情報処理学会論文誌　プログラミング　Vol.6　No.1　17–26 (Jan. 2013)

アトミックグループで拡張された正規表現のオートマトンへの変換

杉山 聡[1,a)]　南出 靖彦[2,b)]

受付日 2012年4月20日, 採録日 2012年6月22日

概要：プログラミング言語における正規表現の拡張の 1 つとしてアトミックグループがある．アトミックグループとは，一度構文内でのマッチが成功し構文を抜けると，構文内へのバックトラックを禁止する構文である．本論文では，アトミックグループで拡張された正規表現のオートマトンへの変換を構成し，その正当性を証明する．拡張された正規表現の意味は Sakuma らによるリストモナドを用いた定義により与える．アトミックグループで拡張された正規表現の表す言語を定義するために，集合モナドを用いた非決定構文解析器を定義する．アトミックグループによるマッチングは，オプションモナドを用いた決定性構文解析器によって表現する．この非決定性構文解析器と決定性構文解析器は相互再帰的な等式となっており，それによって拡張された正規表現の表す言語を表現する．この相互再帰的な等式をもとに，それと等価な先読み付きオートマトンを構成する．先読み付きオートマトンは先読みなしのオートマトンに変換することができるため，拡張された正規表現の表す言語は正則であるといえる．本研究で与えた変換を OCaml を用いて実装し，アトミックグループを含む正規表現を DFA に変換する実験を行った．実験には，Perl ライブラリのアーカイブである CPAN で使用されている正規表現，および Mastering Regular Expressions 3rd Edition に収録されている正規表現を用いた．

キーワード：正規表現，オートマトン，モナド

Translating Regular Expressions Extended with Atomic Grouping to Automata

Satoshi Sugiyama[1,a)]　Yasuhiko Minamide[2,b)]

Received: April 20, 2012, Accepted: June 22, 2012

Abstract: Atomic grouping is one of the extensions of regular expressions used in programing languages. When a word is matched against an atomic group, the construct disables backtracking into the group. In this paper, we translate regular expressions extended with atomic grouping into finite automata and prove the correctness of the translation. The semantics of extended regular expressions is given as a nondeterministic parser defined with the list monad. Then, the language of an extended regular expression is represented by using mutually recursive functions: a nondeterministic parser defined with the set monad and a deterministic parser defined with the option monad. We construct finite automata with regular lookahead based on their definitions and translate them into finite automata without lookahead by a standard construction. We have implemented this translation in OCaml, and conducted experiments of translating regular expressions containing atomic groups into DFA. For the experiments, we use regular expressions that appear in CPAN (Comprehensive Perl Archive Network) and Mastering Regular Expressions 3rd Edition.

Keywords: regular expressions, automata, monads

1. はじめに

ウェブのプログラム等文字列を操作するプログラムにおいて，セキュリティに関する重要な検査等様々な場面で正規表現が用いられ，重要な役割を果たしている．プログラ

[1] 筑波大学大学院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, Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
[2] 筑波大学システム情報系情報工学域
Division of Information Engineering, Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan

a) sugiyama@score.cs.tsukuba.ac.jp
b) minamide@cs.tsukuba.ac.jp

ミング言語における正規表現は表現力を高めるために様々な拡張が行われ，多くの場合，バックトラックを用いて検索や置換を行う関数として実装されている．このような拡張された正規表現が表す言語が，数学的な意味での正則言語になっているかは必ずしも明らかでない．先読みによって拡張された正規表現が表す言語が正則であることは森畑によって証明されている [1]．一方，後方参照によって拡張された正規表現が表す言語は正則でないことが知られている．

本研究では，アトミックグループで拡張された正規表現からオートマトンへの変換を与え，その拡張された正規表現の表す言語が正則であることを示す．アトミックグループとは，一度構文内でのマッチが成功し構文を抜けると，構文内へのバックトラックを禁止する構文である [4]．すなわち，1 つマッチしたら他の可能性を破棄しなければならない．Perl，PHP 等の多くのプログラミング言語では，正規表現 `r` 中のバックトラックを禁止するアトミックグループを `(?>r)` と表記する．PHP において，アトミックグループを用いた場合のマッチングの結果の違いの例を示す．正規表現によるマッチングを行う関数 `preg_match` は第 3 引数に配列をとり，配列の 0 番目の要素にマッチした文字列を格納する．

```
preg_match ('/(a*)a/', "aaa", $matches);
print ($matches[0]);     ==> aaa
preg_match ('/(?>a*)a/', "aaa", $matches);
print ($matches[0]);     ==>
```

正規表現 `(a*)a` は文字列 *aaa* にマッチしている．一方，アトミックグループを用いた `(?>a*)a` は何もマッチしない．アトミックグループ内の `a*` が文字列全体にマッチして，アトミックグループを抜け，続く正規表現 a がマッチする文字列を探すが，アトミックグループによってバックトラックが禁止されるため，マッチを成功させることができない．

本研究の先行研究として，正規表現によるマッチングのトランスデューサへの変換が研究されている [7]．先行研究では，Perl 互換の正規表現の意味論を精確にとらえるために，正規表現マッチングの意味をリストモナドを用いて優先順位を持つ非決定性構文解析器として定義している．この非決定性構文解析器から，オプションモナドを用いて定義される先読みを用いた決定性構文解析器を導出し，決定性構文解析器をもとにトランスデューサを構成している．この先行研究において，アトミックグループについての意味論が与えられていたが，アトミックグループを含む正規表現が表す言語が正則であるかについての議論はされていない．

本研究では，先行研究では行われていないアトミックグループで拡張された正規表現のオートマトンへの変換を与えた．アトミックグループで拡張された正規表現が表す言語は通常の正規表現のように定義することができない．したがって，Sakuma らにより与えられている正規表現マッチングの意味論から正規表現の表す言語を導出する．そのために集合モナドを用いた非決定性構文解析器を導出し，この非決定性構文解析器をもとにオートマトンを構成する．しかし，正規表現マッチングの意味論から直接定義した非決定性構文解析器は，意味論に依存している．オートマトンは優先順位を表現することができないため，非決定性構文解析器から優先順位を排除する必要がある．そのためにアトミックグループによるマッチングをオプションモナドを用いた決定性構文解析器によって表現する．この非決定性構文解析器と決定性構文解析器は相互再帰的な等式となっており，それによって拡張された正規表現の言語を表現する．この相互再帰的な等式をもとに，それと等価な先読み付きオートマトンを構成する．先読み付きオートマトンは先読みなしのオートマトンに変換することができるため，拡張された正規表現が表す言語は正則であるといえる．OCaml を用いてこの変換の実装を行い，アトミックグループを含む正規表現を DFA に変換する実験を行った．実験には，Perl ライブラリのアーカイブである CPAN で使用されている正規表現，および Mastering Regular Expression 3rd Edition [4] に収録されている正規表現を用いた．しかし，CPAN で使用されている正規表現のうち，アトミックグループを使用したものは非常に少なく，実験に使用できた正規表現は 3 つのみだった．実験に使用した正規表現に関しては，少ない状態数でオートマトンを構成することができた．

関連する研究として，先読みで拡張された正規表現をオートマトンに変換する森畑による研究がある [1]．この研究では，先読みで拡張された正規表現を AFA（Alternating Finite Automaton）[2] で構成し，さらに，重み付き正規表現 [3] を導入することで部分マッチの取り出しについても解決策を提案している．

2. 準備

2.1 モナド

正規表現マッチングの意味を定義するために，プログラミング言語におけるモナドを簡単に説明する [5]．モナド M とは次の多相型関数をともなう型構成子 M である．

$$\mathrm{unit}_M :: \alpha \to \alpha\ M$$
$$\mathrm{bind}_M :: \alpha\ M \to (\alpha \to \beta\ M) \to \beta\ M$$

これらの関数は次の 3 つのモナド則を満たす必要がある．ここで，$\mathrm{bind}_M\ m\ f$ は $m \gg\!=_M f$ と表記することにする．

$$(\mathrm{unit}_M\ x) \gg\!=_M f = f\ x$$
$$m \gg\!=_M (\lambda x.\mathrm{unit}_M\ x) = m$$
$$(m \gg\!=_M f) \gg\!=_M g = m \gg\!=_M (\lambda x.f\ x \gg\!=_M g)$$

本論文では，簡潔に表すために，モナドを明示する必要のないところでは上記の関数の添字を省略する．

あるモナド M_1 からそのモナドと関わりのある違うモナド M_2 へ変換する，次の性質を持つ多相型の関数 h を M_1 から M_2 へのモナド射（monad morphism）であるという [8], [10]．

$$h(\mathrm{unit}_{M_1}\ x) = \mathrm{unit}_{M_2}\ x$$

$$h(m \gg\!=_{M_1}\ f) = (h\ m \gg\!=_{M_2}\ \lambda x.h\ (f\ x))$$

本研究では，リストモナドから集合モナドへのモナド射やリストモナドからオプションモナドへのモナド射を用いる．

2.2　先読み付きオートマトン

本研究では，先読み付きオートマトン $A = (Q, \Sigma, \Delta, q_0, F)$ を用いて，アトミックグループで拡張された正規表現をオートマトンに変換する．通常の ϵ-NFA との違いは遷移関数 $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q \times Reg(\Sigma)$ のみである．ここで，$Reg(\Sigma)$ は文字の集合 Σ 上の正則言語の集合である．オートマトン A において，入力文字列 ww' のうち w を読んで，状態 q から状態 q' へ遷移する遷移関係 $q \xrightarrow[A]{w/w'} q'$ を次のように定義する．

$$q \xrightarrow[A]{\epsilon/w'} q \quad \text{if} \quad w' \in \Sigma^*$$

$$q \xrightarrow[A]{\epsilon/w'} q' \quad \text{if} \quad (q, \epsilon, q', R) \in \Delta \ \text{ and } \ w' \in R$$

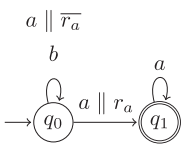
$$q \xrightarrow[A]{a/w'} q' \quad \text{if} \quad (q, a, q', R) \in \Delta \ \text{ and } \ aw' \in R$$

$$q \xrightarrow[A]{ww'/w''} q' \quad \text{if} \quad q \xrightarrow[A]{w/w'w''} q'' \ \text{ and } \ q'' \xrightarrow[A]{w'/w''} q'$$

1 つ目の規則は，遷移が存在しないときの規則である．遷移が存在しない場合，文字を消費せず，他の状態への遷移もしない．2 つ目の規則は，先読みによる ϵ-遷移の規則である．入力文字列 w が先読みの正則言語の集合 R に含まれる場合，q' へ遷移が行われる．3 つ目の規則は，文字を消費する先読みによる遷移の規則である．$(q, a, q', R) \in \Delta$ のとき，入力文字列 aw' が先読みの集合である R に含まれる場合のみ，文字 a によって q' へ遷移する．この遷移関係の定義は，Sakarovitch による定義を先読み付きオートマトンに拡張したものである [6]．

先読み付きオートマトンは先読みを含まないオートマトンに変換できるため，先読み付きオートマトンの表す言語は正則である．

例 2.1（先読み付きオートマトンの例）．$\Sigma = \{a, b\}$ として，1 つ以上の a で終わる文字列を受理するオートマトンを考える．このオートマトンは先読みを用いると下図で表される．ここで，遷移のラベルにおける $w \parallel r$ の r は先読み正則言語を表す．簡単のため，$r = \Sigma^*$ のとき，先読みの表記は省略することとする．



ここで，$r_a = \{a, aa, aaa, \cdots\}$ は 1 つ以上の a からなる正則言語の集合である．したがって，$a \parallel r_a$ は残りの文字列が r_a に含まれるとき，a で遷移することを表している．同様に，$a \parallel \overline{r_a}$ は残りの文字列が r_a に含まれないとき，a で遷移することを表している．このオートマトン A において，文字列 $abaa$ のうち aba を読んで，状態 q_0 から状態 q_1 へ遷移することを考える．

$$q_0 \xrightarrow[A]{aba/a} q_1$$

この遷移を 1 ステップごとに見ていく．初めの文字 a による遷移をするとき，文字列 $abaa$ は r_a に含まれないため，以下の遷移をする．

$$q_0 \xrightarrow[A]{a/baa} q_0$$

次に，b によって以下の遷移をする．

$$q_0 \xrightarrow[A]{b/aa} q_0$$

次に，a によって遷移が行われる．ここで，残りの文字列は r_a に含まれるため，以下の遷移が行われる．

$$q_0 \xrightarrow[A]{a/a} q_1$$

先読み付きオートマトン A による遷移はこのようにして行われる．

3.　先行研究

Sakuma らによる先行研究 [7] では，正規表現によるマッチングの部分マッチの取り出しを行うためにトランスデューサへの変換が行われている．正規表現マッチングの部分マッチの曖昧さを排除するために，正規表現マッチングの意味論をリストモナドを用いて優先順位を持つ非決定性構文解析器として定義している．優先順位を持ったままではトランスデューサを構成することができないため，先読みの導入によって優先順位を排除した決定性構文解析器を導出，それをもとにトランスデューサの構成が行われている．本研究では，Sakuma らによる意味論の定義を利用してアトミックグループで拡張された正規表現について考える．

ここでは，以下の構文の正規表現を考える．

$$\begin{array}{lll} r ::= \epsilon & & \text{（空列）} \\ \quad \mid\ c & & \text{（文字）} \\ \quad \mid\ r_1 r_2 & & \text{（連接）} \\ \quad \mid\ r_1 | r_2 & & \text{（選択）} \\ \quad \mid\ r_1^* & & \text{（繰返し）} \end{array}$$

正規表現 r が表す言語 $L(r)$ は次のように帰納的に定義する．

$$L(\epsilon) = \{\epsilon\}$$
$$L(c) = \{c\}$$
$$L(r_1 r_2) = \{w_1 w_2 \mid w_1 \in L(r_1) \wedge w_2 \in L(r_2)\}$$
$$L(r_1|r_2) = L(r_1) \cup L(r_2)$$
$$L(r_1^*) = L(r_1)^*$$

プログラミング言語における正規表現はマッチングに優先順位が存在する．選択 $r_1|r_2$ では r_1 が r_2 よりも優先順位が高く，どちらにもマッチする文字列の場合，r_1 によってマッチする．繰返し r_1^* では，$r_1^* = r_1 r_1^*|\epsilon$ と展開される．そのため，より多くの r_1 によるマッチを試みる．Sakuma らによる先行研究では，キャプチャした文字列を取り出すための手法としてトランスデューサを構成しており，部分的なマッチの曖昧さを排除するために優先順位を考慮している．また，アトミックグループの意味を定めるためには，優先順位の議論を避けがたいため，本研究でもマッチングの優先順位を考慮して意味を定める必要がある．Sakuma らはプログラミング言語おける正規表現によるマッチングを精確にとらえるために，正規表現の意味論をリストモナド [11] を用いて非決定性構文解析器として定式化している．リストモナドにおける unit と bind の定義は付録 A.1 に示す．非決定性構文解析器は，正規表現 r および文字列 w に対し，$w = w_1 w_2$，$w_1 \in L(r)$ なる w_2 のリストを返す関数として定義する．リストの先頭に向かうほどマッチする文字列の優先順位が高いものとする．これによって，プログラミング言語における正規表現によるマッチングを意味を定義する．

定義 3.1（非決定性構文解析器）．正規表現によるマッチングの意味論を非決定性構文解析器 $N[\![r]\!]w$ として定義する．

$$N[\![r]\!] :: \Sigma^* \to \Sigma^*\ list$$
$$N[\![\epsilon]\!]w = \text{unit } w$$
$$N[\![c]\!]w = \begin{cases} \text{unit } w' & \text{if } w = cw' \\ [] & \text{otherwise} \end{cases}$$
$$N[\![r_1 r_2]\!]w = N[\![r_1]\!]w \gg\!= \lambda w'.N[\![r_2]\!]w'$$
$$N[\![r_1|r_2]\!]w = N[\![r_1]\!]w +\!+ N[\![r_2]\!]w$$
$$N[\![r_1^*]\!]w = (N[\![r_1]\!]w \gg\!= \lambda w'.N[\![r_1^*]\!]w') +\!+ \text{unit } w$$

ここで，$+\!+$ はリストの連結を表している．

非決定性構文解析器 $N[\![r]\!]w$ は優先順位を持つ．トランスデューサでは優先順位を表現することができないため，$N[\![r]\!]w$ から優先順位を取り除く必要がある．そこで，先読みを用いることによって正規表現がマッチする文字列を決定的にした決定性構文解析器を定義する．先読みの正規表現 r_c は，正規表現 r に続く正規表現である．決定性構文解析器 $D[\![r]\!]_{r_c}w$ は，正規表現が表す言語のうち，最も優先順位が高い文字列 w_1 に対して $w = w_1 w_2$ なる w_2 を返す関数である．決定性構文解析器にはオプションモナドを用いる．オプションモナドにおける unit と bind の定義は付録 A.1 に示す．

定義 3.2（決定性構文解析器）．最も優先度の高い結果だけを返す決定性構文解析器を定義する．

$$D[\![r]\!]_{r_c} :: \Sigma^* \to \Sigma^*\ option$$
$$D[\![\epsilon]\!]_{r_c}w = \begin{cases} \text{unit } w & \text{if } w \in L(r_c) \\ \text{None} & \text{if } w \notin L(r_c) \end{cases}$$
$$D[\![c]\!]_{r_c}w = \begin{cases} \text{unit } w' & \text{if } w = cw' \\ & \text{and } w' \in L(r_c) \\ \text{None} & \text{otherwise} \end{cases}$$
$$D[\![r_1 r_2]\!]_{r_c}w = D[\![r_1]\!]_{r_2 r_c}w \gg\!= \lambda w'.D[\![r_2]\!]_{r_c}w'$$
$$D[\![r_1|r_2]\!]_{r_c}w = \begin{cases} D[\![r_1]\!]_{r_c}w & \text{if } w \in L(r_1 r_c) \\ D[\![r_2]\!]_{r_c}w & \text{if } w \notin L(r_1 r_c) \end{cases}$$
$$D[\![r_1^*]\!]_{r_c}w = \begin{cases} D[\![r_1]\!]_{r_1^* r_c}w \gg\!= \lambda w'.\ D[\![r_1^*]\!]_{r_c}w' \\ \quad \text{if } w \in L(r_1 r_1^* r_c) \\ \text{if } w \in L(r_c) \text{ then unit } w \text{ else None} \\ \quad \text{if } w \notin L(r_1 r_1^* r_c) \end{cases}$$

決定性構文解析器 $D[\![r]\!]_{r_c}w$ が正規表現マッチングの意味論に合致することを示す．関数 filter と関数 first の合成である関数 find を次のように定義する．ここで，first m はリスト m の先頭の要素をオプションで返す関数で，関数 filter p m はリスト m のうち，p の条件を満たす要素のリストを返す関数である．

$$\text{first } m = \text{case } m \text{ of } \ e :: l \Rightarrow \text{Some } e \ \mid\ [] \Rightarrow \text{None}$$
$$\text{find } p\ m = \text{first}(\text{filter } p\ m)$$

このとき，次の定理が成り立つ．

定理 3.3. $D[\![r]\!]_{r_c}w = \text{find } (\lambda w'.w' \in L(r_c))(N[\![r]\!]w)$

Sakuma らは，この決定性構文解析器をもとに先読み付きトランスデューサを構成している．本研究では，Sakuma らの先読み付きトランスデューサの構成のすべての出力を ϵ とした先読み付きオートマトンの構成を利用する．この先読み付きオートマトンの構成は 5 章で示す．

繰返し r_1^* において，$\epsilon \in L(r_1)$ であるとき，定義 3.1 の非決定性構文解析器では $r^* = rr^*|\epsilon$ が無限に展開されてしまう．この問題に対応するために，先行研究では，以下のような定義を精密化している．

$$N[\![r_1^*]\!]w = (N[\![r_1]\!]w \gg\!= \lambda w'.\text{if } w = w' \text{ then unit } w \text{ else } N[\![r_1^*]\!]w') +\!+ \text{unit } w$$

アトミックグループに関する意味の定義およびオートマトンの構成には，この問題はほとんど影響がないため，本論文ではこの問題は無視し，繰返し r_1^* において，$\epsilon \notin L(r_1)$

なる場合のみを議論する．一般の場合は，Sakuma らによる構成をもとにした拡張で得られる．

4. アトミックグループによって拡張された正規表現

プログラミング言語における正規表現の拡張の 1 つとして，アトミックグループという構文がある．Sakuma らによる先行研究では，アトミックグループの意味論は与えられているが，トランスデューサへの変換は与えられていない．また，アトミックグループで拡張された正規表現が表す言語が正則であるかの議論は行われていない．そこで，本研究ではアトミックグループで拡張された正規表現からオートマトンへの変換を与え，拡張された正規表現が表す言語が正則であることを示す．本章では，オートマトンを構成するために，アトミックグループで拡張された正規表現の表す言語を集合モナドを用いた非決定性構文解析器で表す．オートマトンの構成は，この非決定性構文解析器に関する等式から導かれる．

前章での正規表現の構文に以下を加える．

$r ::= \ldots \mid (r_1)^{atomic}$　　（アトミックグループ）

先行研究の手法を用いて，アトミックグループで拡張された正規表現によるマッチングの意味論について考える．初めに，アトミックグループの意味論を $N[\![r]\!]w$ で定式化する．

$$N[\![(r_1)^{atomic}]\!]w = \text{case } N[\![r_1]\!]w \text{ of } e :: l \Rightarrow [e] \mid [] \Rightarrow []$$

この定義は Sakuma らによる先行研究において与えられている．アトミックグループはグループ内の最も優先順位の高いマッチだけを返す構文と考えることができるため，上のように定義できる．

例 4.1. $(ab^*)^{atomic}$ の場合を考える．

$$\begin{aligned} &N[\![(ab^*)^{atomic}]\!]abb \\ &\quad = (\text{case } N[\![ab^*]\!]abb \text{ of } e :: l \Rightarrow [e] \mid [] \Rightarrow []) \\ &\quad = (\text{case } [\epsilon, b, bb] \text{ of } e :: l \Rightarrow [e] \mid [] \Rightarrow []) \;=\; [\epsilon] \end{aligned}$$

例 4.2. $N[\![(a^*)^{atomic}b]\!]a \cdots ab = [\epsilon] = N[\![a^*b]\!]a \cdots ab$ となる．どちらの正規表現もこの形の文字列にしかマッチしないため，$N[\![(a^*)^{atomic}b]\!] = N[\![a^*b]\!]$ である．等価な正規表現となっているが，アトミックを用いたものの方が，マッチが失敗する場合のバックトラックが禁止されているため効率的である．このような最適化のため，アトミックグループは導入されている．

文字列 w は，正規表現 $(r_1|r_2)^{atomic}$ に対して，$w \in L(r_1)$ または $w \notin L(r_1) \wedge w \in L(r_2)$ のとき，この正規表現の表す文字列であると考えることができる．このように，アトミックグループとは，グループ内の各選択について，「より優先順位の高い選択肢がマッチしない」ことを表す否定先読みを加えることに等しい．この「先読みを加えることで優先順位を宣言的に扱う」というアプローチは，Sakuma らによる，正規表現の部分マッチのキャプチャの定式化に用いられたものと同じである．よって，Sakuma らの構成をたどることで，アトミックグループの扱いも同様に与えることができると予想される．

アトミックグループで拡張された正規表現は通常の正規表現のように構造に関して帰納的に言語を定義することができない．そのため，拡張された正規表現の言語を定義し直し，それを用いて決定性構文解析器を再定義する必要がある．まず拡張された正規表現の言語を定義するために集合モナドを用いた非決定性構文解析器 $L[\![r]\!]$ を定義する．$L[\![r]\!]w$ はリストモナドから集合モナドへのモナド射 set を用いて以下のように定義する．

$$L[\![r]\!] :: \Sigma^* \to \Sigma^* \; set$$
$$L[\![r]\!]w = \text{set } (N[\![r]\!]w)$$

拡張された正規表現 r が表す言語を，$L[\![r]\!]w$ を用いて定義する．$\epsilon \in L[\![r]\!]w$ であるとき，w は r の表す言語に含まれることを意味する．したがって，そのような文字列の集合を言語 $L(r)$ と定義する．

$$L(r) = \{w \mid \epsilon \in L[\![r]\!]w\}$$

アトミックグループを含まない正規表現に関しては，この定義と前章の定義は一致する．

非決定性構文解析器 $L[\![r]\!]w$ に対応するオートマトンを構成すれば，アトミックグループで拡張された正規表現のオートマトンへの変換を与えることができる．しかし，$L[\![r]\!]w$ の定義はアトミックグループの場合の構造において $N[\![r]\!]w$ に依存している．オートマトンは優先順位を表現することができないため，$N[\![r]\!]w$ に依存した形のままでは，オートマトンを構成することができない．アトミックグループで拡張された正規表現をオートマトンに変換するために，アトミックグループ内のマッチを決定性構文解析器 $D[\![r]\!]_{r_c}w$ を用いて表したい．アトミックグループ内のマッチをこのように表すことができれば，Sakuma らによるトランスデューサの構成をオートマトンの構成として利用できる．先ほど定義した拡張された正規表現が表す言語を用いて，先読みを用いた決定性構文解析器の定義をアトミックグループに対して拡張する．アトミックグループ以外の構文に対しては，決定性構文解析器 $D[\![r]\!]_{r_c}$ の定義は，定義 3.2 で与えたものがそのまま適用できる．ただし，先読みとして現れる正規表現 r の言語 $L(r)$ の定義としては，上の定義を用いる必要がある．

次に，アトミックグループに対する決定性構文解析器の定義を考える．正規表現 $(r_1)^{atomic}$ に対するマッチングでは，アトミックグループに続く正規表現にかかわらず，アトミックグループ内で最も優先度が高い結果を返せばよい．

したがって，先読み正規表現として，任意の文字列にマッチする Σ^* を用いて，$D[\![r_1]\!]_{\Sigma^*}w$ を実行すればよい．さらに，残りの文字列が先読み正規表現 r_c の表す言語に含まれなければならないので，次のように定義できる．

$$D[\![(r_1)^{atomic}]\!]_{r_c}w = \begin{cases} \text{Some } w' & \text{if } D[\![r_1]\!]_{\Sigma^*}w = \text{Some } w' \\ & \text{and } w' \in L(r_c) \\ \text{None} & \text{otherwise} \end{cases}$$

以上で定義したアトミックグループで拡張された正規表現に対する決定性構文解析器は正しい実装となっている．

定理 4.3. 拡張された正規表現 r，r_c と拡張された意味論に対して以下が成り立つ．

$$D[\![r]\!]_{r_c}w = \text{find } (\lambda w'.w' \in L(r_c))(N[\![r]\!]w)$$

証明．r の構造と w の長さに関する辞書式順序による帰納法で証明する．ここでは，$r = (r_1)^{atomic}$ の場合についてのみ示す．他の構造についての証明は Sakuma らによる証明に準ずる．

$$\begin{aligned}
&\text{find } (\lambda w'.w' \in L(r_c))\ (N[\![(r_1)^{atomic}]\!]w) \\
&\quad = \text{find } (\lambda w'.w' \in L(r_c)) \\
&\qquad (\text{case } (N[\![r_1]\!]w) \text{ of } e :: l \Rightarrow [e] | [] \Rightarrow []) \\
&\quad = \text{find } (\lambda w'.w' \in L(r_c)) \\
&\qquad (\text{case } (\text{find } (\lambda w'.w' \in \Sigma^*)\ N[\![r_1]\!]w) \text{ of} \\
&\qquad \text{Some } e \Rightarrow [e] \mid \text{None } \Rightarrow []) \\
&\quad = \text{find } (\lambda w'.w' \in L(r_c)) \\
&\qquad (\text{case } D[\![r_1]\!]_{\Sigma^*}w \text{ of} \\
&\qquad \text{Some } e \Rightarrow [e] \mid \text{None } \Rightarrow []) \\
&\qquad \text{（帰納法の仮定より）} \\
&\quad = \begin{cases} \text{Some } w' & \text{if } D[\![r_1]\!]_{\Sigma^*}w = \text{Some } w' \\ & \text{and } w' \in L(r_c) \\ \text{None} & \text{otherwise} \end{cases}
\end{aligned}$$

□

これによって，アトミックグループ内のマッチングを決定性構文解析器で表すことができる．

$$\begin{aligned}
&N[\![(r_1)^{atomic}]\!]w \\
&\quad = \text{case } N[\![r_1]\!]w \text{ of } w' :: l \Rightarrow [w'] \mid [] \Rightarrow [] \\
&\quad = \text{case } D[\![r_1]\!]_{\Sigma^*}w \text{ of Some } w' \Rightarrow [w'] \mid \text{None} \Rightarrow []
\end{aligned}$$

本論文では，この拡張された正規表現に対する意味と決定性構文解析器の定義から，拡張された正規表現が表す言語を受理するオートマトンの構成法を導出する．そのために，まず，言語を定義するための非決定性構文解析器 $L[\![r]\!]w$ の再帰的な定義を導く．非決定性構文解析器 $N[\![r]\!]w$ の定義と関数 set がモナド射であることから以下の等式が得られる．

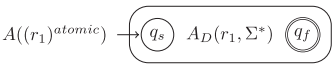
$$\begin{aligned}
&L[\![\epsilon]\!]w = \text{unit } w \\
&L[\![c]\!]w = \begin{cases} \text{unit } w' & \text{if } w = cw' \\ \emptyset & \text{otherwise} \end{cases} \\
&L[\![r_1r_2]\!]w = L[\![r_1]\!]w \gg\!= \lambda w'.L[\![r_2]\!]w' \\
&L[\![r_1|r_2]\!]w = L[\![r_1]\!]w \cup L[\![r_2]\!]w \\
&L[\![r_1^*]\!]w = (L[\![r_1]\!]w \gg\!= \lambda w'.\ L[\![r_1^*]\!]w') \\
&\qquad\qquad \cup \text{ unit } w \\
&L[\![(r_1)^{atomic}]\!]w = \text{case } D[\![r_1]\!]_{\Sigma^*}w \text{ of} \\
&\qquad\qquad \text{Some } w' \Rightarrow \text{unit } w' \mid \text{None} \Rightarrow \emptyset
\end{aligned}$$

アトミックグループ内のマッチングは決定性構文解析器によって表すことができる．すなわち，アトミックグループ内のマッチングと等価なオートマトンは決定性構文解析器をもとに構成すればよい．しかし，決定性構文解析器 $D[\![r]\!]_{r_c}w$ の定義は，正規表現の言語の定義に依存している．そのため，拡張された正規表現が表す言語は，非決定性構文解析器 $L[\![r]\!]w$ と決定性構文解析器 $D[\![r]\!]_{r_c}w$ との相互再帰によって表現されている．

5. オートマトンの構成

前章で定義した $L[\![r]\!]$ と $D[\![r]\!]_{r_c}$ から先読み付きオートマトンを構成する．初めに構成を示し，次に構成が相互再帰的な関数と等価であることを証明することによってアトミックグループで拡張された正規表現が正則であることを証明する．正規表現 r と先読み正規表現 r_c に対して，$L[\![r]\!]$ と $D[\![r]\!]_{r_c}$ に対応するオートマトン $A(r)$，$A_D(r, r_c)$ を構成する．簡単のため，オートマトンの遷移図では先読みが行われない場合，先読みの表記を省略し，加えて ϵ-遷移である場合にはラベルを省略する．

$L[\![r]\!]$ に対応するオートマトン $A(r)$ の構成は，アトミックグループの場合を除き，通常の ϵ-NFA の構成法である Thompson の構成法を用いる [9]．下にアトミックグループの場合の構成を示す．



前章でアトミックグループ内のマッチングは決定性構文解析器から構成されるオートマトンとすればよいことを示したので，アトミックグループが現れたとき，決定性構文解析器に対応する $A_D(r, \Sigma^*)$ を構成する．

次にアトミックグループ内のオートマトン $A_D(r, r_c)$ を $D[\![r]\!]_{r_c}$ の定義から構成する．$A_D(r, r_c)$ の構成は，Sakuma らによるトランスデューサの構成のすべての出力を ϵ としたオートマトンの構成を利用する．**図 1** に $A_D(r, r_c)$ の構成を示す．アトミックグループ内のオートマトン $A_D(r, r_c)$

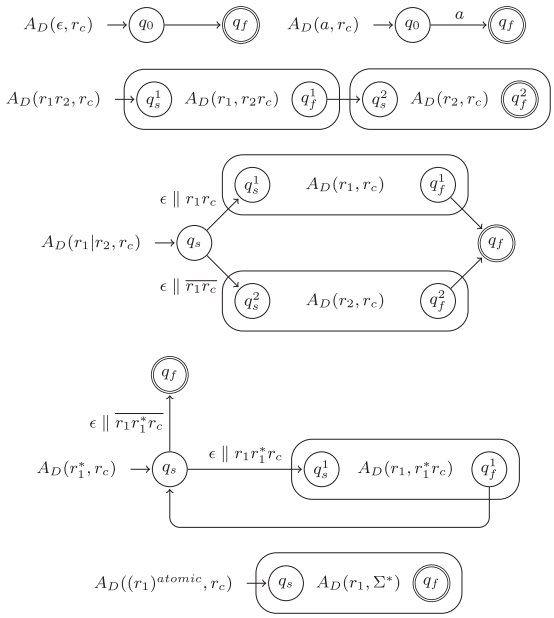


図 1　$A_D(r, r_c)$ の構成
Fig. 1　Construction of $A_D(r, r_c)$.

では，先読みによって遷移を決定的にする．マッチングの結果が複数存在しうる選択と繰返しにおいて，先読み正規表現にマッチする選択肢にのみに遷移する．これによって，マッチングの優先順位を実現している．

正規表現の意味論から導出した相互再帰的な等式とオートマトンの構成が等価であることを示す．これによって，アトミックグループで拡張された正規表現が表す言語が正則であることを示す．

定理 5.1（構成の正しさ）．$A(r)$ の初期状態を q_s，終了状態を q_f とする．このとき，以下が成り立つ．

$$w \in L(r) \quad \Longleftrightarrow \quad q_s \xrightarrow[A(r)]{w/\epsilon} q_f$$

この定理の証明は次の補題 5.2 からなる．

補題 5.2. $A(r)$ によって構成されるオートマトンの状態 q_s，状態 q_f に関して

$$w' \in L[\![r]\!]w \quad \Longleftrightarrow \quad q_s \xrightarrow[A(r)]{ww'^{-1}/w'} q_f$$

$A_D(r, r_c)$ によって構成されるオートマトンの状態 q_s，状態 q_f に関して

$$D[\![r]\!]_{r_c} w = \text{Some } w' \quad \Longleftrightarrow \quad q_s \xrightarrow[A_D(r,r_c)]{ww'^{-1}/w'} q_f$$

ここで，ww'^{-1} は文字列 w の末尾から文字列 w' を取り除いた文字列である．

証明は Sakuma らに準ずる．r の構造と w の長さに関する辞書式順序による帰納法で示すことができる．

例 5.3（オートマトンの構成例）．本研究で与えた構成によるアトミックグループで拡張された正規表現からオートマトンへの変換の例を示す．正規表現 $(a|a^*)^{atomic}b$ をオートマトンへ変換する場合を考える．初めに，この正規表現は $(a|a^*)^{atomic}$ と b の連接であるので，以下のようになる．

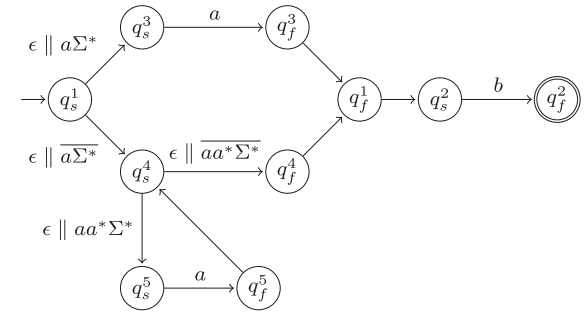
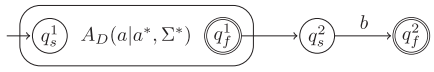


図 2　$A((a|a^*)^{atomic}b)$ の構成
Fig. 2　Construction of $A((a|a^*)^{atomic}b)$.



次に，アトミックグループ内の正規表現に対応するオートマトンを図 1 に従って構成する．これによって得られるオートマトンを**図 2** に示す．図 2 において，q_s^1 と q_s^4 からの遷移が先読みによって行われている．q_s^1 において，残りの文字列が a で始まる文字列ならば q_s^3 へ，そうでなければ q_s^4 へ遷移する．q_s^3 へ遷移した場合，残りの遷移で ab が消費されるので，受理される文字列は ab のみである．q_s^4 へ遷移した場合，残りの文字列が先頭に a が 1 つ以上ある場合 q_s^5 へ，そうでなければ q_f^4 へ遷移する．しかし，q_s^1 から残りの文字列の先頭が a でないという条件で遷移してきたため，q_s^4 からは必ず q_f^4 へ遷移する．したがって，q_s^1 から q_s^4 へ遷移する場合，受理される文字列は b のみである．以上より，このオートマトンが受理する文字列は $\{ab, b\}$ である．

6. 実装と実験

6.1 実装

アトミックグループで拡張された正規表現から，オートマトンへの変換を OCaml を用いて実装した．実装は，Perl 互換正規表現のうち一般的に用いられている文字クラス等の構文をサポートしている．

先読み付きオートマトン $A = (Q, \Sigma, \Delta, q_0, F)$ からそれと等価な先読みを含まないオートマトン A' への変換には，先読み付きオートマトンの状態にサブセット構成を用いる．k 個の先読み正規表現 $r_1, r_2, \ldots, r_k$ を持つ先読み付きオートマトンの変換を考える．$r_1, r_2, \ldots, r_k$ は，DFA $D_i = (Q_i, \Sigma, \delta_i, q_0^i, F_i)$ で表現されているとする．A' の状態は，$Q \times 2^{Q_1} \times 2^{Q_2} \times \cdots \times 2^{Q_k}$ とする．前章の構成法で作られる先読み付きオートマトン A と等価な先読みを含まないオートマトン A' は，以下のどちらかの形式の遷移を持つ．

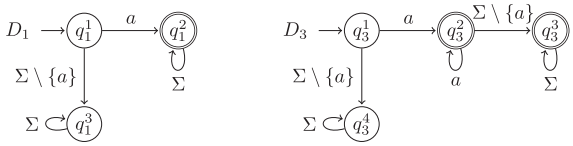
A において，先読み r_i を持つ状態 q から q' への ϵ-遷

移がある場合には，A' は $(q, S_1, S_2, \ldots, S_k)$ から $(q', S_1, S_2, \ldots, S_i \cup \{q_0^i\}, \ldots, S_k)$ への ϵ-遷移を持つ．

A において，先読みを持たない状態 q から q' への入力記号 a に対する遷移がある場合には，A' は $(q, S_1, S_2, \ldots, S_k)$ から $(q', \delta_1(S_1, a), \delta_2(S_2, a), \ldots, \delta_k(S_k, a))$ への遷移を持つ．ここで，δ_i は状態の集合を扱うための拡張がされているものとする．

状態 $(q, S_1, S_2, \ldots, S_k)$ は，$q \in F$ かつ $S_i \subseteq F_i$ のとき，A' の終了状態となる．また，初期状態は $(q_0, \emptyset, \ldots, \emptyset)$ である．A' の各状態 $(q, S_1, S_2, \ldots, S_k)$ に新たに名前をつけると A' は ϵ-NFA に等しい．

例 6.1. 例 5.3 の先読み付きオートマトンを先読みなしのオートマトン A' に変換する．例 5.3 の先読み付きオートマトンにおいて，先読みによる遷移は $a\Sigma^*$，$\overline{a\Sigma^*}$，$aa^*\Sigma^*$，$\overline{aa^*\Sigma^*}$ の 4 つである．これらを r_1 から r_4 とする．r_1 と r_3 に対応する DFA D_1，D_3 を以下に示す．



r_2 に対応する DFA D_2 は D_1 の，r_4 に対応する DFA D_4 は D_3 の終了状態と終了状態ではない状態を入れ替えたものである．先読みなしのオートマトンにおいて $w = ab$ が与えられた場合を考える．初期状態は $(q_s^1, \emptyset, \emptyset, \emptyset, \emptyset)$ である．初期状態において，先読み付きオートマトンの状態 q_s^1 からの遷移が先読みである．したがって，先読み付きオートマトンの状態 q_s^1 を q_s^3 へ ϵ-遷移し，先読み正規表現 r_1 に対応する D_1 の初期状態 q_1^1 を S_1 に追加する．これを遷移先の状態とする．

$(q_s^3, \{q_1^1\}, \emptyset, \emptyset, \emptyset)$

次に，文字 a による遷移をする．先読み付きオートマトンの状態 q_s^3 と D_1 の状態 q_1^1 から a によって遷移する状態によって，次の状態を得る．

$(q_f^3, \{q_1^2\}, \emptyset, \emptyset, \emptyset)$

次に ϵ-遷移によって，次の状態へ遷移する．

$(q_s^2, \{q_1^2\}, \emptyset, \emptyset, \emptyset)$

先読み付きオートマトンの状態 q_s^3 と D_1 の状態 q_1^1 から b によって遷移する状態によって，次の状態を得る．

$(q_f^2, \{q_1^2\}, \emptyset, \emptyset, \emptyset)$

この状態において，先読み付きオートマトンの状態 q_f^2 は先読み付きオートマトン A の終了状態であり，$S_1 = \{q_1^2\}$ は DFA D_1 の終了状態 F_1 の部分集合である．したがって，文字列 ab は例 5.3 の先読み付きオートマトン A と等価な先読みなしのオートマトン A' で受理されることが分かる．

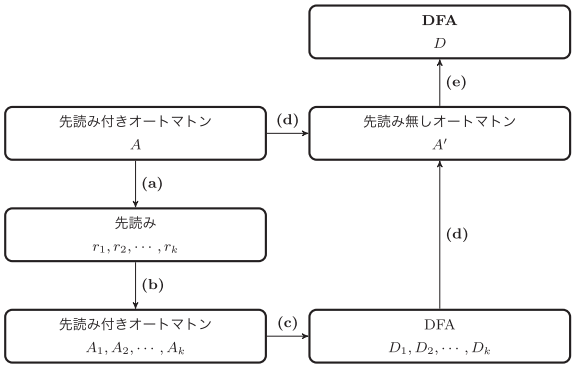


図 3　$A(r)$ の実装
Fig. 3　Implementation of $A(r)$.

アトミックグループが入れ子になる場合は，先読み正規表現にアトミックグループが現れる．その場合，まず，先読み正規表現に対応するオートマトン A' を構成し，A' を DFA に変換することで入れ子のアトミックグループが存在する正規表現もオートマトンに変換することができる．オートマトン A の構成法の概略を**図 3** に示す．

(a)　$A(r)$ と $A_D(r', \Sigma^*)$ によって構成された先読み付きオートマトン A に現れる先読み正規表現 $r_1, r_2, \cdots, r_k$ を集める．

(b)　先読み正規表現 $r_1, r_2, \cdots, r_k$ から先読み付きオートマトン A_i を構成する．先読み正規表現から先にオートマトンを構成していくため，アトミックグループが入れ子になっている場合でもオートマトンを構成することができる．

(c)　A_i を DFA D_i に変換する．A_i は ϵ-NFA と同じであるため，一般的な方法で DFA に変換可能である．

(d)　先読み付きオートマトン A と先読みの DFA から先読み付きオートマトンと等価な先読みを含まないオートマトン A' を構成する．

(e)　構成したオートマトン A' を DFA に変換する．

6.2　実験

アトミックグループを含む正規表現を DFA に変換する実験を行った．実験に使用した正規表現を**表 1** に示す．実験には，Perl ライブラリのアーカイブである CPAN で使用されている正規表現，および解説書 Mastering Regular Expressions 3rd Edition に収録されている正規表現を用いた．上記の解説書では，正規表現マッチングの最適化の手法としてアトミックグループについて詳しく解説している．しかしながら，アトミックグループを実際に使用しているプログラムはとても少なく，CPAN のプログラムで使用されている 1,887 個の正規表現のうち，アトミックグループを使用している正規表現は 2 個しかなかった．さらに，うち 1 つは後方参照を含んでいたため，CPAN で使用されていた正規表現は，ParseWords.pm で使用されている正規表

表 1　実験に使用した正規表現
Table 1　Regular expressions used for the experiment.

	正規表現	出典
(1)	`(")((?>[^\\"]*(?:\\.[^\\"]*)*)"|(')((?>[^\\']*(?:\\.[^\\']*)*)'`	ParseWords.pm
(2)	`(\.\d\d(?>[1-9]?))\d+`	Mastering Regular Expressions 3rd Edition
(3)	`\w+=(?>[^\n\\]*)(?>\\.[^\n\\]*)*`	Mastering Regular Expressions 3rd Edition

表 2　実験結果
Table 2　Experimental result.

	正規表現の大きさ	先読み付きオートマトンの状態数	先読みの数	DFA の状態数
(1)	25	34	6	5
(2)	14	18	1	7
(3)	15	20	2	4

現のみだった．

実験では，アトミックグループが使用されている正規表現の大きさ，先読み付きオートマトンの状態数，先読みの数，DFA の状態数を調べた．実験の結果を**表 2** に示す．ここで，表 2 中の先読み付きオートマトンの状態数は状態数削減等の処理は行っておらず，DFA の状態数は状態数最小化の処理を行った結果である．本実装においては，括弧における部分マッチのキャプチャは実装していないので，`(r)` は単にグルーピングをするものとする．また，キャプチャなし括弧 `(?:r)` も通常の括弧に変換して実験を行う．

表 1 の正規表現について，簡単に説明する．(1) の正規表現はダブルクォート，もしくはシングルクォートで囲まれた文字列を表す正規表現である．クォートに挟まれた部分の正規表現 `[^\\"]*(?:\\.[^\\"]*)*` は改行やエスケープされたクォート等，特殊な文字を含む文字列を表している．クォートに挟まれた部分の正規表現は意味的には `([^\\"]*|\\.)*` と等しいがループ展開 [4] と呼ばれる最適化の手法を用いて表現されている．`[^\\"]*(?:\\.[^\\"]*)*` による処理を終えた後，次に続く文字がクォートでなかった場合，無意味なバックトラックが発生してしまう．それを抑制し，高速化を図るためにアトミックグループが使用されている．(2) の正規表現は小数点以下 2 桁，もしくは 3 桁の数字をキャプチャするための正規表現である．小数点以下 3 桁目が 0 でない場合のみ，3 桁目までキャプチャをする．ここで，一度マッチした 3 桁目がバックトラックによって取り消されてしまうことを防ぐためにアトミックグループが使用されている．(3) の正規表現は，1 文字以上のアルファベットからなる文字列の後 =が続き，その後改行等の特殊文字を含む文字列が続く文字列を表す正規表現である．この正規表現の=の右側の部分は，(1) のクォートの内側の正規表現とほぼ同じである．

表 2 を見ると，先読み付きオートマトンの状態数に対して，DFA の状態数は小さくなっている．しかし，実験の対象が少ないため，先読み付きオートマトンの状態数や先読みの数に対して DFA の状態数がどのような傾向にあるかは実験からは判断できない．

7. 結論

本研究では，プログラミング言語における正規表現の拡張の 1 つであるアトミックグループの意味論について研究し，アトミックグループで拡張された正規表現のオートマトンへの変換を与えた．アトミックグループで拡張された正規表現によるマッチングの意味論を相互再帰的な 2 つの関数として表し，それをもとにオートマトンを構成した．構成したオートマトンの正しさは，定義した関数が正規表現の意味論と合致することと，関数とオートマトンが等価であることを示すことで証明した．この変換によって，アトミックグループを含む正規表現の等価性を検証することができる．アトミックグループで拡張された正規表現のオートマトンへの変換を OCaml を用いて実装し，アトミックグループを含む正規表現をオートマトンに変換する実験を行った．実験には CPAN で実際に使用されている正規表現を使用したが，アトミックグループを使用している正規表現はきわめて少なかった．アトミックグループとほぼ同じ働きをする絶対最大量指定子という構文を使用しているプログラムも存在しなかったため，一般的にこれらの構文はあまり使用されないといえる．

本研究の応用として，アトミックグループを含む正規表現を用いたプログラムの解析に利用できると考えていた．しかし，実際にアトミックグループを使用している正規表現は非常に少ないため，アトミックグループを含む正規表現を用いたプログラムの解析という目的に関しては，実用上はあまり意味がない．別の応用として，アトミックグループを使用していない正規表現をアトミックグループを用いた形に変換することで正規表現の最適化を行うことが考えられる．

参考文献

[1]　森畑明昌：先読み付き正規表現の有限状態オートマトンへの変換，コンピュータソフトウェア，Vol.29, No.1,

pp.147–158 (2012).
[2] Chandra, A.K., Kozen, D.C. and Stockmeyer, L.J.: Alternation, *J. ACM*, Vol.28, pp.114–133 (1981).
[3] Droste, M., Kuich, W. and Vogler, H.: *Handbook of Weighted Automata, 1st edition*, Springer (2009).
[4] Friedl, J.E.F.: *Mastering Regular Expressions, 3rd edition*, O'Reilly (2006).
[5] Moggi, E.: Computational lambda-calculus and monads, *Proc. 4th Annual Symposium on Logic in Computer Science* (*LICS*), pp.14–23 (1989).
[6] Sakarovitch, J.: *Elements of Automata Theory*, Cambridge University Press (2009).
[7] 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).
[8] 香川孝司：Monad Morphism による局所的状態の表現，コンピュータソフトウェア，Vol.11, No.5, pp.21–30 (1994).
[9] Thompson, K.: Regular Expression Search Algorithm, *Comm. ACM*, Vol.11, No.6, pp.419–422 (1968).
[10] Wadler, P.: Comprehending monads, *Mathematical Structures in Computer Science*, Vol.2, pp.461–422 (1992).
[11] Wadler, P.: The essence of functional programming, *the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp.1–14 (1992).

付　　録

A.1　各モナドにおける unit，bind の定義

- リストモナドにおける unit，bind 定義
  初めに，リストモナドを操作するための関数を定義する．

$$\text{map } f\ m = \begin{cases} [] & \text{if } m = [] \\ (f\ e)\ ::\ (\text{map } f\ m') & \\ & \text{if } m = e\ ::\ m' \end{cases}$$

$$\text{concat } l = \begin{cases} [] & \text{if } l = [] \\ m{+\!\!+}(\text{concat } l') & \text{if } l = m\ ::\ l' \end{cases}$$

  上記の関数を用いてリストモナドにおける unit 関数と bind 関数 ≫= を定義する．

$$\text{unit}_{\text{list}}\ :: \alpha \to \alpha\ \mathit{list}$$
$$\text{unit}_{\text{list}}\ x = [x]$$
$$\gg\!=_{\text{list}} :: \alpha\ \mathit{list} \to (\alpha \to \beta) \to \beta\ \mathit{list}$$
$$m \gg\!=_{\text{list}} f = \text{concat (map } f\ x)$$

- オプションモナドにおける unit，bind の定義

$$\text{unit}_{\text{option}}\ :: \alpha \to \alpha\ \mathit{option}$$
$$\text{unit}_{\text{option}}\ x = \text{Some } x$$
$$\gg\!=_{\text{option}} :: \alpha\ \mathit{option} \to (\alpha \to \beta) \to \beta\ \mathit{option}$$
$$m \gg\!=_{\text{option}} f = \text{case } m \text{ of Some } x \Rightarrow f\ x$$
$$|\ \text{None} \Rightarrow \text{None}$$

- 集合モナドにおける unit，bind の定義

$$\text{unit}_{\text{set}}\ :: \alpha \to \alpha\ \mathit{set}$$
$$\text{unit}_{\text{set}}\ x = \{x\}$$
$$\gg\!=_{\text{set}} :: \alpha\ \mathit{set} \to (\alpha \to \beta) \to \beta\ \mathit{set}$$
$$m \gg\!=_{\text{set}} f = \bigcup_{x \in m} fx$$

杉山 聡

2012 年筑波大学情報学群情報科学学類卒業．現在，同大学大学院システム情報工学研究科コンピュータサイエンス専攻修士課程在籍．

南出 靖彦　（正会員）

1993 年京都大学大学院理学研究科数理解析専攻修士課程修了．同年同大学数理解析研究所助手．1999 年筑波大学電子・情報工学系講師．2004 年筑波大学大学院システム情報工学研究科講師．2007 年同准教授．現在，筑波大学システム情報系准教授．博士（理学）．プログラミング言語およびソフトウェア検証に興味を持つ．