バックトラックによる正規表現マッチングの時間計算量解析

○中川 みなみ[1,a)]　南出 靖彦[2,b)]

概要：正規表現マッチングは文字列を操作するウェブプログラムなど，様々な場面で用いられており，その実装の多くはバックトラックに基づいている．そのため，正規表現マッチングにかかる時間が文字列の長さに関して線形でないことがあり，最悪の場合指数関数時間となる．
本発表では正規表現マッチングの時間計算量を判定する手法を提案する．対象の正規表現から先読み付き木トランスデューサを構成する．その先読み付き木トランスデューサから準同型写像や有限遷移系を構成し，それらを用いた遷移過程の中に反復補題が成り立つ遷移過程が存在しているか調べることで計算量を判定することができる．この判定には，Aho と Ullman による木トランスデューサの増加率の判定法を先読み付き木トランスデューサに拡張したものを利用する．
先行研究では，Engelfriet と Maneth のマクロ木トランスデューサの増加率の判定法を用い，正規表現マッチングの計算時間が入力文字列に対して線形であるかどうかを判定する手法が提案された．本発表で提案する手法は，時間計算量が線形であるかどうかの判定だけでなく $O(n^2), O(n^3), \ldots$ になることも判定できる．
この提案手法を OCaml によって実装し，既存の PHP プログラムで使用されている正規表現を対象に実験を行った．実験の結果，393 個中入力文字列の長さに対して 338 個が線形，44 個が 2 乗，6 個が 3 乗の計算量であると判定された．

キーワード：正規表現，木トランスデューサ

Analyzing Time Complexity of Regular Expression Matching Based on Backtracking

Minami Nakagawa[1,a)]　Yasuhiko Minamide[2,b)]

Abstract: Regular expression matching is used in various situations such as web programming. Most of its implementations are based on backtracking. Thus, its execution time may not be linear with respect to the length of an input string. In the worst case, it takes exponential time. In previous research, they proposed a method checking whether the matching of a given regular expression runs in linear time. They translated a regular expression to a tree transducer with regular lookahead and applied the result of Engelfriet and Maneth on proliferation rate of macro tree transducers.
In this presentation, we propose a new method of analyzing time complexity of regular matching. This method can check a regular expression matching runs in $O(n), O(n^2), O(n^3), \ldots$ with respect to the length of input. Instead of the result of Engelfriet and Maneth, we extend the result of Aho and Ullman about proliferation rate of a tree transducer to a tree transducer with lookahead. We obtain a set of homomorphisms from a transducer and then construct a transition system to check whether or not there exist a transition satisfying the condition of the pumping lemma.
We implemented this method by OCaml and conducted experiments analyzing execution time of regular expression matching. We applied out method to regular expressions used on existing PHP programs. Our experiments showed that the time complexity of 338 regular expressions are linear, 44 ones are quadratic, and 6 ones are cubic.

Keywords: regular expression, tree transducer

1. はじめに

正規表現マッチングは文字列を操作するウェブプログラムなど，様々な場面で用いられており，その実装の多くはバックトラックに基づいている．そのため，正規表現マッチングにかかる時間が文字列の長さに関して線形でないことがあり，最悪の場合指数関数時間となる．これは DoS 攻撃の足がかりになると指摘されている [8]．これは時間の問題だけではない．Perl 互換の正規表現の実装である PCRE では，正規表現の計算回数にリミットが設けられており，計算回数がリミットを超えると本来マッチするはずの文字列もマッチ失敗と判断されてしまう．そのため，計算回数が容易にリミットに達してしまう正規表現はプログラムの正当性に影響を及ぼす可能性がある．

このような状況をふまえて，Sugiyama と Minamide の先行研究 [1] では木トランスデューサを用いて正規表現マッチングの計算時間が入力文字列に対して線形であるかどうかを判定する手法が提案された．木トランスデューサとは，入力して与えられた木を別の木に変換する変換機である．この先行研究は，対象の正規表現から先読み付きトップダウン木トランスデューサを構成することにより，線形であるか判定を行うものである．構成する木トランスデューサは，入力文字列を正規表現マッチングの計算過程を表す木に変換する．このため，構成した木トランスデューサが入力の大きさに対して，定数倍の大きさに収まる木しか出力しないという性質を持つかどうかを検査することによって，計算量が文字列の長さに対して線形であることを判定することができる．

この先行研究では，計算量が線形であるかどうかの判定しか行うことができず，計算量が入力文字列の長さに対して 2 乗や 3 乗になることを判定できるようにアルゴリズムを拡張することが課題であった．また，先行研究は Engelfriet と Maneth のマクロ木トランスデューサの線形増加性判定 [6] を利用して判定を行っているため，プログラムが複雑であった．そのような状況をふまえ，本研究では計算量が線形であるかどうかの判定だけでなく $O(n^2)$ や $O(n^3)$ になることも判定できる新たなアルゴリズムを構成する．なお，本研究では Aho と Ullman によって与えられている木トランスデューサの計算量判定 [5] を先読み付き木トランスデューサに拡張したものを用いて計算量の判定を行う．

[1] 筑波大学 システム情報工学研究科 コンピュータサイエンス専攻
Department of Computer Science, University of Tsukuba
[2] 東京工業大学 数理・計算科学専攻
Department of Mathematical and Computing Sciences, Tokyo Institute of Technology
[a)] nakagawa@score.cs.tsukuba.ac.jp
[b)] minamide@is.titech.ac.jp

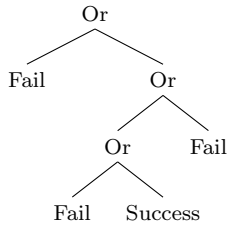

図 1 $a|b^*a$ に対する ba のマッチングの過程を表す木

本研究で提案する手法は，正規表現から先読み付き木トランスデューサを構成し，そのサイズ増加率を決定するものである．サイズ増加率を決定するために，先読み付き木トランスデューサの遷移規則から状態列を生成する準同型写像を構成する．次に，準同型写像から有限遷移系を構成し，その有限遷移系を用いて初期状態から遷移を行ったとき，特定の遷移過程が存在するかどうかの探索を行うことで計算量の判定を行う．

この提案手法を OCaml によって実装し，既存の PHP プログラムで使用されている正規表現を対象に実験を行った．実験の結果，393 個中入力文字列の長さに対して 338 個が線形，44 個が 2 乗，6 個が 3 乗の計算量であると判定された．

2. 先行研究

Sugiyama と Minamide による木トランスデューサを用いた正規表現マッチングの線形性判定の手法を説明する．先行研究で扱う正規表現は，以下の構文のものである．

$$
\begin{array}{rcll}
r & ::= & \epsilon & \text{（空列）} \\
 & | & c & \text{（文字）} \\
 & | & r_1 r_2 & \text{（連接）} \\
 & | & r_1 | r_2 & \text{（選択）} \\
 & | & r_1^* & \text{（繰り返し）}
\end{array}
$$

正規表現マッチングの過程を模倣する木とは次のような木構造である．

$$
\begin{array}{rcl}
ctree & ::= & \text{Success} \\
 & | & \text{Fail} \\
 & | & \text{Or } ctree \times ctree
\end{array}
$$

例えば，文字列 ba の $a|b^*a$ に対するマッチングの過程を表す木は図 1 のようになる．

図 1 の木は正規表現 $a|b^*a$ のマッチを全探索したものであるが，実際のバックトラックに基づく実装ではマッチングが一つ成功するとそこで計算は終了する．そのため，全探索を行う木を一つ目のマッチ成功までの木に構成し直す必要がある．そこで，上記の木構造を以下の木構造に変換する．

$$
\begin{array}{rcl}
ctree & ::= & \mathrm{Success} \\
& | & \mathrm{Fail} \\
& | & \mathrm{Or}_1\ ctree \\
& | & \mathrm{Or}_2\ ctree \times ctree
\end{array}
$$

また，全探索する計算を模倣した木から一つ目のマッチまでの計算を模倣した木に変換する関数や，部分木が成功するマッチを含むか検査する関数を使用する．

そのような木を出力する先読み付き木トランスデューサを構成する．構成した木トランスデューサが入力の大きさに対して線形な木しか出力しないならば，正規表現マッチングの計算時間が線形であるといえる．木トランスデューサによって変換された木 t' が入力木 t に対して線形サイズ増加 (linear size increase) であるとは，$size(t') \leq c \cdot size(t)$ という関係が成り立つときである．

この判定は，Engelfriet と Maneth のマクロ木トランスデューサの線形サイズ増加の判定手法を，先読み付き木トランスデューサに構成し直して行っている．

この先行研究の手法では，線形サイズ増加であることの判定しか行えず，入力文字列に対して計算量が $O(n^2)$ や $O(n^3)$ であるかの判定を行うことができなかった．また，マクロ木トランスデューサの線形サイズ増加の判定手法を用いていたことがプログラムを複雑にしていた．

3. 先読み付き木トランスデューサ

3.1 木

集合 Σ と，Σ から自然数を返す関数である $Arity$ で表される組 $(\Sigma, Arity)$ をランク付きアルファベットとする．$\Sigma^{(k)}$ は集合 $\{\sigma \in \Sigma \mid Arity(\sigma) = k\}$ を表し，$Arity(\sigma) = k$ のとき $\sigma^{(k)}$ と表記する．Σ から成る木の集合を T_Σ と表し，木 t 内における記号 σ の出現回数を $\#_\sigma(t)$ と表す．また，木 t に対して $|t|$ は木の大きさを表す．

3.2 木トランスデューサ

木トランスデューサとは，木から木への変換を行う機械である．本研究で扱う木トランスデューサは，文字列を木で表したものを入力とし，木を出力するトップダウン木トランスデューサである．ここで，文字列を木で表したものとは，$\sigma \in \Sigma$ のランクを 1 とし，文字列の終わりを表す ϵ のランクを 0 として表したものである．例えば文字列 ba の場合，$b(a(\epsilon))$ と表す．

木トランスデューサ M の定義は $M = (Q, \Sigma, \Delta, q_0, R)$ で与えられる．ここで，Q は状態の集合を，Σ と Δ はそれぞれ入力記号と出力記号の集合を表している．$q_0 \in Q$ は初期状態を表し，R は次の形式の遷移規則の集合を表している．

$$
\begin{array}{ll}
q(a) \to u & a \in \Sigma,\ u \in T_{\Delta,Q} \\
q(\epsilon) \to u & u \in T_\Delta
\end{array}
$$

文字列から木への変換であるため，遷移規則には一つの変数しか現れない．そのため，遷移規則には変数を省略した記法を用いている．

例 3.1 (トップダウン木トランスデューサ)．入力木が正規表現 $a|b^*a$ にマッチするかどうかを判定する木を出力する木トランスデューサ $M = (Q, \Sigma, \Delta, [\![a|b^*a]\!], R)$ を考える．ここで，$Q = \{[\![a|b^*a]\!], [\![b^*a]\!], [\![\epsilon]\!]\}$，$\Sigma = \{b^{(1)}, a^{(1)}, \epsilon^{(0)}\}$，$\Delta = \{\mathrm{Or}^{(2)}, \mathrm{Success}^{(0)}, \mathrm{Fail}^{(0)}\}$ とする．遷移規則 R は以下の通りに与えられる．

$$
\begin{array}{lcl}
[\![\epsilon]\!](\epsilon) & \to & \mathrm{Success} \\
[\![\epsilon]\!](a) & \to & \mathrm{Fail} \\
[\![\epsilon]\!](b) & \to & \mathrm{Fail} \\
[\![b^*a]\!](\epsilon) & \to & \mathrm{Or}(\mathrm{Fail}, \mathrm{Fail}) \\
[\![b^*a]\!](a) & \to & \mathrm{Or}(\mathrm{Fail}, [\![\epsilon]\!]) \\
[\![b^*a]\!](b) & \to & \mathrm{Or}([\![b^*a]\!], \mathrm{Fail}) \\
[\![a|b^*a]\!](\epsilon) & \to & \mathrm{Or}(\mathrm{Fail}, \mathrm{Or}(\mathrm{Fail}, \mathrm{Fail})) \\
[\![a|b^*a]\!](a) & \to & \mathrm{Or}([\![\epsilon]\!], \mathrm{Or}(\mathrm{Fail}, [\![\epsilon]\!])) \\
[\![a|b^*a]\!](b) & \to & \mathrm{Or}(\mathrm{Fail}, \mathrm{Or}([\![b^*a]\!], \mathrm{Fail}))
\end{array}
$$

この M に文字列 ba を表す木 $b(a(\epsilon))$ を入力として与えたときの変換を以下に示す．

$$
\begin{array}{ll}
 & [\![a|b^*a]\!](b(a(\epsilon))) \\
\to & \mathrm{Or}(\mathrm{Fail}, \mathrm{Or}([\![b^*a]\!](a(\epsilon)), \mathrm{Fail})) \\
\to & \mathrm{Or}(\mathrm{Fail}, \mathrm{Or}(\mathrm{Or}(\mathrm{Fail}, [\![\epsilon]\!](\epsilon)), \mathrm{Fail})) \\
\to & \mathrm{Or}(\mathrm{Fail}, \mathrm{Or}(\mathrm{Or}(\mathrm{Fail}, \mathrm{Success}), \mathrm{Fail}))
\end{array}
$$

3.3 先読み付き木トランスデューサ

先読み付きトップダウン木トランスデューサ M の定義は $M = (Q, \Sigma, \Delta, q_0, R, P, p_0, \delta)$ で与えられる．ここで，Q は状態の集合を，Σ と Δ はそれぞれ入力記号と出力記号の集合を表している．$q_0 \in Q$ は初期状態を表し，R は次の形式の遷移規則の集合を表している．

$$
\begin{array}{lll}
q(a) \to u & \langle p \rangle & a \in \Sigma,\ u \in T_{\Delta,Q} \\
q(\epsilon) \to u & & u \in T_\Delta
\end{array}
$$

ここで，$\langle p \rangle$ は入力の残りの文字列を先読みオートマトンで検査した結果が p であることを表している．また，(P, p_0, δ) は先読みオートマトンを表しており，P は状態の集合を，$p_0 \in P$ は初期状態を表している．δ は $P \times \Sigma \to P$ である遷移規則で $\delta(p, \epsilon) = p$，$\delta(p, wa) = \delta(\delta(p, a), w)$ となり，文字列の最後の文字から適用していく．$\delta(p_0, w) = p$ となるとき，$w \in L(p)$ と表す．

全ての遷移規則において右辺に一つ以上の Δ 内の記号を含む木トランスデューサを非消去 (non-erasing) であるという．本研究では，非消去な先読み付き木トラン

スデューサを扱う．

$u \in T_\Delta$ において $q(s) \xrightarrow{*} u$ となるとき，$M_q(s)$ と表記する．ここで，$M_q(s)$ の増加率を次のように定める．

$$g_M(n) = \max_{|s|=n} |M_q(s)|$$

例 3.2 (先読み付きトップダウン木トランスデューサ)**.** 例 3.1 を先読み付きトップダウン木トランスデューサにした $M' = (Q, \Sigma, \Delta', [\![a|b^*a]\!], R, P, p_0, \delta)$ を考える．ここで，$\Delta' = \{\mathrm{Or}_2^{(2)}, \mathrm{Or}_1^{(1)}, \mathrm{Success}^{(0)}, \mathrm{Fail}^{(0)}\}$, $P = \{p_\varnothing, p_0, p_1\}$ とする．なお, $p_\varnothing = \varnothing$, $p_0 = \{[\![\epsilon]\!]\}$, $p_1 = \{[\![a|b^*a]\!], [\![b^*a]\!]\}$ である．先読みオートマトンの遷移規則 δ は以下のようになる．

$$\begin{array}{rclcrcl} \delta(p_\varnothing, a) & = & p_\varnothing\ , & & \delta(p_\varnothing, b) & = & p_\varnothing \\ \delta(p_0, a) & = & p_1\ , & & \delta(p_0, b) & = & p_\varnothing \\ \delta(p_1, a) & = & p_\varnothing\ , & & \delta(p_1, b) & = & p_1 \end{array}$$

遷移規則 R' は，例 3.1 の遷移規則において状態 $[\![b^*a]\!]$ と $[\![a|b^*a]\!]$ からの遷移が以下のように変わったものである．

$$\begin{array}{lcll} [\![b^*a]\!](\epsilon) & \to & \mathrm{Or}_2(\mathrm{Fail}, \mathrm{Fail}) & \\ [\![b^*a]\!](a) & \to & \mathrm{Or}_2(\mathrm{Fail}, [\![\epsilon]\!]) & \\ [\![b^*a]\!](b) & \to & \mathrm{Or}_1([\![b^*a]\!]) & \langle p_1 \rangle \\ [\![b^*a]\!](b) & \to & \mathrm{Or}_2([\![b^*a]\!], \mathrm{Fail}) & \langle p_\varnothing \rangle, \langle p_0 \rangle \\ [\![a|b^*a]\!](\epsilon) & \to & \mathrm{Or}_2(\mathrm{Fail}, \mathrm{Or}_2(\mathrm{Fail}, \mathrm{Fail})) & \\ [\![a|b^*a]\!](a) & \to & \mathrm{Or}_1([\![\epsilon]\!]) & \langle p_0 \rangle \\ [\![a|b^*a]\!](a) & \to & \mathrm{Or}_2([\![\epsilon]\!], \mathrm{Or}_2(\mathrm{Fail}, [\![\epsilon]\!])) & \langle p_\varnothing \rangle, \langle p_1 \rangle \\ [\![a|b^*a]\!](b) & \to & \mathrm{Or}_2(\mathrm{Fail}, \mathrm{Or}_1([\![b^*a]\!])) & \langle p_1 \rangle \\ [\![a|b^*a]\!](b) & \to & \mathrm{Or}_2(\mathrm{Fail}, \mathrm{Or}_2([\![b^*a]\!], \mathrm{Fail})) & \langle p_\varnothing \rangle, \langle p_0 \rangle \end{array}$$

この M' に文字列 ba を表す木 $b(a(\epsilon))$ を入力として与えたときの変換を以下に示す．

$$\begin{array}{ll} & [\![a|b^*a]\!](b(a(\epsilon))) \\ \to & \mathrm{Or}_2(\mathrm{Fail}, \mathrm{Or}_1([\![b^*a]\!](a(\epsilon)))) \\ \to & \mathrm{Or}_2(\mathrm{Fail}, \mathrm{Or}_1(\mathrm{Or}_2(\mathrm{Fail}, [\![\epsilon]\!](\epsilon)))) \\ \to & \mathrm{Or}_2(\mathrm{Fail}, \mathrm{Or}_1(\mathrm{Or}_2(\mathrm{Fail}, \mathrm{Success}))) \end{array}$$

例 3.1 では Success の右側にも部分木が存在していたが，先読み付き木トランスデューサの例では Success の右側に部分木を構成していない．

4. 時間計算量解析

与えられた正規表現から正規表現マッチングの時間計算量を判定するアルゴリズムを構成する．木トランスデューサ M が与えられたとき，サイズ増加率の計算量を決定する．すなわち，自然数 i に対して $g_M(n) \in \Theta'(n^i)$ が成り立つか判定を行う．ここで，関数 $g(n)$ に対してのオーダ $\Theta'(g(n))$ の定義を以下に示す．

$$O(g(n)) = \{f(n) \mid \exists k. \forall n. f(n) \le kg(n)\}$$
$$\Omega(g(n)) = \{f(n) \mid \exists k. \forall n. \exists n' > n. f(n') \ge kg(n')\}$$
$$\Theta'(g(n)) = O(g(n)) \cap \Omega(g(n))$$

本研究で構成するアルゴリズムは [5] の定義や定理を基礎としている．まず，正規表現から木トランスデューサを構成し，その木トランスデューサの遷移規則から状態列を生成する準同型写像を構成する．次に準同型写像から有限遷移系を構成し，その有限遷移系を用いて初期状態から遷移を行う．そのとき，特定の遷移過程が存在するかどうかの探索を行うことでオーダの判定を行う．

4.1 準同型写像

Δ と Q 上の木から状態列を生成する関数 $\pi : T_{\Delta,Q} \to Q^*$ を次のように定める．

$$\pi(f(t_1, ..., t_n)) = \pi(t_1) \dots \pi(t_n)$$
$$\pi(q(x)) = q$$

準同型写像 $h_{a,p} : Q \to Q^*$ は，$q(a) \to u\ \langle p \rangle \in R$ のとき $h_{a,p}(q) = \pi(u)$ とする．また，状態 q' が $h_{a,p}(q)$ に含まれることを $q' \in h_{a,p}(q)$ と表記し，集合 $\mathcal{H}_{p,p'}$ を次のように定義する．

$$\mathcal{H}_{p,p'} = \{h_{a,p'} \mid \delta(p', a) = p\}$$

$p'_1 = p_2$ である場合のみ, 2 つの準同型写像 $h_1 \in \mathcal{H}_{p_1,p'_1}$, $h_2 \in \mathcal{H}_{p_2,p'_2}$ は合成される．このとき，$\mathcal{H}^n_{p,p'}$ と $\mathcal{H}^*_{p,p'}$ を次のように定義する．

$$\begin{array}{rcll} \mathcal{H}^0_{p,p} & = & \{\mathrm{id}\} & \\ \mathcal{H}^0_{p,p'} & = & \varnothing & \text{if } p \neq p' \\ \mathcal{H}^{n+1}_{p,p'} & = & \bigcup_{p'' \in P} \mathcal{H}^n_{p,p''} \circ \mathcal{H}_{p'',p'} & \\ \mathcal{H}^*_{p,p'} & = & \bigcup_{n \ge 0} \mathcal{H}^n_{p,p'} & \end{array}$$

次の定理は，[5] で与えられている定理を [6] における議論をもとに詳細化したものである．

定理 4.1 (反復補題)**.** $\mathcal{H}_{p,p'}$ を Q 上の準同型写像とする．$Q' \subseteq Q$，$q \in Q$，$\alpha \in \mathcal{H}^*_{p_1,p_3}$ であり，$\#_{Q'}(\alpha(q))$ に上限がないとき，次のような $\beta_1 \in \mathcal{H}^*_{p_1,p_2}$，$\beta_2 \in \mathcal{H}^*_{p_2,p_2}$，$\beta_3 \in \mathcal{H}^*_{p_2,p_3}$ と $q_1, q_2 \in Q$ が存在する．

- $q_1 \in \beta_3(q)$
- q_1 と q_2 は $\beta_2(q_1)$ 内の別々の場所に出現する
- $q_2 \in \beta_2(q_2)$
- $\beta_1(q_1) \cap Q' \neq \varnothing$，$\beta_1(q_2) \cap Q' \neq \varnothing$

4.2 状態の増加率とそのオーダ

先読みの状態 p での状態 q の増加率 $f_{q,p}(n)$ を次のように定める．

$$f_{q,p}(n) = \max\{|\alpha(q)| \mid \exists p'.\alpha \in \mathcal{H}^n_{p,p'}\}$$

$i \geq 0$, $j \geq -1$ に対して，状態の集合 $Q_p^{(i)} \subseteq Q$ と $Q_p^{[j]} \subseteq Q$ を次のように定義する．

$$Q_p^{[-1]} = \varnothing$$

$$Q_p^{(i)} = \{q \mid q \in Q - Q_p^{[i-1]} \wedge f_{q,p}(n) \leq \exists c.cn^i\}$$

$$Q_p^{[i]} = Q_p^{[i-1]} \cup Q_p^{(i)}$$

定理 4.2. M を非消去な先読み付き木トランスデューサとする．このとき，$q \in Q_p^{(i)}$ であることと $g_{M_q}(n) \in \Theta'(n^{i+1})$ であることは同値である．

また，以下の定理は $Q_p^{(i)}$ の要素を帰納的に判定するものである．

定理 4.3 (Lemma 4.3[5]). $q \in Q - Q_p^{[i]}$, $i \leq -1$ のとき，全ての p' と $\alpha \in \mathcal{H}^*_{p,p'}$ において，$\#_{Q-Q_{p'}^{[i]}}\alpha(q)$ が有限であるならば $q \in Q_p^{(i+1)}$ である．

4.3 有限遷移系

定理 4.1 と定理 4.3 に基づき状態の増加率を決定するために，準同型写像の集合 $\mathcal{H}_{p,p'}$ から有限遷移系の遷移規則を次のように生成する．ここで，$h_{a,p'} \in \mathcal{H}_{p,p'}$ である．

- $\langle q,p\rangle \to \langle q',p'\rangle$ if $q' \in h_{a,p'}(q)$
- $\langle q,p\rangle \to \langle (q_i,q_j),p'\rangle$ if $q_i, q_j \in h_{a,p'}(q),\ i \neq j$
- $\langle (q_1,q_2),p\rangle \to \langle (q_1',q_2'),p'\rangle$ if $q_1' \in h_{a,p'}(q_1),\ q_2' \in h_{a,p'}(q_2)$
- $\langle (q_1,q_2),p\rangle \to \langle (q_i,q_j,q_k),p'\rangle$ if $q_i,q_j \in h_{a,p'}(q_1), q_k \in h_{a,p'}(q_2), i \neq j$
- $\langle (q_1,q_2,q_3),p\rangle \to \langle (q_1',q_2',q_3'),p'\rangle$ if $q_1' \in h_{a,p'}(q_1),\ q_2' \in h_{a,p'}(q_2),\ q_3' \in h_{a,p'}(q_3)$

このとき，初期状態を q_0 とすると，有限遷移系を使って初期状態から遷移したときに $\langle q_0,p\rangle \xrightarrow{*} \langle (q_1,q_2),p_1\rangle \xrightarrow{*} \langle (q_1,q_2,q_2),p_2\rangle$ となる遷移過程がある場合，定理 4.1 より反復補題が成り立つといえる．その場合，$\alpha \in \mathcal{H}^*_{p,p'}$ において，$\#_Q(\alpha(q_0))$ は有限ではない．

4.4 アルゴリズム

準同型写像 $\mathcal{H}_{p,p'}$ が与えられたときの増加率を判定するアルゴリズムを以下に示す．

---

function *ComputeOrder*($\mathcal{H}_{p,p'}$)
begin
  $i := -1$;
  repeat
    $Q' := Q - Q_p^{[i]}$;
    $\hat{\mathcal{H}}_{p,p'} = \{\hat{h}_{a,p'} \mid h_{a,p'} \in \mathcal{H}_{p,p'}\}$ を構成する;
    $\hat{h}_{a,p'} = h_{a,p'}(q)$ から $Q_p^{[i]}$ 内の状態を除去した状態列 ($q \in Q'$);
    $\hat{\mathcal{H}}_{p,p'}$ から有限遷移系を構成する;
    $Q_p^{(i+1)} = \{q \mid q \in Q' \wedge$ 有限遷移系の遷移過程より反復補題が成り立たない $\}$;
    if $q_0 \in Q_p^{(i+1)}$ **then break**;
    else if $Q_p^{(i+1)} = \varnothing$ **then** $i := -3$; **break**;
    else $i := i + 1$;
  until false;
  ComputeOrder $:= i + 2$;
end;

上記のアルゴリズム 2(b) で行われる $\hat{h}_{a,p'}$ の計算では，次の $\hat{q}$ を使用する．

$$\hat{q} = \begin{cases} q & \text{if } q \notin Q_p^{[i]} \\ \epsilon & \text{if } q \in Q_p^{[i]} \end{cases}$$

$h_{a,p}(q) = q_1q_2\ldots q_n$ とすると，$\hat{h}_{a,p}(q) = \hat{q_1}\hat{q_2}\ldots\hat{q_n}$ となる．

また，$Q_p^{(i+1)}$ の判定では，$q \in Q'$ において有限遷移系から $\langle q,p\rangle \xrightarrow{*} \langle (q_1,q_2),p_1\rangle \xrightarrow{*} \langle (q_1,q_2,q_2),p_2\rangle$ となる遷移過程が存在するか探索を行う．存在しなかった場合，反復補題が成り立たず $\#_Q(\alpha(q))$ は有限となり，定理 4.3 より $q \in Q_p^{(i+1)}$ であるといえる．

$q_0 \in Q_p^{(i)}$ だった場合，オーダは $\Theta'(n^{i+1})$ になる．また $Q_p^{(i)}$ が空であった場合，オーダは入力文字列に対して指数関数的になる．

5. 実装と実験

4 章で説明した方法で正規表現マッチングの過程を模倣する先読み付き木トランスデューサを構成し，5 章で表した方法でそのオーダを判定する．実装は OCaml を用いて行った．正規表現の構文解析には Sakuma ら [3] が作成したライブラリを用いた．Perl 準拠の標準的な正規表現をサポートし，特に 2 章の構文に以下の構文を加えたものを対象としている．

r	::=	...
	|	$[c_1c_2\ldots c_n]$ （文字クラス）
	|	$r^?$ （0 回または 1 回の r）
	|	$r^{*?}$ （貪欲でない繰り返し）
	|	r^+ （1 回以上の繰り返し）
	|	$r\{n\}\{m\}$ （n 回以上 m 回以下の繰り返し）

貪欲でない繰り返しとは，より少ない r の繰り返しにマッチする構文で，$r^{*?} = \epsilon|rr^{*?}$ と展開される．先読みやアドミックグループ，後方参照といった拡張はサポートしていない．なお，行頭を意味する ^ から始まらない正規表現は $.^{*?}$ から始まるものとし，行末を意味する $ で終わらない正規表現は $.^*$ で終わるものとする．

先行研究 [1] のプログラムは 6000 行を超えるもので

あったが，本研究のプログラムは 1500 行と大幅に簡単になった．

6. 実験

先行研究 [1] で使用された正規表現 393 個を対象に実験を行った．これは，PHO-Fusion，phpMyAdmin，SquirrelMail，TorrentFlux，XOOPS の五つのプログラムで使用されている正規表現である．なお，実験では 900s の制限時間を設定し，それを超えた場合はタイムアウトとした．実験の結果，正規表現マッチングの計算量が入力文字列の長さに対して線形であると判定された正規表現は 338 個，$\Theta'(n^2)$ であると判定された正規表現は 44 個，$\Theta'(n^3)$ であると判定された正規表現は 6 個，タイムアウトとなった正規表現は 5 個であった．実験結果の一部を表 1 に示す．また，非線形と判定された正規表現に関しては，出力された反例を先読み付き木トランスデューサで実行し，計算量を確認した．

r_1 は / で区切られた 6 個の任意の文字列からなる文字列にマッチする．この正規表現は先行研究でも線形であるという結果が得られているが，判定にかかった計算時間が先行研究では 757s なのに対し，本研究では 3.28s となり大幅に時間を短縮することができた．

r_2 は `content-transfer-encoding:` の後，任意個の空白が続き，その後 1 つ以下の - を含む任意の長さの英字からなる文字列を含む文字列にマッチする．この正規表現は先行研究ではタイムアウトであったが，本研究では線形であるという結果が得られた．

r_3 は `BAD` か `NO` が含まれる文字列にマッチする．先行研究では非線形と判定され，本研究では $\Theta'(n^2)$ と判定された．これは `BAD` も `NO` も含まれない文字列に対して $\Theta'(n)$ にならないと考えられ，$\Theta'(n)$ の反例として文字列 DD^nDO が出力された．しかし，PCRE で反例の文字列を実行した結果，$n = 10,000$ のとき 0.71ms，$n = 20,000$ のとき 1.26ms，$n = 40,000$ のとき 2.4ms となり，線形に近い結果となった．r_3 は行頭の ^ がないため，$.^{*?}(.*)$ から始まることになる．PCRE ではこの部分で最適化が行われているため，このような結果になったと予想される．

r_4 は数字，`a` から `h` までの文字と記号 : からなる任意の文字列 2 つが :: によって繋がっている文字列にマッチする．先行研究では線形と判定されていたが，本研究では $\Theta'(n^2)$ と判定された．また，$\Theta'(n)$ の反例として $0 :: 0(:: 0)^nA0$ が出力されている．PCRE で反例の文字列を実行した結果，$n = 100$ のとき 0.49ms，$n = 200$ のとき 1.55ms，$n = 400$ のとき 5.8ms となり $O(n^2)$ に近いことが確認できた．

r_5 は `<image` の後に任意の文字列が続き `>` で終わる文字列と `</image>` の間に任意の文字列を含む文字列にマッチする．先行研究では非線形と判定され，本研究では $\Theta'(n^3)$ と判定された．また，反例の文字列として $>< imag(e >< image)^n >>^n> /imag$ が出力された．PCRE で反例の文字列を実行した結果，$n = 100$ のとき 0.079s，$n = 200$ のとき 0.58s，$n = 400$ のとき 4.3s となり $O(n^3)$ に近いことが確認できた．

r_6 は数字と小文字のアルファベットを 32 回繰り返す文字列にマッチする．先行研究では線形と判定されたが，本研究では時間内に計算が終わらずタイムアウトとなった．これは先読み付き木トランスデューサの状態数が 32，先読みオートマトンの状態数が 34 となり，準同型写像の数が 39168 と多くなってしまい，制限時間内に有限遷移系を構成できなかったためである．

7. 関連研究

山口は，NFA よって最左優先一致を行う手法を正規表現の微分とモナドを用いて形式的に表現し，それがバックトラックによる最左優先一致と結果が一致することを示した [4]．本研究では，ここで示されている正規表現の微分に関する考え方を用いて，木トランスデューサの遷移規則を構成している．

Kirrage らは，正規表現マッチングの計算時間が指数関数的になるかどうかの判定を行う研究を行っている [2]．彼らは正規表現マッチングを模倣する非決定性抽象機械を用いて判定を行った．この研究では，実際のバックトラックに基づく実装と同じように一つ目のマッチ成功までの正規表現マッチングに関しての計算時間を判定することが目的であった．しかし，彼らは全探索をする正規表現マッチングに関しての計算時間の判定しか行えていない．

Drewes は，木トランスデューサの出力木の大きさが入力木の大きさに対して指数関数的に増えるかどうかを線形時間で判定を行っている [7]．この判定には，定理 4.1 の反復補題が成り立つかどうかを検査するために導入した 5.3 節の有限遷移系に近い考え方が用いられている．

8. 結論

本研究では，バックトラックに基づく正規表現マッチングの時間計算量を判定する新たな手法を提案した．正規表現から先読み付き木トランスデューサを構成し，そのサイズ増加率を決定することでオーダの判定を行った．Aho と Ulman が示した定義や定理 [5] を基に，木トランスデューサから状態列を取り出す準同型写像を用い，その準同型写像から有限遷移系を構成した．構成した遷移系を用いて初期状態から遷移したとき，反復補題が成り立つ遷移過程が存在しているかどうかを調べることでオーダを判定することができる．また，オーダが $\Theta'(n^2)$ 以上だった場合はその反例の文字列も出力できる．

表 1 実験結果

正規表現	計算量	判定にかかった計算時間
r_1 = `^.*\/.*\/.*\/.*\/.*$`	$\Theta'(n)$	3.28s
r_2 = `^.*(content-transfer-encoding:)\s*(\w+-?(\w+)?).*`	$\Theta'(n)$	129s
r_3 = `(.*)(BAD|NO)(.*)$`	$\Theta'(n^2)$	0.990s
r_4 = `^([0-9a-h\:]+)::([0-9a-h\:]+)$`	$\Theta'(n^2)$	0.145s
r_5 = `<image.*?>(.*?)</image>`	$\Theta'(n^3)$	8.10s
r_6 = `^[0-9a-z]{32}$`	timeout	900s

構成したアルゴリズムを用いて，既存の PHP プログラムで使用されている正規表現を対象にオーダの判定を行った．先行研究では非線形と判定されたものも，$\Theta'(n^2)$ や $\Theta'(n^3)$ であることを判定することができた．

今後の課題として，実際のセキュリティツールを用いてテストを行うことが挙げられる．Kirrage ら [2] は，オープンソースの侵入防止システムである *Snort*[9] 等をテストに用いている．それらを用いるためには，正規表現の構文解析のライブラリを構成し直す必要がある．

また，アルゴリズム自体のオーダを検査することも課題として挙げられる．先読み付き木トランスデューサの構成では，先読みオートマトンを構成する際にサブセット構成をするため，オーダは指数関数的，その他のアルゴリズムでは多項式になると予想される．全体では指数関数的になると予想しているが，詳細を確認する必要がある．

参考文献

[1] Sugiyama, S. and Minamide, Y.: Checking Time Linearity of Regular Expression Matching Based on Backtracking. *IPSJ Transactions on Programming*, Vol.7, No.3, pp.1-11, 2014.

[2] Kirrage, J., Rathnayake, A. and Thielecke, H.: Static Analysis for Regular Expression Denial-of-Service Attacks. *International Conference on Network and System Security(NSS 2013)*, pp.135-148, 2013.

[3] Sakuma, Y., Minamide, Y. and Voronkov, A.: Translating Regular Expression Matching into Transducers, *Journal of Applied Logic*, Vol.10, No.1, pp.32-41, 2012.

[4] 山口 隆成: 正規表現の微分を用いた線形時間マッチングアルゴリズム，卒業研究論文, 2015.

[5] Aho, A.V. and Ullman, J.D.: Translations on a context free grammar, *Information and Control*, Vol.19, Issue 5, pp.453-456, 1971.

[6] Engelfriet, J. and Maneth, S.: Macro Tree Translations of Linear Size Increase are MSO Definable, *SIAM J.Comput*, Vol.32, pp.950-1006, 2003.

[7] Drewes, F.: The complexity of the exponential output size problem for top-down and bottom-up tree transducers, *Information and Computation, 169*, pp.264-283, 2001.

[8] OWASP.: Regular expression Denial of Service - ReDoS. `https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS`, 2012.

[9] Snort(IDS/IPS).: `http://www.snotr.org/`, 2012.