情報処理学会論文誌 Vol.57 No.12 2769–2783 (Dec. 2016)

決定性有限オートマトンによる正規表現の貪欲な部分照合と部分式による捕獲

奥居 哲[1,a)] 増田 拓也[2,b)] 鈴木 大郎[3,c)]

受付日 2016年3月14日, 採録日 2016年9月6日

概要：決定性有限オートマトン（DFA）を用いて正規表現の貪欲な部分照合を実現する手法について述べる．通常の DFA の状態が NFA の状態の部分集合からなるのに対し，本手法における DFA の状態は重複のない NFA 状態の列とすでに終状態に達したか否かを表すフラグからなる．この DFA が貪欲な部分照合と部分式による捕獲を正しく実現することを証明する．バックトラックによる試行を行う多くの実装では照合に要する計算コストが最悪の場合，指数的に増大するのに対し，本手法の照合に要する最悪の時間計算量は，DFA を漸進的に構築する場合では $O(nm)$，DFA をあらかじめ構築する場合では $O(m)$ に抑えられる．ここで n は正規表現の部分式の個数，m は照合文字列の長さである．一方，照合結果から部分式による捕獲を計算するのに要する時間計算量は $O(mk)$ である．ここで k は捕獲に用いる部分式の個数である．本手法の試験実装に基づき照合に要する計算時間を計測した結果，Google の正規表現ライブラリ RE2 と比較してより良好な結果が得られた．古典的な文字列照合手法である KMP アルゴリズムあるいは Aho-Corasick オートマトンとの関連にも言及する．

キーワード：正規表現，貪欲照合，決定性有限オートマトン，KMP 文字列照合

A DFA-based Approach to Greedy Partial Regular Expression Matching with Subterm Addressing

Satoshi Okui[1,a)] Takuya Masuda[2,b)] Taro Suzuki[3,c)]

Received: March 14, 2016, Accepted: September 6, 2016

Abstract: We present a DFA-based approach which enables greedy partial regular expression matching and subterm addressing without relying on any backtrack search. Each state of our DFA consists, compared to the standard construction regarding subsets of NFA-states as DFA-states, of an non-duplicating sequence of NFA-states and a flag indicating whether the final state has been found or not. The worst case computational cost for matching is $O(nm)$ in the case of on-the-fly construction of DFA, and $O(m)$ if DFA has been constructed in advance where n is the number of subexpressions in the regular expression and m the length of the input string, while the cost for subterm addressing requires $O(mk)$ where k is the number of subexpression to be addressed. We prove the correctness of our algorithm with respect to a formalization of matching semantics. Our experimental implementation shows certain improvement to Google RE2 library for the test case they have given. We also mention a relation with a classical string matching algorithm given by Knuth, Morris and Pratt, or Aho-Corasick automata.

Keywords: regular expression, greedy matching, deterministic finite automaton, KMP string matching

1. はじめに

正則言語の認識に有限オートマトン（FA）を用いる手

[1] 中部大学
Chubu University, Kasugai, Aichi 487–8501, Japan
[2] なし
[3] 会津大学
The University of Aizu, Aizu Wakamatsu, Fukushima 965–8580, Japan
[a)] okui@cs.chubu.ac.jp
[b)] masuda.1990.04@gmail.com
[c)] taro@u-aizu.ac.jp

法は，コンパイラや形式言語の教科書（たとえば文献 [2]）で一般的である．その一方で，実際的な正規表現と文字列の照合処理には単なる認識問題にはない側面がある．第 1 に，文字列全体との照合（全体照合）だけでなくその一部分との照合（部分照合）が許されること，第 2 に，正規表現の各部分式が文字列のどの位置に正確に照合されるのか（以下，部分式による捕獲）を計算する必要があること，第 3 に，前方参照（back reference）を含む正規表現が許されること等があげられる．このため，実用的な正規表現エンジンの実装では照合の解をバックトラックで探索する仮想機械が用いられることが多い．

最初の 2 つの特徴は，2 通り以上の照合が可能であるという曖昧さをもたらす．この曖昧さを解消するために導入されるのが，最左最長照合や貪欲照合といった制約である．本論文では貪欲照合を対象とする．

前方参照を含む正規表現は一般に文脈依存言語を表す [4] ため FA による処理が不可能なのは当然であるが，前方参照を含まない正規表現に対しては FA に基づく，より効率的な手法が提案されている．Frisch らは，貪欲な全体照合の定式化に基づく 2 パスアルゴリズムを与えている [11]．Cox は Thompson の古典的論文 [22] の現代的意義と，それを用いた正規表現ライブラリ RE2 の実装手法について述べている [5], [6], [7], [8]．Grathwohl らは部分式による捕獲に Nielsen らの最適化された方法 [17] を用いている [12], [13]．いずれの手法も部分集合構成による決定性有限オートマトン（DFA）の漸近的な構築を用いており，非効率的なバックトラックを回避しているが，そのアルゴリズムの正当性については十分議論されていない（特に RE2 ライブラリの詳細はソースコードを参照するしかない）．Frisch らの手法と Grathwohl らの手法は全体照合のみを扱っており，これらをいかに部分照合に適合させられるかはまったく明らかではない．また，新屋らは遷移モノイドのアイデアを取り入れることで，投機的な並列実行に関わる処理を事前に行うことで照合実行時の効率を大きく改善する手法を提案しているが，部分式の捕獲には対応していない [20], [24]．

そこで，DFA を用いて貪欲な部分照合と部分式による捕獲を可能にする手法を提示し，その正当性を示し，さらに試験実装に基づいて実際的な照合効率を明らかにしようというのが本論文である．

まず 2 章で，Frisch らの全体照合の意味論に基づき貪欲な部分照合を定式化する．貪欲照合は実装主導に依るところが大きく，微妙に異なる意味で用いられる場合がある．このため本論文における貪欲照合の意味を厳密に規定しておくことは，本論文全体の議論を正確にするために不可欠な前提である．

次に 3 章では，正規表現から Thompson 構成により得られた非決定性有限オートマトン（NFA）の経路が前章で定式化した貪欲な部分照合の解と正確に対応していることを示す．本章の所要の結果を得るためには，反復の実現に現代の教科書等で一般的な構成法ではなく Thompson の原論文 [22] で提示されている方法を用いる必要があることを指摘する．

続く 4 章では，3 章の NFA から DFA を構築する手法について述べる．本論文で提案する DFA は，通常の DFA が NFA の状態の部分集合を状態と見なすのに対し，NFA の状態からなる重複のない列とすでに終状態に到達したか否かを表すフラグの組を状態と見なす点が異なる．この DFA により貪欲な部分照合と部分式の捕獲が可能になることを示し，照合アルゴリズムの計算量について論じる．また，古典的な KMP 文字列照合アルゴリズム [14] あるいは Aho-Corasick オートマトン [1] との関連にも言及する．

5 章では，提案する手法の試験実装に基づく照合実験の結果について述べ，最後の章で結論を述べる．

本論文では以下の記法等を断りなく使用する．有限個の要素 $a_1, \ldots, a_n$ の**列**（並び）を場合によってはコンマを省略して $a_1 \ldots a_n$ のように表す．あるいは，**リスト**の表記法を用いて $[a_1, \ldots, a_n]$ のように表す．要素 a が列 $\mathbf{a}$ に出現するとき，集合の表記を流用して $a \in \mathbf{a}$ で表す．要素 $a \in \mathbf{a}$ が要素 $a' \in \mathbf{a}$ に先行して出現するとき $a \prec_{\mathbf{a}} a'$ と表す．列 u と v をこの順に連結したものを u, v あるいは単に uv と書くか，リストの表記法を用いて $u +\!+ v$ のように表す．**空列**は ε，あるいはリストの表記を用いて [] で表す．列 w を n（≥ 0）個連結してできる文字列を w^n で表す．列 w の**長さ**（要素の数）を $|w|$ で表す．集合 S の要素の（有限）**列**（空列を含む）の集合を S^* で表す．$w \in S^*$ の**接頭辞**の集合 $\{u \in S^* \mid \exists v \in S^*.\ uv = w\}$ を $\rho(w)$ で表す．w からその接頭辞 u を除いた残りの部分を w/u で表す．s と t が互いに他方の接頭辞でないとき，すなわち $s \notin \rho(t) \wedge t \notin \rho(s)$ のとき，$s \rhd\!\lhd t$ で表す．

2. 正規表現の貪欲な部分照合

2.1 正規表現と照合

アルファベット Σ 上の正規表現は以下の表現からなる．

$$
\begin{array}{ll}
r ::= & 1 \qquad \text{（空列）} \\
\mid & a \quad (a \in \Sigma) \quad \text{（文字）} \\
\mid & (r{+}r) \qquad \text{（和）} \\
\mid & (r{\cdot}r) \qquad \text{（連結）} \\
\mid & (r{*}) \qquad \text{（反復）}
\end{array}
$$

部分式による捕獲を考慮する場合には，結合順序が照合の結果に微妙な違いをもたらす [10]．そのため，本論文では和や連結に関して結合律を仮定しない．また，貪欲照合においては和の交換律も仮定しないことに注意が必要である．本論文では和，連結ともに右結合であると規定し，さらに通常通り反復は結合より強く，結合は和より強く結び

付くとして適宜，括弧を省略する．また，空な言語を表す正規表現（0）は使用しない．

Σ 上の正規表現の集合を $\mathrm{Reg}(\Sigma)$ で表す．正規表現 r の抽象構文木としての**高さ** $\mathrm{h}(r)$ を，その部分木の高さの最大値に 1 を加えた値と定義する．1 と a の高さは 1 とする．

正規表現 r によって表現される文字列の集合を $\mathrm{L}(r)$ で表す．

$$\mathrm{L}(1) = \{\varepsilon\}$$
$$\mathrm{L}(a) = \{a\} \quad (a \in \Sigma)$$
$$\mathrm{L}(r_1{+}r_2) = \mathrm{L}(r_1) \cup \mathrm{L}(r_2)$$
$$\mathrm{L}(r_1{\cdot}r_2) = \{w_1w_2 \mid w_1 \in \mathrm{L}(r_1), w_2 \in \mathrm{L}(r_2)\}$$
$$\mathrm{L}(r_1{*}) = \mathrm{L}(r_1)^*$$

アルファベット Σ と文字列 $w \in \Sigma^*$, 正規表現 $r \in \mathrm{Reg}(\Sigma)$ の組 $\langle \Sigma, w, r\rangle$ を**照合問題**と呼び，$w_1w_2w_3 = w$ かつ $w_2 \in \mathrm{L}(r)$ となる文字列 w_1, w_2, w_3 が存在するとき，r は w に**部分照合**できるという．特に $w_1 = w_3 = \varepsilon$ であるとき，r は w に**全体照合**できるという．

全体照合は，どんな正規表現 r や照合文字列 w にも含まれない文字を r と w の前後に補って始点と終点を表すことで，部分照合の問題に容易に帰着可能である．そこで，以下では部分照合のみを扱う．

正規表現 r が文字列 w に部分照合できるとき，w_1, w_2, w_3 の分割の仕方は必ずしも唯一に決まらないという意味で部分照合は曖昧さをはらんでいる．また，正規表現の各部分式による捕獲（各部分式ごとの照合）まで考慮すると，さらに曖昧さが生じる．この曖昧さを正確に議論するために，正規表現の各部分式と文字列の各部分がどのように対応するかを導出木（構文解析木）の概念を用いて表現する．正確には正規表現 r 上の**導出木**の集合 $\mathrm{D}(r)$ を以下のように定義する[*1]．

$$\mathrm{D}(1) = \{1\}$$
$$\mathrm{D}(a) = \{a\} \quad (a \in \Sigma)$$
$$\mathrm{D}(r_1{+}r_2) = \{\mathtt{L}(t) \mid t \in \mathrm{D}(r_1)\} \cup \{\mathtt{R}(t) \mid t \in \mathrm{D}(r_2)\}$$
$$\mathrm{D}(r_1{\cdot}r_2) = \{t_1{\cdot}t_2 \mid t_1 \in \mathrm{D}(r_1), t_2 \in \mathrm{D}(r_2)\}$$
$$\mathrm{D}(r_1{*}) = \{\mathtt{I}(\mathbf{t}) \mid \mathbf{t} \in \mathrm{D}(r_1)^*\}$$

$\mathrm{D}(r)$ の要素は，ドット（$\cdot$），`L`，`R`，`I` を分岐ノードとする木であり，葉は 1 か文字か `I` のいずれかである（`I` が葉として出現するのは $\mathbf{t}$ が空列のときであり，これを `I`() で表す）．`I` の子は 0 個以上いくつあってもよいので，導出木は一般にはランク不定木（unranked tree）である．

$\mathrm{D}(r)$ の要素 t の**高さ** $\mathrm{h}(t)$ を t の直接の部分木の高さの最大値に 1 を加えた値と定義する．葉の高さは 1 とする．また，$\mathrm{D}(r)$ の要素 t の**大きさ** $|t|$ をそれに含まれるノード（葉も含む）の総数と定義する．

導出木 t の葉における文字の出現を左から順に並べることで，t によって**導出**される文字列が得られる．これを $\mathrm{d}(t)$ と書く．$\mathrm{L}(r) = \{\mathrm{d}(t) \mid t \in \mathrm{D}(r)\}$ が成り立つ．

導出木の概念を用いると，部分および全体照合は以下のように述べられる．

定義 1（照合の解） 照合問題 $\langle \Sigma, w, r\rangle$ にたいして，$uw'v = w$ となる文字列 u, w', v と $\mathrm{d}(t) = w'$ となる導出木 $t \in \mathrm{D}(r)$ が存在するとき照合問題 $\langle \Sigma, w, r\rangle$ は部分照合可能であるといい，$\langle u, t\rangle$ をこの部分照合の解と呼ぶ．

部分照合の解は一般には唯一に決まらない．

例 1 照合問題 $\langle \{a\}, aa, a\rangle$ は部分照合可能で，その解は $\langle \varepsilon, a\rangle$ と $\langle a, a\rangle$ の 2 つある．

さらに，正規表現の部分式の対応の仕方が一般に複数あることも解が唯一に決まらない要因となる．

例 2 照合問題 $\langle \{a, b, c\}, abc, (a{\cdot}b{+}a){\cdot}(b{\cdot}c{+}c)\rangle$ は部分照合可能で，その解は $\langle \varepsilon, \mathtt{L}(a{\cdot}b){\cdot}\mathtt{R}(c)\rangle$ と $\langle \varepsilon, \mathtt{R}(a){\cdot}\mathtt{L}(b{\cdot}c)\rangle$ の 2 つ存在する．

例 3 照合問題 $\langle \{a\}, a, (a{+}1){*}\rangle$ は部分照合可能で，その解は無数に存在する．たとえば，

$$\langle \varepsilon, \mathtt{I}(\mathtt{L}(a))\rangle, \langle \varepsilon, \mathtt{I}(\mathtt{L}(a), \mathtt{R}(1))\rangle, \langle \varepsilon, \mathtt{I}(\mathtt{L}(a), \mathtt{R}(1), \mathtt{R}(1))\rangle, \ldots$$

はすべてこの部分照合の解である．

2.2 貪欲照合

照合の解を唯一に決めるために広く用いられている戦略の 1 つが**貪欲照合**（greedy matching）である．貪欲照合の基本的な考え方は実装主導であり，正規表現に対する導出木をトップダウンに構築するアルゴリズムをバックトラックを用いて実装した際に最初に発見される解を選出する．この際，和 $(r_1{+}r_2)$ の探索では r_1 の探索が r_2 の探索より優先される．また，反復 $r = r_1{*}$ は再帰的に $r = r_1{\cdot}r{+}1$ と見なしたうえで，やはり最初（左）の選択肢を優先して（つまり，できるだけ多く反復するように）探索する（「貪欲」の所以）．探索の結果最初に見つかる解が貪欲な照合の解と見なされる．これを [11] に従い以下のように定式化する．

定義 2（導出木間の優先順位） $\mathrm{D}(r)$ 上の二項関係 $\lessdot_r$ を以下の条件を満たす最小の二項関係（すなわち以下の条件を満たすすべての二項関係の交わり）として定義する．

(or 1) $\forall s \in \mathrm{D}(r_1), t \in \mathrm{D}(r_2).\ \mathtt{L}(s) \lessdot_{r_1+r_2} \mathtt{R}(t)$

(or 2) $\forall s, t \in \mathrm{D}(r_1).(s \lessdot_{r_1} t \Rightarrow \mathtt{L}(s) \lessdot_{r_1+r_2} \mathtt{L}(t))$

(or 3) $\forall s, t \in \mathrm{D}(r_2).(s \lessdot_{r_2} t \Rightarrow \mathtt{R}(s) \lessdot_{r_1+r_2} \mathtt{R}(t))$

(cat 1) $\forall s_1, t_1 \in \mathrm{D}(r_1),\ s_2, t_2 \in \mathrm{D}(r_2).$
$$(s_1 \lessdot_{r_1} t_1 \Rightarrow s_1{\cdot}s_2 \lessdot_{r_1{\cdot}r_2} t_1{\cdot}t_2)$$

(cat 2) $\forall u_1 \in \mathrm{D}(r_1),\ s_2, t_2 \in \mathrm{D}(r_2).$
$$(s_2 \lessdot_{r_2} t_2 \Rightarrow u_1{\cdot}s_2 \lessdot_{r_1{\cdot}r_2} u_1{\cdot}t_2)$$

[*1] 本論文では，導出木を項として表す（導出木は項の抽象構文木として与えられる）．

(star 1) $\forall \mathbf{s} \in \mathrm{D}(r_1)^+.\ (\mathtt{I}(\mathbf{s}) \lessdot_{r_1*} \mathtt{I}())$

(star 2) $\forall s_1, t_1 \in \mathrm{D}(r_1), \mathbf{s}, \mathbf{t} \in \mathrm{D}(r_1)^*.$
$(s_1 \lessdot_{r_1} t_1 \Rightarrow \mathtt{I}(s_1, \mathbf{s}) \lessdot_{r_1*} \mathtt{I}(t_1, \mathbf{t}))$

(star 3) $\forall u_1 \in \mathrm{D}(r_1), \mathbf{s}, \mathbf{t} \in \mathrm{D}(r_1)^*.$
$(\mathtt{I}(\mathbf{s}) \lessdot_{r_1*} \mathtt{I}(\mathbf{t}) \Rightarrow \mathtt{I}(u_1, \mathbf{s}) \lessdot_{r_1*} \mathtt{I}(u_1, \mathbf{t}))$

各項目で仮定に出現する $\lessdot$ の両辺の項の大きさは結論に出現する $\lessdot$ の両辺の項の大きさより厳密に小さい．よって，これは帰納的な定義になっている．文脈から明らかな場合には添字を省略する．

二項関係 R（$\subseteq A \times A$）は，推移律（$\forall a, b, c \in A.(aRb \wedge bRc \Rightarrow aRc)$）と非反射律（$\neg\exists a \in A.(aRa)$）を満たすとき**厳格半順序**（strict partial order）であるといわれ，さらに $\forall a, b \in A.(aRb \vee bRa \vee a = b)$ を満たすとき**厳格全順序**（strict total order）であるといわれる．$\langle A, R\rangle$ はそれぞれ厳格半（あるいは全）順序集合と呼ばれる．

命題 1 r を任意の正規表現とする．$\langle \mathrm{D}(r), \lessdot_r\rangle$ は厳格全順序集合であり，任意の導出木 $s, t \in \mathrm{D}(r)$ に対して (a) $s = t$, (b) $s \lessdot_r t$, (c) $t \lessdot_r s$ のうち 1 つだけが成り立つ（三分律）．

証明は素直な帰納法で可能なので省略する（詳細な証明は [25] を参照）．

直観的には，上記の順序でより小さい導出木ほど貪欲照合としてよりふさわしい解を表している．しかしながら，厳格順序集合 $\langle \mathrm{D}(r), \lessdot_r\rangle$ には最小元は必ずしも存在しない．たとえば，例 3 において無限の下降列

$$\mathtt{I}(\mathtt{L}(a)) \succ \mathtt{I}(\mathtt{L}(a), \mathtt{R}(1)) \succ \mathtt{I}(\mathtt{L}(a), \mathtt{R}(1), \mathtt{R}(1)) \succ \ldots$$

が得られるので最小元は存在しない．

そこで $\mathrm{D}(r)$ への制限として**正準**（canonical）な導出木の集合 $\mathrm{C}(r)$（$\subseteq \mathrm{D}(r)$）を以下のように導入する．

$$\mathrm{C}(1) = \{1\}$$
$$\mathrm{C}(a) = \{a\} \quad (a \in \Sigma)$$
$$\mathrm{C}(r_1{+}r_2) = \{\mathtt{L}(t) \mid t \in \mathrm{C}(r_1)\} \cup \{\mathtt{R}(t) \mid t \in \mathrm{C}(r_2)\}$$
$$\mathrm{C}(r_1{\cdot}r_2) = \{t_1{\cdot}t_2 \mid t_1 \in \mathrm{C}(r_1), t_2 \in \mathrm{C}(r_2)\}$$
$$\mathrm{C}(r_1{*}) = \{\mathtt{I}(\mathbf{t}) \mid \mathbf{t} \in (\mathrm{C}(r_1) \backslash \mathrm{D}_\varepsilon(r_1))^*\}$$

ここで $\mathrm{D}_\varepsilon(r) = \{t \in \mathrm{D}(r) \mid \mathrm{d}(t) = \varepsilon\}$ である．

さらに，$\mathrm{C}(r)$ の要素のうちで w の接頭辞を導出するものの集合を $\mathrm{C}(r, w)$ と書く（すなわち，$\mathrm{C}(r, w) = \{t \in \mathrm{C}(r) \mid \mathrm{d}(t) \in \rho(w)\}$）．

命題 2 任意の正規表現 r と文字列 w にたいして，$\mathrm{C}(r, w)$ は有限集合である．

証明 $\mathrm{C}(r, w)$ の任意の要素 t について $h(t) \leq h(r)$ が成り立つ．次に，t の各ノードの子の数がたかだか $|w| + 2$ であることを示す．$\mathtt{I}$ 以外のノードの子の数は 2 以下なので，どのような w に対しても $|w| + 2$ 以下になる．一方，ノード $\mathtt{I}$ の子の数を n とすると，正準の条件より $n \leq |\mathrm{d}(\mathtt{I}(t_1, \ldots, t_n)| \leq |w|$．したがって，すべてのノードの子の数はたかだか $|w| + 2$ である． □

文献 [11] では正準でない導出木のことを**問題の生じる事例**（problematic case）と呼んでいる．文献 [18] では最左最長照合の場合の正準な導出木を定式化しているが，そこではノード $\mathtt{I}$ の直接の部分木が 2 つ以上ある場合にのみ各部分木が導出する文字列が非空であることを要求している．同じく最左最長照合を扱った [21] では，導出木の順序付けを厳しくすることで問題の生じる事例を実質的に除外している．貪欲照合を扱う本論文における正準の定義はそれらより条件がやや厳しい．

なお，与えられた正規表現をあらかじめ**反復標準形**（star normal form）[3] に変換すれば，問題の生じる事例は除外できる．しかし，与えられた正規表現の形が変化するため，本論文の目的である部分式による捕獲が不可能になる．

照合問題 $\langle \Sigma, w, r\rangle$ の部分照合解のうち正準な導出木を与えるものの集合を $\mathrm{M}(r, w)$ と置く．すなわち，

$$\mathrm{M}(r, w) = \{(u, t) \mid u \in \rho(w), t \in \mathrm{C}(r, w/u)\}.$$

命題 2 より，$\mathrm{M}(r, w)$ は有限集合である．次に，定義 2 の順序を以下のように辞書式順序に拡大する．

$$(u_1, t_1) \lessdot_r (u_2, t_2) \Leftrightarrow |u_1| < |u_2| \vee u_1 = u_2 \wedge t_1 \lessdot_r t_2.$$

$\langle \mathrm{C}(r, w), \lessdot\rangle$ は厳格全順序なので，$\langle \mathrm{M}(r, w), \lessdot_r\rangle$ も厳格全順序である．よって，以下の系を得る．

系 1 照合問題 $\langle \Sigma, w, r\rangle$ に対して $\langle \mathrm{M}(r, w), \lessdot_r\rangle$ は最小要素を持つ．

$\mathrm{M}(r, w)$ の最小要素を**貪欲な**（greedy）部分照合の解と呼ぶ．

3. 貪欲な部分照合を実現する NFA

3.1 Thompson NFA

正規表現 r と $p \in \{1, 2\}^*$ に対して部分式の**ポジション**の集合 $Pos(r, p)$（$\subseteq \{1, 2\}^* \cup \{\overline{p} \mid p \in \{1, 2\}^*\}$）を以下のように定義する．

$$Pos(1, p) = \{p, \overline{p}\}$$
$$Pos(a, p) = \{p, \overline{p}\}$$
$$Pos(r_1{+}r_2, p) = \{p, \overline{p}\} \cup Pos(r_1, p1) \cup Pos(r_2, p2)$$
$$Pos(r_1{\cdot}r_2, p) = \{p, \overline{p}\} \cup Pos(r_1, p1) \cup Pos(r_2, p2)$$
$$Pos(r_1{*}, p) = \{p, \overline{p}, p2\} \cup Pos(r_1, p1)$$

直観的には，$p, \overline{p}$ は r の各部分式に明示的に括弧を補った際の括弧の対に対応しているが，r_1* の場合のみ余分な要素 $p2$ を加えている点に注意が必要である．また，ε と $\overline{\varepsilon}$ とが区別されることにも注意されたい．$Pos(r) = Pos(r, \varepsilon)$ と略記する．

5 つ組 $\langle \Sigma, Q, \mathrm{I}, \mathrm{F}, \Delta\rangle$ を正規表現 $r \in \mathrm{Reg}(\Sigma)$ とポジションの集合 $Pos(r, p)$ から作られる Thompson NFA（**TNFA**）

と呼び $TA(r,p)$（$p=\varepsilon$ のときは $TA(r)$）と表記する．ただし，$Q=Pos(r,p)$ であり，$\mathrm{I}=p$，$\mathrm{F}=\overline{p}$，$\Delta=\Delta(r,p)$ とする．ここで $\Delta(r,p)$ は r の構造に従い以下のように帰納的に定義される．

$$\Delta(1,p)=\{p\xrightarrow{\varepsilon}\overline{p}\}$$
$$\Delta(a,p)=\{p\xrightarrow{a}\overline{p}\}\ (a\in\Sigma)$$
$$\Delta(r_1{+}r_2,p)=\Delta(r_1,p1)\cup\Delta(r_2,p2)\cup\{p\xrightarrow{L}p1,p\xrightarrow{R}p2,\overline{p1}\xrightarrow{\varepsilon}\overline{p},\overline{p2}\xrightarrow{\varepsilon}\overline{p}\}$$
$$\Delta(r_1{\cdot}r_2,p)=\Delta(r_1,p1)\cup\Delta(r_2,p2)\cup\{p\xrightarrow{\varepsilon}p1,\overline{p1}\xrightarrow{\varepsilon}p2,\overline{p2}\xrightarrow{\varepsilon}\overline{p}\}$$
$$\Delta(r_1{*},p)=\Delta(r_1,p1)\cup\{p\xrightarrow{\varepsilon}p2,p2\xrightarrow{L}p1,p2\xrightarrow{R}\overline{p},\overline{p1}\xrightarrow{\varepsilon}p2\}$$

I を**始状態**, F を**終状態**と呼ぶ．また Δ の要素 $q\xrightarrow{a}q'$ を a による q から q' への**遷移**と呼び，特に $q\xrightarrow{x}q', x\in\{\varepsilon,L,R\}$ を**空遷移**と呼ぶ．

図 1 に反復の構成法を図式的に示す．省略されたラベルはすべて ε である．図中の反復の構成法は [22] に由来し，現在の教科書等で一般的なもの（**図 2**）と異なる．この構成法を採用する理由については 3 章の定理 1 の後で述べる．

図 3 に $TA((ab{+}a{*}){*})$ を例示する．

有向グラフとしてみたとき，各状態 $q\in Q$ に入る（から出る）空遷移の数を $\mathrm{in}(q)$（$\mathrm{out}(q)$）で表す．Q に対して $Q_S=\{q\in Q\mid \mathrm{in}(q)=0\}$，$Q_T=\{q\in Q\mid \mathrm{out}(q)=0\}$ と表記する．特に $\mathrm{I}\in Q_S, \mathrm{F}\in Q_T$ である．また，$TA(a)$（$a\in\Sigma$）においては，$\mathrm{I}\in Q_T, \mathrm{F}\in Q_S$ も成り立つ．任意の $q\in Q_S\backslash\{\mathrm{I}\}$ に対して $q'\xrightarrow{a}q$ となる q' が唯一存在するが，その q' を $src(q)$ で表す．同様に，任意の $q\in Q_T\backslash\{\mathrm{F}\}$ に対して，$q\xrightarrow{a}q'$ となる q' を $dest(q)$ で表す．明らかに $src(dest(q))=q, dest(src(q))=q$ である．

$q\xrightarrow{L}q_1, q\xrightarrow{R}q_2$ となる状態の組 $\langle q,q_1,q_2\rangle$ を**分岐**と呼ぶ．TNFA において分岐が出現するのは和と反復においてそれぞれ一カ所のみである．図 3 の例では分岐は以下の 3 つである．

$$\langle 2,1,\mathrm{F}\rangle,\ \langle 1,11,12\rangle,\ \langle 122,121,\overline{12}\rangle$$

直観的には，和における分岐は正規表現の照合における左右の部分式の選択に対応しており，反復における分岐は（さらに）反復するか否かの選択に対応している．

3.2 ステップと経路

TNFA $M=\langle\Sigma,Q,\mathrm{I},\mathrm{F},\Delta\rangle$ が与えられたとする．Q の状態の**重複のない**非空の並び $q_1,\ldots,q_n$ が $q_i\xrightarrow{x}q_{i+1}\in\Delta$ $(1\le i<n, x\in\{\varepsilon,L,R\})$ を満たし $q_1\in Q_S, q_n\in Q_T$ であるとき，この並びを M の**ステップ**と呼ぶ．ステップ s

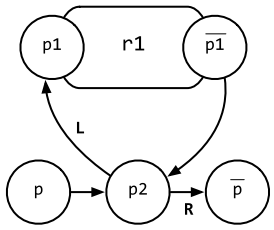

図 1 反復に対する Thompson NFA
Fig. 1 An Thompson NFA for repetitions.

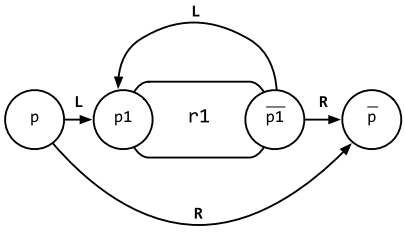

図 2 反復の別の構成法
Fig. 2 Another construction of an NFA for repetitions.

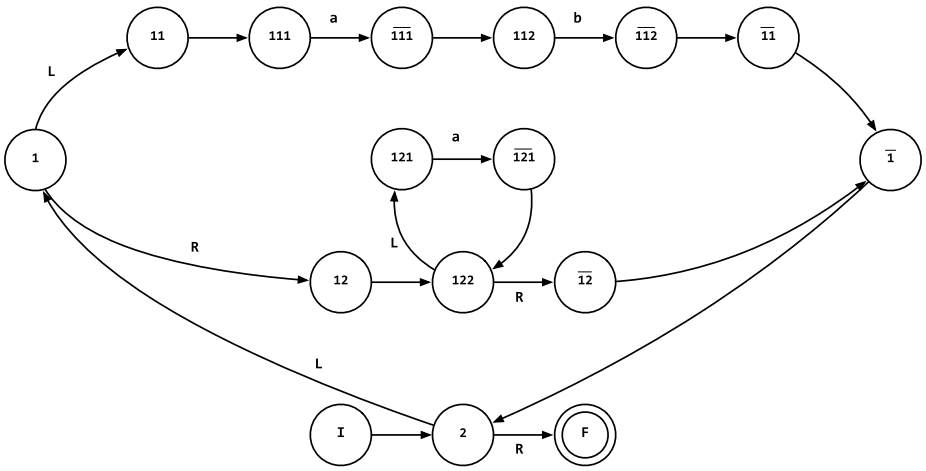

図 3 $(ab+a{*}){*}$ から構築した Thompson NFA
Fig. 3 The Thompson NFA constructed from $(ab+a{*}){*}$.

の起点（あるいは終点）にある状態を $head(s)$（あるいは $tail(s)$）で表す．さらに，ステップの列 $\mathbf{s} = s_1, \ldots, s_n$ に対して $head(s_i)$（あるいは $tail(s_i)$）（$1 \leq i \leq n$）をこの順に並べたものを $Heads(\mathbf{s})$（あるいは $Tails(\mathbf{s})$）で表す．$q \in Q$ を起点とする M のステップの集合を $Step_M(q)$ と書く（文脈から明らかな場合添字 M は省略）．明らかにこれは有限集合である．

さらに，Σ の要素で区切られた 1 つ以上の M のステップの並び $\alpha = s_0, a_1, s_1, \ldots, a_n, s_n \in (Q \cup \Sigma)^*$ が $tail(s_{i-1}) \xrightarrow{a_i} head(s_i) \in \Delta$（$1 \leq i \leq n$）を満たすとき α を M における**経路**と呼ぶ．$head(s_0)$ をこの経路の**始点**，$tail(s_n)$ を**終点**と呼ぶ．Σ の要素を 1 つも含まない単一のステップも M の経路である．M の経路 α における Σ の要素の出現をこの順に並べてできる文字列を α から導出される文字列と呼び，$\mathrm{d}(\alpha)$ で表す．最後に，$TA(r, p)$ における始点が q で終点が q' であるすべての経路からなる集合を $Path^p_{q,q'}(r)$ と表記する．$Path^{\varepsilon}_{q,q'}(r)$ を $Path_{q,q'}(r)$ と略記する．また，$Path^p_{q,\mathrm{F}}(r)$ を $Path^p_q(r)$ と略記し，さらに $Path^p_{\mathrm{I}}(r)$ を $Path^p(r)$ と略記する．$Path^p_{q,q'}(r)$ の要素のうちで w の接頭辞を導出するものの集合を $Path^p_{q,q'}(r, w)$ と書く．すなわち，$Path^p_{q,q'}(r, w) = \{\pi \in Path^p_{q,q'}(r) \mid \mathrm{d}(\pi) \in \rho(w)\}$．

経路を構成する各ステップにおいて状態の重複が許されないことに注意が必要である．たとえば，図 3 において，$\mathrm{I}, 2, 1, 12, 122, \overline{12}, \overline{1}, 2, \mathrm{F}$ は 2 が重複しているので経路ではない．一方，$\mathrm{I}, 2, 1, 12, 122, 121, a, \overline{121}, 122, \overline{12}, \overline{1}, 2, \mathrm{F}$ は 2 つのステップから成り，各ステップにおいては状態の重複がないので経路である．

3.3 貪欲照合の意味論との対応

TNFA M の異なるステップ $s, t \in Step_M(q)$ を考える．$\rho(s) \cap \rho(t)$ の要素のうち最長のもの（最長共通接頭辞）を u と置く．$s \bowtie t$ なので，s/u も t/u も空列ではない．また，$head(s) = q = head(t)$ なので u も空列ではない．そこで，$\langle tail(u), head(s/u), head(t/u)\rangle$ が M の分岐であるとき $s \lessdot t$ と定義すると，Δ の定義より $\langle Step_M(q), \lessdot\rangle$ は厳格全順序集合である．

さらに，$\alpha \bowtie \beta$ であるような経路 $\alpha, \beta \in Path_{p,q}(r, w)$ が最初に異なるステップ（それぞれ順に s, t と置く）を考えると，$Path_{p,q}(r, w)$ 上の二項関係 $\alpha \lessdot \beta \Leftrightarrow \alpha \bowtie \beta \wedge s \lessdot t$ が定義できる．

命題 3 正規表現 r から作られる TNFA $TA(r) = \langle \Sigma, Q, \mathrm{I}, \mathrm{F}, \Delta\rangle$ が与えられている．任意の $p, q \in Q, w \in \Sigma$ に対して $\langle Path_{p,q}(r, w), \lessdot\rangle$ は厳格半順序集合である．特に $\langle Path_p(r, w), \lessdot\rangle$ は厳格全順序集合である．

証明 非反射律を満たすことは明らか．推移律を満たすことは，$\langle Step_M(q), \lessdot\rangle$ の推移性に帰着できる．$TA(r)$ には F からの遷移は存在しないので，$Path_p(r, w)$ の相異なる経路 α, β は $\alpha \bowtie \beta$ を満たす．よって，$\langle Path_p(r, w), \lessdot\rangle$ が全順序であることは $\langle Step_M(q), \lessdot\rangle$ が全順序であることに帰着できる． □

次に，正規表現 r と $p \in \{1, 2\}^*$ にたいして写像 $\Phi^p_r : \mathrm{C}(r) \to (Q \cup \Sigma)^*$ を以下のように定義する．

$$
\begin{aligned}
&\Phi^p_1(1) = [p, \overline{p}]\\
&\Phi^p_a(a) = [p, a, \overline{p}] \quad (a \in \Sigma)\\
&\Phi^p_{r_1+r_2}(\mathtt{L}(t)) = [p]{+}{+}\Phi^{p1}_{r_1}(t){+}{+}[\overline{p}]\\
&\Phi^p_{r_1+r_2}(\mathtt{R}(t)) = [p]{+}{+}\Phi^{p2}_{r_2}(t){+}{+}[\overline{p}]\\
&\Phi^p_{r_1 \cdot r_2}(t_1 \cdot t_2) = [p]{+}{+}\Phi^{p1}_{r_1}(t_1){+}{+}\Phi^{p2}_{r_2}(t_2){+}{+}[\overline{p}]\\
&\Phi^p_{r_1 *}(\mathtt{I}()) = [p, p2, \overline{p}]\\
&\Phi^p_{r_1 *}(\mathtt{I}(t_1, \ldots, t_n))\\
&\quad = [p, p2]{+}{+}\Phi^{p1}_{r_1}(t_1){+}{+}[p2]{+}{+}\Phi^{p1}_{r_1}(t_2)\\
&\qquad \ldots [p2]{+}{+}\Phi^{p1}_{r_1}(t_n){+}{+}[p2, \overline{p}]
\end{aligned}
$$

$\Phi^{\varepsilon}_r(t)$ を $\Phi_r(t)$ と略記する．さらに文脈から明らかな際には添字 r も省略する．

直観的には，$\Phi(t)$ は導出木 t を根から出発して巡回しつつ根に戻る過程を表している．ただし，根のラベルが $\mathtt{I}$ であるような木に関しては，余分のノードを根の直下に補ったうえで反復的に巡回するように扱っている．余分なノードを補うのは，後述する $TA(r)$ の経路との対応のためのテクニカルな要請からきている．

写像 $f : \mathrm{C}(r) \to (Q \cup \Sigma)^*$ は $\mathrm{d}(t) = \mathrm{d}(f(t))$ を満たすとき**導出を保存**する（derivation-preserving）という．

補題 1 写像 $\Phi^p_r : \mathrm{C}(r) \to (Q \cup \Sigma)^*$ は導出を保存する．

証明 r の構造に関する帰納法による． □

補題 2 Φ^p_r の値域は $Path^p(r)$ である．

証明 $R = \{\Phi^p_r(t) \mid t \in \mathrm{C}(r)\}$ と置く．$R = Path^p(r)$ であることを r の構造に関する帰納法で示す．以下，$r = r_1*$ の場合のみ扱い，ほぼ自明なその他の場合は省略する．

[$Path^p(r) \subseteq R$ の証明] $Path^p(r_1*)$ の要素のうち $[p, p2, \overline{p}]$ については $t = \mathtt{I}()$ とすればただちに題意が得られる．それ以外の要素は，

$$[p, p2]{+}{+}\alpha_1{+}{+}[p2] \ldots [p2]{+}{+}\alpha_n{+}{+}[p2, \overline{p}]$$

の形をしている．ただし $\alpha_i \in Path^{p1}(r_1)$ かつ $\mathrm{d}(\alpha_i) \neq \varepsilon$（$1 \leq i \leq n$）である（さもないと $p2$ が重複して出現し，ステップの定義に反する）．r_1 に帰納法の仮定を適用すれば，ある $t_i \in \mathrm{C}(r_1)$（$1 \leq i \leq n$）に対して

$$[p, p2]{+}{+}\Phi^{p1}_{r_1}(t_1){+}{+}[p2] \ldots [p2]{+}{+}\Phi^{p1}_{r_1}(t_n){+}{+}[p2, \overline{p}]$$

を得る．補題 1 より $t_i \notin \mathrm{D}_{\varepsilon}(r_1)$（$1 \leq i \leq n$）であるので，$t = \mathtt{I}(t_1, \ldots, t_n)$ とおけば $t \in \mathrm{C}(r)$ であり，写像 Φ の定義より上式は $\Phi^p_r(t)$ に等しい．

[$R \subseteq Path^p(r)$ の証明] $\Phi^p_r(\mathtt{I}()) \in Path^p(r_1*)$ は明らか．$\Phi^p_r(\mathtt{I}(t_1, \ldots, t_n))$ のとき，$\mathtt{I}(t_1, \ldots, t_n)$ が正準であることと

補題 1 より $\mathrm{d}(\Phi^p_{r_1}(t_i))$ $(1 \leq i \leq n)$ が非空であることが分かる．よって，$\Phi^p_{r_1*}(\mathtt{I}(t_1,\ldots,t_n)) \in Path^p(r_1*)$ であることが帰納法の仮定より確認できる． □

二項関係 α $(\subseteq A\times A)$，β $(\subseteq B\times B)$ が与えられているとする．写像 $f : A \to B$ が $\forall a, a' \in A.(a\alpha a' \Rightarrow f(a)\beta f(a'))$ を満たすとき f は $\langle A,\alpha\rangle$ から $\langle B,\beta\rangle$ への準同型写像（homomorphism）であるといわれる．

補題 3 Φ^p_r は $\langle \mathrm{C}(r), \lessdot_r\rangle$ から $\langle Path^p(r), \lessdot\rangle$ への準同型写像である．

証明 順序関係 $\lessdot_r$ の定義（定義 2）に従い，r の構造に関する帰納法で証明する．以下，定義 2 の場合分けのうち重要な (or1) と (star1) の場合のみ扱う（残りの場合は帰納法の仮定に素直に従う）．

$[\mathtt{L}(s) \lessdot_{r_1+r_2} \mathtt{R}(t)$ の場合$]$ $\alpha = \Phi^p_{r_1+r_2}(\mathtt{L}(s)) = [p]{+}{+}\Phi^{p1}_{r_1}(s){+}{+}[\overline{p}]$, $\beta = \Phi^p_{r_1+r_2}(\mathtt{R}(t)) = [p]{+}{+}\Phi^{p2}_{r_2}(t){+}{+}[\overline{p}]$ と置くと分岐 $\langle p, p1, p2\rangle$ が存在するので $\alpha \lessdot_{r_1+r_2} \beta$ を得る．

$[\mathtt{I}(s_1,\ldots,s_n) \lessdot_{r_1*} \mathtt{I}()$ の場合$]$ $\alpha = \Phi^p_{r_1*}(\mathtt{I}(s_1,\ldots,s_n))$ $(n>0)$ と $\beta = \Phi^p_{r_1*}(\mathtt{I}())$ には分岐 $\langle p2, p1, \overline{p}\rangle$ が存在するので $\alpha \lessdot_{r_1*} \beta$ を得る． □

$\langle A,\alpha\rangle, \langle B,\beta\rangle$ は，準同型写像 $f : A \to B, g : B \to A$ が存在して $\forall a \in A.(g(f(a)) = a)$, $\forall b \in B.(f(g(b)) = b)$ となるとき，同型であるといい $\langle A,\alpha\rangle \cong \langle B,\beta\rangle$ と表す．特に $\langle A,\alpha\rangle$ と $\langle B,\beta\rangle$ の両方が厳格全順序集合の場合には，準同型写像 f が全射でありさえすれば $\langle A,\alpha\rangle \cong \langle B,\beta\rangle$ となる．

以下は本論文の最初の主要な結果であり，$TA(r)$ における経路の集合を 2 章で導入した正準導出木を用いた貪欲照合の意味論と関連付けている．

定理 1 任意の正規表現 $r \in \mathrm{Reg}(\Sigma)$ と文字列 $w \in \Sigma^*$ にたいして $\langle \mathrm{C}(r,w), \lessdot\rangle \cong \langle Path(r,w), \lessdot\rangle$ が成り立つ．

証明 命題 3，1 より，$\langle \mathrm{C}(r,w), \lessdot\rangle$ と $\langle Path(r,w), \lessdot\rangle$ は共に厳格全順序集合である．よって，Φ_r が $\langle \mathrm{C}(r,w), \lessdot\rangle$ から $\langle Path(r,w), \lessdot\rangle$ への全射準同型写像であることを示せばよいが，これは補題 1，2，3 にただちに従う． □

すなわち，$Path(r,w)$ 上の順序構造は $\mathrm{C}(r,w)$ 上の順序構造と正確に対応している．特に，命題 2 より $Path(r,w)$ も $\mathrm{C}(r,w)$ も最小の要素を持つが，$Path(r,w)$ における最小の経路は $\mathrm{C}(r,w)$ における最小の導出木に対応している．

この定理は，図 1 の反復の構成の代わりに図 2 を用いると成立しないことに注意が必要である．実際，$\mathrm{C}(a{*}{*}, \varepsilon)$ は $\mathtt{I}()$ だけからなる一元集合だが，図 1 の反復の構成の代わりに図 2 を用いて作られる NFA では $Path(a{*}{*}, \varepsilon)$ は I, F と $\mathrm{I}, 1, \overline{1}, \mathrm{F}$ からなるので同型にはなり得ない．

本定理と同様の結果は [12], [13] にも見られる．そこでは，Thompson NFA の一種（Augumented NFA）の経路と 2 進列として符号化された導出木の対応を論じている．本論文との（部分照合と全体照合の違い以外の）大きな違いは，NFA の構成において正規表現の連結を結合的に取り扱っている[*2]点にある．これは入力文字列に対して漸近的に導出列を生成するストリーム的な処理の最適化を可能にするためである（その代償として，一般に通用している部分式の捕獲との整合性を欠き，たとえば RE2 等と異なる捕獲結果になる場合がある）．

次章以降では $\mathrm{C}(r,w)$ の要素と $Path(r,w)$ の要素を適宜混同し，導出木に対して定義された概念を経路に対しても流用する．たとえば，$\mathrm{M}(r,w)$ の要素として組 $\langle u,t\rangle$ の代わりに $\langle u, \Phi_r(t)\rangle$ を用いる．

4. 部分照合と部分式による捕獲を実現する DFA

貪欲照合の最も広く用いられている実現方法は，正規表現から作られる NFA 上での遷移を左の分岐を優先しつつ深さ優先探索で行うことである．遷移先がなければ直近の分岐までバックトラックし，他方を探索しなおす．さらにすべての探索に失敗した場合は与えられた文字列上で次の文字にシフトし再び初期状態から遷移を試みる．この手法では，最悪の場合には与えられた文字列長に対して指数的に探索空間が増大することになる．

それに対して**部分集合構成**[19]（subset construction）を用いて決定性有限オートマトン（DFA）を構築する手法がある．あらかじめ DFA を構築しようとすれば最悪の場合には NFA の状態数に対して指数的な計算コストが必要であるが，与えられた文字列の照合に必要な DFA の一部分のみを構築する（いわゆる on-the-fly 方式）[2] ことで，文字列長に比例する計算コストで解を求めることが可能である．

TNFA から部分照合と部分式による捕獲を実現する DFA を構築するときも，部分集合構成と同様の方法を用いる．ただし，DFA の状態として，NFA の状態の集合の代わりに NFA の状態の列を用いることで経路間の優先順序を表現する．

4.1 有効なステップ

一般的な部分集合構成では，NFA 状態の集合の各要素から空遷移により到達可能なすべての NFA 状態をまとめた集合を，DFA の状態とする．TNFA $M = \langle \Sigma, Q, \mathrm{I}, \mathrm{F}, \Delta\rangle$ から部分照合と部分式による捕獲を実現する DFA を構築するときには，ステップの列の終点である NFA 状態の列をもとに DFA の状態を作る．このとき，経路間の優先順位を保存するために，以下の順序を導入する．

Q_S の要素の重複のない列 $\mathbf{q} = q_1,\ldots,q_n$ に対して，$Step_M(\mathbf{q}) = \bigcup_{1\leq i\leq n} Step_M(q_i)$ とする．$Step_M(\mathbf{q})$ 上の二項関係 $\lessdot_{\mathbf{q}}$ を

[*2] したがって，文献 [13] では導出木の集合とその 2 進符号の集合は同型であると述べているが，これは誤りであろう．

```
 1: function EFFECTIVE(q_1, ..., q_n)
 2:     frontier ← stack of [q_1, ..., q_n]; // The top is q_1
 3:     visited ← ∅;
 4:     next ← [];
 5:     prev ← {};
 6:     while frontier is non-empty do
 7:         q ← frontier.pop();
 8:         visited ← visited ∪ {q};
 9:         next ← next ++ [q] if q ∈ Q_T;
10:         break if q = F;
11:         for x in [ε, R, L] do
12:             if q -x-> q' ∈ Δ and q' ∉ visited then
13:                 prev ← prev ∪ {(q', q)};
14:                 frontier.push(q');
15:             end if
16:         end for
17:     end while
18:     return ⟨next, prev⟩;
19: end function
```

図 4 *Effective*$(q_1, \ldots, q_n)$ と *Closure*$(q_1, \ldots, q_n)$ の計算
Fig. 4 The computation of *Effective*$(q_1, \ldots, q_n)$ and *Closure*$(q_1, \ldots, q_n)$.

$$s \lessdot_{\mathbf{q}} t \Leftrightarrow \quad head(s) \prec_{\mathbf{q}} head(t) \lor (head(s) = head(t) \land s \lessdot t)$$

のように定義する．$\lessdot_{\mathbf{q}}$ は $Step_M(\mathbf{q})$ 上の厳格全順序である．$Step_M(\mathbf{q})$ の要素を $\lessdot_{\mathbf{q}}$ に関して昇順に整列した列を作る．この列に終点が同じステップが複数現れるとき，最初に現れるものだけを残して得られるステップの列を $\mathbf{s} = s_1, \ldots, s_m$ と置く（すなわち，任意の $s \in Step_M(\mathbf{q})$ について，集合 $\{t \in Step_M(\mathbf{q}) \mid tail(s) = tail(t)\}$ 中の $\lessdot_{\mathbf{q}}$ に関する最小要素だけが $\mathbf{s}$ に含まれる）．$\mathbf{s}$ の異なる要素 s, s' が $tail(s) = tail(s')$ となることはないので，$s_1, \ldots, s_m$ の要素 s のうち $tail(s) = \mathrm{F}$ となるものはたかだか 1 つである．この要素の添字を i（$1 \le i \le m$）とする（なければ $i = m$ とする）とき，列 $s_1, \ldots, s_i$ を $\mathbf{q}$ からの**有効**（effective）なステップの並びと呼び $Effective_M(\mathbf{q})$ で表す．$Tails(Effective_M(\mathbf{q}))$（$\in (Q_T)^*$）を $Closure_M(\mathbf{q})$ と略記し，$\mathbf{q}$ の有効な**閉包**と呼ぶ．文脈から明らかな場合には添え字 M は省略する．*Effective*$(\mathbf{q})$ は相異なるステップの列であり，*Closure*$(\mathbf{q})$ もまた相異なる状態の列になっている．$E = Effective(\mathbf{q})$ と置くとき，$q \in Closure(\mathbf{q})$ に対して $tail(s) = q$ となる E の唯一の要素 s を $E(q)$ で表す．

図 4 に *Effective*$(\mathbf{q})$ と *Closure*$(\mathbf{q})$ を計算するアルゴリズムを示す．これは終状態が見つかればただちに計算を終了すること（10 行目）を除けば，基本的に有向グラフを左の分岐から深さ優先探索（depth-first search）し全域木（spanning tree）を求めるアルゴリズムである．11 行目から始まる for 文中において $q \xrightarrow{R} q'$ は $q \xrightarrow{L} q'$ より先にスタック *frontier* にプッシュされるので，左の分岐から先に探索されることになる．

6 行目から始まるループの冒頭の時点で *frontier* に格納されている要素を $e_1, \ldots, e_n$（$n \ge 1$，e_1 がスタックの頂点）とすると，以下の不変条件（loop invariant）が成り立っている（Dom(*prev*) は二項関係 *prev* の定義域を意味する）．

(1) $\forall\, 1 \le i \le n.\ (e_i \notin visited)$
(2) $\mathrm{Dom}(prev) \subseteq \{e_1, \ldots, e_n\} \cup visited$
(3) $\forall\, 1 \le i, j \le n.\ (i \ne j \Rightarrow e_i \ne e_j)$
(4) $\forall\, 2 \le i \le n.\ (e_i \in Q_S \lor \exists\, e_i'.\ (e_i' \xrightarrow{R} e_i \land e_i' \in visited))$
(5) $\forall\, q \in \mathrm{Dom}(prev).\ ((q, q_1), (q, q_2) \in prev \Rightarrow q_1 = q_2)$

初めてループに入る時点では *prev* と *visited* は空で，*frontier* は Q_S の相異なる状態からなるので，これらの条件は自明である．次にループの冒頭（6 行目）でこれらの条件を仮定する．8 行目で e_1 が *visited* に加えられるが，条件 (3) より $e_1 \notin \{e_2, \ldots, e_n\}$ なので，10 行目でループを抜けたときにも全条件が成り立つ．また，12 行目の if 文の条件が一度も成立しないなら，次回も全条件が成り立つ．一方，if 文の条件が一度でも成り立つときは，TNFA の構成より，次回の *frontier* は $e_\varepsilon e_2 \ldots e_n$ か $e_L e_R e_2 \ldots e_n$（$e_1 \xrightarrow{x} e_x$ とする）のいずれかである．前者では Dom(*prev*) に e_ε が，後者では e_L と e_R が追加される．このとき，仮定と 12 行目の条件および 14 行目より次回も条件 (1)(2) が成り立つ．次に，任意の $x \in \{\varepsilon, L, R\}$ について $e_x \notin \{e_2, \ldots, e_n\}$ を示す．$e_x = e_k$（$2 \le k \le n$）とすると，$e_k \notin Q_S$ となるので，TNFA の構成とループ冒頭での条件 (4) より，6 行目の時点で $e_1 \xrightarrow{R} e_k$ と $e_1 \in visited$ が成立しなければならない．しかし，これはループ冒頭での条件 (1) より $e_1 \notin visited$ であることと矛盾する．TNFA の構成より $e_L \ne e_R$ なので，上の結果と合わせると，次回も条件 (3) が成り立つ．条件 (4) は，$e_1 \xrightarrow{R} e_R$ と $e_1 \in visited$ および仮定より次回も成り立つ．条件 (5) は，12 行目の if 文の条件と，条件 (3) が次回も成り立つことから分かる．以上より，条件 (1) から (5) は確かに不変条件である．

不変条件 (1) より $|visited|$ は反復ごとに厳密に増加するので，このループはたかだか $|Q|$ 回反復される．よって，このアルゴリズムの時間計算量は（集合 *visited* への要素の読み書きを O(1) で実現して）O(n)（n は与えられた TNFA の状態数）である．最終的に *next* に *Closure*$(q_1, \ldots, q_n)$ が得られる．また，不変条件 (5) より *prev* は関数である．関数 f の冪（power）f^n（$n \ge 0$）を $f^0(x) = x; f^{n+1}(x) = f(f^n(x))$ と定義する．*next* の任意の要素 q に対して重複のない状態の列 $prev^0(q), prev^1(q), \ldots$ を考えると $q_1, \ldots, q_n$ のいずれかに到達する列が得られるが，これが q で終わる *Effective*$(q_1, \ldots, q_n)$ の要素を（逆順に）与えている．

4.2 DFA の構築

TNFA から DFA を構築する際，NFA 状態の重複のない列とすでに NFA の終状態に達したか否かを表す真偽値のフラグの組を DFA の状態とし，フラグの値が偽の場合のみ NFA 状態の列の末尾に *Closure*([I]) を加えることで，部

```
1: function PROCEED(⟨current, foundF⟩, a)
2:     next ← [];
3:     for q ∈ current do
4:         next ← next ++ [q′] if q -a→ q′ ∈ Δ;
5:     end for
6:     next ← next ++ [I] unless foundF;
7:     eps ← Effective(next);
8:     next ← Closure(next);
9:     return ⟨⟨next, foundF ∨ F ∈ next⟩, eps⟩;
10: end function
```

図 5 proceed 関数
Fig. 5 The function proceed.

分照合を効率的に実現する．この考え方を形式化すると以下のようになる．

$M = \langle \Sigma, Q, I, F, \Delta \rangle$ を正規表現 $r \in \mathrm{Reg}(\Sigma)$ から作られる TNFA とするとき，6 つ組 $\langle \Sigma, Q', I', E_0, Q_F, \Delta' \rangle$ を r からつくられる**貪欲な DFA**（greedy DFA, **GDFA**）と呼び，$GA(\Sigma, r)$ で表す．ここで Q' は，Q の相異なる要素の 1 つ以上の並び $\mathbf{q}$ と真偽値 f の対 $\langle \mathbf{q}, f \rangle$ の集合であり，これを GDFA の状態と呼ぶ．真偽値は真を T，偽を F で表す．$I' = \langle \mathit{Closure}([\mathrm{I}]), \mathrm{F} \in \mathit{Closure}([\mathrm{I}]) \rangle$ であり，これを始状態と呼ぶ．$E_0 = \mathit{Effective}([\mathrm{I}])$ とする．Q_F は $\mathrm{F} \in \mathbf{q}$ を満たす GDFA 状態 $\langle \mathbf{q}, f \rangle$（これを終状態と呼ぶ）の集合である．最後に Δ' は 4 つ組 $\langle \langle \mathbf{q}, f \rangle, a, E, \langle \mathbf{q}', f' \rangle \rangle$ の集合である．ただし，$\langle \mathbf{q}, f \rangle, \langle \mathbf{q}', f' \rangle \in Q', a \in \Sigma$ である．$E, \mathbf{q}', f'$ は以下で与えられる．

- $E = \mathit{Effective}([q' \mid q \in \mathbf{q}, q \xrightarrow{a} q'] ++ [\mathrm{I} \mid f = \mathsf{F}])$
- $\mathbf{q}' = \mathit{Tails}(E)$
- $f' = f \vee \mathrm{F} \in \mathbf{q}'$

Δ' の要素 $\langle t, a, E, t' \rangle$ を $t \xrightarrow{a}_E t'$ と表す．

上記の定義中では Haskell 等の関数型言語で一般的なリスト内包表記を流用しており，$[q' \mid q \in \mathbf{q}, q \xrightarrow{a} q']$ は $\mathbf{q}$ の各要素 q を $q \xrightarrow{a} q'$ となる q' でこの順に置換して得られる列を表す（そのような q' がない q は取り除かれる）．$[q' \mid q \in \mathbf{q}, q \xrightarrow{a} q'] ++ [\mathrm{I} \mid f = \mathsf{F}]$ が相異なる要素の列であることは，TNFA の構成法より明らかである．

図 5 に Δ' を計算するアルゴリズムを示す．$t \xrightarrow{a}_E t' \Leftrightarrow \langle t', E \rangle = \mathrm{proceed}(t, a)$ である．

図 6 は $GA(\{a, \ldots, z\}, (ab{+}a*)*)$ を例示している（始状態は左端の状態 $q1$．これ以降の図でも GDFA の始状態は左端に書く）．破線で示された状態 $q4 = \langle \varepsilon, \mathsf{T} \rangle$ は，他の状態へ至る経路のない状態，いわゆる**シンク**（sink）[*3]である．$[a-z]$ でラベル付けされた矢印のところには実際は $a, b, \ldots, z$ の各文字に対応する遷移がある（$[b-z], [c-z]$ についても同様）．E_0, E_1, E_2, E_3 の要素は見やすいように有向グラフとして表示している．

[*3] 一般には死状態（dead state）とも呼ばれるが，本論文の枠組みでは死状態への到達は照合の成功を意味するので，誤解のないようにシンクと呼ぶ．実際，GDFA 状態 $\langle [], \mathrm{F} \rangle$ は決して出現しない．

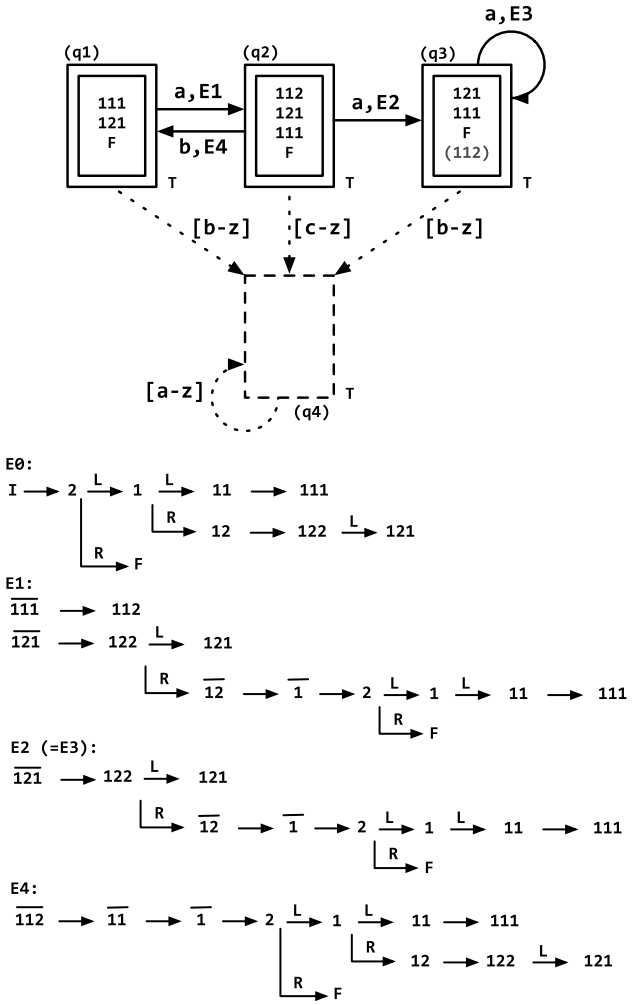

図 6 $(ab{+}a*)*$ から構築した GDFA
Fig. 6 The GDFA constructed from $(ab{+}a*)*$.

次に，本手法における部分照合の様子を見るために，$\Sigma = \{a, b\}$ で正規表現が単なる文字列 *ababbaaa* である場合を例として取り上げる．**図 7** の (a) は $TA(ababbaaa)$ である．ただし，議論の本筋と無関係な連続する ε-遷移を 1 つにまとめ，状態には通し番号を振っている．(b) はこれから生成された GDFA である（ただし，終状態から出る遷移は除く）．この GDFA は文字列 *ababbaaa* から作られる Aho-Corasick オートマトン（以下，AC オートマトン）[1]（ただし，終状態から出る遷移は除く）と同型である．AC オートマトンは古典的な文字列照合アルゴリズム KMP [14] と密接に関連している．実際，Morris-Pratt（MP）アルゴリズムの後戻り表から AC オートマトンの遷移を構築することや，逆に AC オートマトンの遷移から MP アルゴリズムの後戻り表を得ることが可能である（たとえば，文献 [9]）．このことは，GDFA における部分照合が KMP アルゴリズムと同様のアイデアであることを意味している．正規表現が任意の文字列として与えられた場合に GDFA と AC オートマトンが同型になることについては文献 [23] で議論している．文献 [16] では AC オートマトン構築におけ

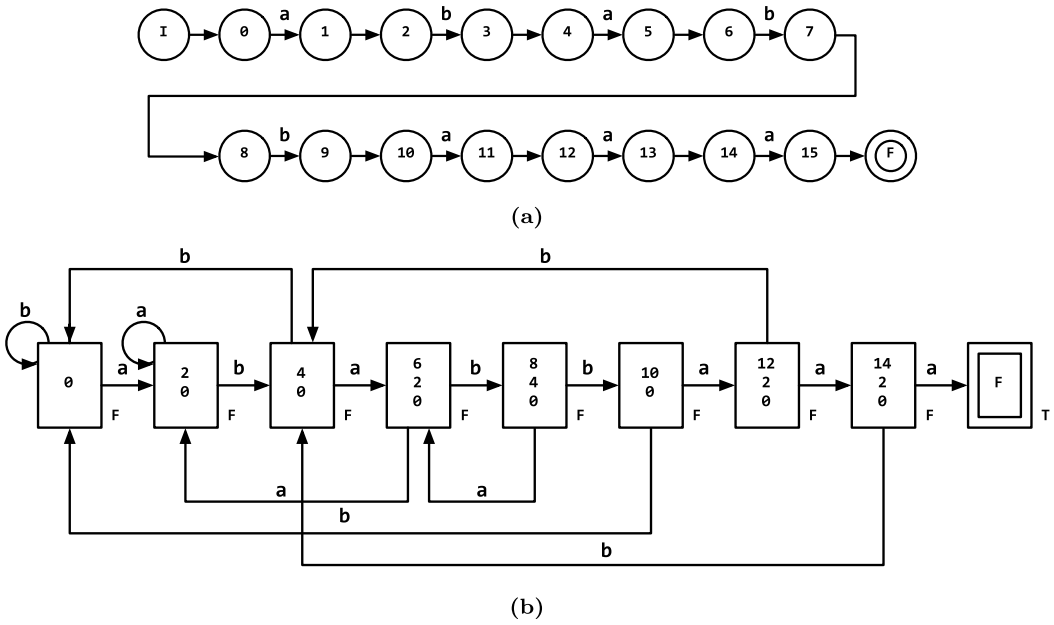

図 7 *ababbaaa* から構築した Thompson NFA と GDFA
Fig. 7 The Thompson NFA and GDFA constructed from *ababbaaa*.

るトライ木（trie）を任意の DFA に一般化しているが，これを GDFA の枠組みに適合させ，全体照合を行う GDFA から部分照合を行う GDFA を（部分集合構成を用いるよりも）効率的に生成できるかどうかは興味深い問題である．

GDFA $M = \langle \Sigma, Q, I, E_0, Q_F, \Delta \rangle$ を考える．与えられた文字列 $w = a_1 \dots a_n$ に対して

- $t_0 = I$,
- $t_{i-1} \xrightarrow{a_i}_{E_i} t_i \in \Delta$ $(1 \le i \le n)$,

であるとき，列 $E_0, t_0, a_1, \dots, E_{n-1}, t_{n-1}, a_n, E_n, T_n$ を（入力文字列 $a_1 \dots a_n$ に対する）M の**トレース**（trace）と呼ぶ．正規表現 $r \in \mathrm{Reg}(\Sigma)$ から作られた GDFA のトレース

$$\Pi = E_0, \langle \mathbf{q}_0, f_0 \rangle, a_1, \dots, a_n, E_n, \langle \mathbf{q}_n, f_n \rangle$$

に関していくつか表記と定義を設ける．以下では，$0 \le i \le n$ とする．まず，$\mathit{Heads}(E_i)$ を $\Pi(i)$ で表す．状態 $q \in \Pi(i)$ に対して，$\mathit{Path}_q(r, a_{i+1} \dots a_n)$ を $\mathit{Follow}_i^{\Pi}(q)$ で表す（明らかなとき Π は省略）．f_i が真であるか，あるいは $\mathit{Follow}_i^{\Pi}(q) \neq \emptyset$ となる $q \in \Pi(i)$ が存在するとき，$\Pi(i)$ は**活性**（active）であるという（活性でない $\Pi(i)$ は不活性と呼ばれる）．このとき，$\mathit{Follow}_i^{\Pi}(q) \neq \emptyset$ となる $\Pi(i)$ の最初の状態 q を $\Pi(i)$ の**主要な状態**（principal state）と呼ぶ．

補題 4 M を TNFA M' から作られる GDFA，Π を入力 $a_1 \dots a_n$ に対する M のトレースとし，任意の $0 \le i < n$ を考える．$\Pi(i)$ が活性であるならば $\Pi(i+1)$ も活性である．さらに，M' の初期状態 I が $\Pi(i+1)$ の主要な状態になることはない．

証明 f_i が真の場合には題意は自明．そこで f_i は偽と仮定すると，GDFA の遷移の定義より $\mathrm{F} \notin \mathit{Tails}(E_i)$ である．よって，$\Pi(i)$ に主要な状態があれば $\Pi(i+1)$ には I 以外の主要な状態が存在しなければならない． □

補題 5 M を GDFA，$\Pi = E_0, t_0, a_1, \dots, a_n, E_n, t_n$ を M のトレースとし，任意の $0 \le i < n$ を考える．q が $\Pi(i+1)$ の主要な状態で，かつ，$q \neq I$ であるならば，$q' = \mathit{head}(E_i(\mathit{src}(q)))$ も $\Pi(i)$ の主要な状態である．さらに，α が $\mathit{Follow}_{i+1}^{\Pi}(q)$ の最小の経路であるならば $E_i(\mathit{src}(q))a_{i+1}\alpha$ も $\mathit{Follow}_i^{\Pi}(q')$ の最小の経路である．

証明 $q \neq I$ かつ $\mathit{Follow}_{i+1}^{\Pi}(q) \neq \emptyset$ より，M の構成から $\mathit{Follow}_i^{\Pi}(q') \neq \emptyset$ を得る．よって，q' が $\Pi(i)$ の主要な状態であることを示すには，$q'' \prec_{\Pi(i)} q'$ かつ $\mathit{Follow}_i^{\Pi}(q'') \neq \emptyset$ を満たす $q'' \in \Pi(i)$ が存在しないことを示せばよい．そのような q'' の存在を仮定する．$\mathit{head}(s) = q''$ となる任意のステップ s を考えると，E_i の構成より $q'' \prec_{\Pi(i)} q'$ ならば $\mathit{tail}(s) \prec_{\mathit{Tails}(E_i)} \mathit{src}(q)$. よって $\mathit{tail}(s) \neq \mathrm{F}$ である．このことから，$\mathit{tail}(s) \xrightarrow{a_{i+1}} \mathit{dest}(\mathit{tail}(s)), \mathit{Follow}_{i+1}^{\Pi}(\mathit{dest}(\mathit{tail}(s))) \neq \emptyset$ と $\mathit{head}(s) = q''$ を満たすステップ s が存在する．$\mathit{dest}(\mathit{tail}(s)) \prec_{\Pi(i+1)} \mathit{dest}(\mathit{src}(q)) = q$ なので，これは q が主要な状態であるという前提に反する．よって，q' は $\Pi(i)$ の主要な状態である．

次に，$\beta = E_i(\mathit{src}(q))a_{i+1}\alpha$ が $\mathit{Follow}_i^{\Pi}(q')$ の最小経路であることを示す．β が最小経路でないと仮定すると，$\beta' \lessdot \beta$ かつ β' の最初のステップ β_1 が $\beta_1 \in E_i$ を満たす $\beta' \in \mathit{Follow}_i^{\Pi}(q')$ が存在する．すると $\mathit{tail}(\beta_1) \prec_{\mathit{Tails}(E_i)} \mathit{src}(q)$ となるので，$\mathit{tail}(\beta_1) \neq \mathrm{F}$. $q'' = \mathit{dest}(\mathit{tail}(\beta_1))$ と置くと $\mathit{Follow}_{i+1}^{\Pi}(q'') \neq \emptyset$ を得る．$q'' \prec_{\Pi(i+1)} q$ なので，これは q が $\Pi(i+1)$ の主要な状態であるという前提に反する．よって β は $\mathit{Follow}_i^{\Pi}(q')$ の最小経路である． □

以上をふまえ，与えられた経路 $\alpha \in \mathit{Follow}_i^{\Pi}(q)$ に対して $\mathrm{M}(r, a_1 \dots a_n)$ の要素 $\mathit{sol}_i^{\Pi}(\alpha)$ を以下のように定義する．

$$sol_i^{\Pi}(\alpha) = \begin{cases} \langle a_1 \dots a_i, \alpha \rangle & (q = \mathrm{I}) \\ sol_{i-1}^{\Pi}(E_{i-1}(src(q))a_i\alpha) & (q \neq \mathrm{I}) \end{cases}$$

補題 6 正規表現 $r \in \mathrm{Reg}(\Sigma)$ を考え，Π を入力 $w = a_1 \dots a_n$ に対する $GA(\Sigma, r)$ のトレースとする．$\Pi(i)$（$0 \leq i \leq n$）の主要な状態 q が存在し α が $Follow_i^{\Pi}(q)$ の最小経路であるならば，$sol_i^{\Pi}(\alpha)$ は $\mathrm{M}(r, w)$ の最小解である．

証明 i に関する数学的帰納法による．$i = 0$ の場合は $\Pi(0) = \mathrm{I}$ であり，$sol_i^{\Pi}(\alpha) = \langle \varepsilon, \alpha \rangle$ が最小解であることは自明．以下 $i > 0$ とし，q が I か否かで場合分けする．

[1]（$q = \mathrm{I}$ の場合）補題 4 より $\Pi(j)$（$0 \leq j < i$）はすべて不活性であるので，$\langle a_1 \dots a_j, \beta \rangle$（$0 \leq j < i$）という形の解はあり得ない．よって $sol_i^{\Pi}(\alpha) = \langle a_1 \dots a_i, \alpha \rangle$ が $\mathrm{M}(r, w)$ の最小解である．

[2]（$q \neq \mathrm{I}$ の場合）$\Pi = E_0, t_0, a_1, \dots, a_n, E_n, t_n$，および，$q' = head(E_{i-1}(src(q)))$ と置くと，補題 5 より q' は $\Pi(i-1)$ の主要な状態であり $\alpha' = E_{i-1}(src(q))a_i\alpha$ は $Follow_{i-1}(q')$ の最小経路である．よって帰納法の仮定より $sol_{i-1}^{\Pi}(\alpha')$ は $\mathrm{M}(r, w)$ の最小解である．定義より $q \neq \mathrm{I}$ のとき $sol_i^{\Pi}(\alpha) = sol_{i-1}^{\Pi}(\alpha')$ なので，題意が示された． □

次の定理は本論文の第 2 の主要結果であり，GDFA を用いて貪欲な部分照合の解が得られることを述べている．

定理 2 正規表現 $r \in \mathrm{Reg}(\Sigma)$ と入力 $w = a_1 \dots a_n$ を考え，$\Pi = E_0, t_0, a_1, \dots, a_n, E_n, t_n$ を w に対する $GA(\Sigma, r)$ のトレースとする．t_i（$0 \leq i \leq n$）の中に終状態があると仮定し，その最大の添え字を m とする．このとき，$sol_m^{\Pi}(E_m(\mathrm{F}))$（F は $TA(r)$ の終状態）は $\mathrm{M}(r, a_1 \dots a_n)$ の最小解を与える．

証明 t_m を $\langle q_1 \dots q_k, f \rangle$ と置くと，$t_m \in Q_F$ より $q_k = \mathrm{F}$，$q_j \neq \mathrm{F}$（$1 \leq j < k$）である．$i > m$ のとき $t_i \notin Q_F$ なので，$q'' \in \Pi(m+1)$ ならば $Follow_{m+1}(q'') = \emptyset$．よって，$q' = head(E_m(\mathrm{F}))$ と置くと，$head(E_m(q_j)) \neq q'$ を満たすどの q_j（$1 \leq j < k$）に対しても $Follow_m(head(E_m(q_j))) = \emptyset$ が得られる．したがって，q' は $\Pi(m)$ の主要な状態である．さらに $head(s) = q'$ と $s \lessdot E_m(\mathrm{F})$ を満たす任意のステップ s は，ある j $(1 \leq j < k)$ について $tail(s) = q_j$ を満たすので，$Follow_{m+1}(q_j) = \emptyset$ より，$sa_i\alpha \in Follow_m(q')$ を満たす α はない．よって，$E_m(\mathrm{F})$ は $Follow_m(q')$ の最小経路であり，補題 6 から $sol_m^{\Pi}(E_m(F))$ は $\mathrm{M}(r, a_1 \dots a_n)$ の最小解である． □

図 8 に GDFA を漸進的に構築しつつ部分照合を行うアルゴリズムを示す．これは，実質的に与えられた文字列に対するトレースを構築しているにほかならない．ただし，NFA 状態の列 Q が空になった場合には計算を打ち切っている．このとき必然的に $foundF$ が真であることが容易に分かる．Q が空になったか，あるいは，与えられた文字列を最後まで読み終えたときに $foundF$ が真ならば終状態に到達したことになり，部分照合の解が存在することを意味する．$foundF$ が偽であれば，解がないことを意味する．5 行目の proceed 関数の時間計算量が $\mathrm{O}(n)$（n は TNFA の状態数，あるいは正規表現の部分式の総数）なので，この照合アルゴリズムの時間計算量は $\mathrm{O}(nm)$（m は照合文字列の長さ）である．

```
1: function ON-THE-FLY-MATCH(s)
2:     ⟨Q, foundF⟩ ← ⟨Closure([I]), F ∈ Closure([I])⟩;
3:     α ← [Effective([I])];
4:     for a in s do
5:         ⟨⟨Q, foundF⟩, E⟩ ← proceed(⟨Q, foundF⟩, a);
6:         break if Q = []; /* Note: Q = [] ⇒ foundF */
7:         α ← α++[E];
8:     end for
9:     return ⟨foundF, α⟩;
10: end function
```

図 8 部分照合手続き（漸進的に GDFA を構築）
Fig. 8 The partial match procedure by constructing a GDFA incrementally.

例として部分照合問題 $\langle \{a, b\}, abaaabaa, (ab{+}a*)* \rangle$ を考える．図 8 のアルゴリズムによって，図 6 の GDFA（の実線で記した部分）が構築され，接頭辞 $abaaab$ に対してシンクに至るトレース

$$E_0, q_1, a, E_1, q_2, b, E_4, q_1, a, \\ E_1, q_2, a, E_2, q_3, a, E_3, q_3, b, E, q_4$$

（ただし，$E = \langle [], \mathsf{T} \rangle$）が得られる．よってこの照合は解を持つ．最後に出現する終状態 $q3$ に定理 2 を適用して最小の（すなわち貪欲な）解 $\langle \varepsilon, \alpha \rangle$ が得られる．ここで α は以下の経路である．

$$\mathrm{I}, 2, 1, 11, 111, a, \overline{111}, 112, \\ b, \overline{112}, \overline{11}, \overline{1}, 2, 1, 12, 122, 121, \\ a, \overline{121}, 122, 121, \\ a, \overline{121}, 122, 121, \\ a, \overline{121}, 122, \overline{12}, \overline{1}, 2, \mathrm{F}$$

定理 1 によれば，これは以下の導出木に相当する．

$$\mathtt{I}(\mathtt{L}(a{\cdot}b), \mathtt{R}(\mathtt{I}(a, a, a)))$$

すなわち，正規表現全体は与えられた文字列 $abaaabaa$ の $[0, 5)$ の範囲（最初の 5 文字分）を捕獲し，ポジション $1, 11, 111, 112, 12, 121$ における各部分式はそれぞれ以下の範囲を捕獲することが分かる．

1: $[0,2)$, $[2,5)$	112: $[1,2)$
11: $[0,2)$	12: $[2,5)$
111: $[0,1)$	121: $[2,3)$, $[3,4)$, $[4,5)$

ここで $[i, j)$（$i, j \geq 0$）は i 以上 j 未満の範囲を表す．

上の経路 α から上記の各範囲を求めるには経路全体を一

```
1: function SUBSET-CONS
2:     Δ ← ∅;
3:     Q ← {};
4:     frontier ← {⟨Closure([I]), F ∈ Closure([I])⟩};
5:     while frontier is non empty do
6:         src ← frontifer.remove();
7:         Q ← Q ∪ {src};
8:         for a ∈ Σ do
9:             ⟨dest, E⟩ ← proceed(src, a);
10:            frontier ← frontier ∪ {dest} if dest ∉ Q;
11:            Δ ← Δ ∪ {src -a->_E dest};
12:        end for
13:    end while
14:    return ⟨Q, Δ⟩;
15: end function
```

図 9 あらかじめ GDFA を構築
Fig. 9 The construction procedure of a GDFA.

```
1: function MATCH(s)
2:     ⟨q, foundF⟩ ← I′;
3:     α ← [E_0];
4:     for a in s do
5:         ⟨q, foundF⟩ ← ⟨q′, foundF′⟩;
6:             where ⟨q, foundF⟩ -a->_E ⟨q′, foundF′⟩ ∈ Δ′;
7:         break if q = []; /* Note: q = [] ⇒ foundF */
8:         α ← α++[E];
9:     end for
10:    return ⟨foundF, α⟩;
11: end function
```

図 10 部分照合手続き（事前構築した GDFA を使用）
Fig. 10 The partial match procedure using a GDFA constructed beforehand.

度走査すればよい．各ステップの長さは部分式の数，よって TNFA の状態数 n に比例するので，この時間計算量は $O(mn)$ である（m は与えられた文字列の長さ）．一般には，捕獲したい部分式は部分式のごく一部だけである．そこで，あらかじめ捕獲したい部分式のポジションのみを走査できるようにしておけば，これは $O(mk)$ になる（k は捕獲したい部分式の個数）．

本アルゴリズムはシンクが出現するか入力文字列の最後に到達するまで走査を完了して初めて逆方向にトレースを辿り捕獲を計算するので，文献 [12] のように漸近的に捕獲を計算することはできない．これは，本研究がより一般的な部分照合を扱うためであり，文献 [12] の最適化されたストリーム処理が部分照合の場合に一般化できるかどうかは明らかではない．

GDFA の状態数が多くない場合には，GDFA 全体をあらかじめ構築しておく方法も有効である．**図 9** にそのアルゴリズムを示す．**図 10** の照合アルゴリズムは，全体の構造は図 8 と同じだが，各ステップで proceed 関数を呼ぶ代わりに，あらかじめ構築しておいた GDFA を参照している点が異なる．このため，この照合アルゴリズムの時間計算量は TNFA の状態数（あるいは，正規表現の部分式の総数）には無関係で $O(m)$（m は照合文字列の長さ）である．

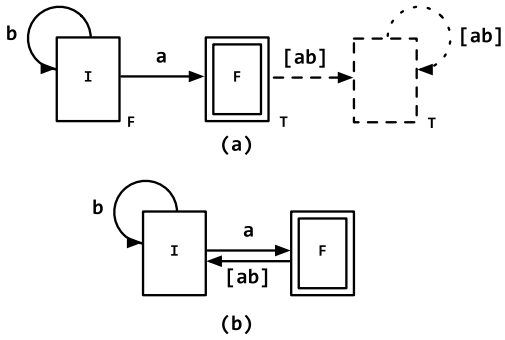

図 11 正規表現 a に対する GDFA
Fig. 11 The GDFA for regular expression a.

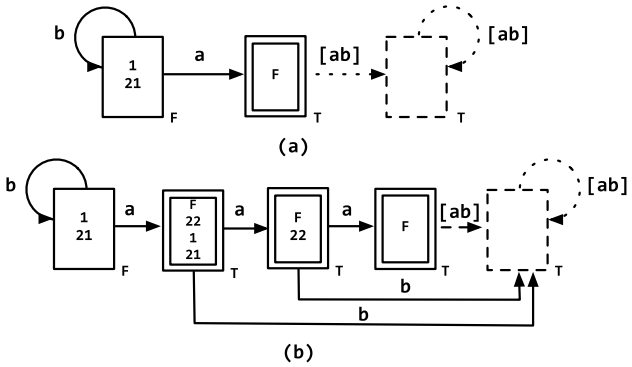

図 12 正規表現 a+aa に対する GDFA
Fig. 12 The GDFA for regular expression a+aa.

GDFA の遷移の定義において仮に [I | f = F] を [I] で置き換えつねに始状態 I を付加することにすれば，定理 2 はもはや成り立たなくなることに注意が必要である．部分照合問題 $\langle\{a, b\}, aba, a\rangle$ を例として考えると，正規表現 a から構築した GDFA（**図 11** の (a)，図では各遷移における有効なステップの表示は省略）を用いるとシンクに至る経路

$$\langle[\mathrm{I}], \mathsf{F}\rangle \xrightarrow{a} \langle[\mathrm{F}], \mathsf{T}\rangle \xrightarrow{b} \langle[], \mathsf{T}\rangle$$

が得られ，定理 2 より貪欲な部分照合の解 $\langle\varepsilon, a\rangle$ が得られる．一方，フラグを用いずにつねに始状態を付加すれば同図 (b) の GDFA より経路

$$[\mathrm{I}] \xrightarrow{a} [\mathrm{F}] \xrightarrow{b} [\mathrm{I}] \xrightarrow{a} [\mathrm{F}]$$

が得られ，これに定理 2 を適用すると誤った（最小でない）解 $\langle ab, a\rangle$ が得られてしまう．よって，すでに終状態に到達したことがあるか否かを表すフラグを GDFA 状態に付加することは重要である．

また，有効なステップの計算においては終状態に至るステップを発見次第，それ以降のステップの計算を打ち切る（図 4 の関数 effective() の 10 行目）が，仮に打ち切らないことにすれば定理 2 はもはや成り立たなくなる．実際，部分照合問題 $\langle\{a, b\}, aaa, a{+}aa\rangle$ を考えると，提案する構築法で得られた GDFA（**図 12** (a)）では経路

$$\langle[1, 21], \mathsf{F}\rangle \xrightarrow{a} \langle[\mathrm{F}], \mathsf{T}\rangle \xrightarrow{a} \langle[], \mathsf{T}\rangle$$

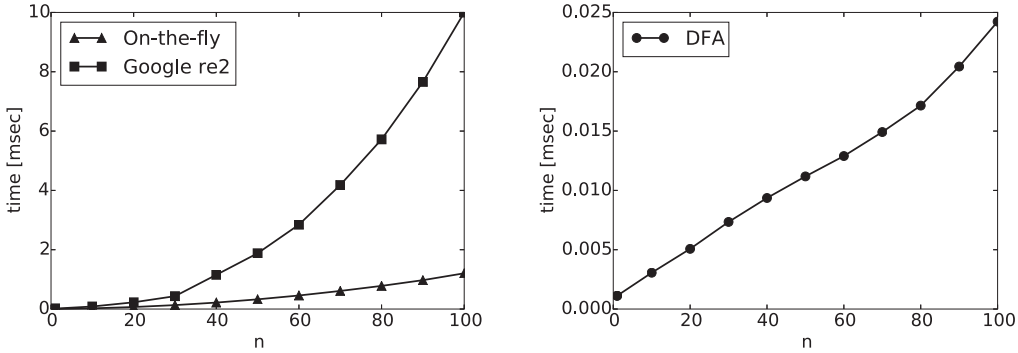

図 13 本手法と Google RE2 の照合・部分式の捕獲時間．正規表現：$(a{+}1)^n a^n$ $(n = 1, 10, 20, \ldots, 100)$，文字列：$a^n$ $(n = 1, 10, 20, \ldots, 100)$

Fig. 13 Comparison of the partial matching and capturing of subexpressions between the proposed method and Google RE2. regular expression: $(a{+}1)^n a^n$ $(n = 1, 10, 20, \ldots, 100)$, string: a^n $(n = 1, 10, 20, \ldots, 100)$.

が得られ，定理 2 より貪欲な部分照合の解 $\langle \varepsilon, \mathtt{L}(a) \rangle$ が得られるが，終状態に至るステップで計算を打ち切らない場合には同図 (b) の GDFA から経路

$$\langle [1, 21], \mathsf{F} \rangle \xrightarrow{a} \langle [\mathrm{F}, 22, 1, 21], \mathsf{T} \rangle \xrightarrow{a} \langle [\mathrm{F}, 22], \mathsf{T} \rangle \xrightarrow{a} \langle [\mathrm{F}], \mathsf{T} \rangle$$

が得られ，定理 2 を適用すると誤った（最小でない）解 $\langle a, \mathtt{R}(a{\cdot}a) \rangle$ が得られてしまう．終状態に至るステップより大きいステップを有効なステップから除外することには単に無駄な探索を除外する最適化以上の意味があることが分かる．

5. 照合に要する時間の計測

提案手法の実際的な効果を確認するために，C++による試験実装を行い，GDFA 構築と照合に要する計算時間および GDFA の状態数を計測した．計測に用いた計算機の CPU は Intel Core i5 2557M 1.7 GHz，メモリは 4 GB 1,333 MHz DDR3 であり，コンパイラは clang++ 3.5（最適化オプション-O3）を使用した．計算時間の計測には，10,000 回反復実行した際の CPU 時間を累積し 10,000 で割った値を用いた．

5.1 正規表現エンジン RE2 との比較

まず，正規表現 $(a{+}1)^n a^n$ と文字列 a^n（$n > 0$）の照合および部分式の捕獲に要する計算時間を n を増やしつつ計測し，正規表現エンジン RE2（20140304 版）との比較を行った．この例は RE2 の開発者が [5] で取り上げているもので，バックトラックに基づく従来型の正規表現エンジンでは計算時間が急激に増大し事実上計算不可能な例である．RE2 は照合問題ごとに最適化された複数のアルゴリズムからなるが，そのうちの部分照合と部分式による捕獲を行う関数 `RE2::PartialMatchN` を用いて計測を行った．また，捕獲はすべての部分式について行った．その結果を**図 13** に示す．

左のグラフは図 8 のアルゴリズムによる照合と部分式の捕獲に要する時間を RE2 と比較したもので，全般に本手法のほうがより高速であることが見て取れる．問題のサイズ n が増大するにつれ両者の差は拡大し，$n = 100$ の場合で本手法は RE2 の 8 倍程度高速であることが分かる．

右のグラフは図 9 のアルゴリズムを用いてあらかじめ構築した GDFA に図 10 のアルゴリズムを用いた場合の照合と部分式による捕獲に要する時間を計測したものである（GDFA 構築に要する時間は含まれていない）．当然ながら GDFA を漸進的に構築しつつ照合する場合と比較して高速（$n = 100$ の場合で 20 倍程度）である．

5.2 部分照合に要する時間の計測

次に，KMP 的な部分照合の効果を確認するために，正規表現 a∗c と文字列 $\mathrm{a}^n\mathrm{bc}$（$n > 0$）の部分照合に要する時間を n の値を増やしつつ計測し Perl（5.16.2）および Python（3.4.2）の正規表現エンジンを用いた場合と比較した．その結果を**図 14** に示す．左のグラフは Perl（左軸）および Python（右軸）の照合に要する時間を示している．およそ N の自乗で計算時間が増大している．一方，右のグラフは図 8 のアルゴリズムを用いた場合（On-the-fly）と図 9 のアルゴリズムで構築した GDFA に図 10 を用いた場合（DFA）の照合に要する時間を示している（GDFA 構築に要する時間は含まれていない）．いずれも n に関して線形に増大しており，提案手法により効率的な部分照合が可能であることを示している．

5.3 オートマトンの状態数

以下の (1) から (5) までの 5 つの正規表現から構築される TNFA および GDFA の状態数を**表 1** に示す．

(1) [0-9]{3}-[0-9]{4}
(2) ([*a-zA-Z*][*a-zA-Z*0-9]∗) : //([ˆ /]+)(/[ˆ]∗)?
(3) ([ˆ @]+)@([ˆ @]+)

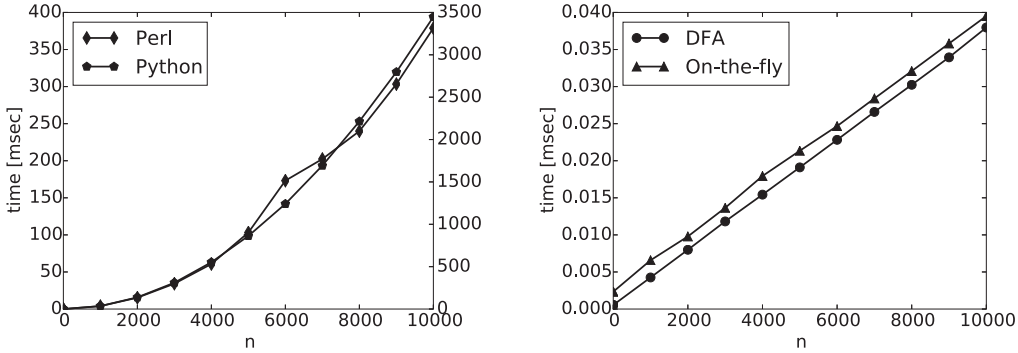

図 14 本手法（DFA, On-the-fly）と Perl, Python（Backtrack）の照合時間．正規表現：$a*c$, 文字列：$a^n bc$

Fig. 14 Comparison of matching between the proposed method (DFA and On-the-fly) and Perl, Python (Backtrack). regular expression: $a*c$, string: $a^n bc$.

表 1 オートマトン（TNFA，GDFA）の状態数

Table 1 The number of states in the TNFA and GDFA.

	(1)	(2)	(3)	(4)	(5)
TNFA	30	40	14	44	85
GDFA	10	8	5	12	1537

(4) ([0-9][0-9]?)/([0-9][0-9]?)/([0-9][0-9]([0-9][0-9])?)
(5) $(a|b)*a(a|b)\{9\}$

表記の簡略化のため（前章までと異なり）POSIX の正規表現の表記を用いている．(1) から (4) は実用的な正規表現の例（順に郵便番号，URI，メールアドレス，日付を簡略化したもの）で，(2) から (4) はベンチマークサイト [15] で取り上げられているものである．(5) は状態数が指数的に爆発する作為的な例である．この結果をみる限りでは状態数が爆発する作為的な例 (5) を除けば GDFA の状態数はそれほど多くはなっていない．このことから現実的な方法として（1）あらかじめ設定した状態数を超えるまでは，あらかじめ GDFA を構築しておき，(2) 設定値を超えた場合は残りの部分を照合実行時に漸進的に構築するという折衷案が有効であろうと考えられる．

6. おわりに

本論文では，正規表現の貪欲な部分照合と部分式による捕獲を効率的に行う新たな DFA 構築法について述べ，部分照合の貪欲な解が得られることを証明した．また，本手法による部分照合の計算コストについて論じ，最悪の場合でも与えられた文字列に比例することを述べた．さらに試験実装に基づき，最新の正規表現エンジンの 1 つである RE2 等と比較してより良好な結果が得られることを示した．また，本研究の手法と古典的な KMP 文字列照合および Aho-Corasick オートマトンとの関連についても言及し，正規表現が単なる文字列である場合，本手法によって Aho-Corasick オートマトンと同型の DFA が得られることを例示した．

貪欲照合に関する先行研究がもっぱら全体照合のみを扱っているのに対し，部分照合を含めて議論し手法の正しさの証明を与えている点は本研究の新規な点である．本論文では手法を厳密に定式化し，その正しさの証明に議論の多くを割いているが，これは今後さらに増えるであろう DFA に基づく正規表現エンジンの実装に対し，その基本的な手法の正当性の根拠を与えたという点に意義がある．

今後の研究の方向としては，Boyer-Moore 法を始めとする文字列の部分照合の手法の導入を検討し，さらなるアルゴリズムの改良を試みることがあげられる．

参考文献

[1] Aho, A.V. and Corasick, M.: Efficient string matching: An aid to bibliographic search, *Comm. ACM*, Vol.18, pp.333–340 (1975).
[2] Aho, A.V., Lam, M.S., Sethi, R. and Ullman, J.D.: *Compilers: Principles, Techniques, and Tools*, Addison Wesley (2006).
[3] Brüggemann-Klein, A.: Regular Expressions into Finite Automata, *Theoretical Computer Science*, Vol.120, No.2, pp.197–213 (1993).
[4] Câmpeanu, C., Salomaa, K. and Yu, S.: Regex and extended regex, *CIAA'02*, pp.77–84 (2002).
[5] Cox, R.: Regular expression matching can be simple and fast (2007), available from ⟨http://swtch.com/∼rsc/regexp/regexp1.html⟩.
[6] Cox, R.: Regular expression matching: The virtual machine approach (2009), available from ⟨http://swtch.com/∼rsc/regexp/regexp2.html⟩.
[7] Cox, R.: Regular expression matching in the wild (2010), available from ⟨http://swtch.com/∼rsc/regexp/regexp3.html⟩.
[8] Cox, R.: Regular expression matching with a trigram index or how Google Code Search worked (2012), available from ⟨http://swtch.com/∼rsc/regexp/regexp4.html⟩.
[9] Crochemore, M. and Rytter, W.: *Jewels of Stringology — Text Algorithms*, World Scientific (2002).
[10] Fowler, G.: An interpretation of the POSIX regex standard (2003), available from ⟨http://www2.research.att.com/∼gsf/testregex/re-interpretation.html⟩.

[11] Frisch, A. and Cardelli, L.: Greedy Regular Expression Matching, *ICALP'04* (*LNCS 3142*), pp.618–629 (2004).
[12] Grathwohl, N.B.B., Henglein, F. and Rasmussen, U.: Optimally Streaming Greedy Regular Expression Parsing, *ICTAC'14* (*LNCS 8687*), pp.224–240 (2014).
[13] Grathwohl, N.B.B., Henglein, F. Nielsen, L. and Rasmussen, U.: Two-Pass Greedy Regular Expression Parsing, *CIAA'13* (*LNCS 7982*), pp.60–71 (2013).
[14] Knuth, D.E., Morris, J.H. and Pratt, V.R.: Fast pattern matching in strings, *SIAM J. Comput.*, Vol.6, pp.323–350 (1977).
[15] Li, H.: Benchmark of regex libraries (2010), available from ⟨http://lh3lh3.users.sourceforge.net/reb.shtml⟩.
[16] Mohri, M.: String matching with automata, *Nordic Journal of Computing*, Vol.4, No.2, pp.217–231 (1997).
[17] Nielsen, L. and Henglein, F.: Bit-coded Regular Expression Parsing, *LATA 2011* (*LNCS 6638*), pp.402–413 (2011).
[18] Okui, S. and Suzuki, T.: Disambiguation in Regular Expression Matching via Position Automata with Augmented Transitions, *CIAA 2010* (*LNCS 6482*), pp.231–240 (2010).
[19] Rabin, M.O. and Scott, D.: Finite automata and their decision problems, *IBM Journal of Research and Development*, Vol.3, No.2, pp.114–125 (1959).
[20] Sin'ya, R., Matsuzaki, K. and Sassa, M.: Simultaneous Finite Automata: An Efficient Data-Parallel Model for Regular Expression Matching, *ICPP-2013*, pp.220–229 (2013).
[21] Sulzmann, M. and Lu, K.Z.M.: POSIX regular expression parsing with derivatives, *FLOPS 2014* (*LNCS 8475*), pp.203–220 (2014).
[22] Thompson, K.: Regular expression search algorithm, *Comm. ACM*, Vol.11, No.6, pp.419–422 (1968).
[23] 奥居 哲, 増田拓也：正規表現の貪欲な部分照合と Morris-Pratt アルゴリズム, 中部大学情報科学ジャーナル, Vol.22, pp.101–104 (2015).
[24] 新屋良磨，光成滋生，佐々政孝：並列化と実行時コード生成を用いた正規表現マッチングの提案，コンピュータソフトウェア，Vol.30, No.2, pp.191–206 (2013).
[25] 増田拓也：正規表現の貪欲な部分照合に関する研究，修士論文，中部大学工学部情報工学専攻修士論文 (2015).

奥居 哲

昭和 42 年生．平成 7 年筑波大学大学院博士課程工学研究科電子・情報工学専攻修了．博士（工学）．平成 8 年三重大学工学部情報工学科助手．平成 11 年講師．平成 13 年より中部大学工学部情報工学科准教授．オートマトン，単一化，書換え系，関数論理型言語に関する研究に従事．

増田 拓也

平成 2 年生．平成 25 年中部大学工学部情報工学科卒業．平成 27 年中部大学大学院工学研究科情報工学専攻博士前期課程修了．修士（工学）．オートマトンに関する研究に興味を持つ．

鈴木 大郎 （正会員）

昭和 39 年生．平成 7 年筑波大学大学院博士課程工学研究科電子・情報工学専攻中退．博士（理学）．平成 7 年筑波大学電子・情報工学系助手．平成 10 年北陸先端科学技術大学院大学助手．平成 12 年東北大学電気通信研究所助手．平成 13 年会津大学講師．現在，会津大学コンピュータ理工学部コンピュータ理工学科上級准教授．オートマトン，XML，単一化，書換え系，関数論理型言語に関する研究に従事．著書『コンパイラ』（近代科学社）．ACM，ソフトウェア科学会各会員．