決定性有限オートマトンによる正規表現の貪欲な照合

DFA-based Implementation of Greedy Regular Expression Matching *

奥居哲　増田拓也　藤田佳宏　鈴木大郎 +

中部大学工学部情報工学科　会津大学コンピュータ理工学部 +

Satoshi Okui　Takuya Masuda　Yoshihiro Fujita　Taro Suzuki

Department of Computer Science, Chubu University

School of Computer Science and Engineering, The University of Aizu+

概要

正規表現の貪欲な文字列照合を可能にする決定性オートマトン（DFA）を構築する手法について述べる．構築される DFA は，正規表現から構築される古典的な DFA と異なり，NFA 状態の列を DFA 状態とみなし，もとの正規表現の部分式の情報を遷移毎に保持している．このため，貪欲な部分照合を行い各部分式毎の照合を得ることが可能になる．非決定性オートマトン（NFA）に基づく従来手法では照合の各ステップの計算コストが NFA のサイズに依存して増加するのにたいし，提案手法では定数になる．提案手法の（実行時の）最悪の場合の計算コストは $O(n)$（n は照合される文字列の長さ）であり，バックトラックを用いる手法や NFA を直接用いる手法と比較して改善されている．

1　はじめに

正規表現の貪欲（greedy）な照合を実現する最も一般的な手段は，バックトラックを用いる方法である．これに対して，正規表現から構築した**非決定性オートマトン**（NFA）に対して**部分集合構成**を実行時に（すなわち照合すべき文字列が与えられる毎に）行うことでバックトラックを排除する手法が近年いくつか提案されている [5, 2, 3, 7]．これらの手法にはバックトラックを行う方法と比較して最悪の場合の計算コストが遥かに小さい（与えられた文字列長さに関して線形のオーダ）という利点がある一方，実行時に部分集合構成を行うコストがかさむ．

そこで，NFA から正規表現の部分式の情報を付加した**決定性オートマトン** (DFA) を予め（すなわち，照合すべき文字列が与えられる以前に）構築することで，照合の各ステップにおける計算コストを定数にする手法を提案する．

以下，形式言語とオートマトンの基本的な概念については既知であるとする．文字の集合（アルファベット）を Σ で表し，正規表現（の抽象構文木）の記法としては和（R_1+R_2），積（$R_1 \cdot R_2$），空集合（0），空列（1），文字 a ($\in \Sigma$)，クリーネ閉包（$R*$）を用いる．正規表現 R が表現する言語を

* 本研究は科研費基盤研究 (C)No.22500019 の助成を受けて行われている．

L(R) で表す．有限集合 S の要素数は $|S|$ で表す．S の要素の有限列（$[s_1, s_2, ..., s_n]$ と表記）の集まりを S^* と書く．列 s の最初と最後の要素をそれぞれ first(s), last(s) で表し，列 s, t をこの順に連結したものを s ++ t と表す．列の集合 s, t に対して $s \cdot t = \{x$ ++ $y \mid x \in s, y \in t\}$ と定義する．木の各ノードには位置が割り当てられる．正規表現の構文木は高々二分木であるので, 位置の表現として集合 $P = \{1, 2\}*$ の要素（列の表記を簡略化して例えば $[1, 2, 1]$ を 121 のように表す）を用い，あるノードの位置を p とするとき，その直下のノードの位置を左から $p1, p2$ で表す.[*1] ルート位置が ε で開始する木 t の位置 p におけるノードのラベルを（もしあれば）$t(p)$ で表す．ルート位置が ε で開始する正規表現の構文木 R において文字でラベル付けされた葉ノード（以下，文字ノード）の位置の集合を特に $\alpha(R)$ で表す.

2　正規表現の貪欲照合

文字列 s と正規表現 R の組を**照合問題**と呼ぶ．$s = uwv, w \in \mathrm{L}(R)$ となる文字列 u, v, w があるとき，照合された部分文字列の範囲を表す非負整数の組 $(|u|, |u| + |v|)$ をこの照合問題の**解**と呼ぶ．解は，もし存在すれば一般に複数あり，この意味で照合には曖昧さがある．

$w \in \mathrm{L}(R)$ のとき R の各部分式が w のどの部分と照合されるかまで考慮するなら，照合にはさらに曖昧さが生じる（[3] や [7] では，w の構文解析木のようなものを考えることで，この曖昧さを形式化している）．この曖昧さを解消する一般的な方法のひとつが**貪欲**（greedy）照合である．貪欲照合の意味を非形式的に述べると以下のようになる．正規表現の各部分式に非終端記号を割り当て，正規表現を文法規則で表現したとする．このとき，クリーネ閉包 $R*$ は $S \to RS \mid 1$ のように再帰的に表現する．この文法規則にたいして，左優先でバックトラックを許す再帰下降型の解析（深さ優先探索）を行うことで貪欲な照合が実現される．ただし，この方法ではいわゆる**左再帰**の問題が生じる場合がある．たとえば，正規表現 a**b を文法になおすと

$$S \to Ab$$
$$A \to BA \mid 1$$
$$B \to \mathrm{a}\, B \mid 1$$

のようになるが，これと文字列 b を照合しようとすると

$$S \Rightarrow Ab \Rightarrow BAb \Rightarrow 1Ab(= Ab) \Rightarrow \ldots$$

のような無限の導出が生じ，照合が停止しない．このような問題を避けるため，貪欲照合では $R*$ の部分式 R が空列とマッチするような導出を禁止している．

3　拡張ポジションオートマトンの構築

貪欲照合は，その意味論からして，バックトラックによる深さ優先探索で実装するのが自然であるが，最悪の場合の計算コストは指数的に爆発する恐れがある．そこで正規表現から Thompson 構

[*1] 通常，木のルート位置は空列（ε）で始めるが，本論文では任意の位置から始まるような位置付けを許す.

成によって得られる NFA を用い，照合すべき文字列が与えられる毎に部分集合構成を行い，バックトラックなしで照合を行う手法がいくつか提案されている．一方，[7] では部分式の情報を付加したポジションオートマトン（Position Automata with Augumented Transitions; 以下 **PAT**）を用いる最左最長照合の手法を提案している．以下に述べる PAT の構築法は，古典的なポジション NFA と同じく正規表現の抽象構文木を巡回するが，古典的な構築法 [6, 4] とは異なり，巡回途中のノードの情報を保存するために位置の集合 P に符号を付与して得られる集合 $\mathrm{T} = \{+p, -p \mid p \in P\}$ の要素を**タグ**と呼んで用いる．直観的には + の付いたタグ（$+p$）は位置 p のノードを下降しながら巡回したことを意味し，− の付いたタグ $(-p)$ は上昇しながら巡回したことを意味する．

正規表現の抽象構文木 R とそのルート位置 p が与えられたとき，$\epsilon(R, p)$ $(\subseteq \mathrm{T}^*)$ は文字ノードを一切通らずに R を巡回する可能な経路の集合を表す．形式的には以下のように定義される．

$$
\begin{aligned}
\epsilon(0, p) &= \{\} \\
\epsilon(1, p) &= \left\{[+p, -p]\right\} \\
\epsilon(a, p) &= \{\} \quad (\text{for } a \in \Sigma) \\
\epsilon(R_1{+}R_2, p) &= \left\{[+p]\right\}\cdot(\epsilon(R_1, p1) \cup \epsilon(R_2, p2)) \cdot \left\{[-p]\right\} \\
\epsilon(R_1{\cdot}R_2, p) &= \left\{[+p]\right\}\cdot\epsilon(R_1, p1) \cdot \epsilon(R_2, p2) \cdot \left\{[-p]\right\} \\
\epsilon(R{*}, p) &= \left\{[+p, -p]\right\}
\end{aligned}
$$

0 は失敗を表すので，可能な経路はない．R が文字の場合も同様である．空列でラベル付けされたノード（以下，空ノード）を通過する経路は $[+p, -p]$ だけである．和については，各部分木に対する可能な経路を併せたものになる．積については，各部分木を巡回する可能な経路のすべての組み合わせについて経路を連結したものになる．貪欲照合ではクリーネ閉包 $R*$ の部分式 R を空文字列と照合してはならないので，可能な経路は 1 と同様に $[+p, -p]$ のみになることに注意が必要である．

$\epsilon(R, \varepsilon)$ を $\epsilon(R)$ と略記する．$\epsilon(R)$ が非空のとき R は**通過可能**（nullable）であると呼ばれる．

正規表現の抽象構文木 R とそのルート位置 p，さらに位置の集合 D $(\subseteq P)$ が与えられたとき，$\mathrm{First}(R, p, D)$ $(\subseteq \mathrm{T}^*)$ は R のルートから始め，D に属する位置の文字ノードに（途中 R の文字ノードを通過せずに）至るまでの可能な経路の集合を表す．形式的には以下のように定義される．

$$
\begin{aligned}
\mathrm{First}(0, p, D) &= \{\} \\
\mathrm{First}(1, p, D) &= \{\} \\
\mathrm{First}(a, p, D) &= \begin{cases} \left\{[+p]\right\} & (\text{for } a \in \Sigma \text{ if } p \in D) \\ \{\} & (\text{for } a \in \Sigma \text{ if } p \notin D) \end{cases} \\
\mathrm{First}(R_1{+}R_2, p, D) &= \left\{[+p]\right\}\cdot\mathrm{First}(R_1, p1, D) \cup \left\{[+p]\right\}\cdot\mathrm{First}(R_2, p2, D) \\
\mathrm{First}(R_1{\cdot}R_2, p, D) &= \left\{[+p]\right\}\cdot\mathrm{First}(R_1, p1, D) \cup \left\{[+p]\right\}\cdot\epsilon(R_1, p1) \cdot \mathrm{First}(R_2, p2, D) \\
\mathrm{First}(R{*}, r, D) &= \left\{[+p]\right\}\cdot\mathrm{First}(R, p1, D)
\end{aligned}
$$

0, 1 にたいしては，そのような経路はない．D に属する位置 r の葉に到達した際には，その開始タグ $+r$ が経路に含まれる．和については，各部分木にたいする経路を併せたものになる．積につい

ては，R_1 が通過可能ならば R_2 にたいする経路も加える必要がある．

First$(R, \varepsilon, \alpha(R))$ を First(R) と略記する．

正規表現の抽象構文木 R とそのルート位置 p，さらに位置の集合 S $(\subseteq P)$ が与えられたとき，Last(R, p, S) $(\subseteq \mathrm{T}^*)$ は S に属する位置の文字ノードから開始して R のルートに（途中 R の文字ノードを通過せずに）至る可能な経路の集合を表す．Last は First の巡回の方向が逆になったものであり，形式的定義は省略する．Last$(R, \varepsilon, \alpha(R))$ を Last(R) と略記する．

正規表現の抽象構文木 R とそのルート位置 p，さらに位置の集合 S, D $(\subseteq P)$ が与えられたとき，Follow(R, r, S, D) $(\subseteq \mathrm{T}^*)$ は S に属する位置の文字ノードから開始して D に属する位置の文字ノードに到達する可能な経路の集合を表す．形式的には以下のようになる．

$$
\begin{aligned}
\mathrm{Follow}(0, p, S, D) &= \{\} \\
\mathrm{Follow}(1, p, S, D) &= \{\} \\
\mathrm{Follow}(a, p, S, D) &= \{\} \qquad (\text{for } a \in \Sigma) \\
\mathrm{Follow}(R_1{+}R_2, p, S, D) &= \mathrm{Follow}(R_1, p1, S, D) \cup \mathrm{Follow}(R_2, p2, S, D) \\
\mathrm{Follow}(R_1{\cdot}R_2, p, S, D) &= \mathrm{Follow}(R_1, p1, S, D) \\
&\quad \cup \mathrm{Last}(R_1, p1, S) \cdot \mathrm{First}(R_2, p2, D) \\
&\quad \cup \mathrm{Follow}(R_2, p2, S, D) \\
\mathrm{Follow}(R{*}, p, S, D) &= \mathrm{Follow}(R, p1, S, D) \mathbin{+\!\!+} \mathrm{Last}(R, p1, S) \cdot \mathrm{First}(R, p1, D)
\end{aligned}
$$

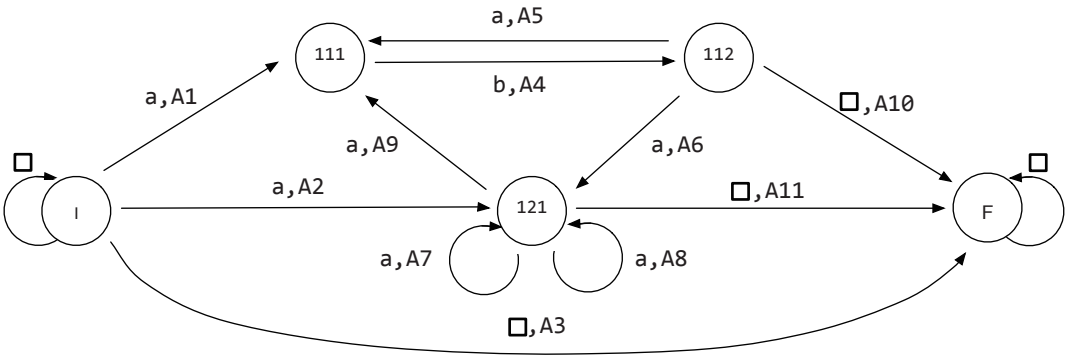
0，1，$a \in \Sigma$ にたいしては，そのような経路は無い．和にたいしては，各部分木毎に経路を求め，それらを併せればよい．積にたいしては，各部分木毎の経路に加え，R_1 から R_2 にまたがるような $\mathrm{Last}(R_1, r.1, S) \cdot \mathrm{First}(R_2, r.2, D)$ も考慮する必要があるクリーネ閉包 $R*$ については，その部分式 R の抽象構文木を再び巡回するような経路 $\mathrm{Last}(R, r.1, S) \cdot \mathrm{First}(R, r.1, D)$ が加わる．

Follow$(R, \varepsilon, \alpha(R), \alpha(R))$ を Follow(R) と略記する．

最後に，**状態**の集合 P を $\{\mathrm{I}, \mathrm{F}\} \cup \alpha(R)$ と定義 (I, F はそれぞれ**始状態**，**終状態**）し，**遷移関係** $\delta \subseteq P \times \Sigma \times \mathrm{T}^* \times P$ を

$$
\begin{aligned}
\delta = \ & \left\{(\mathrm{I}, a, t, p) \mid t \in \mathrm{First}(R),\ +p = \mathrm{last}(t),\ a = R(p)\right\} \\
& \cup \left\{(p, a, t, q) \mid t \in \mathrm{Follow}(R),\ -p = \mathrm{first}(t),\ +q = \mathrm{last}(t),\ a = R(q)\right\} \\
& \cup \left\{(p, a, t, \mathrm{F}) \mid t \in \mathrm{Last}(R),\ -p = \mathrm{first}(t),\ a \in \Sigma\right\} \\
& \cup \left\{(\mathrm{I}, a, t, \mathrm{F}) \mid t \in \epsilon(R),\ a \in \Sigma\right\} \\
& \cup \left\{(\mathrm{I}, a, \varepsilon, \mathrm{I}), (\mathrm{F}, a, \varepsilon, \mathrm{F}) \mid a \in \Sigma\right\}
\end{aligned}
$$

と定義した上で，6 つ組 $(\Sigma, P, \mathrm{T}, \delta, \mathrm{I}, \mathrm{F})$ を拡張ポジションオートマトン（PAT）と定義する．δ の要素 (p, a, t, q) を**遷移**と呼び，$p \xrightarrow{a,t} q$ と書く．直観的には状態 p から a を読んで q に遷移し，そのとき通過した経路が t であることを意味する．p を**遷移元**状態，q を**遷移先**状態と呼ぶ．遷移を 4 つ組として定式化したが，p, q の情報はすべて t から決まるので実は冗長である．また，PAT はポジションオートマトンであるので文字 a の情報も冗長である（遷移先の状態から一意に決まる）．ここでは従来の NFA の定義により合わせるためにこのように定式化している（t の情報を捨てれば，通常の NFA の遷移になる）．



A1: [+ε,+1,+11,+111]
A2: [+ε,+1,+12,+121]
A3: [+ε,-ε]
A4: [-111,+112]
A5: [-112,-11,-1,+1,+11,+111]
A6: [-112,-11,-1,+1,+12,+121]
A7: [-121,+121]
A8: [-121,-12,-1,+1,+12,+121]
A9: [-121,-12,-1,+1,+11,+111]
A10:[-112,-11,-1,-ε]
A11:[-121,-12,-1,-ε]

図 1　正規表現(ab+a∗)∗から生成される PAT

図 1 に，正規表現 (ab+a∗)∗ から生成される PAT を示す．□ で記した遷移は任意の文字 $a \in \Sigma$ による遷移を表す．A1～A11 はタグ列を表す．

4　貪欲照合を実現する決定性有限オートマトンの構築

一般に NFA に部分集合構成（subset construction）を適用することで DFA を得ることができる．生成される DFA の遷移はもとの NFA における複数の遷移を並列して行うことに相当し，DFA の状態は到達可能な NFA の状態の集合になる．つまり，DFA の経路はもとの NFA における潜在的に可能な複数の経路をまとめたものである．ただし，複数の NFA 遷移をひとつの DFA 遷移に統合する際に各 NFA 遷移毎の情報は消失するため，DFA の経路からもとの NFA の経路の情報を知ることはできない．そこで，貪欲照合を可能にするためには，以下のふたつの情報を表現できるよう DFA に拡張を加える必要がある．

1. DFA 経路が表現する複数の NFA 経路の間の順序付けの情報
2. 各 DFA 遷移に対応するもとの複数の NFA 遷移の情報（PAT のタグ列の情報）

上記の項目 1 を表現するために，NFA 状態の（集合の代わりに）列を DFA 状態とする．貪欲照合では，([7] で述べられている POSIX 流の最左最長照合とは異なり）経路の優先順位が照合の進行による遷移の伸張に伴い逆転することはない．よって，各 DFA 状態毎にそれが表現する NFA 状態を順序付けしておけばよい．生成済の DFA 状態から次の遷移先の DFA 状態を生成するために，遷移元を同じくする PAT の遷移をタグ列の情報を用いて予め順序付けしておく．貪欲照合の意味

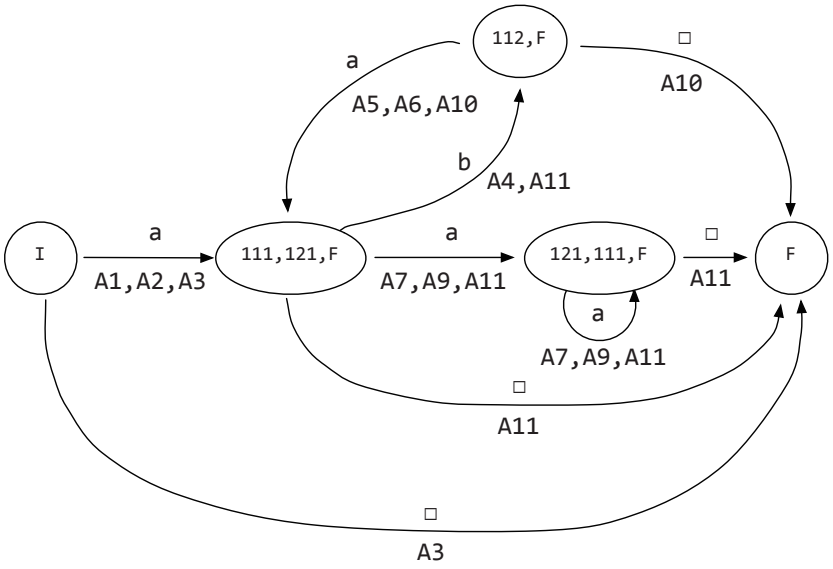


図 2　正規表現(ab+a∗)∗から図 1 の PAT を経由して最終的に生成される DFA

論と合致させるには，タグ列同士の優先順位を以下のように考えればよい．

まず，最初の要素が共通する 2 つの異なるタグ列を考える．一方が他方のプレフィックスになることはない．そこで，最初に異なるタグが見つかるまで先頭から比較する．これら異なる二つのタグの少なくとも一方は +タグである．片方のみが +タグの場合はそちらのタグ列の優先順位が高い．両方が +タグ（$+p, +q$）の場合は，p と q を辞書順（1 が 2 より優先）に比較して優先順位を決定する．また空タグ列は空でないタグ列より優先順位が低いとする．例えば，図 1 で遷移元が I の 4 つ遷移を考えると，タグ列の優先順序は（高い順に）A1,A2,A3 となる．また，121 からの 4 つの遷移に関しては A7,A9,A8,A11 の順になる．

あるタグ列が別のタグ列より優先されるという関係はタグ列の集合上の厳格半順序（strict partial order）であるが，開始タグを同じくするタグ列からなる集合上では厳格全順序になる．よって，始状態から終状態に至る異なる経路を最初に異なるタグによって上述のように比較するなら，これは始状態から終状態に至る経路の有限集合上の厳格全順序を与える．よって，最も優先される経路がひとつ存在するが，これが貪欲照合で求める経路である．

次の遷移先の DFA 状態を生成する際に，ある遷移先の NFA 状態に異なる遷移元から合流する場合には最も優先順位の高い遷移元からの遷移だけを残し他の遷移は捨て去る（枝刈り）．

図 2 に図 1 の PAT から生成された DFA を示す．各状態は元になった PAT の状態の列でラベル付けしている．I だけからなる状態が始状態，F だけからなる状態が終状態である．□ の付いた遷移は遷移元を同じくする他の遷移以外の任意の文字での遷移を表す．例えば，[111, 121, F] から [F] への遷移は a,b 以外の任意の文字による遷移を表す．

各 DFA 状態には NFA 状態の列でラベル付けしているが，列中で F より後に出現するはずの状態は取り除いてある．これは F に到達する経路より優先度の低い経路は不要だからである．この最適化により，部分集合構成で生成される DFA 状態の個数が抑制される．

上記の項目 2 で述べたように，各 DFA 遷移にはそれを生成する際に用いた（枝刈りされなかった）PAT の遷移のタグ列を関連づけている．例えば，[111, 121, F] から a を読んで [121, 111, F] に遷移する箇所では枝刈りされる A8 を除いた A7,A9,A11 が関連付けられている．

最終的に生成された DFA を用いて照合を行うには，以下のようにする．まず，与えられた文字列の末尾には終端を表す特別な文字（仮に $\$ \in \Sigma$ とする）がひとつ補われていると仮定する．始状態から開始し可能な限り遷移を続ける．遷移先は決定的なので，この手続きは与えられた文字列の長さを n とすると $O(n)$ のコストで計算できる．終了時に DFA の終状態にあれば照合は成功，それ以外では照合は失敗となる．

例えば，図 2 の DFA に文字列 abaaabaa$を与えると以下のように遷移する．

$$[\mathrm{I}] \overset{\mathrm{a,A1,A2,A3}}{\rightarrow} [111, 121, \mathrm{F}] \overset{\mathrm{b,A4,A11}}{\rightarrow} [121, \mathrm{F}] \overset{\mathrm{a,A5,A6,A10}}{\rightarrow} [111, 121, \mathrm{F}] \overset{\mathrm{a,A7,A9,A11}}{\rightarrow} [121, 111, \mathrm{F}] \overset{\mathrm{a,A7,A9,A11}}{\rightarrow} [121, 111, \mathrm{F}] \overset{\mathrm{b,A11}}{\rightarrow} [\mathrm{F}]$$

よって照合は成功し，解 $(0, 5)$ を得る．

図 2 において状態 [111, 121, F] と [121, 111, F] は集合としては等価だが区別されていることが重要である．仮にこれらを同一視するなら，文字列 abaaabaa$を与えると解 $(0, 8)$ を返すことになる．これは（部分）照合の貪欲な解ではないので誤りである．

照合に成功した場合には，DFA の遷移列を逆向きにたどることで部分式の照合が可能になる．各 DFA 遷移に関連付けられた PAT 遷移のタグ列のうち実際にどれが使用されたかは DFA の遷移列を逆向きにたどることで判明する．前節で言及したように PAT のタグ列は遷移元，遷移先の状態の情報も含んでいる．さらに，遷移先を同じくする PAT 遷移はあらかじめ枝刈りされ最も優先順位の高い遷移のみが残されているはずである．このことから，どのタグ列をたどればよいのかは一意に決まる．

例えば上の例の遷移では，A11 → A7 → A7 → A6 → A4 → A1 のようにたどることができるので，以下のように各部分式に対する照合の解を知ることができる（各部分式を構文木の位置で指定した）．[*2]

$$1 : (2, 5),\quad 11 : (0, 2)\dagger,\quad 111 : (0, 1)\dagger,\quad 112 : (1, 2)\dagger,\quad 12 : (2, 5),\quad 121 : (4, 5)$$

DFA の構築時に，関連付けられるタグ列の末尾の要素（の構文木上の位置）をキーとしたハッシュ表を作成しておくならば，タグ列を逆向きにたどる際の計算コストは $O(n)$（n は入力文字列の長さ）となる．[*3] よって，DFA を用いた（部分式の照合も含めた）貪欲照合にかかる実行時の全体の計算コストも，$O(n)$ である．これは，バックトラックを用いる照合手法よりはるかに小さいのは無論であるが，バックトラックを用いない従来手法と比較しても小さい．

一方，DFA をあらかじめ構築する本手法では DFA 構築の計算コストが生じる．実際の一般的な DFA 状態の個数は正規表現のサイズ（部分式の個数）程度 [1] とされているが，理論的には最悪の

[*2] クリーネ閉包の部分式は 0 個以上の部分文字列と照合されるが，このうち最後の照合による解のみを表示し，それ以外は $(-1, -1)$ とするのが通例である．これにしたがうなら，† の箇所はすべて $(-1, -1)$ になる．

[*3] 最後の反復以外の解を $(-1, -1)$ と報告する場合にもスタックを用いる等の工夫により，計算コストは $O(n)$ で済む．

場合 $n!$ となり許容できない．この場合は，実行時に部分集合構成を行い DFA を漸進的に構築する折衷案的な手法のほうが有効であろう．

[8] では，試作したシステムを用いて計測を行い，DFA を用いた照合手法を PAT を直接用いる手法やその他の既存の正規表現ライブラリと比較し，大幅な効率改善を確認している．

5　おわりに

DFA を用いて正規表現の（部分式の照合も含む）貪欲な照合を効率的に実現する手法を提案した．本手法は DFA を構築するための計算コストと空間コストが必要になる代わりに，与えられた文字列毎の照合に要する計算コストが大きく改善される．このため，同一の正規表現に対して繰り返し文字列を照合するような用途や，あるいは，正規表現のサイズに比較して与えられる文字列のサイズが遥かに大きい場合には，本研究の手法がきわめて有効である．

本研究は現在も進行中であり，今後は DFA の構築に要するコストの評価，漸進的に DFA を構築する手法との比較検討などを行う予定である．

参考文献

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principle, Techniques, and Tools*. Prentice Hall, 2006.

[2] R. Cox. Regular expression matching can be simple and fast. Available online: `http://swtch.com/~rsc/regexp/regexp1.html`, 2007.

[3] A. Frisch and L. Cardelli. Greedy regular expression matching. In *ICALP04 (LNCS 3142)*, pp. 618–629, 2004.

[4] V. M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, Vol. 16, No. 5, pp. 1–53, 1961.

[5] V. Laurikari. Efficient submatch addressing for regular expressions. Master's thesis, Helsinki University of Technology, 2000.

[6] R. McNaughton and H Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, Vol. 9, pp. 39–47, 1960.

[7] S. Okui and T. Suzuki. Disambiguation in regular expression matching via position automata with augmented transitions. In *CIAA 2010 (LNCS 6482)*, pp. 231–240, 2010.

[8] 藤田佳宏, 増田拓也. 拡張ポジションオートマトンの決定性有限オートマトンへの変換. 卒業論文, 中部大学情報工学科, 2013.