

マルチバイトキャラクタを扱う決定性有限状態オートマトンの構成法*

1M-4

長谷川 勇
日本アイ・ビー・エム株式会社 ソフトウェア開発研究所

1. はじめに

正規表現¹⁾とは文字列中のパターンを発見/操作するためのパターンマッチングの機構であり、sed, grep, awkを始めとするテキスト処理ツールに広く利用されている。例えば、正規表現“(ab)*”は“ab”の0回以上の繰り返し(“”, “ab”, “abab”, ...)にマッチし、“[acd]”は“a”または“c”または“d”にマッチする。

DFAは正規表現によるパターンマッチングの評価器であり、処理速度に優れており、GNU grep, awkなどのツールが採用している。DFAは、正規表現からNFAを構成し、そこから変換することで構成する。この変換の手順²⁾は理論上容易であるが、日本語などマルチバイトキャラクタを扱う実装をするには、困難な点がある。

本論文では、NFAからDFAへの変換について説明し、それをマルチバイトキャラクタに応用する上での問題、我々の提案する解決法を述べ、その成果を示す。

2. 有限状態オートマトン

有限状態オートマトンは開始ノードと停止ノードを持つラベル付きの有向グラフである。ここでは、NFAを5つ組 (K, T, t, k_1, F) で定義する。ただし、 K : 状態の有限集合、 T : 入力文字の有限集合、 t : 現在の状態、入力、遷移先の状態の関係 $(K, 2^T, 2^K)$ の有限集合、 k_1 : $k_1 \in K$ 開始状態、 $F: F \subset K$ 停止状態の集合である。

例えば、正規表現“(ab)c*ac”からは以下のNFAを構成できる。この手順は容易なので本論文では省略する。

$(\{0, 1, 2, 3\}, \{a, b, c\}, \{0, \{a, b\}, \{1\}\}, (1, \{c\}, \{0\}), (0, \{a\}, \{2\}), (2, \{c\}, \{3\}), k_1, \{3\})$

これをグラフで表したのが図1である。

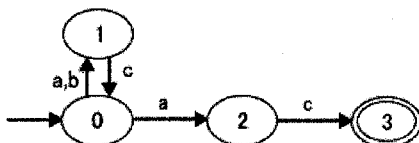


図1: NFA (ab)c*ac

なお、ある状態、入力に対し、複数の遷移先を持つのでNFAは“非決定性”である¹⁾。

次にNFAからDFA $(K', T', t', \{k_1\}, F')$ に変換する手順を示す。

1. $K' = \{\{k_1\}\}, t' = \phi, M = \{k_1\}$
2. $\forall a \in T$ に対して、
 - a. $L = \{l(k, N, O) \in t, k \in M, a \in N, l \in O\}$
 - b. $K' \cup \{L\}$ を新たな K とする
 - c. $t' \cup \{(M, a, L)\}$ を新たな t' とする
3. 2で増えたすべての K' の要素を M として2,3を適用
4. $F' = \{l \in K', \exists k(k \in l, k \in F)\}$

ここで定義するDFAの t' は (K, T, K) であり、現在の状態と入力文字から一意に遷移先の状態が決定するため、このオートマトンは“決定性”である。

図1のNFAを変換すると次のDFAが得られる(図2)。
 $(\{\{0\}, \{1\}, \{1, 2\}, \{0, 3\}\}, \{a, b, c\}, \{\{0\}, a, \{1, 2\}\}, \{\{0\}, b, \{1\}\}, \{\{1\}, c, \{0\}\}, \{\{1, 2\}, c, \{0, 3\}\}, \{\{0, 3\}, a, \{1, 2\}\}, \{\{0, 3\}, b, \{1\}\}\}, \{0\}, \{\{0, 3\}\})$

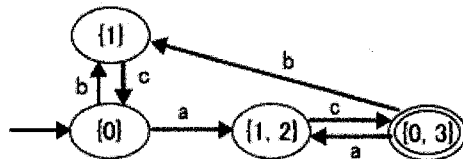


図2: DFA (ab)c*ac

3. マルチバイトキャラクタへの応用

3.1. 問題点

応用する上で次の2点が問題となる。

- a. 正規表現のマルチバイトを認識
- b. 入力文字列のマルチバイトを認識

aは正規表現“a*”が正しく“a”の0回以上の繰り返しと認識されることである。対応していない場合“a”の1byte目と“a”の2byte目の0回以上の繰り返しと誤って認識されてしまう。この問題は解決が比較的容易なので本論文では省略する。bは正規表現“(任意の1文字にマッチする正規表現)”が、“a”1文字にマッチし、“a”の1byte目だけにマッチしないことである。これを実現するには、キャラクタセットの大きさと可変長のキャラクタの2つが問題となる。本節ではこれらの問題について述べる。

3.1.1. キャラクタセットの大きさ

上記の手順2において、 $\forall a \in T$ に対する処理を行っているが、これは T がASCIIコードなど十分小さい集合であることを前提としている。例えば T がUCS-4などの大きな集合の場合、UCS-4の40億個のコードポイントを調べるのは実用的な時間では不可能である。

これを回避する方法として、文字の割り当てられていない部分を飛ばしてチェックを行う方法が考えられる。しかし、Single Unix Specification³⁾などの規格において、あるコードポイントに文字が割り当てられているかどうかを調べるAPIが用意されておらず、ポータビリティを損なわずに実装するのは不可能である。さらに、現在割り当てられていないコードポイントに将来文字が割り当てられる可能性もあるため、この方法は好ましくない。

3.1.2. 可変長のキャラクタ

マルチバイトキャラクタでは、各キャラクタは可変長である。マルチバイトキャラクタ自体はbyte列と考えれば8bit cleanな実装であれば問題は起きない。しかし、例

¹⁾ この例では、状態0、入力aに対して、遷移先が1, 2の2通りある。

*A method for constructing DFA considering with multibyte character set
Isamu Hasegawa, Software Development Lab., IBM Japan, Ltd.

例えば正規表現“(a|.)”を“(a(1)a(2))”と考えたとする(“a”の1byte目をa(1)と表記)。NFAの場合はこれで問題ないが、DFAの場合、“.”は1文字つまり2もしくは3byteとマッチするため、a(1)とは消費する入力長の長さが異なり、DFAの状態に変換できない。

一方、マルチバイトキャラクタをすべて固定長のワイドキャラクタに変換してから処理する方法も、国際化では一般的である。しかしワイドキャラクタへの変換によるオーバーヘッド、二重に必要となるバッファ、入力をワイドキャラクタに変換できなかった場合の誤動作など、デメリットも多く、今回のアプローチではマルチバイトキャラクタをそのまま扱うこととした。

3.2. 解決法

3.2.1. “キャラクタセットの大きさ”の解決法

まず、キャラクタが固定長であり、単にキャラクタの種類が多い場合を考える。我々が提案する方法では、上記の手順2を次の2'で置き換える。

2'. $\exists l((k, C, l) \in t, k \in M, a \in C)$ であるマルチバイトキャラクタaが

i. 存在しない場合 → 上記の手順2を適用

ii. 存在する場合

a. $L = \{l((k, x, l) \in t, k \in M, x \in T)\}$ を求める。

b. 2^L から空集合を除いた集合を L' として、 $K' \cup L'$ を新たな K' とする。

ここで、手順2'において、 $L \subset K$ であり、 $2^L \subset 2^K$ なので、 $K' \subset 2^K$ 。従って、この手順は必ず停止する。

さらに、 $\exists l((k, C, l) \in t, k \in M, a \in C)$ であるマルチバイトキャラクタaが存在する状態kにおいて、集合 $N = \{(k, M_i, L_i) | (k, M_i, L_i) \in t\}$ を求め、入力cに対する遷移先の状態Mを次式で求め、状態遷移を行う。 $M = \{l | c \in M_i, l \in L_i, (k, M_i, L_i) \in N\}$

例えば、正規表現“(a|[あ-お]b)”から構成したNFAと、それをこの手順で変換したDFAを図3に示す。ここで、入力“aa”に対する状態遷移は、 $\{0\} \rightarrow \{1, 2\} \rightarrow \{3\}$ であり、入力“アa”に対する状態遷移は、 $\{0\} \rightarrow \{1\} \rightarrow \{3\}$ である。

ここで述べた手順において、2'.iiにかかる時間はtの大きさだけに依存し、キャラクタセットの大きさとは無関係である。従って、大きなキャラクタセットに対しても、実用的な時間でDFAを構成できる。

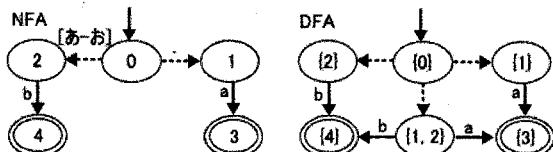


図3: 正規表現“(a|[あ-お]b)”

3.2.2. “可変長のキャラクタ”の解決法

次に一般的な場合、つまりキャラクタが可変長の場合を考える。我々の方法では、DFAの開始状態のみ用意しておき、状態遷移の過程で動的に状態を生成する。

1'. M を現在のDFAの状態(初期値は $\{k_1\}$)とする。

2'. $\exists l((k, C, l) \in t, k \in M, a \in C)$ であるマルチバイトキャラクタaが

i. 存在しない場合

tと遷移先の状態がまだ構築されていない場合は上述の手順2を適用し、通常の状態遷移を行い、新たなMとする。

ii. 存在する場合

a. 3.2.1で述べた遷移先の状態によりMを求め、 M' とする。

b. 入力 $c = c_1, \dots, c_i, \dots, c_n, M_0 = M,$

$M_i = \{m | (k, c_i, m) \in t, k \in M_{i-1}\}$ として、

$M' \cup M_n$ を次の状態Mとして、 K' に加える。

3". 入力の終りであり、 $\exists k(k \in M, k \in F)$ ならば終了。終了でない場合は2'に戻る。

例えば、正規表現“(a|あb)”から構成したNFAを図4に示す。これに、入力“aa”を与えると、 $M = \{0\}$ より、“a”に対して $M' = \{3\}, M_n = \{2\}$ であり $\{2, 3\}$ に遷移し、“a”に対して $\{5\}$ に遷移し終了する。また入力“いa”を与えると、 $M = \{0\}$ より、“い”に対して $M' = \{3\}, M_n = \emptyset$ であり $\{3\}$ に遷移し、“a”に対して $\{5\}$ に遷移し終了する。この様子を図5に示す。

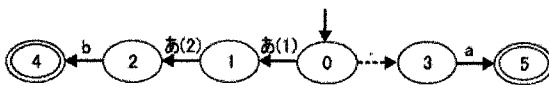


図4: NFA (a|あb)

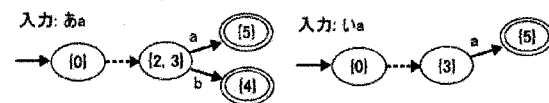


図5: DFA (a|あb)

以上の方法により、キャラクタが可変長である場合に対してもDFAを構成できる。

4. 成果

本論文で提案したDFAの構成法により、GNU grep, GNU awk をエンコーディングの種類に依存せずマルチバイトキャラクタを扱えるよう拡張することができた。²さらにgrep-2.5のβ版にはこのコードが含まれている。

5. 謝辞

DFAのマルチバイトキャラクタ対応にあたり、助言をいただいた方々、特に杉山 昌治氏、鷲澤 正英氏、Alain Magloire氏に深く感謝いたします。

参考文献

- [1] Jeffrey E.F.Friedl著, 歌代 和正監訳: 詳説正規表現, オライリー・ジャパン(1999)
- [2] V.J.Rayward-Smith著, 井上謙蔵監訳: 言語理論入門, 共立出版(1986)
- [3] The Single UNIX Specification, Version 2 (<http://www.opengroup.org/onlinepubs/007908799/>)

² パッチは<http://www.lil8nux.org/subgroups/utilddev/dli18npatch2.html>で公開している。