$ は t で与えられ、 $CHECK[t]$ と s が一致しなければ $g(s, a_j) = 0$ となる。そして、repeat 文を出たとき、状態 s が出力状態 F に含まれるならば、入力ストリング x は K に含まれ、そうでなければ x は K に含まれない。

ダブル配列法では、 $g(s, a) = s'$ なる遷移が、BASE と CHECK により、次の様に表現されることになる。

$$BASE[s] + a = s', \quad CHECK[BASE[s] + a] = s.$$

3. 有限状態機械への拡張

3.1 一般の状態遷移表での問題点

有限状態機械の状態遷移は、 $indegree(s) \geq 2$ なる状態 s を持つ。この状態 s に対して、次の問題点が生じる。

$g(s', a) = s$, $g(s'', b) = s$, $a \neq b$ なる状態 s を仮定するとき、次の二つの t', t'' により新しい状態番号 $new(s)$ を定義してしまい、状態 s に対する新しい状態番号が一意的に決定できなくなる。

$$t' = BASE[new(s')] + a, \quad t'' = BASE[new(s'')] + b$$

ここで、 $new(s)$ が定義する新しい状態番号の集合を $NEW[s]$ で表す。そして、 $NEW[s]$ がシングルトンでないならば、状態 s を重複状態と呼び、その要素の状態番号を重複状態 s に対する複合状態番号と呼ぶ。

本稿では次の条件を満足するダブル配列を提案する。

(条件 A) 状態 s が重複状態ならばそのときに限って次が成立する。

$NEW[s] = \{t_1, t_2, \dots, t_n\}$ で統一状態番号を t_1 とおくと、 $-t_1 = BASE[t_2] = \dots = BASE[t_n]$ となる。

条件 A より、統一状態 t_1 にまとめられるべき t_i , $2 \leq i \leq n$ なる複合状態番号に対して $BASE[t_i] < 0$ が成立する。従って、統一状態番号は $t_1 = -BASE[t_i]$ として決定できるので、余分な配列は不要となる。但し、ここでは便宜上統一状態を t_1 とおいたが、実際は t_i ($1 \leq i \leq n$) のどれでもよい。

(条件 B) 状態 s が出力状態のとき、入力記号に含まれない特別な記号 $\#$ に対して次が成立する。

$$CHECK[BASE[s] + \#] = -s$$

条件 B より出力状態の判定はダブル配列の情報だけで行えるので、出力状態のための余分な表が不要となる。この方法により、 $outdegree(s) \neq 0$ でない出力状態 s に対しても $BASE[s]$ は未使用となるので、この場所に出力情報或はそのポイントが格納できるようになる。

また、アルゴリズム 1 では、 $t = a_j + BASE[s]$ がダブル配列のインデックスの範囲を越える場合の判定がなされていないので、次の条件を定義する。

(条件 C) 初期状態番号 1 に対する $CHECK[1]$ の値はダブル配列の最大インデックスと一致する。

(例1) 図2に $L(M) = \{a(bd)^*c, c, cc, ca\}$ なる言語を受理する有限状態機械の状態遷移図の例を示す。但し、出力状態は 4, 5 であり、() 内の状態番号は次節以後に説明される新しい状態番号である。

図2の状態遷移図において、状態2と4が重複状態となる。

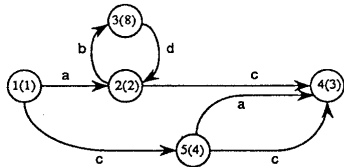


図2 有限状態機械の状態遷移図の例

Fig.4 An example of a transition graph for a finite state machine

3.2 拡張されたダブル配列の構成法

拡張されたダブル配列の構成アルゴリズム2を図3に示す。但し、条件Bを実現するために出力状態 s に対する $list[s]$ の要素には記号 # を入れておくものとする。

```

Algorithm 2. Construction of the extended double-array.
Input: Goto function g and a set list[s] for each s in S.
Output: Function new, and a double-array holding the conditions A and B.
Method:
begin
  s := 1; new[s] := 1; push(s);
  repeat
    s := pop();
    (2-1) BASE[new[s]] := X_CHECK(1, list[s]);
    (2-2) for each b in list[s] do
      begin
        (2-3) if b ≠ # then
          begin
            (2-4) CHECK[BASE[new[s]] + b] := new[s];
            (2-5) s' := g(s, b);
            (2-6) if new[s'] = 0 then
              begin
                (2-7) new[s'] := BASE[new[s]] + b;
                (2-8) push(s');
              end
            end
          end
        else BASE[new[s] + b] := -new[s];
      end
    end
    (2-10) else CHECK[BASE[new[s] + #] := -new[s]
  until the stack becomes empty;
  (2-11) CHECK[1] := MAX_SIZE();
end.

function X_CHECK(base, list)
begin
  overlap: for each a in list do
    if CHECK[base + a] ≠ 0 then
      begin
        base := base + 1;
        goto overlap;
      end
    end
  return(base);
end;

```

図3 拡張されたダブル配列の構成アルゴリズム。
Fig. 3 An algorithm of constructing the extended double-array.

アルゴリズム2では、行(2-9)で複合状態から統一状態への写象を決定するので(BASE[s]を負の値にする)。行(2-1)で設定するBASEの要素は負になってはならない。従って、正の値を必ず返す関数 X_CHECK を使用する。行(2-4, 2-5)は変更はないが、行(2-3, 2-10)では出力状態に対する手順が追加されている。また、行(2-6)は複合状態を統一状態に写象するタイミングを決定する。アルゴリズム2では、行(2-6)の $new[s']$ が未定義の場合にのみ関数 new が決定されるので、重複状態への遷移の中で最初の処理対象となる遷移(一本目の遷移と呼ぶ)によって定義される新しい状態番号が統一状態番号となる。そして、重複状態への二本目以後の遷移で定義される新しい状態番号は、行(2-9)で条件Aに従った統一状態への写象が実行される。即ち、行(2-6)で $new[s'] \neq 0$ が成立することは、状態 s' が重複状態であるための必要十分条件である。更に、行(2-11)の $MAX_SIZE()$ はダブル配列の最大のインデックスを返す関数であり、ここで $CHECK[1]$ に値が設定される。

(例2) 図2の状態遷移図に対するアルゴリズム2の処理結果を表1に示す。但し、# の内部コードは 5 とする。

表1 図2の状態遷移図に対するダブル配列

	1	2	3	4	5	6	7	8	9	10
BASE	1	6	1	2	-3	0	0	6	-3	-2
CHECK	10	1	4	1	4	-3	-4	2	2	8

重複状態 2 と 4 の新しい状態番号は、それぞれ2, 3 となり、これらが統一状態番号となる。そして、統一状態 2 に写象される複合状態は 10; 統一状態 3 に写象される複合状態番号は 5 と 9 になる。

4. 評価

4.1 理論的評価

入力記号の総数を e とおくと、 $BASE[s]$ は関数 X_CHECK により 1 から $n + e$ 迄の値をとるので、状態番号 i に対する $BASE$ の計算は、 $O((n + e) \cdot outdegree(i))$ 時間となる。従って、ダブル配列の構成時間は

$$O((n + e) \cdot \sum_{i=1}^n outdegree(i)) = O((n + e)n) = O(n^2 + en) \text{ となる。}$$

以上は、理論的な評価が与えられるが、ダブル配列の記憶量が $O(n)$ となることの保証は得られない。

4.2 具体的評価

本手法のダブル配列の構成システムは、言語Cで記述されており、6種類の状態数に対して、乱数により状態遷移図を構成して実験を行った結果を表2に示す。表の値は、総遷移数に対するダブル配列の未使用要素数の割合(単位は%)を表し、圧縮率と呼ばれる。但し、状態遷移表はスパースな表としてよく知られているので⁽³⁾各状態から出る平均遷移数(av とする)3, 4, 5 の場合を想定し、入力記号の総数は128 とした。

総状態数	100	200	300	400	500	1,000
av = 3	16.8	19.1	15.4	19.9	19.4	22.7
av = 4	16.4	17.7	14.5	15.5	17.6	15.3
av = 5	18.6	22.0	16.1	18.2	16.4	14.7

表2より、圧縮率は25%以下となりよい結果を得た。即ち、記憶量は圧縮率を25%とすれば、総遷移数 e に対して $2.5e$ バイトで表される。

ダブル配列の構成時間は総状態数1,000で、 av が5の場合で約4秒となった。但し、この時間はリスト形式で表現された goto 関数を入力ファイルから読み込み、ダブル配列を構成した後、出力ファイルに結果を書き込む迄の時間である。

5. むすび

以上本稿では、パターンマッチングマシンの静的な状態遷移表に対する表現法として提案されていたダブル配列法を一般の有限状態機械に対する遷移表にも適用できるように拡張した。本手法は、有限状態機械に関係しない分野、例えば有向グラフ⁽⁴⁾を表現するデータ構造或はスパース行列の圧縮⁽³⁾にも応用できる。

文 献

- (1) 青江外: "パターンマッチングマシンの効率的記憶検索法", 情報論, 24, 4, pp. 414-420, (S58-07).
- (2) Aho, J. et al.: "An Efficient Implementation of Static String Pattern Matching Machines, Conf. on Supercomputing Sys. pp. 491-498 (1985).
- (3) 青江外: "行置換によるスパース行列の効率的縮小アルゴリズム", 情報論, 26, 2, pp. 211-218, (S60-03).
- (4) Aho, A. V., et al.: The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading Mass. Ch.2 (1974).
- (5) Aho, A. V. et al.: "Efficient string matching: An aid to bibliographic search", Comm. ACM., 18, 6, pp.333-340 (1975).