A novel algorithm for the conversion of shuffle regular expressions into non-deterministic finite automata

Ajay Kumar* and Anil Kumar Verma

Department of Computer Science and Engineering, Thapar University, Patiala, Punjab-147004, India

*Corresponding author, e-mail: (ajayloura@gmail.com)

Received: 19 September 2012 / Accepted: 7 October 2013 / Published: 14 October 2013

Abstract: Regular expressions with shuffle operators are widely used in diverse fields of computer science. The work presented here investigates the shuffling of regular expressions and their conversion into non-deterministic finite automata. The aim of the paper is to design a novel algorithm for constructing ε-free non-deterministic finite automata from the shuffling of regular expressions. Non-deterministic finite automata generated using the proposed approach requires, in the worst case, $2^{m+s+1}$ states. This is a significant improvement over other existing approaches in the literature, where the number of states reaches $2^{2(m+u+k+s)-c}$ in the worst case.

Keywords: regular expression, parallel regular expression, non-deterministic finite automaton, shuffle operator

INTRODUCTION

Regular expressions (REs) are used in various applications such as compiler designs, XML schema languages, text processors, and as a programming tool for multifarious scripting languages such as PERL and PHP [1-4]. REs used in these applications require standard operators (union, concatenation and Kleene closure). Extended REs [5] consist of shuffle, intersection and counting operators, in addition to standard operators. Although the use of these additional operators do not increase the expressive power of REs, succinctness occurring due to the addition of these operators is difficult to handle with the standard operators [5]. Extended REs are used in diverse applications such as XML schema languages [6], text processors and programming languages [7]. A semi-extended RE [5] consists of union, concatenation, Kleene closure and intersection operators. REs with union, concatenation, Kleene closure and counting operators are used in different areas such as egrep, PERL and XML schema [5].

A shuffle operator is used in the modelling of process algebra, concurrent systems and critical section problems [2, 8-9]. A parallel RE (PRE) consists of disjunction (+), concatenation (.), Kleene
closure (*) and shuffle (III) operators [2]. Shuffle operators appear in diverse forms of computer science such as process algebra, multipoint communications, XML schema Relax NG and concurrency of processes [6, 9-11].

PRELIMINARY CONCEPTS

For the rest of the paper, $\Sigma$ denotes an alphabet depicting a non-empty set of symbols [3]. A string $s=a_1a_2...a_n$ is a finite sequence of symbols taken from the alphabet. The position of symbol $a_i$ is equal to $i$. The length of a string $s$ (denoted by $|s|$) represents the number of occurrence of symbols presented in $s$. $\Sigma^*$ [3] depicts a set of all strings generated from $\Sigma$. An empty string (denoted by $\varepsilon$) is a string which has no symbols. The null language (denoted by $\phi$) does not contain any string. A language $L$ is a subset of $\Sigma^*$ [3]. A language can be classified into regular, context-free, context-sensitive and recursive enumerable. A regular language can be represented by an RE or a finite automaton (FA).

$RE \alpha$, where $\alpha \in \Sigma^*$, depicts a regular language $\{\alpha\}$. $\varepsilon$ and $\phi$ are the REs describing null string ($\varepsilon$) and null language ($\phi$) respectively. If $r_1$ and $r_2$ are REs depicting the languages $L_1$ and $L_2$ respectively, then union, concatenation and Kleene closure are described by the REs $r_1 + r_2$, $r_1r_2$ and $r_1^*$ respectively. Union, concatenation and Kleene closure rules can be finitely applied many times [3].

$FA$ is a quintuple [3] $M = (Q, \Sigma, \delta, q_0, F)$, where $Q$ is the set of states, $\Sigma$ is an alphabet, $\delta$ is a transition function $\delta \subseteq Q \times \{\Sigma \cup \varepsilon\} \times Q$, $q_0 \in Q$ is the starting state and $F \subseteq Q$ is the set of final states. Deterministic finite automaton (DFA) can be represented by a partial function $(\delta : Q \times \Sigma \rightarrow Q)$. Non-deterministic finite automaton (NFA) may or may not involve $\varepsilon$-transitions; $\varepsilon$-free NFA does not contain any $\varepsilon$-transitions and its transition function can be represented by $\delta \subseteq Q \times \Sigma \times Q$.

Shuffling [3] of two strings $x$ and $y$ comprises all strings $z$ such that each position of $z$ is assigned either to $x$ or $y$ for each string $z$. If we concatenate the symbols of the positions prescribed to $x$ from left to right, we obtain the string $x$. Similarly, if we concatenate the symbols of the positions prescribed to $y$ from left to right, we obtain the string $y$. Formally, shuffling can be defined by $a \alpha \lambda \beta b \beta = a(\alpha \lambda \beta b \beta) \cup b(a \alpha \lambda \beta \beta)$ for $(a, b \in \Sigma) \land (\alpha, \beta \in \Sigma^*)$. For example, given $\alpha = 01$ and $\beta = ab\gamma$, then $\alpha \lambda \beta = \{01ab, 0a1b, 0a1b, ab01, 0a0b1, a01b\}$. Shuffling of two languages $L_i$ and $L_j$ can be defined by $L_i \lambda L_j = \{w | w = w_1 \lambda w_2, (\exists w_1 \in L_i) \land (\exists w_2 \in L_j)\}$.

A PRE can be described using a parallel finite automaton (PFA) [12], which consists of 7-tuple $(Q, q_0, N, \Sigma, F, \delta, \gamma)$, where $Q \subseteq 2^N$ is a finite set of states, $q_0$ is the starting state, $N$ is a finite non-empty set of nodes, $\Sigma$ is an alphabet, $F \subseteq N$ is a set of final nodes, $\delta$ is a state transition function defined by $Q \times (\Sigma \cup \lambda) \rightarrow 2^Q$, and $\gamma$ is a node transition function defined by $2^N \times (\Sigma \cup \lambda) \rightarrow 2^{2^N \times N}$. A $\lambda$-transition represents the joining of two languages with a shuffle operator.

A shuffle RE (SRE) is a confined form of PRE, in which the shuffle operators explicitly occur between REs. Conversion of an SRE to an RE or a DFA has been studied widely with respect to the concurrency aspects.
LITERATURE REVIEW

Estrade et al. [2] investigated the explicitly PRE, proposed an algorithm for the conversion of SREs to NFAs using PFA as an intermediate step and designed a tool kit FLAT in the PERL language. Figure 1 delineates the PFA corresponding to SRE $r = (10)\mathbb{I}^* 0^*$ using Estrade et al.’s methodology. Biegler et al. [13] reasserted that shuffle decomposition is not unique over the finite sets. Gelade [5] proved that a double exponential size is required for the conversion of PREs to REs. Hovland [14] proposed a membership algorithm for an RE with unordered concatenation, which is a limited form of shuffle operator, for example $(ach\&ge) = \{acbge,geacb\}$, where & represents the unordered concatenation. Standard generalised markup language [14] allows the use of unordered concatenation. Gelade et al. [15] worked on REs with numerical constraints and a shuffle operator. They depicted the overview of complexity for the decision problems of XML schema languages, namely DTDs, RELAX NG, and XSDs. Berstel et al. [16] studied different kinds of language classified based on the shuffle operations and proved partial results for the star-free commutative language.

![Figure 1. PFA for $r = (10)\mathbb{I}^* 0^*$ using Estrade et al.’s methodology](image)

SREs can be used for depicting the number of processes competing for the critical section [8]. Each critical section of a process can be described by a symbol. For example, critical sections of three processes can be represented by $a$, $(ab)^*$ and $b^*$. For a single CPU, the SRE describes the probabilistic ways of scheduling. An equivalent RE commensurate with the SRE renders the way for tasks to be completed. In the case of resource reduction in the modelling of concurrent systems, partial serialisation can be useful [2]. In this paper, we propose a novel algorithm for the conversion of SREs into $\varepsilon$-free NFAs without using intermediate steps.

Methodologies exist in the literature for the conversion of REs into NFAs, namely Thompson’s construction [17], partial derivatives [18] and follow automata [19]. Estrade et al. [2] worked on the conversion of SRE into NFA using PFA as an intermediate state. SRE-to-PFA conversion can be achieved by using modified Thompson’s construction. PFA with $n$ states can be converted into an NFA requiring $2^n$ states in the worst case [2]. However, an RE can be converted into DFA directly with fewer states than the trivial Thompson’s construction. The motivation for this research stems from the scope for the conversion of an SRE into a DFA by extending the direct conversion method.
PROPOSED APPROACH

In the direct method of conversion of REs into DFAs [1, 20], a syntax tree is constructed for the augmented RE. Firstpos, lastpos and nullable are determined using the rules given in Table 1. Additional rules for the shuffle operator are described with those for the direct conversion of RE into DFA [1, 20]. Followpos are determined using the rules given in Table 2. Firstpos, lastpos, followpos and nullable are determined for the nodes of the syntax tree, and using these followpos, an equivalent DFA can be generated.

Table 1. Rules for firstpos(r), lastpos(r) and nullable

<table>
<thead>
<tr>
<th>RE</th>
<th>Firstpos(r)</th>
<th>Lastpos(r)</th>
<th>Nullable</th>
</tr>
</thead>
<tbody>
<tr>
<td>r = φ</td>
<td>φ</td>
<td>φ</td>
<td>false</td>
</tr>
<tr>
<td>r = ε</td>
<td>φ</td>
<td>φ</td>
<td>true</td>
</tr>
<tr>
<td>r = a</td>
<td>position(a)</td>
<td>position(a)</td>
<td>false</td>
</tr>
<tr>
<td>r = r + r</td>
<td>firstpos(r) ∪ firstpos(r)</td>
<td>lastpos(r) ∪ lastpos(r)</td>
<td>nullable(r) ∨ nullable(r)</td>
</tr>
<tr>
<td>r = r r</td>
<td>If ε ∈ L(r)</td>
<td>firstpos(r) ∪ firstpos(r)</td>
<td>lastpos(r) ∪ lastpos(r)</td>
</tr>
<tr>
<td></td>
<td>Else</td>
<td>firstpos(r)</td>
<td>lastpos(r)</td>
</tr>
<tr>
<td>r = r*</td>
<td>firstpos(r)</td>
<td>lastpos(r)</td>
<td>true</td>
</tr>
<tr>
<td>r = r ∨ r</td>
<td>firstpos(r) ∪ firstpos(r)</td>
<td>lastpos(r) ∪ lastpos(r)</td>
<td>nullable(r) ∧ nullable(r)</td>
</tr>
</tbody>
</table>

Table 2. Computation of followpos

<table>
<thead>
<tr>
<th>RE</th>
<th>Followpos</th>
</tr>
</thead>
<tbody>
<tr>
<td>r r ∈ r</td>
<td>For p ∈ lastpos(r)</td>
</tr>
<tr>
<td></td>
<td>followpos(p) ← followpos(p) ∪ firstpos(r)</td>
</tr>
<tr>
<td>r* ∈ r</td>
<td>For p ∈ lastpos(r)</td>
</tr>
<tr>
<td></td>
<td>followpos(p) ← followpos(p) ∪ firstpos(r)</td>
</tr>
</tbody>
</table>

SRE r with s shuffle operators can be dispensed to s+1 sub-regular expressions r_1, r_2, ..., r_{s+1} such that:
1. r=r_1 III r_2 ... III r_{s+1}
2. r_1, r_2, ..., r_{s+1} are the REs.

An augmented SRE can be obtained by adding # before each shuffle operator and at the end of SRE r. A syntax tree (similar to that shown in Figure 2) is constructed for the augmented SRE such that the symbols and #’s appear at leaf nodes and operators appear as the internal nodes. For all leaf nodes, except a node labelled with #, followpos are calculated. Followpos(#) is taken as null set.
Followpos for each sub-regular expression is calculated severally. Consider SRE \( r = r_i \text{III} r_j \) having a single shuffle operator. Firstpos, lastpos and followpos of \( r_i \) and \( r_j \) are calculated severally. The starting state of the NFA is the union of firstpos\((r_i)\) and firstpos\((r_j)\). Lastpos of the root node is determined by taking the union of all positions labelled with \# . State \( q_i \) is a final state if lastpos\((\text{root})\) \( \subseteq q_i \). Using the definition of shuffle, the resulting NFA must include all strings generated from the shuffling of \( r_i \) and \( r_j \). The final state of the NFA accepts all strings generated from \( r_i \text{III} r_j \). At any instant of time during the processing of string \( w \), consider \( q_i \) is the current state. On reading symbol \( a \), from the sub-REs \( r_i \) and \( r_j \), the following two cases can occur after reading symbol \( a \) as shown in Figure 3.

- Reading of symbol \( a \) from \( r_i \):
  \[ q_j \leftarrow \text{followpos}(\text{position}(r_i) \text{labelled with } a \text{ in } q_i) \cup (\text{positions}(r_i) \text{ in } q_i) \]

- Reading of symbol \( a \) from \( r_j \):
  \[ q_k \leftarrow \text{followpos}(\text{position}(r_j) \text{labelled with } a \text{ in } q_j) \cup (\text{positions}(r_j) \text{ in } q_j) \]

Similarly, there is a possibility of reading symbol \( a \) on states \( q_j \) and \( q_k \) from sub-expressions \( r_j \) and \( r_i \) respectively. Hence, if \( (\text{symbols}(r_i) \cap \text{symbols}(r_j) \neq \emptyset) \land (r_j \text{III} r_j \in r) \) , then an NFA is generated from \( r \) using the proposed approach. Repeating this procedure, all possible shuffled strings can be generated. The final state of the NFA will accept all strings which are obtained by \( r_i \text{III} r_j \). In general, SRE \( r \) consists of \((s+1)\) sub-REs \((r_1, r_2, r_3, ..., r_{s+1})\), such that \( r = r_1 \text{III} r_2 \text{III} r_3 \text{III} ... \text{III} r_{s+1} \). At any instant of time during the processing of a string \( w \), consider \( q_i \) is the current state. Symbol \( a \) can be taken from sub-REs \( r_s, r_z \) and \( r_5 \) of the state \( q_i \). The following cases can occur after reading symbol \( a \):

![Figure 2. Syntax tree for augmented SRE](image)

![Figure 3. Reading symbol a from r_i III r_j](image)
Reading symbol \( a \) from \( r_z \):
\[
q_i \leftarrow \text{followpos}(\text{position}(r_i) \text{ labelled with } a) \cup (q_i - \text{positions}(r_i))
\]

Reading symbol \( a \) from \( r_y \):
\[
q_i \leftarrow \text{followpos}(\text{position}(r_i) \text{ labelled with } a) \cup (q_i - \text{positions}(r_i))
\]

Reading symbol \( a \) from \( r_z \):
\[
q_i \leftarrow \text{followpos}(\text{position}(r_i) \text{ labelled with } a) \cup (q_i - \text{positions}(r_i))
\]

Further, reading of symbol \( a \) from the sub-expression \( r_y \) and \( r_z \) on state \( q_j \) is possible. Similarly, there is a possibility of reading symbol \( a \) on states \( q_k \) and \( q_l \). Repeating this procedure, all possible shuffled strings can be generated from the sub-regular expressions \( r_1, r_2, \ldots, r_{s+1} \). The final state will accept all strings of the SRE \( r \). If a common symbol appears in different sub-REs of \( r \) involved in shuffling, then DFA cannot be generated directly from SRE \( r \). Algorithms for the conversion of SRE into NFA are depicted in Schemes 1-3.

---

**Algorithm 1: SRE\_NFA (p,NFA)**

**Input:** SRE \( p \)

**Output:** \( \varepsilon \)-free NFA equivalent to SRE \( p \)
1. Generate augmented SRE \( p \).
2. Construct a syntax tree for augmented SRE \( p \). // Syntax tree construction
3. Calculate\_Follow(\( p \)). // Call to Procedure Calculate\_Follow
4. Create\_NFA(firstpos, lastpos, followpos). // Call to Procedure Create\_NFA

---

**Scheme 1.** Algorithm for the conversion of SRE into \( \varepsilon \)-free NFA

Notations used in the algorithm \( \text{Calculate\_Follow}(p) \) are as follows. Consider \( c_1 \) and \( c_2 \) are the left and right child during traversal of a node delineated by union, concatenation and shuffle operators. Consider \( c_j \) is the only child of a node delineated by Kleene closure. Let \( a \) indicate symbol from \( \Sigma \). Position(\( n \)) depicts the position of the symbol embarked at RE \( n \). Using Brueggemann-Klein’s methodology [20], firstpos, lastpos and followpos are determined for all nodes except for shuffle operator. The rules for determination of firstpos, lastpos, nullable and followpos are depicted in Tables 1-2. The same are represented by the procedure \( \text{Calculate\_Follow}(p) \) as shown in the Scheme 2.

The reading of a symbol of a state is processed by two steps. In the first step we find the followpos of the current symbol from a sub-RE. In the second step we add the positions of other sub-REs involved in a shuffling of the current state to the result of the first step. This process is repeated for each \( q \in Q \) and each \( a \in \Sigma \). The initial and final states are determined using the firstpos and lastpos of the root node respectively. The complete procedure is specified in Scheme 3.
Algorithm 2: Calculate_Follow(p)

Input: Syntax tree for the SRE \( r \)

Output: Firstpos, lastpos and followpos

1. Repeat steps 2 to 6 for each node \( n \) during the post-order traversal of the syntax tree do

2. \( \text{If } (n = 'a' \lor n = '#') \text{ then } \) // Leaf node

\( \text{firstpos}(n) \leftarrow \text{position}(n) \)
\( \text{lastpos}(n) \leftarrow \text{position}(n) \)

End If

3. \( \text{If } n = '+' \text{ then } \) // Union node

\( \text{firstpos}(n) \leftarrow \text{firstpos}(c_1) \cup \text{firstpos}(c_2) \)
\( \text{lastpos}(n) \leftarrow \text{lastpos}(c_1) \cup \text{lastpos}(c_2) \)

End If

4. \( \text{If } n = ',' \text{ then } \) // Concatenation node

\( \text{firstpos}(n) \leftarrow \text{firstpos}(c_1) \)
\( \text{lastpos}(n) \leftarrow \text{lastpos}(c_2) \)

If \( c_1 = '*' \) then // nullable at left child is true

\( \text{firstpos}(n) \leftarrow \text{firstpos}(n) \cup \text{firstpos}(c_2) \)

End If

If \( c_2 = '*' \) then // nullable at right child is true

\( \text{lastpos}(n) \leftarrow \text{lastpos}(n) \cup \text{lastpos}(c_1) \)

End If

For each \( i \in \text{lastpos}(c_1) \) do

\( \text{followpos}(i) \leftarrow \text{followpos}(i) \cup \text{firstpos}(c_2) \)

End For

End If

5. \( \text{If } n = '*' \text{ then } \) // Kleene closure

\( \text{firstpos}(n) \leftarrow \text{firstpos}(c_1) \)
\( \text{lastpos}(n) \leftarrow \text{lastpos}(c_1) \)

For each \( i \in \text{lastpos}(n) \) do

\( \text{followpos}(i) \leftarrow \text{followpos}(i) \cup \text{firstpos}(n) \)

End For

End If

6. \( \text{If } n = 'III' \text{ then } \) // Shuffle operator

\( \text{firstpos}(n) \leftarrow \text{firstpos}(c_1) \cup \text{firstpos}(c_2) \)
\( \text{lastpos}(n) \leftarrow \text{lastpos}(c_1) \cup \text{lastpos}(c_2) \)

End If

Scheme 2. Procedure for computing firstpos, lastpos and followpos of the SRE \( p \)
**Algorithm 3:** Create$_{NFA}$ (firstpos, followpos, lastpos)

**Input:** Firstpos, lastpos and followpos of SRE $p$

**Output:** NFA equivalent to SRE $p$

1. $q_0 \leftarrow \text{firstpos}(\text{root})$
   $s \leftarrow \text{number of shuffle operator in } p$
2. $Q \leftarrow q_0$ and make $q_0$ as unmarked.
3. **Repeat** steps 4-9, while there is an unmarked state $q \in Q$ do
4. Choose an unmarked state $q$ and mark it.
5. $\text{Prevlast} \leftarrow 0$  //indicates the last position of the sub-expression recently processed
   $i \leftarrow 1$
6. **Repeat** steps 7-9 while $(i \leq s + 1)$  // Reading of $a$ on the current sub-expression
7. $\text{currentlast} \leftarrow i^{th}$ value of $\text{lastpos}(\text{root})$
   $T \leftarrow \phi$
   **For** $(\forall pos \in q)$ do
     **If** $(\text{Prevlast} < pos \leq \text{currentlast})$
     $T \leftarrow T \cup pos$
   **End If**
   **End For**
   $N \leftarrow q - T$  // Remaining positions of the other sub-expressions
8. **For** $(\forall a \in \Sigma)$ do
   $\text{Newstate} \leftarrow \phi$
   **For** $(\forall t \in T)$ do
     **If** $(\text{symbol}(t) = a)$ then
     $\text{Newstate} \leftarrow \text{Newstate} + \text{followpos}(t)$
   **End If**
   **End For**
   **If** $\text{Newstate} \neq \phi$
   $\text{Newstate} \leftarrow \text{Newstate} \cup N$
   $\text{Tran}[q,a] \leftarrow \text{Newstate}$
   **If** $\text{Newstate} \notin Q$ then
   $Q \leftarrow Q \cup \text{Newstate}$
   Make $\text{Newstate}$ as unmarked state.
   **End If**
   **End If**
9. $\text{Prevlast} \leftarrow \text{currentlast}$
   $i \leftarrow i + 1$
10. $q_0 = \{ q \mid q = \text{firstpos}(\text{root}) \}$  //Starting state
    $F = \{ q_f \mid q_f \in Q \land \text{lastpos}(\text{root}) \subseteq q_f \}$  //Final state

**Scheme 3.** Algorithm for generating equivalent NFA from SRE $p$
The complete procedure for the conversion of SRE to NFA is expounded by the following numerical example.

**Example.** Given \( r = (a(a+b)^{*}) \text{III} (a^*b^{*}) \)

Augmented \( r = (a_{1}(a_{2}+b_{2})^{*})\#_{1} \text{III} (a_{3}b_{3})\#_{2} \)

Figure 4 delineates the syntax tree for the augmented SRE. The *firstpos* of a node is depicted on the left side of the node in blue colour and the *lastpos* of a node is depicted on the right side of the node in green colour.

The following values are determined using algorithm 2:

- \( \text{followpos}(1) = \{2,3,4\} \)
- \( \text{followpos}(2) = \{2,3,4\} \)
- \( \text{followpos}(3) = \{2,3,4\} \)
- \( \text{followpos}(5) = \{5,6,7\} \)
- \( \text{followpos}(6) = \{6,7\} \)
- \( \text{followpos}(4) = \text{followpos}(7) = \emptyset \)

Figure 5 delineates the NFA \( M = (\{q_{0}, q_{1}, F_{0}, F_{1}\}, \{a,b\}, \delta, q_{0}, \{F_{0}, F_{1}\}) \) corresponding to SRE \( r \).

![Figure 4](image1.png)

*Figure 4.* Syntax tree for \( r = (a(a+b)^{*}) \text{III} (a^*b^{*}) \)

![Figure 5](image2.png)

*Figure 5.* NFA for \( r = (a(a+b)^{*}) \text{III} (a^*b^{*}) \)
DESCRIPTIONAL COMPLEXITY OF THE OBTAINED NFA

In this section, analysis is carried out on state complexity of the NFA obtained using Estrade et al. methodology [2] and the proposed approach. It is well known that a PFA with \( n \) states can be converted into an NFA with \( 2^n \) states. This leads to the following theorems:

**Theorem 1.** If \( C \) is the number of occurrence of the concatenation operator in SRE \( r \), then the NFA obtained using Estrade et al.’s methodology [2] requires \( 2^{2|r|-3C} \) states in the worst case.

**Proof:** Estrade et al.’s methodology converts \( r \) to a PFA using the modified Thompson’s construction [8]. The number of states of the PFA using the modified Thompson’s construction is equal to \( 2^{|r|-3C} \). A PFA with \( n \) states can be converted to an NFA using the subset construction that requires \( 2^n \) states in the worst case [2]. Hence, a PFA with \( 2^{|r|-3C} \) states can be converted to an NFA using a subset construction requiring \( 2^{2^{|r|-3C}} \) states in the worst case. \( \square \)

**Theorem 2.** If \( m \) denotes the number of instances of symbols and \( s \) denotes the number of shuffle operators in the SRE \( r \), then a \( \varepsilon \)-free NFA can be constructed using the proposed algorithm requiring \( 2^{m+s+1} \) states in the worst case.

**Proof.** In the proposed algorithm, the syntax tree is constructed such that all symbols and \('#'s appear as leaf nodes. Each state can be constructed using the positions assigned to the leaf nodes. The total number of leaf nodes in the syntax tree is \((m+s+1)\). The maximum \( 2^{m+s+1} \) states can be constructed using the positions of the syntax tree. Hence, using the proposed algorithm, the worst case state complexity of the NFA is \( 2^{m+s+1} \). \( \square \)

**Theorem 3.** The state complexity of the proposed algorithm is less than the state complexity of Estrade et al.’s methodology [2] for the conversion of an SRE to an NFA.

**Proof.** SRE \( r \) with \( m, u, k, s \) and \( c \) represent instances of symbols, union, Kleene closure, shuffle and concatenations respectively. The worst-case state complexity of the proposed algorithm and of Estrade et al.’s methodology are \( 2^{m+s+1} \) and \( 2^{2(m+u+k+s)-C} \) respectively. That the proposed algorithm will generate an NFA with fewer states can be proved by the relation:

\[
m + s + 1 < 2(m + u + k + s) - C \Rightarrow 1 < m + 2u + 2k + s - C
\]

Each concatenation operation is performed between two symbols. Similarly, \( n \) concatenation can be performed with at least \((n+1)\) symbols, which means \( m - C \geq 1 \).

Hence, the relation \( 1 \leq m + 2u + 2k + s - C \) always holds and the worst-case state complexity of the proposed algorithm is always less than that of Estrade et al.’s methodology. \( \square \)

The worst-case state complexity of the \( \varepsilon \)-NFA is \( 2^{2|r|-3C} \) using Estrade et al.’s methodology [2]. The pattern matching will decelerate if the NFA consists of \( \varepsilon \)-transitions. Using the proposed algorithm, a \( \varepsilon \)-free NFA is generated.

The worst case-state complexity of the NFA is \( 2^{m+s+1} \). Table 3 describes the differences between the proposed approach and Estrade et al.’s methodology [2].
Table 3. Comparison between the proposed algorithm and Estrade et al.’s methodology

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Proposed algorithm</th>
<th>Estrade et al.’s methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst-case state complexity</td>
<td>$2^{m+s+1}$</td>
<td>$2^{2r-3c}$</td>
</tr>
<tr>
<td>Output</td>
<td>$\varepsilon$-free NFA</td>
<td>$\varepsilon$-NFA</td>
</tr>
<tr>
<td>Conversion chain</td>
<td>SRE $\rightarrow$ NFA $\rightarrow$ DFA</td>
<td>SRE $\rightarrow$ PFA $\rightarrow$ NFA $\rightarrow$ DFA</td>
</tr>
</tbody>
</table>

CONCLUSIONS AND FUTURE SCOPE

An algorithm has been designed for the conversion of SREs into $\varepsilon$-free NFAs without using any intermediate steps. Using the proposed approach, a svette NFA is generated. The number of states of the NFA generated using the proposed approach is equal to $2^{m+s+1}$ states in the worst case, which is a significant improvement over the existing approaches in the literature. Another advantage of this approach is that the $\varepsilon$-free NFAs is produced. Pattern matching is delayed if the NFA consists of $\varepsilon$-transitions. Further, if $\forall(r_i \in R_i) \in r \land ((\text{symbols}(r_i) \cap \text{symbols}(r_j)) \neq \emptyset)$, then it is implausible to generate a DFA directly from the SRE. This approach will be useful in the field of process algebra and concurrent aspects. In the future, this work can be extended to the conversion of PREs into REs. The reduction of time and state complexity for the conversion of SREs into NFAs can also be carried out.

REFERENCES


