Journal of Information Processing Vol.27 422–430 (June 2019)

[DOI: 10.2197/ipsjjip.27.422]

Regular Paper

Derivatives of Regular Expressions with Lookahead

Takayuki Miyazaki[1,a)] Yasuhiko Minamide[1,b)]

Received: September 28, 2018, Accepted: December 24, 2018

Abstract: Lookahead is an extension of regular expressions that has been adopted in many implementations and is widely used. Lookahead represents what is allowed as the rest of input. Morihata developed a conversion from regular expressions with lookahead (REwLA) to deterministic finite automata by extending Thompson's construction. In this paper, we develop a conversion from REwLA to deterministic finite automata by extending derivatives of regular expressions. First, we formalize the semantics of REwLA. An REwLA has information about the rest of the input, so the definition of the semantics of REwLA is not languages but structures different from those of regular expressions. Thus, we introduce languages with lookahead as sets of pairs of strings with several operations and define the semantics of REwLA as languages with lookahead. Next, we define two kinds of left quotient for languages with lookahead and give corresponding derivatives. Then, we show that the types of expressions obtained by repeatedly applying derivatives are finite under some equivalence relation and give a conversion to deterministic finite automata. We also show that the semantics of REwLA is a finite union of sets of the form $A \times B$, where A and B are regular languages.

Keywords: regular expressions with lookahead, derivatives

1. Introduction

Regular expression matching is widely used for searching and replacing strings. In these implementations, standard regular expressions are extended with several operations. To accommodate such extensions, many implementations use backtracking, and such implementations have the weakness that the matching takes exponential time in the worst case. A method of matching without using backtracking is required. For related works, please refer to Refs. [6], [14], [19], [21].

One of the practically important operations added is lookahead. For example, the regular expressions representing comments of C language and passwords can be described concisely and clearly by using lookahead. Despite its importance, regular expressions with lookahead (REwLA) have not been studied sufficiently because of the difficulty of handling lookahead.

Lookahead is also widely used theoretically. For example, in the theory of parsing, the concept of lookahead has been used for a long time. Moreover, Refs. [7], [8] studied objects having lookahead. Because REwLA is the most fundamental object having lookahead, it is important to study the properties of REwLA.

Lookahead is a constraint that shows what comes subsequently. There are positive lookahead and negative lookahead. In many implementations, (?=e) and (?!e) denote positive lookahead of e and negative lookahead of e, respectively. In this paper, we adopt the notation of PEG [8], &e and !e denoting positive lookahead and negative lookahead, respectively. Positive lookahead &e represents a constraint that e comes subsequently, while negative lookahead represents a constraint that e does not come subsequently.

As an example, we explain the behavior of REwLA $a\&b$.

input is $a \implies$ failure

input is $aa \implies$ failure

input is $ab \implies$ success (match a, rest b)

input is $aba \implies$ success (match a, rest ba)

The former two cases fail because ε or a comes after a, and it is not a string beginning with b. The latter two cases succeed because b or ba comes after a, and it is a string beginning with b. Since the matching string does not contain the part matched with lookahead, the matching string is a.

Lookahead is mainly used to represent intersection and complement. For example, an REwLA representing the intersection of languages represented by e_1, e_2, and e_3 can be written as follows: $\&(e_1\$)\&(e_2\$)\&(e_3\$).^*\$$, where the dot "." represents an arbitrary letter, and \$ represents the end of a string. Furthermore, an REwLA representing the complement of the language represented by e can be written as $!(e\$).^*\$$.

As a practical example, an extended regular expression representing passwords can be written as the following.

$$\&(.^*[a\text{-}z])\,\&(.^*\backslash d)\,[a\text{-}z\backslash d]\{8, 100\}\$$$

This represents the set of strings of 8 to 100 letters including both lowercase letters [a-z] and digits $\backslash d$. Furthermore, by taking advantage of lookahead, an REwLA representing comments of the C language, which cannot be nested, can be written as the following.

$$/*\,(!(*/).)^*\,*/$$

Note that $*$ is a letter, and * represents iteration. This REwLA

[1] School of Computing, Tokyo Institute of Technology, Meguro, Tokyo 152–855, Japan
[a)] miyazaki.t.af@m.titech.ac.jp
[b)] minamide@is.titech.ac.jp

represents that a comment starts with $/*$, followed by characters such that each of them is not at the start of $*/$, and finally ends with $*/$.

A formal discussion about REwLA was initiated by Morihata [15], and he showed a conversion similar to Thompson's construction [22] from REwLA to Boolean automata [4]. A Boolean automaton is an automaton where the initial state is a logical formula of states, and the transition function is a function from states and letters to logical formulas of states. A similar extension of automata is also called alternating finite automata [5]. An REwLA can be converted to a deterministic finite automaton (DFA) that accepts the same language because a Boolean automaton can be converted to a DFA. To define the language of REwLA $L(e)$, Morihata introduced the function $L'(e, x)$ from REwLA and input strings to the set of remaining strings. His definition is operational in the sense that $L'(e^*, x)$ is defined recursively using $L'(e^*, x')$. This definition is different from the standard definition. Moreover, he did not discuss the properties of L'.

Therefore, in this paper, we call a set of pairs of matching strings and remaining strings as a language with lookahead and define several operations on languages with lookahead. We define the semantics of REwLA $B(e)$ as a function from REwLA to languages with lookahead, noting that our definition is close to the standard definition. We discuss the properties of languages with lookahead and show that languages with lookahead form a Kleene algebra with tests [12]. Furthermore, we show that the semantics of REwLA is a finite union of sets of the form $A \times B$ where A and B are regular languages. A similar representation appears also in recognizable relation [2] and ω regular languages [11].

In addition, we use derivatives [3] as a method of conversion. It is known that the conversion by derivatives is relatively easy to implement in programming languages with pattern matching, and the number of states of DFA obtained by derivatives is small [17].

We briefly explain derivatives. First, the left quotient is defined as $x^{-1}L = \{y \mid xy \in L\}$. Derivatives compute the left quotient on regular expressions. The set of states obtained by repeatedly applying derivatives is finite under some equivalence relation; thus, a regular expression can be converted to DFA with derivatives. Moreover, by using partial derivatives [1] of regular expressions, it has been shown that the number of states of the DFA is bounded by $2^{\|e\|} + 1$ under some equivalence relation, where $\|e\|$ represents the number of occurrence of letters.

In the case of languages with lookahead, we introduce letters with tilde $\widetilde{a}$ and define two types of left quotient. Two derivatives corresponding to each left quotient are extensions of derivatives and their auxiliary functions. Then, no auxiliary function is needed. The set of states obtained by derivatives is not finite in general, as is the case with regular expressions. However, we show that the number of states of the DFA is bounded by $2^{2^{\|e\|}} + 1$ under some equivalence relation.

In Section 2, we define language with lookahead and REwLA. In Section 3, we first define the left quotient and derivatives. Next, we show the upper bound $2^{2^{\|e\|}} + 1$ as the size of the states in the worst case and the lower bound $2^{2^{\Omega(\sqrt{m})}}$ where m is the size of REwLA. Lastly, we show conversion to DFA, equivalence checking, and that the semantics of an REwLA is a finite union of sets of the form $A \times B$ where A and B are regular languages. In Section 4, we briefly describe the implementation.

2. Regular Expressions with Lookahead

In this section, we introduce languages with lookahead and define REwLA.

2.1 Languages with Lookahead

First, we define languages with lookahead. A *language with lookahead* is a subset of $\Sigma^* \times \Sigma^*$. This subset contains what is accepted and what comes subsequently. For example, $\{a\} \times b\Sigma^*$ expresses that the matching string is a, and the remaining string starts with b.

The *concatenation* operation $\cdot$ is defined on languages with lookahead as the following.

$$R \cdot S = \{(xy, z) \mid (x, yz) \in R, (y, z) \in S\}$$

By considering only the matching part, if x is matched in R, and y is matched in S, then xy is matched in $R \cdot S$. By also considering the lookahead part, if x is matched in R, yz is allowed as the remaining string in R, y is matched in S, and z is allowed in S, then xy is matched in $R \cdot S$, and z is allowed in $R \cdot S$.

Languages with lookahead form a monoid under concatenation and the unit element $I = \{\varepsilon\} \times \Sigma^*$. A subset of I is just a constraint because it is expressed as $\{\varepsilon\} \times A$ and only matches ε. For $R, S \subseteq I$, we have $R \cdot S = R \cap S$. Therefore, the concatenation operation is commutative and idempotent for $R, S \subseteq I$. That is, $R \cdot S = S \cdot R$, and $R \cdot R = R$.

The *star* operation is defined in the same manner as for a usual language.

$$R^* = \bigcup_{i=0}^{\infty} R^i \qquad (R^0 = I,\ R^{n+1} = R^n \cdot R)$$

For the star operation, the following lemma holds.

Lemma 2.1.

$$R^* = I \cup (R \setminus I) \cdot R^*$$

Proof. Since $(I \cup S)^n = I \cup S \cup \cdots \cup S^n$ holds, $(I \cup S)^* = S^*$ is obtained. By substituting $(I \cup S)^*$ for S^* in $S^* = I \cup S \cdot S^*$, we obtain $(I \cup S)^* = I \cup S \cdot (I \cup S)^*$. By substituting $R \setminus I$ for S, we obtain what we wanted. □

We describe the lookahead operations. The *positive lookahead* operation & is defined as follows.

$$\&R = \{\varepsilon\} \times \{xy \mid (x, y) \in R\}$$

The language $\{xy \mid (x, y) \in R\}$ represents successful input for R. Thus, positive lookahead $\&R$ represents that successful input for R is allowed. For a positive lookahead operation, $\&(R \cup S) = \&R \cup \&S$ holds. If $R \subseteq I$, then $\&(R \cdot S) = R \cdot \&S$ holds. These properties are similar to the linear mapping of vector spaces. In fact, it can be regarded as a homomorphism of the left semimodule [10], which is a generalization of vector space. Furthermore, $\&(\&R) = \&R$ holds.

The *negative lookahead* operation ! is defined as the following.

$$!R = \{\varepsilon\} \times (\Sigma^* \setminus \{xy \mid (x, y) \in R\})$$

The negative lookahead can be written as $!R = I \setminus (\&R)$ by using positive lookahead and the set difference. Hence, for $R \subseteq I$, $!R = I \setminus R$ holds; thus, $(\mathcal{P}(I), \emptyset, I, \cup, \cdot, !)$ is a Boolean algebra. The positive lookahead is written as $\&R = !(!R)$ by using negative lookahead. Therefore, positive lookahead can be treated as an abbreviation. For negative lookahead, $!(R \cup S) = !R \cdot !S$ holds. If $R \subseteq I$, then $!(R \cdot S) = !R \cup !S$ holds.

Next, we define a Kleene algebra with lookahead.

Definition 2.2. A *Kleene algebra with lookahead* $(K, B, 0, 1, +, \cdot, ^*, !)$ satisfies the following.

(1) $B \subseteq K$

(2) $(K, 0, 1, +, \cdot, ^*)$ is Kleene algebra.

(3) $(B, 0, 1, +, \cdot, !)$ is Boolean algebra.

(4) $!(k_1 + k_2) = !k_1 \cdot !k_2$

(5) $!(b \cdot k) = !b + !k$

If $(K, B, 0, 1, +, \cdot, ^*, !)$ satisfies conditions (1), (2), and (3), it is called a Kleene algebra with tests [12]. Languages with lookahead $(\mathcal{P}(\Sigma^* \times \Sigma^*), \mathcal{P}(I), \emptyset, I, \cup, \cdot, ^*, !)$ form a Kleene algebra with lookahead.

2.2 Regular Expressions with Lookahead

We define *REwLA* by the following grammars:

$$e ::= \emptyset \mid \varepsilon \mid a \mid e|e \mid ee \mid e^* \mid !e,$$

where $a \in \Sigma$.

We define the *semantics of REwLA* $B(e)$ [*1]. B is a mapping from REwLA to languages with lookahead.

$$B(\emptyset) = \emptyset$$
$$B(\varepsilon) = I$$
$$B(a) = \{a\} \times \Sigma^*$$
$$B(e_1|e_2) = B(e_1) \cup B(e_2)$$
$$B(e_1e_2) = B(e_1) \cdot B(e_2)$$
$$B(e^*) = B(e)^*$$
$$B(!e) = !B(e)$$

We define the *language of REwLA* $L(e)$. L is a mapping from REwLA to languages.

$$L(e) = \{x \mid (x, \varepsilon) \in B(e)\}$$

This definition extends that of the language of regular expressions. For a regular expression r, $B(r) = L(r) \times \Sigma^*$ holds. Because the lookahead part is Σ^*, an arbitrary string can come subsequently, or there is no constraint.

$\|e\|$ is the number of occurrence of letters (elements of Σ) in REwLA e.

We define several abbreviations. First, the dot expression "." represents any letter and is an abbreviation for $a_1|\ldots|a_n$ when $\Sigma = \{a_1, \ldots, a_n\}$. Thus, $B(.) = \Sigma \times \Sigma^*$. The expression \$ represents the end of input and is an abbreviation for "!.". $B(\$) = \{\varepsilon\} \times \{\varepsilon\}$. Moreover, $B(e\$) = L(e) \times \{\varepsilon\}$. Positive lookahead $\&e$ is an abbreviation for $!(!e)$. $B(\&e) = \{\varepsilon\} \times \{xy \mid (x, y) \in B(e)\}$. Moreover, $B(\&(e\$)) = \{\varepsilon\} \times L(e)$. Let e^n be the expression in which e is concatenated n times. $B(e^n) = B(e)^n$.

REwLA given by the following grammar are called *logical expressions of REwLA*.

$$l ::= \emptyset \mid \varepsilon \mid l|l \mid ll \mid !e$$

The reason these REwLA are called logical expressions is that concatenation operation behaves similarly to intersection. For any l, $B(l) \subseteq I$ holds.

3. Construction of DFA by Derivatives

In this section, we first define left quotient and derivatives. Next, we show that the set of states obtained by repeatedly applying derivatives is finite under some equivalence relation. Afterward, we show the upper bound $2^{2^{\|e\|}} + 1$ of the size of the set of states in the worst case and discuss the lower bound. Lastly, we show the conversion to DFA and prove several theorems as its corollaries.

3.1 Left Quotient

Left quotient is defined as follows.

$$a^{-1}R = \{(x, y) \mid (ax, y) \in R\}$$
$$\widetilde{a}^{-1}R = \{(\varepsilon, y) \mid (\varepsilon, ay) \in R\}$$

Here, $a^{-1}R$ is the left quotient for the matching part, which is the operation to remove a from the matching part. This operation is an extension of the left quotient of usual languages without lookahead.

$\widetilde{a}^{-1}R$ is the left quotient for the remaining part. We write it as $\widetilde{a}^{-1}R$ using a letter with tilde as $\widetilde{a}$. Because it is the left quotient for the remaining part, it is the operation to remove a from the remaining part of a pair that does not consume any letters. Thus, it removes a only from a pair where the matching part is ε.

The definition is extended to a string $w \in (\Sigma \cup \widetilde{\Sigma})^*$ by the following.

$$\varepsilon^{-1}R = R$$
$$(cw)^{-1}R = w^{-1}(c^{-1}R) \quad (c \in \Sigma \cup \widetilde{\Sigma},\ w \in (\Sigma \cup \widetilde{\Sigma})^*)$$

The following holds from the definition.

$$(\widetilde{a}b)^{-1}R = \emptyset$$

Therefore, for $w \in (\Sigma \cup \widetilde{\Sigma})^* \setminus (\Sigma^*\widetilde{\Sigma}^*)$, $w^{-1}R = \emptyset$ holds. Furthermore, for $x\widetilde{y} \in \Sigma^*\widetilde{\Sigma}^*$, the following holds.

$$(x, y) \in R \iff (\varepsilon, \varepsilon) \in (x\widetilde{y})^{-1}R$$

From the above, in order to decide whether (x, y) is included in R, it is sufficient to calculate the left quotient of R by $x\widetilde{y}$ and check whether it includes the end of input $(\varepsilon, \varepsilon)$. If it is possible to calculate the left quotient on REwLA and judge whether the end of input is included, then we can implement matching of REwLA.

For the left quotient by letter a, the following holds.

[*1] B derives from behavior.

Lemma 3.1.

$$a^{-1}(R \cup S) = (a^{-1}R) \cup (a^{-1}S)$$
$$a^{-1}(R \cdot S) = (a^{-1}R) \cdot S \cup (\widetilde{a}^{-1}R) \cdot (a^{-1}S)$$
$$a^{-1}(R^*) = (a^{-1}R) \cdot R^*$$
$$a^{-1}(!R) = \emptyset$$

Proof. This lemma follows from the equational reasoning on sets. In the case of concatenation, $R \cdot S = (R \setminus I) \cdot S \cup (R \cap I) \cdot S$ is used. In the case of star, $R^* = I \cup (R \setminus I) \cdot R^*$ is used. □

For the left quotient by letter $\widetilde{a}$, the following holds.

Lemma 3.2.

$$\widetilde{a}^{-1}(R \cup S) = (\widetilde{a}^{-1}R) \cup (\widetilde{a}^{-1}S)$$
$$\widetilde{a}^{-1}(R \cdot S) = (\widetilde{a}^{-1}R) \cdot (\widetilde{a}^{-1}S)$$
$$\widetilde{a}^{-1}(R^*) = I$$
$$\widetilde{a}^{-1}(!R) = !(a^{-1}R \cup \widetilde{a}^{-1}R)$$

Proof. This lemma follows from the equational reasoning on sets. In the case of star, $R^* = I \cup (R \setminus I) \cdot R^*$ is used. In the case of negative lookahead, $R = (R \setminus I) \cup (R \cap I)$ is used. □

3.2 Derivatives

Derivatives calculate the left quotient on REwLA. There are two types of derivatives, d_a and $d_{\widetilde{a}}$, both of which are functions from REwLA to REwLA and are defined by mutual recursion.

Definition 3.3. The *derivative of e by a*, written $d_a e$, is defined as follows.

$$d_a\emptyset = \emptyset$$
$$d_a\varepsilon = \emptyset$$
$$d_a b = \begin{cases} \varepsilon & (a = b) \\ \emptyset & (a \neq b) \end{cases}$$
$$d_a(e_1|e_2) = d_a e_1 | d_a e_2$$
$$d_a(e_1 e_2) = (d_a e_1)e_2 | (d_{\widetilde{a}} e_1)(d_a e_2)$$
$$d_a(e^*) = (d_a e)e^*$$
$$d_a(!e) = \emptyset$$

The *derivative of e by* $\widetilde{a}$, written $d_{\widetilde{a}} e$, is defined as follows.

$$d_{\widetilde{a}}\emptyset = \emptyset$$
$$d_{\widetilde{a}}\varepsilon = \varepsilon$$
$$d_{\widetilde{a}} b = \emptyset$$
$$d_{\widetilde{a}}(e_1|e_2) = d_{\widetilde{a}} e_1 | d_{\widetilde{a}} e_2$$
$$d_{\widetilde{a}}(e_1 e_2) = (d_{\widetilde{a}} e_1)(d_{\widetilde{a}} e_2)$$
$$d_{\widetilde{a}}(e^*) = \varepsilon$$
$$d_{\widetilde{a}}(!e) = !(d_a e | d_{\widetilde{a}} e)$$

These derivatives are extensions of derivatives of regular expressions. Derivatives of regular expressions require an auxiliary function. In contrast, in REwLA, $d_{\widetilde{a}}$ is an extension of the auxiliary function, and the two derivatives only require the corresponding functions.

Example. $d_a(a^*)$, $d_a(a!b)$, and $d_{\widetilde{b}}(!b)$ are calculated as follows.

$$d_a(a^*) = (d_a a)a^* = \varepsilon a^*$$
$$d_a(a!b) = (d_a a)!b | (d_{\widetilde{a}} a)(d_a b) = \varepsilon!b|\emptyset\emptyset$$
$$d_{\widetilde{b}}(!b) = !(d_b b | d_{\widetilde{b}} b) = !(\varepsilon|\emptyset)$$

The following theorem is obtained from the lemma of the left quotient.

Theorem 3.4. (*Correctness of Derivatives*)

$$B(d_a e) = a^{-1}B(e)$$
$$B(d_{\widetilde{a}} e) = \widetilde{a}^{-1}B(e)$$

Derivatives are also extended to a string $w \in (\Sigma \cup \widetilde{\Sigma})^*$.

$$d_\varepsilon e = e$$
$$d_{cw} e = d_w(d_c e) \qquad (c \in \Sigma \cup \widetilde{\Sigma},\ w \in (\Sigma \cup \widetilde{\Sigma})^*)$$

For the extended derivative, the following holds.

$$B(d_w e) = w^{-1}B(e)$$

In addition, from the definition of derivative, $d_{\widetilde{a}} e$ is a logical expression of REwLA. Therefore, if $v \in (\Sigma \cup \widetilde{\Sigma})^* \setminus (\Sigma^*)$, then $d_v e$ is a logical expression of REwLA.

3.3 Conversion to Automata

It is possible to decide whether an REwLA accepts the end of input. ν is a function from REwLA to Boolean values and is defined as follows.

$$\nu(\emptyset) = \nu(a) = \mathit{false}$$
$$\nu(\varepsilon) = \nu(e^*) = \mathit{true}$$
$$\nu(e_1|e_2) = \nu(e_1) \vee \nu(e_2)$$
$$\nu(e_1 e_2) = \nu(e_1) \wedge \nu(e_2)$$
$$\nu(!e) = \neg\, \nu(e)$$

$\nu(e)$ is true if and only if $(\varepsilon, \varepsilon) \in B(e)$. Therefore, matching is achieved: $\nu(d_{x\widetilde{y}} e)$ determines whether $(x, y) \in B(e)$.

We describe the conversion to DFA. The set of states $Q(e)$ is defined as follows.

$$Q(e) = \{d_w e \mid w \in (\Sigma \cup \widetilde{\Sigma})^*\}$$

If $Q(e)$ is finite, then we can convert REwLA to DFA on $\Sigma \cup \widetilde{\Sigma}$. The DFA by derivatives, $\mathcal{A}(e_0)$, is defined as follows.

$$\mathcal{A}(e_0) = \langle Q(e_0), \Sigma \cup \widetilde{\Sigma}, \delta, e_0, F \rangle$$
$$F = \{e \in Q(e_0) \mid \nu(e)\}$$
$$\delta(e, c) = d_c e$$

If $Q(e_0)$ is finite, then the following holds.

$$L(\mathcal{A}(e_0)) \subseteq \Sigma^*\widetilde{\Sigma}^*$$
$$x\widetilde{y} \in L(\mathcal{A}(e_0)) \iff (x, y) \in B(e_0)$$

However, $Q(e_0)$ is not necessarily finite.

If some equivalence relation $\equiv$ exists such that the following is true:

$Q(e)/\equiv$ is finite,

$e_1 \equiv e_2 \implies \delta(e_1, c) \equiv \delta(e_2, c)$,

$e_1 \equiv e_2 \implies (e_1 \in F \iff e_2 \in F)$,

then it is possible to construct the DFA on the quotient of $\mathcal{A}(e)$ by the equivalence relation. In the next subsection, we show that such an equivalence relation exists.

3.4 Finiteness

A congruence on REwLA is an equivalence relation that satisfies the following.

$$e_1 \equiv e_2, e_3 \equiv e_4 \implies e_1|e_3 \equiv e_2|e_4$$
$$e_1 \equiv e_2, e_3 \equiv e_4 \implies e_1e_3 \equiv e_2e_4$$
$$e_1 \equiv e_2 \implies e_1^* \equiv e_2^*$$
$$e_1 \equiv e_2 \implies !e_1 \equiv !e_2$$

We define an equivalence relation that satisfies the conditions presented in the previous subsection.

Definition 3.5. Let $\equiv$ be a congruence generated by the following rule.

$$e_1|(e_2|e_3) \equiv (e_1|e_2)|e_3 \quad (|\text{ assoc}) \quad (1)$$
$$e_1|e_2 \equiv e_2|e_1 \quad (|\text{ comm}) \quad (2)$$
$$e|e \equiv e \quad (|\text{ idem}) \quad (3)$$
$$e|\emptyset \equiv e,\ \emptyset|e \equiv e \quad (\emptyset \text{ unit}) \quad (4)$$
$$e\varepsilon \equiv e,\ \varepsilon e \equiv e \quad (\varepsilon \text{ unit}) \quad (5)$$
$$e\emptyset \equiv \emptyset,\ \emptyset e \equiv \emptyset \quad (\emptyset \text{ zero}) \quad (6)$$
$$(e_1|e_2)e_3 \equiv (e_1e_3)|(e_2e_3) \quad (\text{right dist}) \quad (7)$$
$$e_1(e_2|e_3) \equiv (e_1e_2)|(e_1e_3) \quad (\text{left dist}) \quad (8)$$
$$e_1(e_2e_3) \equiv (e_1e_2)e_3 \quad (\cdot \text{ assoc}) \quad (9)$$
$$l_1l_2 \equiv l_2l_1 \quad (\cdot \text{ comm}) \quad (10)$$
$$ll \equiv l \quad (\cdot \text{ idem}) \quad (11)$$
$$!\emptyset \equiv \varepsilon \quad (!\emptyset \text{ rule}) \quad (12)$$
$$!\varepsilon \equiv \emptyset \quad (!\varepsilon \text{ rule}) \quad (13)$$
$$!(e_1|e_2) \equiv !e_1!e_2 \quad (!| \text{ rule}) \quad (14)$$
$$!(le) \equiv !l|!e \quad (!\cdot \text{ rule}) \quad (15)$$

Equations (1)–(9) are the rules of the idempotent semiring. Equations (10)–(13) are some rules of Boolean algebra. Equations (14) and (15) are the rules corresponding to Eqs. (4) and (5) of Kleene algebra with lookahead.

For regular expressions, the sufficient set of rules to prove finiteness is Eqs. (1)–(6). In Brzozowski's paper [3], it is shown that Eqs. (1)–(3) are sufficient. However, unlike our definition, the derivative of concatenation e_1e_2 is defined by case analysis on the auxiliary function. In order to prove the number of states is bounded by an exponential function, it is sufficient to add right distributivity (7). For REwLA, the sufficient set of rules to prove finiteness is Eqs. (1)–(6) and Eqs. (8)–(11). The sufficient set of rules to prove that the number of states is bounded by a double exponential function is Eqs. (1)–(15).

For the congruence $\equiv$, the following holds.

$$e_1 \equiv e_2 \implies B(e_1) = B(e_2)$$
$$e_1 \equiv e_2 \implies \nu(e_1) = \nu(e_2)$$
$$e_1 \equiv e_2 \implies d_ce_1 \equiv d_ce_2$$

The proposition for B follows because languages with lookahead form a Kleene algebra with lookahead. The proposition for ν follows from the proposition for B and $\nu(e) \iff (\varepsilon, \varepsilon) \in B(e)$. The proposition for d_c is proved by induction on $\equiv$.

We show that $Q(e_0)/\equiv$ is finite. First, we find the normal form of d_we. For $\emptyset, \varepsilon$, and a, the following holds.

$$d_w\emptyset = \emptyset$$
$$d_w\varepsilon = \begin{cases} \varepsilon & (w \in \widetilde{\Sigma}^*) \\ \emptyset & (\text{otherwise}) \end{cases}$$
$$d_wa = \begin{cases} a & (w = \varepsilon) \\ \varepsilon & (w \in a\widetilde{\Sigma}^*) \\ \emptyset & (\text{otherwise}) \end{cases}$$

Moreover, for $e_1|e_2$, the following holds.

$$d_w(e_1|e_2) = d_we_1|d_we_2$$

We define the abbreviation $\sum$ as follows:

$$\sum_{i=1}^{n} e_i = e_1|\ldots|e_n,$$

where $\sum_{i=1}^n e_i = \emptyset$ if $n = 0$. In addition, we write $\sum_{i=1}^n e_i\, e_i'$ for $\sum_{i=1}^n (e_i\, e_i')$. For derivatives, $d_a(\sum_{i=1}^n e_i) = \sum_{i=1}^n (d_ae_i)$ and $d_{\widetilde{a}}(\sum_{i=1}^n e_i) = \sum_{i=1}^n (d_{\widetilde{a}}e_i)$ hold.

In the case of concatenation, we find the following normal form of d_we.

Lemma 3.6. *For any w, there exist some n, $v_i \in (\Sigma \cup \widetilde{\Sigma})^* \setminus (\Sigma^*)$, and $w_i \neq \varepsilon$ such that the following holds.*

$$d_w(e_1e_2) \equiv \begin{cases} (d_we_1)e_2|\sum_{i=1}^n (d_{v_i}e_1)(d_{w_i}e_2) & (w \in \Sigma^*) \\ \sum_{i=1}^n (d_{v_i}e_1)(d_{w_i}e_2) & (\text{otherwise}) \end{cases}$$

Proof. This lemma is proved by induction on w. In the case of ε, we use the fact $\emptyset$ is the unit element. That is, $d_\varepsilon(e_1e_2) \equiv (d_\varepsilon e_1)e_2|\emptyset$. In the case of wc, we use the associativity of $|$. We show the case where $c = a$, and $d_w(e_1e_2) \equiv (d_we_1)e_2|\sum_{i=1}^n (d_{v_i}e_1)(d_{w_i}e_2)$.

$$\begin{aligned} d_{wa}(e_1e_2) &\equiv d_a((d_we_1)e_2|\textstyle\sum_{i=1}^n (d_{v_i}e_1)(d_{w_i}e_2)) \\ &\equiv (((d_{wa}e_1)e_2|(d_{w\widetilde{a}}e_1)(d_ae_2))| \\ &\qquad \textstyle\sum_{i=1}^n ((d_{v_ia}e_1)(d_{w_i}e_2)|(d_{v_i\widetilde{a}}e_1)(d_{w_ia}e_2))) \\ &\equiv (d_{wa}e_1)e_2|\textstyle\sum_{i=1}^{2n+1} (d_{v_i'}e_1)(d_{w_i'}e_2) \end{aligned}$$

□

In the case of negative lookahead, we find the following normal form of d_we.

Lemma 3.7. *For any w, there exist n and $w_i \neq \varepsilon$ such that the following holds.*

$$d_w(!e) \equiv \begin{cases} !((\sum_{i=1}^n d_{w_i}e)|d_we) & (w \in \widetilde{\Sigma}^*) \\ \emptyset & (\text{otherwise}) \end{cases}$$

Proof. This is proved by induction on w. It is proved in a manner similar to the case of concatenation. □

We define the abbreviation $\prod$ as follows:

$$\prod_{i=1}^{n} e_i = e_1 \dots e_n,$$

where $\prod_{i=1}^{n} e_i = \varepsilon$ if $n=0$. For derivatives, $d_{\widetilde{a}}(\prod_{i=1}^{n} e_i) = \prod_{i=1}^{n}(d_{\widetilde{a}} e_i)$ holds. Moreover, we have $d_a(\prod_{j=1}^{n} e_j) \equiv \sum_{i=1}^{n}(\prod_{j=1}^{n}(d_{w_{ij}} e_j))$, where $w_{ij} = \widetilde{a}$ $(i > j)$, a $(i = j)$, ε $(i < j)$.

In the case of the star, we find the following form of $d_w e$.

Lemma 3.8. *For any w, there exist n, m_i, $v_{ij} \in (\Sigma \cup \widetilde{\Sigma})^* \setminus (\Sigma^*)$, and $w_i \in \Sigma^+$ such that the following holds.*

$$d_w e^* \equiv \begin{cases} e^* & (w = \varepsilon) \\ \sum_{i=1}^{n}(\prod_{j=1}^{m_i}(d_{v_{ij}} e))(d_{w_i} e)e^* & (w \in \Sigma^+) \\ \sum_{i=1}^{n}(\prod_{j=1}^{m_i}(d_{v_{ij}} e)) & (\text{otherwise}) \end{cases}$$

Proof. This is proved by induction on w. The case of ε is clear. In the case of wc, we show the case where $c = a$, and $d_w e^* \equiv (d_v e)(d_{w_1} e)e^*$.

$$\begin{aligned} & d_{wa} e^* \\ & \equiv d_a((d_v e)(d_{w_1} e)e^*) \\ & \equiv (d_{va} e)(d_{w_1} e)e^* | (d_{v\widetilde{a}} e)(d_a((d_{w_1} e)e^*)) \\ & \equiv (d_{va} e)(d_{w_1} e)e^* | (d_{v\widetilde{a}} e)((d_{w_1 a} e)e^* | (d_{w_1\widetilde{a}} e)(d_a e)e^*) \\ & \equiv (d_{va} e)(d_{w_1} e)e^* | (d_{v\widetilde{a}} e)(d_{w_1 a} e)e^* | (d_{v\widetilde{a}} e)(d_{w_1\widetilde{a}} e)(d_a e)e^* \\ & \equiv (d_{v_{11}} e)(d_{w_1} e)e^* | (d_{v_{21}} e)(d_{w_2} e)e^* | (d_{v_{31}} e)(d_{v_{32}} e)(d_{w_3} e)e^* \end{aligned}$$

□

Theorem 3.9. *$Q(e)/\equiv$ is finite.*

Proof. Let $f(e) = |Q(e)/\equiv|$. Because $\equiv$ is idempotent and commutative, the following holds for each normal form of $d_w e$.

$$\begin{aligned} & f(\emptyset) = 1 \\ & f(\varepsilon) = 2 \\ & f(a) = 3 \\ & f(e_1|e_2) \le f(e_1) \times f(e_2) \\ & f(e_1 e_2) \le 2^{f(e_1)\times f(e_2)} \\ & f(e^*) \le 2^{2^{2\times f(e)}} + 1 \\ & f(!e) \le 2^{f(e)} + 1 \end{aligned}$$

Hence, $Q(e)/\equiv$ is finite.

We explain the case of the star. From Lemma 3.8, there exist n, m_i, $v_{ij} \in (\Sigma \cup \widetilde{\Sigma})^* \setminus (\Sigma^*)$, w_i, and $e_i \in \{(d_{w_i} e)e^*, \varepsilon\}$ such that the following holds.

$$d_w e^* \equiv \begin{cases} e^* & (w = \varepsilon) \\ \sum_{i=1}^{n}(\prod_{j=1}^{m_i}(d_{v_{ij}} e))e_i & (\text{otherwise}) \end{cases}$$

Then, $d_{v_{ij}} e$ is a logical expression of REwLA. Thus, the number of equivalence classes of $\prod_{j=1}^{m_i}(d_{v_{ij}} e)$ is less than or equal to $2^{f(e)}$. Because the number of equivalence classes of e_i is less than or equal to $f(e) + 1 \le 2^{f(e)}$, the number of equivalence classes of $(\prod_{j=1}^{m_i}(d_{v_{ij}} e))e_i$ is less than or equal to $2^{f(e)} \times 2^{f(e)} = 2^{2\times f(e)}$. Therefore, the number of equivalence classes of $d_w e^*$ is less than or equal to $2^{2^{2\times f(e)}} + 1$. □

We have proved that $Q(e)/\equiv$ is finite; thus, we obtained a conversion to DFA by derivatives. In the next two subsections, we discuss the number of states of the DFA. In the subsection of application, we show several theorems as corollaries of the conversion to DFA.

3.5 Upper Bound

We have proved $Q(e)/\equiv$ is finite. Although it is sufficient for DFA construction, we show a stronger result that

$$|Q(e)/\equiv| \le 2^{2^{\|e\|}} + 1.$$

This is shown in the following manner. First, for $w \neq \varepsilon$, we represent $d_w e$ in the form $\sum_{i=1}^{n}(\prod_{j=1}^{m_i} l_{ij})e_i$. Next, we identify the sets of expressions $V_m(e)$ and $V_l(e)$ that appear as e_i, l_{ij}. Lastly, we show $|Q(e)/\equiv| \le 2^{2^{\|e\|}} + 1$ because $|V_m(e)| + |V_l(e)| \le \|e\|$.

We generalize concatenation and negative lookahead. For a set E_1 of REwLA and an REwLA e_2, $E_1 e_2 = \{e_1 e_2 \mid e_1 \in E_1\}$. For a set E of REwLA, $!E = \{!e \mid e \in E\}$.

The set of REwLA, $V_m(e)$, is defined as follows.

$$\begin{aligned} & V_m(\emptyset) = \emptyset \\ & V_m(\varepsilon) = \emptyset \\ & V_m(a) = \{\varepsilon\} \\ & V_m(e_1|e_2) = V_m(e_1) \cup V_m(e_2) \\ & V_m(e_1 e_2) = V_m(e_1)e_2 \cup V_m(e_2) \\ & V_m(e^*) = V_m(e)e^* \\ & V_m(!e) = \emptyset \end{aligned}$$

If r is a regular expression, $V_m(r) \cup \{r\}$ is the set of regular expressions obtained by repeatedly applying partial derivatives [1].

The set of REwLA, $V_l(e)$, is defined as follows.

$$\begin{aligned} & V_l(\emptyset) = \emptyset \\ & V_l(\varepsilon) = \emptyset \\ & V_l(a) = \emptyset \\ & V_l(e_1|e_2) = V_l(e_1) \cup V_l(e_2) \\ & V_l(e_1 e_2) = V_l(e_1) \cup V_l(e_2) \\ & V_l(e^*) = V_l(e) \\ & V_l(!e) = !(V_m(e) \cup V_l(e)) \end{aligned}$$

$V_l(e)$ is a concept unique to REwLA because $V_l(r) = \emptyset$ for a regular expression r.

$V_m(e)$ and $V_l(e)$ satisfy $|V_m(e)| + |V_l(e)| \le \|e\|$. We confirm this in the case of concatenation.

$$\begin{aligned} & |V_m(e_1 e_2)| + |V_l(e_1 e_2)| \\ & = |V_m(e_1)e_2 \cup V_m(e_2)| + |V_l(e_1) \cup V_l(e_2)| \\ & \le |V_m(e_1)e_2| + |V_m(e_2)| + |V_l(e_1)| + |V_l(e_2)| \\ & \le \|e_1\| + \|e_2\| = \|e_1 e_2\| \end{aligned}$$

Lemma 3.10. *For any w, there exist n, m_i, $l_{ij} \in V_l(e)$, and*

$e_i \in V_m(e)$ *such that the following holds.*

$$d_w e \equiv \begin{cases} e & (w = \varepsilon) \\ \sum_{i=1}^{n} (\prod_{j=1}^{m_i} l_{ij}) e_i & (w \in \Sigma^+) \\ \sum_{i=1}^{n} (\prod_{j=1}^{m_i} l_{ij}) & (\text{otherwise}) \end{cases}$$

Proof. This is proved by induction on e. We use associativity and distributivity. We show a part of the case of concatenation. From Lemma 3.6, there exist N, $v_i \in (\Sigma \cup \widetilde{\Sigma})^* \setminus (\Sigma^*)$, and $w_i \neq \varepsilon$ such that the following holds.

$$d_w(e_1 e_2) \equiv \begin{cases} (d_w e_1) e_2 | \sum_{i=1}^{N} (d_{v_i} e_1)(d_{w_i} e_2) & (w \in \Sigma^*) \\ \sum_{i=1}^{N} (d_{v_i} e_1)(d_{w_i} e_2) & (\text{otherwise}) \end{cases}$$

We prove the case of $w \in \Sigma^*$ and $N = 1$. For the first half part $(d_w e_1) e_2$, from induction hypothesis, the following holds.

$$\begin{aligned} (d_w e_1) e_2 &\equiv (\sum_{i=1}^{n} (\prod_{j=1}^{m_i} l_{ij}) e'_i) e_2 \\ &\equiv \sum_{i=1}^{n} (\prod_{j=1}^{m_i} l_{ij})(e'_i e_2) \end{aligned}$$

For the latter half part $(d_{v_1} e_1)(d_{w_1} e_2)$, we use the following equality.

$$\begin{aligned} (d_{v_1} e_1)(d_{w_1} e_2) &\equiv (\sum_{i=1}^{n} (\prod_{j=1}^{m_i} l_{ij}))(\sum_{k=1}^{n'} (\prod_{j=1}^{m'_k} l'_{kj}) e'_k) \\ &\equiv \sum_{i=1}^{n} (\sum_{k=1}^{n'} (\prod_{j=1}^{m_i} l_{ij})(\prod_{j=1}^{m'_k} l'_{kj}) e'_k) \end{aligned}$$

The other cases are the same except for the negative lookahead. For the case of negative lookahead, the following holds from the rule of congruence for negative lookahead.

$$\begin{aligned} !(d_{w_1} e) &\equiv !(\sum_{i=1}^{n} (\prod_{j=1}^{m_i} l_{ij}) e_i) \\ &\equiv \prod_{i=1}^{n} !((\prod_{j=1}^{m_i} l_{ij}) e_i) && (\text{rules (12), (14)}) \\ &\equiv \prod_{i=1}^{n} ((\sum_{j=1}^{m_i} !l_{ij}) | !e_i) && (\text{rules (13), (15)}) \end{aligned}$$

Then, this case is shown by using associativity and distributivity. □

Theorem 3.11. $|Q(e)/\equiv| \leq 2^{2^{\|e\|}} + 1$

Proof. From the above lemma, for any w, there exist n, m_i, $l_{ij} \in V_l(e)$, and $e_i \in (V_m(e) \cup \{\varepsilon\})$ such that the following holds.

$$d_w e \equiv \begin{cases} e & (w = \varepsilon) \\ \sum_{i=1}^{n} (\prod_{j=1}^{m_i} l_{ij}) e_i & (\text{otherwise}) \end{cases}$$

Therefore, we have the following.

$$\begin{aligned} |Q(e)/\equiv| &\leq 2^{2^{|V_l(e)|}(|V_m(e)+1|)} + 1 \\ &\leq 2^{2^{|V_l(e)|+|V_m(e)|}} + 1 \\ &\leq 2^{2^{\|e\|}} + 1 \end{aligned}$$

This completes the proof of the theorem. □

In the proof of the upper bound, the form $\sum_{i=1}^{n} (\prod_{j=1}^{m_i} l_{ij}) e_i$ is essential. As a result, it is expected that the double exponential size is essential, and the number of states cannot be bounded by an exponential. In the next section, we show that this intuition is correct.

3.6 Lower Bound

When we convert an REwLA to a DFA, we estimate a lower bound of the number of states of the DFA in the worst case. We give a lower bound of $2^{2^{\Omega(\sqrt{m})}}$ where m is the size of REwLA.

Let p_i be an i th prime number; we consider the following REwLA.

$$T_n = .^*a\&((.^{p_1})^*\$) \ldots \&((.^{p_n})^*\$).^*\$$$

$B(T_n)$ is $\Sigma^* a (\Sigma^{p_1 \times \cdots \times p_n})^* \times \{\varepsilon\}$. The language of the corresponding DFA $L(\mathcal{A})$ is $\Sigma^* a (\Sigma^{p_1 \times \cdots \times p_n})^*$.

For the size of T_n, the following holds.

Lemma 3.12. *The size of REwLA T_n is $O(p_n^2)$.*

Proof. The size of T_n is $O(\sum_{i=1}^{n} p_i)$. If $i < j$ then $p_i < p_j$. Hence, $\sum_{i=1}^{n} p_i \leq \sum_{i=1}^{p_n} i \leq p_n^2$. □

For the size of DFA, the following holds.

Lemma 3.13. *The number of states of the minimum DFA on $(\Sigma \cup \widetilde{\Sigma})^*$ that satisfies $L(\mathcal{A}) = \Sigma^* a (\Sigma^N)^*$ is $2^N + 1$.*

Proof. 2^N states are needed to memorize whether a came at the $i + (\text{multiple of } N)$ th position for $1 \leq i \leq N$. In addition, the state of the empty set is necessary because the automaton does not accept strings containing $\widetilde{a}$. Thus, $2^N + 1$ states are necessary. □

We use the following lemma proved in Ref. [18].

Lemma 3.14. $\prod_{i=1}^{n} p_i = 2^{\Omega(p_n)}$

From the presented lemmas, the following theorem is obtained.

Theorem 3.15. *The lower bound of the number of states of DFA in the worst case is $2^{2^{\Omega(\sqrt{m})}}$, where m is the size of an REwLA.*

Proof. From Lemma 3.12, the size of T_n is $O(p_n^2)$. The number of states corresponding to the minimum DFA is $2^{\prod_{i=1}^{n} p_i} + 1$. It is $2^{2^{\Omega(p_n)}}$. □

As a result, we have given a lower bound $2^{2^{\Omega(\sqrt{m})}}$.

Morihata pointed out that a better lower bound is obtained by the application of research on XPath [16]. The lower bound is $2^{2^{\Omega(m)}}$ if the dot expression "." is introduced as a primitive. For $\Sigma = \{a, b, c, x_1, \ldots, x_m\}$, he analyzed $.^*a(b|.\&(.^*x_1)) \ldots (b|.\&(.^*x_m)).^*\$$.

The following results were shown in related work. The lower bound $2^{2^{\Omega(m)}}$ is shown for semi-extended regular expressions that are extended by intersection [9]. The lower bound is not bounded by an elementary function for extended regular expressions that are extended by complement [20].

As pointed out by Morihata [15], it is worth noting that despite REwLA have negative lookahead, the size of the corresponding DFA can still be bounded by a double exponential.

3.7 Applications

We convert the REwLA e_0 to DFA $\mathcal{A}(e_0)/\equiv$ on $\Sigma \cup \widetilde{\Sigma}$. $\mathcal{A}(e_0)/\equiv$ is defined as follows:

$$\begin{aligned} &\mathcal{A}(e_0)/\equiv = \langle Q(e_0)/\equiv, \Sigma \cup \widetilde{\Sigma}, \delta, [e_0], F \rangle, \\ &F = \{[e] \in Q(e_0)/\equiv \mid \nu(e)\}, \\ &\delta([e], c) = [d_c e], \end{aligned}$$

where $[e]$ is equivalence classes defined as $[e] = \{e' \mid e \equiv e'\}$.

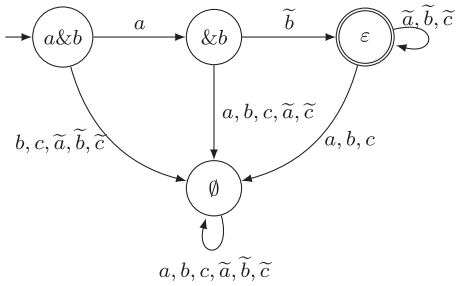

Fig. 1 DFA for $a\&b$.

$Q(e_0)/{\equiv}$ is a finite set by Theorem 3.9. In addition, F and δ are well-defined. Thus, $\mathcal{A}(e_0)/{\equiv}$ is a DFA.

Theorem 3.16. (*Correctness of DFA*)

$$L(\mathcal{A}(e_0)/{\equiv}) \subseteq \Sigma^*\widetilde{\Sigma}^*$$
$$x\widetilde{y} \in L(\mathcal{A}(e_0)/{\equiv}) \iff (x, y) \in B(e_0)$$

Example. When $\Sigma = \{a, b, c\}$, the DFA converted from $a\&b$ is shown in **Fig. 1**.

Several theorems are obtained as corollaries.

Corollary 3.17. (*Matching*) *Let $\mathcal{A}$ be the DFA converted from the REwLA e, n be the length of x, and m be the number of states. It is decidable whether $x \in L(e)$ in $O(n)$ time, and it is possible to calculate $\{(x_1, x_2) \in B(e) \mid x = x_1x_2\}$ in $O(nm)$ time.*

Proof. In order to determine whether $x \in L(e)$, it is sufficient to run $\mathcal{A}$ with input x. Since $\mathcal{A}$ is DFA, it is determined in $O(n)$ time. In order to calculate $\{(x_1, x_2) \in B(e) \mid x = x_1x_2\}$, when $x = a_1 \dots a_n$, for $0 \le i \le n$, it is sufficient to run $\mathcal{A}$ with input $a_1 \dots a_i\widetilde{a_{i+1}} \dots \widetilde{a_n}$. These can run in parallel, which is similar to the behavior of a nondeterministic automaton. The number of states at each step is bounded by the number of states m. Thus, it is calculated in $O(nm)$ time. □

Corollary 3.18 (*Backward Movement of Lookahead*)**.** *For any REwLA e, there exist regular expressions $r_0, \dots, r_n$, $s_0, \dots, s_n$ such that $B(e) = B(r_0\&(s_0\$)| \dots |r_n\&(s_n\$))$.*

Proof. Let $\mathcal{A}$ be the DFA converted from REwLA e, $\mathcal{A} = \langle Q, \Sigma \cup \widetilde{\Sigma}, \delta, q_0, F\rangle$, and $Q = \{q_0, \dots, q_n\}$. Let $\mathcal{A}_i = \langle Q, \Sigma, \delta|_{Q\times\Sigma}, q_0, \{q_i\}\rangle$, and $\mathcal{B}_i = \langle Q, \widetilde{\Sigma}, \delta|_{Q\times\widetilde{\Sigma}}, q_i, F\rangle$. $L(\mathcal{A}) = \bigcup_{i=0}^{n} L(\mathcal{A}_i)L(\mathcal{B}_i)$ holds because $L(\mathcal{A}) \subseteq \Sigma^*\widetilde{\Sigma}^*$. Therefore, $L(\mathcal{A}) = L(r_0\widetilde{s_0}| \dots |r_n\widetilde{s_n})$ where r_i and $\widetilde{s_i}$ are the regular expressions converted from DFA $\mathcal{A}_i$ and $\mathcal{B}_i$. Thus, $B(e) = B(r_0\&(s_0\$)| \dots |r_n\&(s_n\$))$ from Theorem 3.16. □

Corollary 3.19 (*Regularity*)**.** *For any REwLA e, there exist regular languages $A_1, \dots, A_n, B_1, \dots, B_n$ such that $B(e) = \bigcup_{i=1}^{n}(A_i \times B_i)$. Therefore, $L(e)$ is a regular language.*

Proof. This follows from Corollary 3.18. □

Corollary 3.20 (*Equivalence Checking*)**.** *Given REwLA e_1, e_2, it is decidable whether $B(e_1) = B(e_2)$.*

Proof. This can be determined by converting to DFA, minimizing, and comparing. □

Table 1 Experimental result.

REwLA	letters	states
a^*	1	3
$a\&b$	2	4
$(!(ab).)^*$	3	4
$ba(!(ab).)^*ab$	7	6
$\&(.^*a)\&(.^*b)(.^{8}).^*\$$	13	33
$.^*a\&((.^{3})^*\$)\&((.^{5})^*\$).^*\$$	11	32769

4. Implementation

We implemented the conversion from REwLA to DFA in the programming language Scala. It is easy to implement derivatives in programming languages having pattern matching, as indicated in Ref. [17].

In the previous section, states are equivalence classes, and each equivalence class is a possibly infinite set. Therefore, the implementation calculates representatives and makes them states. The computation of representatives is based on the implementation of derivatives of regular expressions in Isabelle/HOL [13].

Some examples were converted to DFA. **Table 1** shows the number of states of obtained DFA. The fourth from the top is an expression for comments of C languages, the fifth is an expression similar to passwords, and the sixth is an example of state explosion. The number of letters of "." is counted as 1, and the number of letters of \$ is counted as 0.

We describe one scheme used in our implementation. The expression $(a|b)(a|b)$ expands to $aa|ab|ba|bb$ by distributivity. We use literal classes rather than letters as primitives, and we represent the expression as $[ab][ab]$, preventing excessive expansion by the distributive law. Furthermore, we can handle letters and the dot expression uniformly as a special case of literal class.

5. Conclusion

We have given conversion from REwLA to DFA by derivatives. We have also given an upper bound on the number of states, which is doubly exponential in the size of the REwLA. As a corollary of conversion to DFA, we show that the semantics of REwLA is a finite union of sets of the form $A \times B$ where A and B are regular languages. In addition, we have implemented the conversion and confirmed it works as expected for some examples.

Acknowledgments This work was supported by JSPS KAKENHI Grant Number 15K00087.

References

[1] Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions, *Theoretical Computer Science*, Vol.155, No.2, pp.291–319 (1996).
[2] Berstel, J.: *Transductions and context-free languages*, Teubner Verlag (1979).
[3] Brzozowski, J.A.: Derivatives of regular expressions, *Journal of the ACM* (*JACM*), Vol.11, No.4, pp.481–494 (1964).
[4] Brzozowski, J.A. and Leiss, E.: On equations for regular languages, finite automata, and sequential networks, *Theoretical Computer Science*, Vol.10, No.1, pp.19–35 (1980).
[5] Chandra, A.K., Kozen, D. and Stockmeyer, L.J.: Alternation, *J. ACM*, Vol.28, No.1, pp.114–133 (1981).
[6] Cox, R.: Implementing regular expressions (2011), available from ⟨http://swtch.com/~rsc/regexp/⟩.
[7] Engelfriet, J.: Top-down tree transducers with regular look-ahead, *Mathematical Systems Theory*, Vol.10, No.1, pp.289–303 (1977).
[8] Ford, B.: Parsing expression grammars: A recognition-based syntactic foundation, *Proc. 31st ACM SIGPLAN-SIGACT Symposium on Prin-*

ciples of Programming Languages, pp.111–122, ACM (2004).
[9] Gelade, W.: Succinctness of regular expressions with interleaving, intersection and counting, *Theoretical Computer Science*, Vol.411, No.31-33, pp.2987–2998 (2010).
[10] Golan, J.S.: *Semirings and their Applications*, Kluwer Academic Publishers, Dordrecht (1999).
[11] Khoussainov, B. and Nerode, A.: *Automata theory and its applications*, Birkhäuser, Boston (2001).
[12] Kozen, D.: Kleene algebra with tests, *ACM Trans. Programming Languages and Systems* (*TOPLAS*), Vol.19, No.3, pp.427–443 (1997).
[13] Krauss, A. and Nipkow, T.: Proof pearl: Regular expression equivalence and relation algebra, *Journal of Automated Reasoning*, Vol.49, No.1, pp.95–106 (2012).
[14] Laurikari, V.: Efficient submatch addressing for regular expressions, Master's thesis, Helsinki University of Technology (2001).
[15] Morihata, A.: Translation of Regular Expression with Lookahead into Finite State Automaton, *Computer Software*, Vol.29, No.1, pp.147–158 (2012). (in Japanese).
[16] Morihata, A.: On the Size of Deterministic Finite Automata Obtained from XPath Expression, *106th IPSJ Workshop on Programming* (2015).
[17] Owens, S., Reppy, J. and Turon, A.: Regular-expression derivatives re-examined, *Journal of Functional Programming*, Vol.19, No.2, pp.173–190 (2009).
[18] Rosser, B.: Explicit bounds for some functions of prime numbers, *American Journal of Mathematics*, Vol.63, No.1, pp.211–232 (1941).
[19] Sakuma, Y., Minamide, Y. and Voronkov, A.: Translating regular expression matching into transducers, *Journal of Applied Logic*, Vol.10, No.1, pp.32–51 (2012).
[20] Stockmeyer, L.J. and Meyer, A.R.: Word problems requiring exponential time (preliminary report), *Proc. 5th Annual ACM Symposium on Theory of Computing*, pp.1–9, ACM (1973).
[21] Sugiyama, S. and Minamide, Y.: Translating Regular Expressions Extended with Atomic Grouping to Automata, *IPSJ Trans. Programming*, Vol.6, No.1, pp.17–26 (2013). (in Japanese).
[22] Thompson, K.: Programming techniques: Regular expression search algorithm, *Comm. ACM*, Vol.11, No.6, pp.419–422 (1968).

Takayuki Miyazaki received his B.Sc. degree in information science from Tokyo Institute of Technology in 2018. He is currently undertaking a master course at Tokyo Institute of Technology. His research interests include formal language theory.

Yasuhiko Minamide received his M.Sc. and Ph.D. degrees from Kyoto University in 1993 and 1997, respectively. Since 2015, he has been a professor at the Department of Mathematical and Computing Science, Tokyo Institute of Technology. His research interests focus on software verification and programming languages. He is also interested in the theory and applications of automata and formal languages. He is a member of ACM, IPSJ, and JSSST.