IEICE TRANSACTIONS

on Information and Systems

DOI:10.1587/transinf.2024FCP0008

Publicized:2024/08/20

This advance publication article will be replaced by the finalized version after proofreading.

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER *Special Section on Foundations of Computer Science*

Bilaterally Colored Finite Automata and Bilaterally Colored Regular Expressions

Akira ITO[†] ***and*** **Yoshiaki TAKAHASHI**[††]**,**

SUMMARY Recently, we introduced and investigated a colored variant of finite automata, so-called "colored finite automata." Its accepting states are able to be differently colored each and therefore a single automaton can classify and distinguish multiple languages at once. In this paper, we further extend the concept of colored accepting states and propose a new automaton, called bilaterally colored finite automaton (biCFA) which can possess as many differently colored initial states as possible, rather than a specified single initial state.

We next introduce its regular expression counterpart, called bilaterally colored regular expression (biCRE), which exactly expresses the same tuple of languages as accepted by the corresponding biCFA. Notably, the mono-colored version of colored regular expression is a succinct and intuitive alternative of the existing ordinary regular expression.

We also demonstrate the usefulness and feasibility of biCRE by applying the concept to extended (i.e., regular right part) context-free grammar and exhibit a super-short pure Python program which parses basic arithmetic expressions with addition and multiplication operators.[†]

key words: *Kleene's Theorem, state elimination method, system of equations, regex engine, compiler*

1. Introduction

Among many concepts in automata and language theory [1, 2], finite automaton and regular expression are the most elemental and yet available to wider areas beyond computer science field [3–5]. Recently, we introduced and investigated a colored variant of finite automata, so-called "colored finite automata." Its accepting states are able to be differently colored each and therefore a single automaton can classify and distinguish multiple languages at once. This mechanism not only provides a structural advantage for the design of language recognizers but also provokes new complexity problems concerning unmixedness of multiple colored languages [6, 7].

In this paper, we further extend the concept of colored accepting states toward the initial state. That is, the newly proposed automaton, called bilaterally colored finite automaton (biCFA) can possess as many differently colored initial states as possible, rather than a specified single initial state. Due to this extension, an n states biCFA is able to accept at most $\mathrm{O}(n^2)$ different languages by itself. To be convinced, imagine a string-shaped FA which accepts the singleton set $L_n^0 = \{0^n 1^n\}, n \geq 1$, with a pair of the leftmost 0-colored initial state q_0 and the rightmost n-colored accepting state q_{2n}. Further, it can accept n^2 different languages $L_i^{j-1} = \{0^{n-j+1}1^i\}$ for i, j $(1 \leq i, j \leq n)$ if it has a j-colored initial state at the $(j-1)$st position from q_0 and an i-colored accepting state at the ith position from the center.

We next introduce its regular expression counterpart, called bilaterally colored regular expression (biCRE), which exactly expresses the same tuple of languages as accepted by some biCFA. That is, just one biCRE can represent the same amount of languages as the corresponding biCFA. Incidentally, there has already existed a unilateral mono-colored version of biCRE (i.e., with accepting state position symbols not internally weaved), named 'pointed regular expression' or 'marked regular expression' in [8–10]. Although an example (described as "$(a + \triangleright b)^* \triangleleft = \triangleright b(a+b)^* \triangleleft$" in our notation) which illustrates its succinctness and intuitiveness was noticed, a main part of their investigation has being devoted to efficient construction of deterministic FAs that correspond to such REs. In contrast, this paper mainly focuses on the opposite direction: the transformation of bilaterally colored FAs to the corresponding biCREs. Notably, the mono-colored version of bilaterally colored regular expression turns out to be a succinct and intuitive alternative to ordinary regular expression. Particularly, the resultant bilateral regular expression transformed from the 'double chain'-like automaton [11] has linear size $\mathrm{O}(n)$ for the left-to-right order of elimination of the states while the ordinary expression has $O(n^3)$ size in the same order during the course of the state-elimination method. Such a phenomenon comes from the fact that our bilateral regular expression is formed in such a way that the start and end points of strings it represents could appear anywhere in the expression explicitly, rather than the implicit leftmost and rightmost boundaries of the expression.

We also demonstrate the usefulness and feasibility of biCRE by applying the concept to extended (i.e., regular right part) context-free grammar and exhibit a short

[†]The author with Non-Affiliated, Ube-shi, 755-8611 Japan. E-mail: aito@c-able.ne.jp

[††]The author with the National Institute of Technology, Oshima College, Yamaguchi-ken, 742-2193 Japan. E-mail: takahashiy@oshima-k.ac.jp (Corresponding author)

[†]This manuscript is a revised version of the conference paper [14].

Python program (25 lines without any extra module) which does parse arithmetic expressions with plus and multiplying operators.

This paper is organized as follows. In Section 2, we introduce bilaterally colored FA and give an example of its usage in lexical analysis. In Section 3, we introduce bilaterally colored RE as its counterpart, then investigate its fundamental properties and give some examples showing how familiar regular expressions are rewritten in bilateral form. In Section 4, we first show a method of transformation from biCFAs to biCREs based on the solution method of simultaneous language equations and give some examples demonstrating how conventional finite automata are converted to bilateral regular expressions without any complex manipulation. We next demonstrate how the inverse transformation can be done by using the example described in the case of biCFA-to-biCRE transformation. Inductively, we have the equivalence of bilaterally colored versions of FAs and REs. In Section 5, we give examples of their application to an improvement of existing regular expression matching engines and a simplification of parser program for well-known context-free grammars.

Now we recall the definition of ordinary regular expression [12].

Definition 1: Regular expression over an alphabet Σ is defined inductively:

1. ε, $\emptyset$, and $a \in \Sigma$ are regular expressions expressing the languages $L(\varepsilon) = \{\varepsilon\}, L(\emptyset) = \emptyset$, and $L(a) = \{a\}$, respectively.
2. If r and s are regular expressions expressing the languages $L(r)$ and $L(s)$, then $r + s, rs$, and r^* are regular expressions expressing the languages $L(r + s) = L(r) \cup L(s), L(rs) = L(r)L(s)$, and $L(r^*) = L(r)^*$, respectively.

2. Bilaterally Colored Finite Automata

Definition 2: ([6,7]) Let L_i be a language over alphabet Σ for $i = 1, \ldots, k, k \geq 1$. A k-tuple $(L_1, L_2, \ldots, L_k)$ of languages is called colored language (vector) of k colors over Σ.

The terminology of language vector or tuple of languages was first used to describe the behavior of a multiple-output sequential circuit in [13].

Definition 3: A *bilaterally colored finite automaton*, abbreviated biCFA, is a 5-tuple $M = (\Sigma, Q, \Sigma_{j=1}^{l} Q_0^j, \delta, \sum_{i=1}^{k} F_i)$, where

1. Q is a finite set of states,
2. Σ is a finite set of input symbols,
3. δ is the transition function from $Q \times (\Sigma \cup \{\varepsilon\})$ to 2^Q,
4. $\Sigma_{j=1}^{l} Q_0^j \subseteq Q$ is a mutually disjoint family $\{Q_0^1, \ldots, Q_0^l\}$ of sets Q_0^j's, where Q_0^j is the set of initial states with jth color,
5. $\Sigma_{i=1}^{k} F_i \subseteq Q$ is a mutually disjoint family $\{F_1, \ldots, F_k\}$ of sets F_i's, where F_i is the set of accepting states with ith color.†

We denote as $\hat{\delta}(q, x)$ the set of reachable states when M starts from state q and finishes after it reads the input string x. Define††

$$
\begin{aligned}
L_i^j(M) &\triangleq \{x \in \Sigma^* \mid \hat{\delta}(q_0^j, x) \bigcap F_i \neq \emptyset, q_0^j \in Q_0^j\},\\
L_i(M) &\triangleq \bigcup_{j=1}^{l} L_i^j(M),\\
L^j(M) &\triangleq \bigcup_{i=1}^{k} L_i^j(M), \text{ and}\\
L(M) &\triangleq \bigcup_{j=1}^{l} L^j(M) = \bigcup_{i=1}^{k} L_i(M)\\
&\qquad = \bigcup_{j=1}^{l} \bigcup_{i=1}^{k} L_i^j(M).
\end{aligned}
$$

$L_i^j(M)$ is said to be the language accepted by M which starts with jth color and ends with ith color and $L(M)$ is called the (unified) language accepted by M.

Note that ordinary non-colored finite automaton M is a special case of biCFA whose families $\Sigma_{j=1}^{l} Q_0^j$ and $\Sigma_{i=1}^{k} F_i$ of sets of initial states and accepting states are equal to $\{\{q_0\}\}$ and $\{F\}$, respectively.

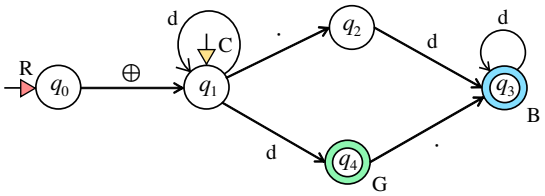


Fig. 1 A biCFA accepting decimal integers or fractional numbers with or without plus signs, simultaneously.

Example 1: Consider a bilaterally colored finite automaton M depicted in Fig. 1, where d is the abbreviation of digit characters $\{0, \ldots, 9\}$. M accepts fractional numbers (having decimal points) with or without plus signs in the same way as Example 2.16 in [12]. In addition, M also accepts integer numbers (not having decimal points) with or without plus signs. More precisely, M distinguishes those four different languages:

†For sets X and Y, direct sum X + Y is the union X ∪ Y satisfying the disjointness X ∩ Y = ∅. Notice that we use the regular expression $r + s$ to denote language $L_1 \cup L_2$, where L_1 and L_2 are the languages expressed by the regular expressions r and s, respectively.

††$X \triangleq Y$ means that X is defined as Y.

$$
\begin{aligned}
L_{\mathrm{B}}^{\mathrm{R}}(M) &= \oplus\, \mathrm{d}^*(\,\cdot\,\mathrm{d} + \mathrm{d}\,\cdot)\,\mathrm{d}^*,\\
L_{\mathrm{B}}^{\mathrm{C}}(M) &= \mathrm{d}^*(\cdot\,\mathrm{d} + \mathrm{d}\,\cdot\,)\,\mathrm{d}^*,\\
L_{\mathrm{G}}^{\mathrm{R}}(M) &= \oplus\, \mathrm{d}^+, \text{and}\\
L_{\mathrm{G}}^{\mathrm{C}}(M) &= \mathrm{d}^+.
\end{aligned}
$$

Furthermore,

$$
\begin{aligned}
L_{\mathrm{B}}(M) &= L_{\mathrm{B}}^{\mathrm{R}}(M) \cup L_{\mathrm{B}}^{\mathrm{C}}(M) = (\oplus + \varepsilon)\,\mathrm{d}^*(\,\cdot\,\mathrm{d} + \mathrm{d}\,\cdot\,)\,\mathrm{d}^*,\\
L_{\mathrm{G}}(M) &= L_{\mathrm{G}}^{\mathrm{R}}(M) \cup L_{\mathrm{G}}^{\mathrm{C}}(M) = (\oplus + \varepsilon)\,\mathrm{d}^+, \text{ and}\\
L(M) &= L_{\mathrm{B}}(M) \cup L_{\mathrm{G}}(M)\\
&= (\oplus + \varepsilon)\,\mathrm{d}^*(\,\cdot\,\mathrm{d} + \mathrm{d}\,\cdot\,)\,\mathrm{d}^* + (\oplus + \varepsilon)\,\mathrm{d}^+.
\end{aligned}
$$

3. Bilaterally Colored Regular Expressions

In this section, we first give the syntactical and semantic definitions of bilaterally colored regular expression, which is a natural generalization of that of ordinary regular expression.

Definition 4: A *bilaterally colored regular expression* (abbreviated biCRE) over an alphabet Σ with $l \geq 1$ initial state colors and $k \geq 1$ accepting state colors is defined inductively:

1. $\varepsilon, \triangleright^j\,(1 \leq j \leq l), \triangleleft_i\,(1 \leq i \leq k), \emptyset$, and $a \in \Sigma$ are bilaterally colored regular expressions.
2. If r and s are bilaterally colored regular expressions, then $r + s, rs$, and r^* are bilaterally colored regular expressions.

The symbols $\triangleright^j\,(1 \leq j \leq l)$ and $\triangleleft_i\,(1 \leq i \leq k)$ are called *j-colored initial state position symbol* and *i-colored accepting state position symbol*, respectively.

Definition 5: Let r be a biCRE with l initial state colors and k accepting state colors. Let $\mathrm{pos}^j(r), 1 \leq j \leq l$, and $\mathrm{pos}_i(r), 1 \leq i \leq k$, denote the sets of positions appearing in r of j-colored initial state position symbols and i-colored accepting state position symbols, respectively. For each $j' \in \mathrm{pos}^j(r), i' \in \mathrm{pos}_i(r), 1 \leq j \leq l, 1 \leq i \leq k$, let $r|_{i'}^{j'}$ denote the regular expression that is obtained from r by replacing with ε's all initial state position symbols and all accepting state position symbols except a pair of the j'th initial state position symbol and the i'th accepting state position symbol. Furthermore, let $r|_{i'}^{0}$ and $r|_{0}^{j'}$ denote the regular expressions that are obtained from $r|_{i'}^{j'}$ by replacing with ε its j'th initial state position symbol and its i'th accepting state position symbol, respectively.

Definition 6: Let r be a biCRE with l initial state and k accepting state colors. For each $j(1 \leq j \leq l)$, $i(1 \leq i \leq k)$, the (j,i)-component of colored language (vector) $L(r)$ expressed by r is recursively defined as

$$
L_i^j(r) \triangleq \bigcup_{j' \in \mathrm{pos}^j(r)} \bigcup_{i' \in \mathrm{pos}_i(r)} L(r|_{i'}^{j'})
$$

(this resolving of r is called **pairs-scattering**), where

(U) If $r|_{i'}^{j'} = s + t$, then $L(r|_{i'}^{j'}) = L(s|_{i'}^{j'}) \bigcup L(t|_{i'}^{j'})$, i.e., ordinary union (Boolean sum) operation.

(Cl) If $r|_{i'}^{j'} = st, \triangleright \notin s \neq \varepsilon$, then $L(r|_{i'}^{j'}) = L(t|_{i'}^{j'})$, i.e., discard of useless prefix s (leftmost substring).

(Cr) If $r|_{i'}^{j'} = st, \triangleleft \notin t \neq \varepsilon$, then $L(r|_{i'}^{j'}) = L(s|_{i'}^{j'})$, i.e., discard of useless suffix t (rightmost substring).

(Si) If $r|_{i'}^{j'} = s^*, \triangleright \in s$, then $L(r|_{i'}^{j'}) = L(s|_{i'}^{j'} s^*|_{i'}^{0})$, i.e., eviction of initial position symbol out of star operator.

(Sa) If $r|_{i'}^{j'} = s^*, \triangleleft \in s$, then $L(r|_{i'}^{j'}) = L(s^*|_{0}^{j'} s|_{i'}^{j'})$, i.e., eviction of accepting position symbol out of star operator.

(E) If $r|_{i'}^{j'} = \varepsilon$ or $r|_{i'}^{j'} = \emptyset$, then $L(r|_{i'}^{j'}) = L(\emptyset) = \emptyset$, i.e., discard of empty string and empty set.

(O) If $r|_{i'}^{j'} = \triangleright s \triangleleft$ (and $\triangleright, \triangleleft \notin s$), then $L(r|_{i'}^{j'}) = L(s)$, where $L(s)$ is defined in Definition 1 for ordinary regular expression s,
i.e., extraction of a string enclosed with both end mark position symbols as meaningful contents.

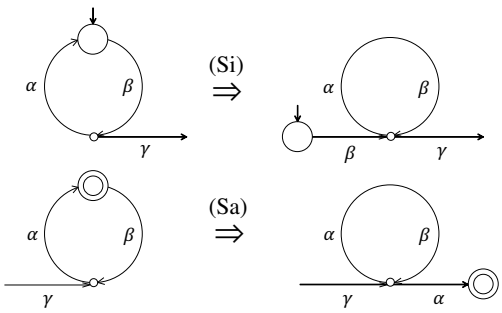


Fig. 2 Illustrations of the rules (Si) and (Sa) of Definition 6.

See Fig. 2 for the graphical meanings of the rules (Si) and (Sa) above. Note that after applying the rule (Si) or (Sa), we must repeat another cycle of pairs-scattering and star-eviction if the resulting expression has more than one pair of position symbols $\triangleright, \triangleleft$.

Hereafter, we abbreviate $\alpha = \beta$ when the two languages expressed by α and β are the same.

The following fact is an extremal case of biCRE (See Fig. 3 for their intuitive meanings).

Fact 1:
1. For $r \in \{\triangleright, \triangleleft, \triangleleft\triangleright, \triangleleft + \triangleright, \triangleright + \triangleleft\}$, $L(r) = \emptyset$.
2. For $r \in \{\triangleright\triangleleft, (\triangleleft\triangleright)^*, (\triangleright\triangleleft)^*, (\triangleleft + \triangleright)^*, (\triangleright + \triangleleft)^*\}$, $L(r) = \{\varepsilon\}$.

(Proof)

1. From the rule (Cr) with $s = \varepsilon, t = \triangleright \neq \varepsilon$ and (E) of Definition 6, we have $L(\triangleright|_{i'}^{j'}) = L(\varepsilon|_{i'}^{j'}) = \emptyset$.
From the rule (Cl) with $s = \triangleleft \neq \varepsilon$, $t = \varepsilon$ and (E) of Definition 6, we have $L(\triangleleft|_{i'}^{j'}) = L(\varepsilon|_{i'}^{j'}) = \emptyset$.
From the rule (Cr) of Definition 6 with $s = \triangleleft$ and $t = \triangleright$, we have $L(\triangleleft\triangleright|_{i'}^{j'}) = L(\triangleleft|_{i'}^{j'}) = \emptyset$.
From the rule (U) of Definition 6 with $s = \triangleleft$ and $t = \triangleright$, we have $L(\triangleleft + \triangleright\,|_{i'}^{j'}) = L(\triangleright + \triangleleft\,|_{i'}^{j'}) = L(\triangleleft|_{i'}^{j'}) +$

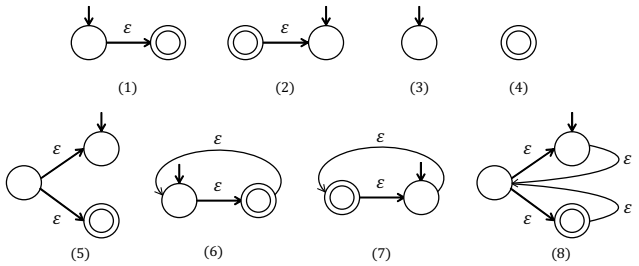


Fig. 3 FAs corresponding to the expressions (1) $\triangleright\triangleleft$, (2) $\triangleleft\triangleright$, (3) $\triangleright$, (4) $\triangleleft$, (5) $\triangleleft+\triangleright$ or $\triangleright+\triangleleft$, (6) $(\triangleright\triangleleft)^*$, (7) $(\triangleleft\triangleright)^*$, and (8) $(\triangleleft+\triangleright)^*$ or $(\triangleright+\triangleleft)^*$.

$$L(\triangleright|_{i'}^{j'}) = \emptyset.$$

2. From the rule (O) of Definition 6 with $s = \varepsilon$, $L(\triangleright\triangleleft|_{i'}^{j'}) = \varepsilon$.
From the rules (Si), (Sa) of Definition 6 (and pairs-scattering), we have
$$\begin{aligned}
(\triangleleft\triangleright)^* &= (\triangleright)^*(\triangleleft\triangleright) = (\triangleright)^*(\triangleleft) + (\varepsilon)^*(\triangleleft\triangleright)\\
&= (\triangleright)(\varepsilon)^*(\triangleleft) + (\triangleleft\triangleright) = (\triangleright)(\triangleleft) + \emptyset = \triangleright\triangleleft,\\
(\triangleright\triangleleft)^* &= (\triangleright)^*(\triangleright\triangleleft) = (\triangleright)^*(\triangleleft) + (\varepsilon)^*(\triangleright\triangleleft)\\
&= (\triangleright)(\varepsilon)^*(\triangleleft) + (\triangleright\triangleleft)\\
&= (\triangleright)(\triangleleft) + \triangleright\triangleleft = \triangleright\triangleleft, \text{ and}\\
(\triangleleft+\triangleright)^* &= (\triangleright+\triangleleft)^* = (\triangleright+\triangleleft)(\varepsilon+\triangleleft)^*\\
&= (\triangleright+\triangleleft)(\varepsilon+\varepsilon)^* + (\triangleright+\varepsilon)(\varepsilon+\triangleleft)^*\\
&= (\triangleright+\triangleleft) + (\triangleright+\varepsilon)(\varepsilon+\varepsilon)^*(\varepsilon+\triangleleft)\\
&= \emptyset + \triangleright\triangleleft = \triangleright\triangleleft.
\end{aligned}$$

□

Example 2: The four regular expressions for $L_i^j(M)$, $j \in \{\mathrm{R}, \mathrm{C}\}, i \in \{\mathrm{G}, \mathrm{B}\}$ in Example 1 are aggregated to a single expression

$$\begin{aligned}
&\triangleright^{\mathrm{R}} \oplus \triangleright^{\mathrm{C}} \mathrm{d}^*(\,\cdot\,\mathrm{d} + \mathrm{d}\triangleleft_{\mathrm{G}}\,\cdot\,)\,\mathrm{d}^*\triangleleft_{\mathrm{B}}\\
&= r|_{\mathrm{G}}^{\mathrm{R}} + r|_{\mathrm{B}}^{\mathrm{R}} + r|_{\mathrm{G}}^{\mathrm{C}} + r|_{\mathrm{B}}^{\mathrm{C}}\\
&= \triangleright^{\mathrm{R}} \oplus \mathrm{d}^*(\,\cdot\,\mathrm{d} + \mathrm{d}\triangleleft_{\mathrm{G}}\,\cdot\,)\,\mathrm{d}^* + \triangleright^{\mathrm{R}} \oplus \mathrm{d}^*(\,\cdot\,\mathrm{d} + \mathrm{d}\,\cdot\,)\,\mathrm{d}^*\triangleleft_{\mathrm{B}}\\
&\quad + \oplus\ \triangleright^{\mathrm{C}} \mathrm{d}^*(\,\cdot\,\mathrm{d} + \mathrm{d}\triangleleft_{\mathrm{G}}\,\cdot\,)\,\mathrm{d}^* + \oplus\triangleright^{\mathrm{C}} \mathrm{d}^*(\,\cdot\,\mathrm{d} + \mathrm{d}\,\cdot\,)\,\mathrm{d}^*\triangleleft_{\mathrm{B}}\\
&= \triangleright^{\mathrm{R}} \oplus \mathrm{d}^*\mathrm{d}\triangleleft_{\mathrm{G}} + \triangleright^{\mathrm{R}} \oplus \mathrm{d}^*(\,\cdot\,\mathrm{d} + \mathrm{d}\,\cdot\,)\,\mathrm{d}^*\triangleleft_{\mathrm{B}}\\
&\qquad + \triangleright^{\mathrm{C}} \mathrm{d}^*\mathrm{d}\triangleleft_{\mathrm{G}} + \triangleright^{\mathrm{C}} \mathrm{d}^*(\,\cdot\,\mathrm{d} + \mathrm{d}\,\cdot\,)\,\mathrm{d}^*\triangleleft_{\mathrm{B}}.
\end{aligned}$$

The last equality is derived by the rules (E), (Cl), and (Cr) of Definition 6.

Example 3: (Example 3.2 in [12]) The language $(01)^* + 1(01)^* + (01)^*0 + 1(01)^*0 = (\varepsilon+1)(01)^*(\varepsilon+0)$, in which neither 0 nor 1 continues, is expressed by a bi-CRE $\triangleright(0\triangleright\triangleleft 1)^*\triangleleft$ because the similar derivation holds as in Example 2 (see Fig. 4 for its graphical meaning):

$$\begin{aligned}
&\triangleright(0\triangleright\triangleleft 1)^*\triangleleft\\
&= \triangleright(01)^*\triangleleft + \triangleright(0\triangleleft 1)^* + (0\triangleright\triangleleft 1)^* + (0\triangleright 1)^*\triangleleft\\
&= \triangleright(01)^*\triangleleft + \triangleright(01)^*0\triangleleft + (0\triangleright\triangleleft 1)(0\triangleleft 1)^* + (0\triangleright 1)(01)^*\triangleleft\\
&\quad\ \vdots\\
&= \triangleright(01)^*\triangleleft + \triangleright(01)^*0\triangleleft + \triangleright 1(01)^*0\triangleleft + \triangleright\triangleleft + \triangleright 1(01)^*\triangleleft.
\end{aligned}$$

The following identities between bilateral regular expressions are derived in the similar manner as before.

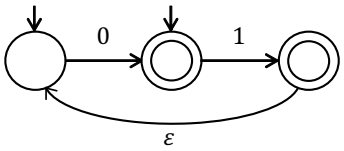


Fig. 4 An FA corresponding to the expression $\triangleright(0\triangleright\triangleleft 1)^*\triangleleft$.

Proposition 1: For any expression α, β which do not contain any state position symbol, the following holds (See Fig. 5 for their intuitive meanings).

1. ("the shifting rule" [15])
$$\triangleright(\alpha\triangleleft\beta)^* = \triangleright(\alpha\beta)^*\alpha\triangleleft = \triangleright\alpha(\beta\alpha)^*\triangleleft = (\beta\triangleright\alpha)^*\triangleleft,$$
2. ("the denesting rule" [15])
$$\begin{aligned}
\triangleright(\alpha^*\triangleleft\beta)^* &= \triangleright(\alpha^*\beta)^*\alpha^*\triangleleft\\
&= \triangleright(\alpha+\beta)^*\triangleleft\\
&= \triangleright\beta^*(\alpha\beta^*)^*\triangleleft = (\alpha\triangleright\beta^*)^*\triangleleft,
\end{aligned}$$
3. (one of "the aperiodic identities" [11])
$$(\alpha\triangleright\triangleleft\beta)^* = \triangleright(\beta\alpha)^*\triangleleft = (\alpha\triangleleft\triangleright\beta)^* + \triangleright\triangleleft.$$

Practically, the identities above should not be used when they are surrounded by other loops of star operators.

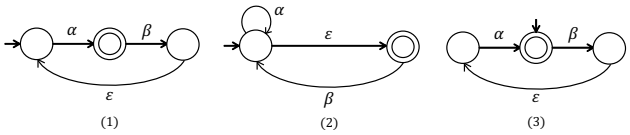


Fig. 5 FAs corresponding to the expressions (1) $\triangleright(\alpha\triangleleft\beta)^*$, (2) $\triangleright(\alpha^*\triangleleft\beta)^*$, and (3) $(\alpha\triangleright\triangleleft\beta)^*$.

Example 4: (Sub-subsection 3.3.3 in [12]) From the shifting rule $\triangleright\alpha(\beta\alpha)^*\triangleleft = \triangleright(\alpha\triangleleft\beta)^*$ of Proposition 1, the Unix-style regular expression
'`[A-Z][a-z]*( [A-Z][a-z]*)*`'
for names of streets can be expressed by a biCRE
'`:>([A-Z][a-z]*<: )*`,'
where '`:>`' and '`<:`' mimic initial and accepting state position symbols $\triangleright$ and $\triangleleft$, respectively (see Fig. 6 for its graphical meaning). The same convention will be seen from now on.

Example 5: The regular expression for email addresses in HTML Living Standard [16]

```
[a-zA-Z0-9.!#$%&'*+\/=?^_$`{|}~-]+@[a-zA-Z
0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:
\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z
0-9])?)*
```

can be rewritten to a bilateral one:

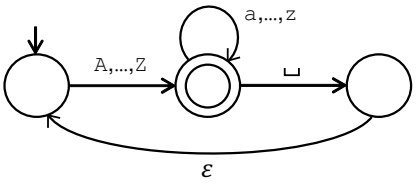


Fig. 6 An FA corresponding to the expression '`:>([A-Z][a-z]*<: )*`.'

```
:>[a-zA-Z0-9.!#$%&'*+\/=?^_`{|}~-]+@(:?[a
-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])
?<:\.)*,
```

since it holds that

$$\triangleright\, xy(zy)^* \triangleleft = \triangleright\, x(yz)^* y \triangleleft = \triangleright\, x(y \triangleleft z)^*,$$

where $x =$ '`[a-zA-Z0-9.!#$%&'*+\/=?^_$`{|}~-]+@,`' $y =$ '`[w]([w-]{0,61}[w])?,`' $z =$'`\.`,' $w =$ '`a-zA-Z0-9`' and the equality each comes from the shifting rule.

Remark 1: We must be aware of the following general principles.

- When more than one pair $\triangleright, \triangleleft$ of left and right triangles appear in a given biCRE (in either mono-colored case or multi-colored case), we must first split them to separated biCREs, each of which has only one pair of triangles (pairs-scattering), and then must do other manipulations, e.g.,

$$\begin{aligned}\triangleright\, a(\triangleright\, b \triangleleft)^* &= \triangleright\, a(\varepsilon\, b \triangleleft)^* + \varepsilon\, a(\triangleright\, b \triangleleft)^* \\ &= \triangleright\, a(b)^*(b \triangleleft) + a(\triangleright\, b)^*(\triangleright\, b \triangleleft).\end{aligned}$$

  The last equality is derived by the rule (Sa) of Definition 6.
- To evict position state triangles out of nested star parentheses, we must proceed in a top-down manner: The whole contents in the outermost star parentheses are copied to its left or right side at first, then the inner triangles of the copied contents can be evicted if they are in another star loop, e.g.,

$$\begin{aligned}\triangleright\, ((a \triangleleft b)^* c)^* &= \triangleright\, ((ab)^* c)^* ((a \triangleleft b)^* c) \\ &= \triangleright\, ((ab)^* c)^* ((ab)^* (a \triangleleft b) c).\end{aligned}$$

  Both equalities are derived by the rule (Sa) of Definition 6.

As the final remark of this section, we point out the well-definedness of Definition 6: When even the rules (Si) and (Sa) are simultaneously applicable, the order of these applications makes no difference in their outcomes, i.e., they are confluent.

Let $\alpha[\triangleright,\triangleleft]^*$ be a star loop sub-expression of some conjunctive (product) term T of a regular expression such that (1) there exists exactly one pair of position symbols $\triangleright$ and $\triangleleft$ in α, (2) no position symbol outside it, and (3) there may be other star loops inside α but not outside α^*, i.e., it is the outermost star loop containing the position symbols $\triangleright,\triangleleft$ within T. Similarly, for example, let $\alpha[\triangleright,]$ denotes the modified α such that its accepting state position symbol $\triangleleft$ is replaced with ε.

Fact 2: $\alpha[\triangleright,\triangleleft]^* = \alpha[\triangleright,\triangleleft] + \alpha[\triangleright,]\alpha[,]^*\alpha[,\triangleleft]$.

(Proof) Let $T[\alpha[\triangleright,\triangleleft]^*]$ denotes such a term T as described above containing sub-expression $\alpha[\triangleright,\triangleleft]^*$. By applying the rule (Si), pairs-scattering, the rules (Cr) and (Sa) in this order, we have $T[\alpha[\triangleright,\triangleleft]^*] = T[\alpha[\triangleright,\triangleleft]\alpha[,\triangleleft]^*] = T[\alpha[\triangleright,\triangleleft]\alpha[,]^*] + T[\alpha[\triangleright,]\alpha[,\triangleleft]^*] = T[\alpha[\triangleright,\triangleleft]] + T[\alpha[\triangleright,]\alpha[,]^*\alpha[,\triangleleft]]$. In the same way, by applying the rule (Sa), pairs-scattering, the rules (Si) and (Cl) in this order, we have $T[\alpha[\triangleright,\triangleleft]^*] = \cdots = T[\alpha[\triangleright,]\alpha[,]^*\alpha[,\triangleleft]] + T[\alpha[\triangleright,\triangleleft]]$, too. On the other hand, by pairs-scattering, the rules (Cl), (Cr) and (E), we have also $T[\alpha[\triangleright,\triangleleft] + \alpha[\triangleright,]\alpha[,]^*\alpha[,\triangleleft]] = \cdots = T[\alpha[\triangleright,\triangleleft]] + T[\alpha[\triangleright,]\alpha[,]^*\alpha[,\triangleleft]]$. □

4. Equivalence of Bilaterally Colored FAs and Bilaterally Colored REs

From the definitions described so far, we can see a multi-colored biCFA (biCRE) as a color-distinguished sum of FAs (REs) each of which has just one pair of initial state and accepting state (these position symbols). We can, therefore, adopt classical algorithms of transformations between ordinary FAs and REs as a whole.

As a result, the termination and correctness of our modified versions follow from the original ones.

4.1 Transformation of BiCFAs to BiCREs

Among several methods to transform finite automata to their equivalent regular expressions, we adopt the fixed point solution method for a system of language equations [11,17–19]. It is known [11] that the most popular (graphical) "state elimination method" and this rather literate (algebraic) system of equations method are the same and therefore produce the same expression from a given automaton, provided that states in the former and indeterminate variables in the latter are eliminated in the same order during the course of transformation.

An equation $X = aY + b$ for some automaton M means that from state X it can go to state Y via the transition labeled with a or go to some accepting state and halt via the transition labeled with b. We also denote such equation as '$X \to aY + b$' interchangeably because we do not require the set theoretical equality between two languages represented by the left-hand side and the right-hand side, but rather require the least fixed-point solution obtained by (infinite) self-substitution of the equations that are regarded as rewriting rules of left-hand side nonterminals to right-hand side sentential forms just like right linear grammars for regular languages.

In this section, we often use the following lemma,

known as Arden's Lemma, in order to eliminate indeterminate variables from right-hand side of the given equations.

Lemma 1: Let A and B be two languages. Then, A^*B is the (least) solution of the equation $X = AX + B$ (or $X \to AX + B$). □

The following is a key to the relative compactness of the resulting expressions of our transformation.

Theorem 1: Let X and Y be an accepting state and a state of some biCFA, respectively. Then, all the equations below are equivalent:

(1) $X \to a\,X + b\,Y + \triangleleft$,
(2) $X \to a\,X + \triangleleft\, b\,Y$,
(3) $X \to \triangleleft\, a\,X + b\,Y$, and
(4) $X \to \triangleleft\, a\,X + \triangleleft\, b\,Y$.

(Proof) Suppose that the equations above become the following forms due to the replacement of Y with $\alpha X + \beta$ just before X is eliminated from the right hand side of the equation: (1) $X \to (a + b\alpha)X + b\beta + \triangleleft$, (2) $X \to (a+b\alpha)X+\triangleleft\, b\beta$, (3) $X \to (\triangleleft\, a+b\alpha)X+b\beta$, and (4) $X \to (\triangleleft\, a + b\alpha)X + \triangleleft\, b\beta$, where α, β do not contain variable X. Below, $\alpha[\,]$, etc., denotes the regular expression α from which $\triangleleft$'s are replaced with ε's. Therefore, e.g., it holds that $\alpha\beta\gamma = \alpha\beta[\,]\gamma[\,] + \alpha[\,]\beta\gamma[\,] + \alpha[\,]\beta[\,]\gamma$. By Lemma 1, we have

$$
\begin{aligned}
(1) : X &= (a + b\alpha)^*(b\beta + \triangleleft) \\
&= (a + b\alpha)^*(b\beta[\,] + \varepsilon) + (a + b\alpha[\,])^*(b\beta + \varepsilon) \\
&\qquad + (a + b\alpha[\,])^*(b\beta[\,] + \triangleleft) \\
&= (a + b\alpha)^* + (a + b\alpha[\,])^*b\beta + (a + b\alpha[\,])^*\triangleleft, \\
(2) : X &= (a + b\alpha)^*(\triangleleft\, b\beta) \\
&= (a + b\alpha)^*(\varepsilon\, b\beta[\,]) + (a + b\alpha[\,])^*(\triangleleft\, b\beta[\,]) \\
&\qquad + (a + b\alpha[\,])^*(\varepsilon\, b\beta) \\
&= (a + b\alpha)^* + (a + b\alpha[\,])^*\triangleleft + (a + b\alpha[\,])^*(b\beta), \\
(3) : X &= (\triangleleft\, a + b\alpha)^*(b\beta) \\
&= (\triangleleft\, a + b\alpha[\,])^*(b\beta[\,]) + (\varepsilon a + b\alpha)^*(b\beta[\,]) \\
&\qquad + (\varepsilon a + b\alpha[\,])^*(b\beta) \\
&= (a + b\alpha[\,])^*(\triangleleft\, a + b\alpha[\,])(b\beta[\,]) + (a + b\alpha)^* \\
&\qquad + (a + b\alpha[\,])^*b\beta \\
&= (a + b\alpha[\,])^*\triangleleft + (a + b\alpha)^* + (a + b\alpha[\,])^*b\beta, \\
&\text{and} \\
(4) : X &= (\triangleleft\, a + b\alpha)^*(\triangleleft\, b\beta) \\
&= (\triangleleft\, a + b\alpha)^*(\varepsilon\, b\beta) + (a + b\alpha)^*(\triangleleft\, b\beta) \\
&= (a + b\alpha)^*(b\beta + \triangleleft) \text{ from the parts (2) and (3) above.}
\end{aligned}
$$

□

Note that the part (1) of the theorem above corresponds to the conventional equation $X = aX + bY + \varepsilon$. Thus, Theorem 1 says that there exists more than one way to indicate that some state is accepting in a given equation. Similarly, there exist many ways for initial state indication but we adopt here the way that is most operative to formulate the system of equations at first and systematic to solve it to the final expression: If X is an initial state of some biCFA, the left-hand side of the equation for X is initially set as $\triangleright\, X$. When the contents of body of the equation is resolved or arranged to be substituted to another equation, the initial state position symbol $\triangleright$ must be transposed from the left-hand side to the right-hand side, e.g., $\triangleright\, X \to \alpha \cdots$ must change to $X = \triangleright\, \alpha \cdots$.

As will be seen below, initial and accepting state position symbols embedded in state equations are properly inserted in intermediate forms of regular expressions during the course of transformation.

Example 6: We transform the biCFA classifying decimal numbers shown in Fig. 1 to an equivalent biCRE as follows. From the figure, we have the system of equations.

$$
\begin{cases}
\triangleright^{\mathrm{R}}\, q_0 \to \oplus\, q_1 & \cdots (0) \\
\triangleright^{\mathrm{C}}\, q_1 \to \mathrm{d}\, q_1 + \cdot\, q_2 + \mathrm{d}\, q_4 & \cdots (1) \\
\quad q_2 \to \mathrm{d}\, q_3 & \cdots (2) \\
\quad q_3 \to \mathrm{d}\, q_3 + \triangleleft_{\mathrm{B}} & \cdots (3) \\
\quad q_4 \to \triangleleft_{\mathrm{G}} \cdot q_3 & \cdots (4)
\end{cases}
$$

Note that the symbol $\triangleleft_{\mathrm{G}}$ for G-colored accepting state in (4) is attached at the front of the term $\cdot\, q_3$.
Appling Lemma 1 to (3), $q_3 = \mathrm{d}^*\triangleleft_{\mathrm{B}}. \cdots (3')$
Substituting it to (2), $q_2 = \mathrm{d}(\mathrm{d}^*\triangleleft_{\mathrm{B}}) = \mathrm{d}\,\mathrm{d}^*\triangleleft_{\mathrm{B}}. \cdots (2')$
Substituting $(3')$ to (4), $q_4 = \triangleleft_{\mathrm{G}} \cdot (\mathrm{d}^*\triangleleft_{\mathrm{B}}) = \triangleleft_{\mathrm{G}} \cdot \mathrm{d}^*\triangleleft_{\mathrm{B}}$.
Substituting it and $(2')$ to (1),
$\triangleright^{\mathrm{C}} q_1 \to \mathrm{d}\, q_1 + \cdot\,(\mathrm{d}\,\mathrm{d}^*\triangleleft_{\mathrm{B}}) + \mathrm{d}(\triangleleft_{\mathrm{G}} \cdot \mathrm{d}^*\triangleleft_{\mathrm{B}})$.
Hence, from Lemma 1,
$\triangleright^{\mathrm{C}} q_1 \to \mathrm{d}^*(\cdot\,\mathrm{d}\,\mathrm{d}^*\triangleleft_{\mathrm{B}} + \mathrm{d}\triangleleft_{\mathrm{G}} \cdot \mathrm{d}^*\triangleleft_{\mathrm{B}}) = \mathrm{d}^*(\cdot\,\mathrm{d} + \mathrm{d}\triangleleft_{\mathrm{G}}\cdot)\,\mathrm{d}^*\triangleleft_{\mathrm{B}}$.
By left-to-right transposition of the initial state position symbol,
$q_1 = \triangleright^{\mathrm{C}}\mathrm{d}^*(\cdot\,\mathrm{d} + \mathrm{d}\triangleleft_{\mathrm{G}}\cdot)\,\mathrm{d}^*\triangleleft_{\mathrm{B}}$.
Substituting it to (0), $\triangleright^{\mathrm{R}} q_0 \to \oplus(\triangleright^{\mathrm{C}}\mathrm{d}^*(\cdot\,\mathrm{d} + \mathrm{d}\triangleleft_{\mathrm{G}}\cdot)\,\mathrm{d}^*\triangleleft_{\mathrm{B}})$.
Hence, by left-to-right transposition of the initial state position symbol,

$$q_0 = \triangleright^{\mathrm{R}} \oplus \triangleright^{\mathrm{C}}\mathrm{d}^*(\cdot\,\mathrm{d} + \mathrm{d}\triangleleft_{\mathrm{G}}\cdot)\,\mathrm{d}^*\triangleleft_{\mathrm{B}}.$$

Example 7: (Example 3.6 in [12]) Fig. 7 is an FA that accepts strings of 0's and 1's such that either the second or third position from the end has a symbol 1. From the figure, we obtain the system of equations as follows.

$$
\begin{cases}
\triangleright A \to (0+1)\,A + 1B & \cdots (1) \\
B \to (0+1)\,C & \cdots (2) \\
C \to \triangleleft\,(0+1)\,D & \cdots (3) \\
D \to \triangleleft & \cdots (4)
\end{cases}
$$

Note that the symbol $\triangleleft$ for accepting state in (3) is attached at the front of the term $(0+1)D$. In the same manner as the example above, we get

$$A = \triangleright\,(0+1)^*1(0+1)\triangleleft\,(0+1)\triangleleft.$$

Compare this expression with the ordinary regular expression $(0+1)^*1(0+1)+(0+1)^*1(0+1)(0+1) = (0+1)^*1(0+1)(\varepsilon+(0+1))$ [12].

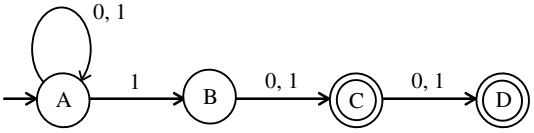


Fig. 7 An NFA accepting strings that have a symbol 1 either two or three positions from the rightmost end [12].

Example 8: Fig. 8 is an FA that was originally designed to accept in state p_0 the binary numbers which are multiples of 6. It can also distinguish the other cases in 3 different colors. From the figure, we obtain the system of equations as follows.

$$\begin{cases} \triangleright p_0 \to 0\,p_0 + 1\,p_1 + \triangleleft_0 & \cdots(0) \\ p_1 \to \triangleleft_1 0\,p_2 + 1\,p_3 & \cdots(1) \\ p_2 \to \triangleleft_2 0\,p_1 + 1\,p_2 & \cdots(2) \\ p_3 \to \triangleleft_3 0\,p_0 + 1\,p_1 & \cdots(3) \end{cases}$$

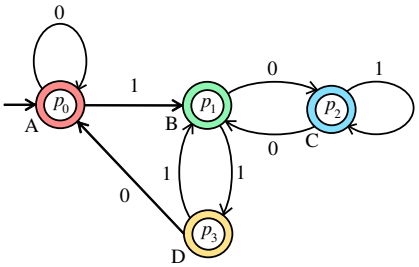


Fig. 8 A deterministic biCFA accepting binary numbers which are multiples of 6 and also classifying the remaining numbers into three categories.

Note that the symbols $\triangleleft_1$, $\triangleleft_2$, and $\triangleleft_3$ for accepting states in (1), (2), and (3) are attached at the front of the terms $0\,p_2, 0\,p_1$, and $0\,p_0$, respectively.
Applying Lemma 1 to (2), we have
$p_2 \to 1^*(\triangleleft_2 0\,p_1).\cdots(2')$
Substituting (2′) and (3) to (1), we have
$p_1 \to \triangleleft_1 0(1^*\triangleleft_2 0\,p_1) + 1(\triangleleft_3 0\,p_0 + 1\,p_1)$
$= (\triangleleft_1 01^*\triangleleft_2 0 + 11)p_1 + 1\triangleleft_3 0\,p_0.$
Therefore from Lemma 1, we get
$p_1 = (\triangleleft_1 01^*\triangleleft_2 0 + 11)^*1\triangleleft_3 0\,p_0.$
Substituting it to (0), we have
$\triangleright p_0 \to 0\,p_0 + 1((\triangleleft_1 01^*\triangleleft_2 0 + 11)^*1\triangleleft_3 0\,p_0) + \triangleleft_0$
$= (0 + 1(\triangleleft_1 01^*\triangleleft_2 0 + 11)^*1\triangleleft_3 0)p_0 + \triangleleft_0.$
Therefore, we get

$$p_0 = \triangleright(0 + 1(\triangleleft_1 01^*\triangleleft_2 0 + 11)^*1\triangleleft_3 0)^*\triangleleft_0.$$

It is known that the order of eliminated states enormously affects the appearance of resulting REs transformed from FAs. The next example shows that a preferable phenomenon happens in the bilateral case.

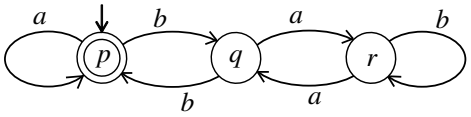


Fig. 9 The FA $\mathcal{D}_3$ in [11] for demonstrating the effect of the order of state elimination.

Example 9: (Example 16 in [11]) Below are the original equations for the FA M in [11].

$$\begin{cases} i \to p & \cdots(1) \\ p \to a\,p + b\,q + t & \cdots(2) \\ q \to a\,r + b\,p & \cdots(3) \\ r \to a\,q + b\,r & \cdots(4) \\ t \to \varepsilon, & \cdots(5) \end{cases}$$

where i is the initial state of M. Abbreviating (1) and (5) and moving the accepting position symbol $\triangleleft$ at the tail of (2) to the head of the term bq, we get a simplified system of equations (See Fig. 9):

$$\begin{cases} \triangleright p & \to a\,p + \triangleleft b\,q & \cdots(2') \\ q & \to a\,r + b\,p & \cdots(3) \\ r & \to a\,q + b\,r & \cdots(4) \end{cases}$$

In case of the elimination order $p < q < r$ (from left to right in the figure): From (2′) and Lemma 1, we have
$p = \triangleright a^*\triangleleft b\,q.$
Substituting it to (3) and using Lemma 1,
$q = a\,r + b(\triangleright a^*\triangleleft b\,q) = b\triangleright a^*\triangleleft b\,q + a\,r$
$= (b\triangleright a^*\triangleleft b)^*a\,r.$
Substituting it to (4),
$r \to a((b\triangleright a^*\triangleleft b)^*a\,r) + b\,r = (a(b\triangleright a^*\triangleleft b)^*a + b)\,r + \emptyset.$
Hence from Lemma 1 (and the rule (Cr) of Definition 6 with $t = \emptyset$),

$$r = (a(b\triangleright a^*\triangleleft b)^*a + b)^*\emptyset = (a(b\triangleright a^*\triangleleft b)^*a + b)^*.$$

Note the following:

1. Contrary to the ordinary elimination method, backward substitution is not executed after the accomplishment of forward elimination for an arbitrary state (not necessarily for the initial state).
2. The respective ordinary regular expression [11]
$$a^*b(ba^*b)^*a(a(ba^*b)^*a + b)^*a(ba^*b)^*ba^* + a^*b(ba^*b)^*ba^* + a^*$$
easily derived from the corresponding bilateral expression above by using the method described so far.
3. In the same order as $p < q < r$, it is easily shown that the transformation of a generalized FA $\mathcal{D}_n$ having n states produces the biCRE
$z_{2n} = (b(a\cdots(a(b\triangleright a^*\triangleleft b)^*a)^*\cdots a)^*b)^* + a)^*,$
$z_{2n+1} = (a(b\cdots(a(b\triangleright a^*\triangleleft b)^*a)^*\cdots b)^*a)^* + b)^*,$
for each $n \geq 1$ and $z_1 = \triangleright(a+\triangleleft b)^* = \triangleright(a+b)^*(a+\triangleleft b) = \triangleright(a+b)^*a + \triangleright(a+b)^*\triangleleft b = \emptyset + \triangleright(a+b)^*\triangleleft =$

$\triangleright(a+b)^*\triangleleft$, thus its length satisfies $|z_n| = O(n)$. On the other hand, the ordinary regular expression for $\mathcal{D}_n$ can be described as

$$y_{2n} = z'_1 b z'_2 a \cdots a z'_{2n-1} b z''_{2n} b z'_{2n-1} a \cdots a z'_2 b z'_1 + Y_{2n-1},$$
$$y_{2n+1} = z'_1 b z'_2 a \cdots b z'_{2n} a z''_{2n+1} a z'_{2n} b \cdots a z'_2 b z'_1 + Y_{2n},$$

for each $n \geq 1$ and $y_1 = (a+b)^*$, where

$$z'_{2n} = (b(a(\cdots(a(ba^*b)^*a)^*\cdots)^*a)^*b)^*,$$
$$z'_{2n+1} = (a(b(\cdots(a(ba^*b)^*a)^*\cdots)^*b)^*a)^*,$$

for each $n \geq 1$ and $z'_1 = a^*$,

$$z''_{2n} = (b(a(\cdots(a(ba^*b)^*a)^*\cdots)^*a)^*b + a)^*,$$
$$z''_{2n+1} = (a(b(\cdots(a(ba^*b)^*a)^*\cdots)^*b)^*a + b)^*,$$

for each $n \geq 1$, and

$$Y_{2n} = z'_1 b z'_2 a \cdots a z'_{2n-1} b z'_{2n} b z'_{2n-1} a \cdots a z'_2 b z'_1 + Y_{2n-1},$$
$$Y_{2n+1} = z'_1 b z'_2 a \cdots b z'_{2n} a z'_{2n+1} a z'_{2n} b \cdots a z'_2 b z'_1 + Y_{2n},$$

for each $n \geq 1$ and $Y_1 = a^*$, thus its length satisfies $|y_n| = O(n^2) + |Y_{n-1}| = O(n^3)$.

4.2 Transformation of BiCREs to BiCFAs

This subsection will assert that the inverse direction of the previous subsection also holds, i.e., any biCRE can be simulated by a certain biCFA. Our transformation is essentially the same as conventional one since the appearance of biCRE directly reflects the structure of some biCFA.

There are several well-known algorithms to transform ordinary REs to their corresponding FAs [11, 17]. Although any of them can be adapted to our purpose, we adopt here the concept of 'position automaton' (or 'Glushkov automaton') [20, 21], which allows us a concise implementation of the algorithm and is constructed by so-called 'follow' function [22, 23]. The follow function of a regular expression r maps a position of input symbol in r to the set of positions where the corresponding automaton can go through only with ε-transitions.

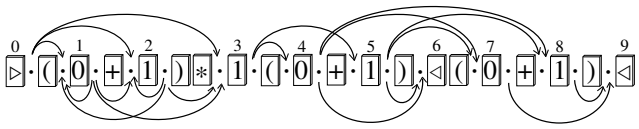


Fig. 10 The ε-transitions derived from the 'follow' function of the expression $\triangleright(0+1)^*1(0+1)\triangleleft(0+1)\triangleleft$.

Example 10: Fig. 10 shows the ε-transitions derived from the follow function of the bilaterally monotone-colored regular expression $\triangleright(0+1)^*1(0+1)\triangleleft(0+1)\triangleleft$ in Example 10.

More precisely, consider a biCRE r over Σ with l kinds of initial state position symbols and k kinds of accepting state position symbols. Let the all symbols excluding operators '+', '$\cdot$', '$*$' and parentheses '(',')' appearing in r be numbered starting with 0 and ending with a natural number N from left to right as shown in Fig. 10. Let $pos(r) = \{0, 1, \ldots, N\}$ and for each $m \in pos(r)$, let $a_m \in \Sigma \bigcup \{\triangleright^j \mid 1 \leq j \leq l\} \bigcup \{\triangleleft_i \mid 1 \leq i \leq k\}$ denotes the mth symbol in such a numbering.

Furthermore, consider the syntax tree t of r whose internal nodes are labeled with expression operators. To construct the function *follow*, which is a mapping from $pos(r)$ to $2^{pos(r)}$, and its three helper functions *nullable*, which maps nodes of t to boolean values, and *first*, *last*, which both are mappings from nodes of t to $2^{pos(r)}$, we can adopt the ordinary procedure described in [22,23] except the setting values for new symbols: In the course of post-order traversal of t, if a visited node v is labeled with an initial state position symbol $\triangleright$, then set $nullable(v) =$ true and $first(v) = \emptyset$, which means that no transition can reach at this symbol but can skip it over. Similarly, if v is labeled with an accepting state position symbol $\triangleleft$, then set $nullable(v) =$ true and $last(v) = \emptyset$, which means that no transition can depart from this symbol but can skip it over.

Definition 7: Let r be a biCRE over Σ with l initial state position symbols and k accepting state position symbols. The *position ε-automaton* for r is a biCFA $M_r = (Q, \Sigma, \delta, \Sigma_{j=1}^{l} Q_0^j, \Sigma_{i=1}^{k} F_i)$, where

- $Q_0^j = \{m\cdot \mid a_m = \triangleright^j, m \in pos(r)\}, 1 \leq j \leq l$,
  $F_i = \{\cdot m \mid a_m = \triangleleft_i, m \in pos(r)\}, 1 \leq i \leq k$,
  $Q = \{\cdot m, m\cdot \mid a_m \in \Sigma, m \in pos(r)\} \bigcup \Sigma_{j=1}^{l} Q_0^j \bigcup \Sigma_{i=1}^{k} F_i$, and
- $\delta(m\cdot, \varepsilon) = \{\cdot n \mid n \in follow(m)\}$ and especially if $a_m \in \Sigma$, $\delta(\cdot m, a_m) = \{m\cdot\}$ for each $m \in pos(r)$.

For example, the resulting automaton $M_r = (Q, \Sigma, \delta, q_0, F)$ converted from r of Fig. 10 is as follows:

- $Q = \{0\cdot, \cdot 1, 1\cdot, \cdot 2, 2\cdot, \cdot 3, 3\cdot, \cdot 4, 4\cdot, \cdot 5, 5\cdot, \cdot 6, \cdot 7, 7\cdot, \cdot 8, 8\cdot, \cdot 9\}$, $q_0 = 0\cdot$, $F = \{\cdot 6, \cdot 9\}$,
- $\delta(0\cdot, \varepsilon) = \{\cdot 1, \cdot 2, \cdot 3\}$ since $follow(0) = \{1, 2, 3\}$, $\delta(\cdot 1, \text{`}0\text{'}) = \{1\cdot\}, \ldots, \delta(8\cdot, \varepsilon) = \{\cdot 9\}$.

Based on this information, we can simulate the behavior of the automaton on any given input string. Below is the running process of M_r on string 00101 from the initial configuration to an accepting configuration, where we use the derivative notation [13, 24], i.e., $\partial\, w$ denotes the regular expression which expresses the situation when the automaton have read the prefix w of 00101.

$$
\begin{aligned}
\partial\,\varepsilon &= \triangleright\,(0+1)^*1(0+1)\triangleleft\,(0+1)\triangleleft \\
&\quad\sim (\triangleright\,0+\triangleright\,1)^*\triangleright\,1(0+1)\triangleleft\,(0+1)\triangleleft, \\
\partial\,0 &= (0\triangleright\,+1)^*1(0+1)\triangleleft\,(0+1)\triangleleft \\
&\quad\sim (\triangleright\,0+\triangleright\,1)^*\triangleright\,1(0+1)\triangleleft\,(0+1)\triangleleft = \partial\,\varepsilon, \\
&\quad\vdots \\
\partial\,00101 &= (0+1\triangleright\,)^*1\triangleright\,(0+1)\triangleleft\,(0+1\triangleright\,)\triangleleft \\
&\qquad\sim (\triangleright\,0+\triangleright\,1)^*\triangleright\,1(\triangleright\,0+\triangleright\,1)\triangleleft\,(0+1)\triangleright\triangleleft.
\end{aligned}
$$

The relational notation $s \sim t$ above for two bilateral regular expressions s, t is not meant a kind of similarity but just the equality of the languages expressed by them, e.g., $\triangleright\,(0+1)^*1(0+1)\triangleleft\,(0+1)\triangleleft = (\triangleright\,0+\triangleright\,1)^*\triangleright\,1(0+1)\triangleleft\,(0+1)\triangleleft$ can be verified in the same way as described so far. Note that such s and t correspond to so-called 'mark-after' automaton and 'mark-before' automaton, respectively in [20].

5. Applications of Bilaterally Colored Acceptance

5.1 Enhancement of Regular Expression Engines

In the previous section, we have seen that there exist a bilateral finite automaton for a given bilateral regular expression and vice versa. Based on these observations, we have implemented a Unix-style biCRE simulator [25] which is equipped with most basic features that any regex engine might have (including support of repeated concatenation quantifier {m,n}). By using this tool, we assert that all the examples of (bilaterally colored or not) regular expressions appearing in this paper certainly accord with their original intentions. Fig. 11[†] is a code to demonstrate its usage for matching task of the biCRE in Example 6.

```
import bicre
for line in @["12.34","-1.23","1234","+123",".123","123.","."]:
  echo "[text = \"" & line & "\"]:"
  if line =~ r":>^R[+-]:>^C[0-9]*(\.[0-9]|[0-9]<:_G\.)[0-9]*<:_B":
    if "R-B" in kinds: echo "fractional number with sign"
    if "C-B" in kinds: echo "fractional number without sign"
    if "R-G" in kinds: echo "integer number with sign"
    if "C-G" in kinds: echo "integer number without sign"
    # there exisit only four kinds of acceptance enumerated above
  else: echo "something else"
```

Fig. 11 A programming example using the biCRE matching engine built in Nim language for matching a bilateral colored regular expression with fractional numbers and others simultaneously (See Example 6).

5.2 Simplification of Regular Right Part Context-Free Grammar Parsers

In principle, we cannot obtain a complete solution for the equations composed of rewriting rules of a context-free grammar due to their inherently nonlinear recursions. We can, however, simplify these equations by making use of the technique developed for bilateral regular expressions in case of context-free grammars whose rewriting rules have star operators in their right side bodies.

Example 11: A bilaterally colored extended (i.e., regular right part) context-free grammar for arithmetic expressions with plus and multiplying operators is given as follows.

$$
\begin{cases}
\triangleright^1 E \to T\{\oplus T\}^*\triangleleft_1 = \{T\oplus\}^*T\triangleleft_1 = \{T\triangleleft_1\oplus\}^* & \cdots(1)\\
\triangleright^2 T \to F\{\odot F\}^*\triangleleft_2 = \{F\odot\}^*F\triangleleft_2 = \{F\triangleleft_2\odot\}^* & \cdots(2)\\
\triangleright^3 F \to (\,[\,E\,]+i\,)\triangleleft_3, & \cdots(3)
\end{cases}
$$

where E, T, F, and i represents the nonterminal variables for expressions, terms, factors, and identifiers, respectively. The rightmost equalities of (1) and (2) above are derived by the rule (Sa) of Definition 6.
From (3),
$F = \triangleright^3(\,[\,E\,]+i\,)\triangleleft_3$.
Substituting it to (2),
$\triangleright^2 T \to \{\triangleright^3(\,[\,E\,]+i\,)\triangleleft_3\triangleleft_2\odot\}^*$.
Then, we have
$T = \triangleright^2\{\triangleright^3(\,[\,E\,]+i\,)\triangleleft_3\triangleleft_2\odot\}^*$.
Substituting it to (1),
$\triangleright^1 E \to \{\triangleright^2\{\triangleright^3(\,[\,E\,]+\ i\,)\triangleleft_3\triangleleft_2\odot\}^*\triangleleft_1\oplus\}^*$.
Therefore, we have

$$E \to \triangleright^1\{\triangleright^2\{\triangleright^3(\,[\,E\,]+\ i\,)\triangleleft_3\triangleleft_2\odot\}^*\triangleleft_1\oplus\}^*.$$

By viewing a pair $\triangleright, \triangleleft$ of state position symbols as a pair of left and right parentheses, we observe in the above that those and other ordinary parentheses {,} are improperly nested (i.e., crossing each other).

We can directly translate the one-line grammar above to an actual parser as shown in Fig. 12. The position symbols $\triangleright^j$'s, $\triangleleft_i$'s and their eviction from star operators are useful in this case just as in bilateral regular expression. For example, the derivation tree for a terminal string $a \odot b \oplus c$ is explicitly visible in the sequence of derivations as shown below.

$$
\begin{aligned}
E &\Rightarrow \triangleright^1\{\triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\odot\}^*\triangleleft_1\oplus\}^* \\
&= \triangleright^1\{\triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\odot\}^*\oplus\}^* \\
&\qquad\qquad \{\triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\odot\}^*\triangleleft_1\} \\
&= \triangleright^1\{\triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\odot\}^*\oplus\}^* \\
&\qquad\qquad \triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\odot\}^*\triangleleft_1 \\
&\Rightarrow \triangleright^1\{\triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\odot\}^*\oplus\} \\
&\qquad\qquad \triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\odot\}^*\triangleleft_1 \\
&= \triangleright^1\triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\odot\}^*\oplus \\
&\qquad\qquad \triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\odot\}^*\triangleleft_1 \\
&\Rightarrow \triangleright^1\triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\odot\}^*\oplus \\
&\qquad\qquad \triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\}\triangleleft_1 \\
&= \triangleright^1\triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\odot\}^*\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\}\oplus \\
&\qquad\qquad \triangleright^2\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\triangleleft_1 \\
&\Rightarrow \triangleright^1\triangleright^2\{\triangleright^3(\,[\,E\,]+i)\triangleleft_3\odot\}\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\oplus \\
&\qquad\qquad \triangleright^2\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\triangleleft_1 \\
&= \triangleright^1\triangleright^2\triangleright^3(\,[\,E\,]+i)\triangleleft_3\odot\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\oplus \\
&\qquad\qquad \triangleright^2\triangleright^3(\,[\,E\,]+i)\triangleleft_3\triangleleft_2\triangleleft_1 \\
&\overset{*}{\Rightarrow} \triangleright^1\triangleright^2\triangleright^3 a\triangleleft_3\odot\triangleright^3 b\triangleleft_3\triangleleft_2\oplus\triangleright^2\triangleright^3 c\triangleleft_3\triangleleft_2\triangleleft_1.
\end{aligned}
$$

[†]If the colors of languages accepted by biCFA transformed from a given biCRE is guaranteed to be "unmixed" [6, 7], we can adopt `case` statement instead of repetition of `if` statements just as proposed in [14].

Example 12: In the same way as Example 11, we can obtain a bilaterally colored extended context-free grammar

$$E \to \triangleright^1\{\triangleright^2\{\triangleright^3([\,E\,]+i)\triangleleft_3\circledast\triangleleft_3\triangleleft_2\}^*\triangleleft_1\oplus\}^*$$

for ordinary regular expressions whose concatenating operators are implicit (the symbol $\circledast$ denotes Kleene star unary operator).

In fact, the parser module in the biCRE engine mentioned in the previous subsection was constructed based on the one-line grammar described above (its final code consists of about 100 lines).

```
def E():
  global i
  exp = 0
  while (True):   # :>^1
    term = 1
    while (True):   # :>^2
      if inp[i] == '[':   # :>^3
        i += 1; factor = E()
        if inp[i] != ']': raise Exception("NOT ']'")
      elif not inp[i].isdigit(): raise Exception("NOT a digit")
      else: factor = int(inp[i])
      term = term * factor   # <:_3
      i += 1
      if inp[i] != '*': break   # <:_2
      i += 1   # else: inp[i] == '*'
    exp = exp + term
    if inp[i] != '+': break   # <:_1
    i += 1   # else: inp[i] == '+'
  return exp
while (True):
  i = 0
  inp = input("expression or q >>") + "$"
  if inp == "q$": break
  print(E())
  if inp[i] != '$': raise Exception("NOT '$'")
```

Fig. 12 An arithmetic expression parser in Python language which is directly translated from the grammar $E \to \triangleright^1\{\triangleright^2\{\triangleright^3([\,E\,]+i)\triangleleft_3\triangleleft_2\odot\}^*\triangleleft_1\oplus\}^*$ with starting rule $S \to E\$$.

6. Summary and Further Research

This paper proposed new computational models, called bilaterally colored finite automata and bilaterally colored regular expressions whose powers of acceptance or expressiveness are the same. Beyond the conventional framework, these concepts give more compact representations of set of regular languages and more flexibility to designing of them, despite the fact that they are entangled clusters of conventional models. Particularly, even mono-colored bilateral regular expression has fair advantages worth the cost of new ingredients of triangle-shaped symbols '$\triangleright$' and '$\triangleleft$': Thanks to the omission of backward substitution step in the process of Gaussian state elimination, it can be quickly derived from a system of equations, which is equivalent to the transition diagram of a finite automaton, then in most cases the obtained expression is shorter than the ordinary one, and its conversion to the form of ordinary expressions is rather laborious but merely a certain boilerplate routine in accordance with its definition. In fact, we have straightforwardly built a tool that decomposes biCRE into a tuple of ordinary regular expressions [25].

At the end of this paper, we propose future research topics: (1) complete implementation of biCRE as pattern matching engine, which currently lacks of constructs such as catching mechanism of matched substring and (2) rigorous definitions (and their possible proofs) of the newly introduced notions, such as a bilaterally colored extended context-free grammar in Section 5.

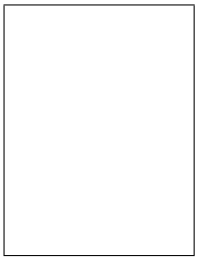
Acknowledgements

We deeply thanks reviewers and editor for their valuable comments especially to the improvement of Definition 5, 6, and Fact 2.

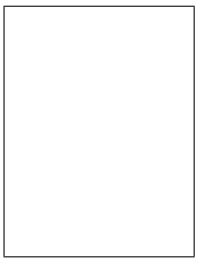
References

[1] S. Yu, Regular languages, Handbook of Formal Languages Vol. 1, pp. 43–110, Springer, 1997.
[2] D. Perrin, Finite automata, Handbook of Theoretical Computer Science Vol. B, The MIT Press, pp. 1–57,1994.
[3] L. D. Antoni and M. Veanes, Symbolic automata better balances how automata are implemented in practice, Communication of the ACM 64, No. 5, pp. 86–95, 2021.
[4] S. T. O'Neil, A Primer for Computational Biology, Oregon State University Press, 2017.
[5] K. N. Kumar and S. Sukumaran, A survey on network intrusion detection system techniques, International Journal of Advanced Technology and Engineering Exploration, Vol. 5(47), pp. 385-393, 2018.
[6] Y. Takahashi and A. Ito, Finite Automata with Colored Accepting States and Their Unmixedness Problems, IEICE Transactions on Information and Systems, Vol. E 105-D, No. 3, pp. 491–502, 2022.
[7] Y. Takahashi and A. Ito, On the Unmixedness Problems of Colored Pushdown Automata, IEICE Transactions on Information and Systems, Vol.E106-D, No. 3, pp. 303–308, Mar. 2023.
[8] A. Asperti, C. S. Coen, and E. Tassi, Regular expressions, au point. CoRRabs/1010.2604 (2010), `http://arxiv.org/abs/1010.2604.pdf` (extracted 2024).
[9] A. Asperti, A compact proof of decidability for regular expression equivalence, LNCS 7406, pp. 283–298, Springer, 2012.
[10] T. Nipkow and D. Traytel, Unified decision procedures for regular expression equivalence, LNCS 8558, pp. 450—466, Springer, 2014.
[11] J. Sakarovitch, Automata and rational expressions, Handbook of Automata Theory, J.-E. Pin ed., EMS Publishing, 2021.
[12] J. E. Hopcroft, R. Motowani, and J. D. Ullman, Intoroduction to Automaton Theory, Languages, and Computation, Addison-Wesley Longman, 3rd ed, Pearson Education, 2007.
[13] J. A. Brzozowski, Derivatives of Regular Expressions, J. ACM 11 (4), pp. 481–494, 1964.
[14] A. Ito and Y. Takahashi, Bilaterally Colored Finite Au-

tomata and their Regular Expressions with Application to Context-Free Parsing, to appear in Proceedings of 2024 the 3rd International Conference on Computer Technologies, 2024.
[15] D. C. Kozen, Automata and Computability, Springer Science & Business Media, 2007.
[16] WHATWG.org, HTML Standard 4.10.5.1.5 Email state (type=email), `https://html.spec.whatwg.org/multipage/input.html#email-state...1` (extracted 2024).
[17] H. Gruber and M. Holzer, From Finite Automata to Regular Expressions and Back — A Summary on Descriptional Complexity, Int. J. of Foundations of Computer Science 26 (08), pp. 1009–1040, 2015.
[18] M. V. Lawson, Finite Automata, CRC Press, 2004.
[19] J. Sakarovitch, Elements of Automata Theory, Cambridge University Press, 2009.
[20] S. Broda, M. Holzer, E. Maia, N. Moreira, and R. Reis, On the Mother of All Automata: the Position Automaton, LNCS 10396, pp. 134–146, Springer, 2017.
[21] S. Fischer, F. Huch, and T. Wilke, A play on regular expressions: functional pearl, ACM SIGPLAN Notices, Vol. 45, pp. 357–368, 2010.
[22] A.V. Aho, et al., Compilers : principles, techniques, and tools - 2nd ed., Pearson Education, 2007.
[23] A. Brüggemann-Klein, Regular expressions into finite automata, Theoretical Computer Science 120, pp. 197–213, 1993.
[24] V. Antimirov, Partial derivatives of regular expressions and finite automaton constructions, Theoretical Computer Science 155, pp. 291–319, 1996.
[25] A. Ito, biCREtools, GitHub repository, `https://github.com/ubeito/biCREtools` (extracted 2024).



Akira Ito received the D.E. degrees in Faculty of Engineering, Nagoya University in 1992. During 1983–2023, he stayed in Faculty of Engineering, Yamaguchi University and is now an independent researcher. Engaged in research on automata, formal language, algorithm, and complexity theory.



Yoshiaki Takahashi received the M.E. degrees and D.E. degrees in Faculty of Engineering, Yamaguchi University in 2008 and 2022, respectively. He is now an associate professor at the National Institute of Technology, Oshima College. Engaged in research on automata and language theory.