Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

Maejo International Journal of Science and Technology

ISSN 1905-7873
Available online at www.mijst.mju.ac.th

Full Paper

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

Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

closure (*) and shuffle (Ш) 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, $\sum$ 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. $\sum^*$ [3] depicts a set of all strings generated from $\sum$. An empty string (denoted by ε) is a string which has no symbols. The null language (denoted by ϕ) does not contain any string. A language L is a subset of $\sum^*$ [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 a, where $a \in \sum$, depicts a regular language $\{a\}$. ε and ϕ are the REs describing null string (ε) and null language (ϕ) 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] $M = (Q, \sum, \delta, q_0, F)$, where Q is the set of states, $\sum$ is an alphabet, δ is a transition function $\delta \subseteq Q \times \{\sum \cup \varepsilon\} \times Q$, $q_0 \in Q$ 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 $(\delta : Q \times \sum \to Q)$. Non-deterministic finite automaton (NFA) may or may not involve ε-transitions; *ε-free* NFA does not contain any ε-transitions and its transition function can be represented by $\delta \subseteq Q \times \sum \times Q$.

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 $a\alpha \text{ Ш } b\beta = a(\alpha \text{ Ш } b\beta) \bigcup b(a\alpha \text{ Ш } \beta)$ for $(a, b \in \sum) \wedge (\alpha, \beta \in \sum^*)$. For example, given $\alpha = 01$ and $\beta = ab$, then $\alpha \text{ Ш } \beta = \{01ab, 0a1b, 0ab1, ab01, a0b1, a01b\}$. Shuffling of two languages L_i and L_j can be defined by $L_i \text{ Ш } L_j = \{w \mid w = w_1 \text{ Ш } w_2, (\exists w_1 \in L_i) \wedge (\exists w_2 \in L_j)\}$.

A PRE can be described using a parallel finite automaton (PFA) [12], which consists of 7-tuple $(Q, q_0, N, \sum, F, \delta, \gamma)$, 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, $\sum$ is an alphabet, $F \subseteq N$ is a set of final nodes, δ is a state transition function defined by $Q \times (\sum \cup \lambda) \to 2^Q$, and γ is a node transition function defined by $2^N \times (\sum \cup \lambda) \to 2^{2 \wedge N}$. A λ-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.

Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

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 $r = (10) \text{ш } 0^*$ 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 $(acb \,\&\, ge) = \{acbge, geacb\}$, 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.

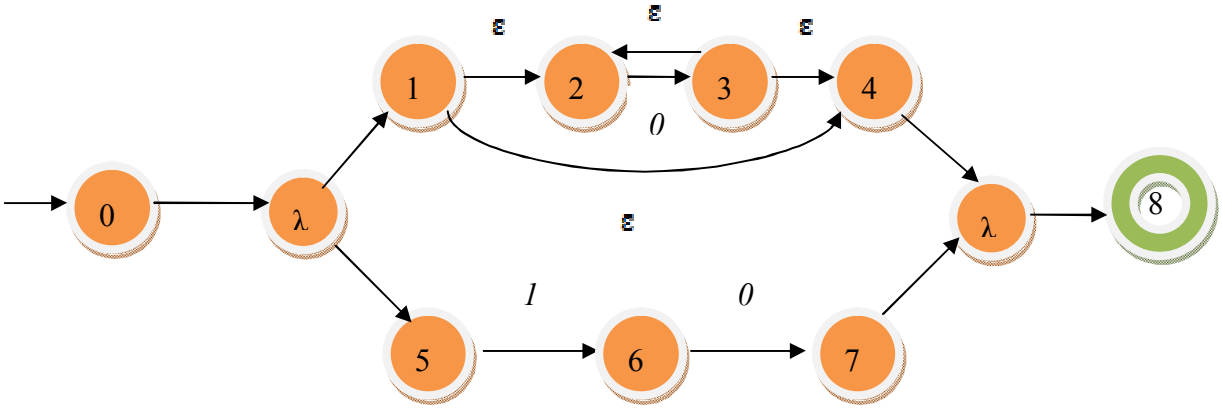

Figure 1. PFA for $r = (10) \text{ш } 0^*$ using Estrade *et al.*'s methodology

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 ε-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.

Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

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*

RE	Firstpos(r)	Lastpos(r)	Nullable
$r = \phi$	ϕ	ϕ	*false*
$r = \varepsilon$	ϕ	ϕ	True
$r = a$	$position(a)$	$position(a)$	*false*
$r = r_i + r_j$	$firstpos(r_i) \cup firstpos(r_j)$	$lastpos(r_i) \cup lastpos(r_j)$	$nullable(r_i) \vee nullable(r_j)$
$r = r_i r_j$	**If** $\varepsilon \in L(r_i)$ $firstpos(r_i) \cup firstpos(r_j)$ **Else** $firstpos(r_i)$	**If** $\varepsilon \in L(r_j)$ $lastpos(r_i) \cup lastpos(r_j)$ **Else** $lastpos(r_j)$	$nullable(r_i) \wedge nullable(r_j)$
$r = r_i^*$	$firstpos(r_i)$	$lastpos(r_i)$	*true*
$r = r_i \text{ Ш } r_j$	$firstpos(r_i) \cup firstpos(r_j)$	$lastpos(r_i) \cup lastpos(r_j)$	$nullable(r_i) \wedge nullable(r_j)$

Table 2. Computation of *followpos*

RE	Followpos
$r_i r_j \in r$	**For** $p \in lastpos(r_i)$ $followpos(p) \leftarrow followpos(p) \cup firstpos(r_j)$
$r_i^* \in r$	**For** $p \in lastpos(r_i)$ $followpos(p) \leftarrow followpos(p) \cup firstpos(r_i)$

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. $r = r_1 \text{ Ш } r_2 \text{ Ш} ... \text{ Ш} r_{s+1}$
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.

Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

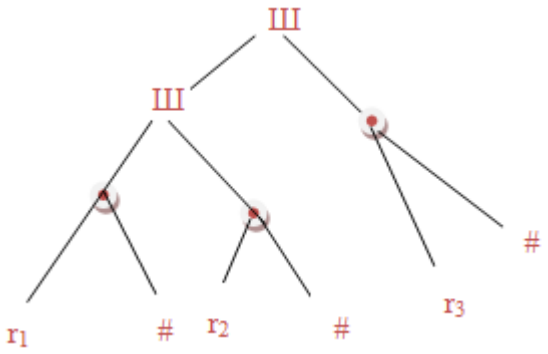

Figure 2. Syntax tree for augmented SRE

Followpos for each sub-regular expression is calculated severally. Consider SRE $r = r_i \text{ Ш } r_j$ 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 $lastpos(root) \subseteq q_i$. 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 $r_i \text{ Ш } r_j$. 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:
  $q_j \leftarrow followpos((position(r_i)\ labelled\ with\ \ a)\ in\ \ q_i) \cup (positions(r_j)\ in\ \ q_i)$
- Reading of symbol *a* from r_j:
  $q_k \leftarrow followpos((position(r_j)\ labelled\ with\ \ a)\ in\ \ q_i) \cup (positions(r_i)\ in\ \ q_i)$

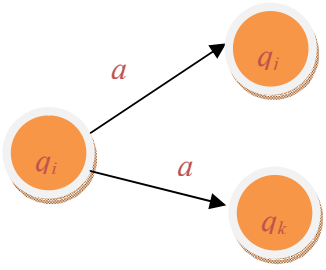

Figure 3. Reading symbol *a* from $r_i \text{ Ш } r_j$

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 $(symbols(r_i) \cap symbols(r_j) \neq \phi) \wedge (r_i \text{ Ш } r_j \in r)$, 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 $r_i \text{ Ш } r_j$. In general, SRE *r* consists of $(s+1)$ sub-REs $(r_1, r_2, r_3, \ldots, r_{s+1})$, such that $r = r_1 \text{ Ш } r_2 \text{ Ш } r_3 \text{ Ш } \ldots \text{ Ш } r_{s+1}$. 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_x, r_y and r_z of the state q_i. The following cases can occur after reading symbol *a*:

Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

- Reading symbol a from r_x:
  $q_j \leftarrow followpos((position(r_x)\ labelled\ with\ \ a)\ in\ \ q_i) \cup (q_i - positions(r_x))$
- Reading symbol a from r_y:
  $q_k \leftarrow followpos((position(r_y)\ labelled\ with\ \ a)\ in\ \ q_i) \cup (q_i - positions(r_y))$
- Reading symbol a from r_z:
  $q_l \leftarrow followpos((position(r_z)\ labeled\ with\ \ a)\ in\ \ q_i) \cup (q_i - positions(r_z))$

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: ε -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 ε -free NFA

Notations used in the algorithm *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_1 is the only child of a node delineated by Kleene closure. Let a indicate symbol from $\sum$. 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 *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 $a \in \sum$. 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.

Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

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. **If** $(n = 'a' \vee n = '\#')$ then // Leaf node
   $firstpos(n) \leftarrow position(n)$
   $lastpos(n) \leftarrow position(n)$
   End If
3. **If** $n = '+'$ then // Union node
   $firstpos(n) \leftarrow firstpos(c_1) \cup firstpos(c_2)$
   $lastpos(n) \leftarrow lastpos(c_1) \cup lastpos(c_2)$
   End If
4. **If** $n = '.'$ then // Concatenation node
   $firstpos(n) \leftarrow firstpos(c_1)$
   $lastpos(n) \leftarrow lastpos(c_2)$
   If $c_1 = '*'$ then // nullable at left child is true
   $firstpos(n) \leftarrow firstpos(n) \cup firstpos(c_2)$
   End If
   If $c_2 = '*'$ then // nullable at right child is true
   $lastpos(n) \leftarrow lastpos(n) \cup lastpos(c_1)$
   End If
   For each $i \in lastpos(c_1)$ do
   $followpos(i) \leftarrow followpos(i) \cup firstpos(c_2)$
   End For
   End If
5. **If** $n = '*'$ then // Kleene closure
   $firstpos(n) \leftarrow firstpos(c_1)$
   $lastpos(n) \leftarrow lastpos(c_1)$
   For each $i \in lastpos(n)$ do
   $followpos(i) \leftarrow followpos(i) \cup firstpos(n)$
   End For
   End If
6. **If** $n = ' ш '$ then // Shuffle operator
   $firstpos(n) \leftarrow firstpos(c_1) \cup firstpos(c_2)$
   $lastpos(n) \leftarrow lastpos(c_1) \cup lastpos(c_2)$

   End If

Scheme 2. Procedure for computing *firstpos*, *lastpos* and *followpos* of the SRE p

Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

Algorithm 3: *Create*_NFA (*firstpos, followpos, lastpos*)
Input: *Firstpos*, *lastpos* and *followpos* of SRE *p*
Output: NFA equivalent to SRE *p*

1. $q_0 \leftarrow firstpos(root)$
   $s \leftarrow$ *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. $\Pr evlast \leftarrow 0$ //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. $currentlast \leftarrow i^{th}$ value of $lastpos(root)$
   $T \leftarrow \phi$
   For $(\forall pos \in q)$ do
   If $(\Pr evlast < pos \leq currentlast)$
   $T \leftarrow T \bigcup pos$
   End If
   End For
   $N \leftarrow q - T$ // Remaining positions of the other sub-expressions
8. **For** $\forall a \in \sum$ do
   $Newstate \leftarrow \phi$
   For $\forall t \in T$ do
   If $(symbol(t) = a)$ then
   $Newstate \leftarrow Newstate + followpos(t)$
   End If
   End For
   If $Newstate \neq \phi$
   $Newstate \leftarrow Newstate \bigcup N$
   $Tran[q, a] \leftarrow Newstate$
   If $Newstate \notin Q$ then
   $Q \leftarrow Q \bigcup Newstate$
   Make *Newstate* as unmarked state.
   End If
   End If
9. $prevlast \leftarrow currentlast$
   $i \leftarrow i+1$
10. $q_0 = \{q \mid q = firstpos(root)\}$ //Starting state
    $F = \{q_f \mid q_f \in Q \wedge lastpos(root) \subseteq q_f\}$ //Final state

Scheme 3. Algorithm for generating equivalent NFA from SRE *p*

Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

The complete procedure for the conversion of SRE to NFA is expounded by the following numerical example.

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

Augmented $r = (a_1(a_2+b_3)^*)\#_4 \text{Ш} (a_5^*b_6^*)\#_7$

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.

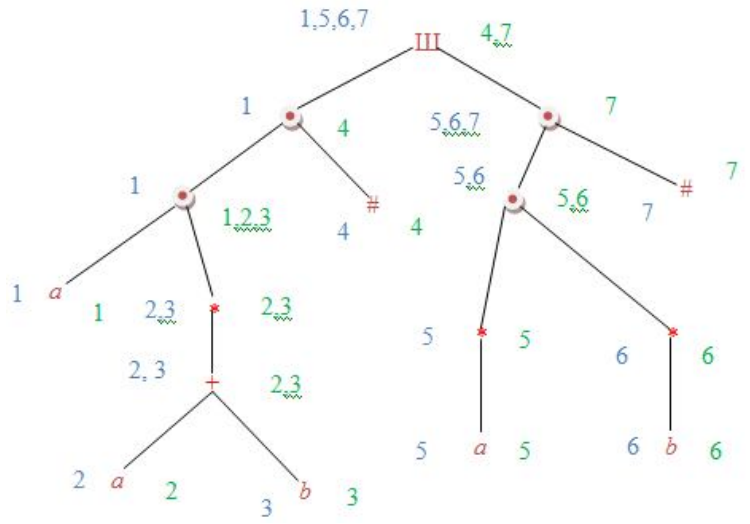

Figure 4. Syntax tree for $r = (a(a+b)^*) \text{Ш} (a^*b^*)$

The following values are determined using algorithm 2:

$$followpos(1) = \{2,3,4\}$$
$$followpos(2) = \{2,3,4\}$$
$$followpos(3) = \{2,3,4\}$$
$$followpos(5) = \{5,6,7\}$$
$$followpos(6) = \{6,7\}$$
$$followpos(4) = followpos(7) = \phi$$

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*.

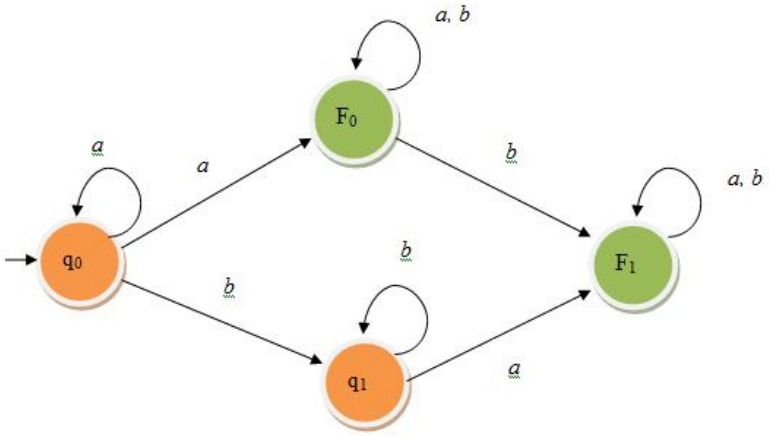

Figure 5. NFA for $r = (a(a+b)^*) \text{Ш} (a^*b^*)$

Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

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. □

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 ε-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}. □

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. □

The worst-case state complexity of the ε-NFA is $2^{2|r|-3C}$ using Estrade *et al.*'s methodology [2]. The pattern matching will decelerate if the NFA consists of ε-transitions. Using the proposed algorithm, a ε-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].

Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

Table 3. Comparison between the proposed algorithm and Estrade *et al.*'s methodology

Criterion	Proposed algorithm	Estrade *et al.*'s methodology
Worst-case state complexity	2^{m+s+1}	$2^{2\|r\|-3C}$
Output	ε-free NFA	ε-NFA
Conversion chain	SRE $\rightarrow$ NFA $\rightarrow$ DFA	SRE $\rightarrow$PFA$\rightarrow$ NFA $\rightarrow$ DFA

CONCLUSIONS AND FUTURE SCOPE

An algorithm has been designed for the conversion of SREs into ε-free NFAs without using any intermediate steps. Using the proposed approach, a svelte 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 ε-free NFAs is produced. Pattern matching is delayed if the NFA consists of ε-transitions. Further, if $\forall(r_i \text{ Ш } r_j) \in r \wedge ((symbols(r_i) \bigcap symbols(r_j)) \neq \phi)$, 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

1. A. V. Aho, R. Sethi and J. D. Ullman, "Compilers: Principles, Techniques and Tools", 19th Edn., Pearson Education, Singapore, **2005.**
2. B. D. Estrade, A. L. Perkins and J. M. Harris, "Explicitly parallel regular expressions", Proceedings of 1st International Multi-symposiums on Computer and Computational Sciences, **2006,** Hangzhou, China, pp.402-409.
3. J. E. Hopcroft, R. Motwani and J. D. Ullman, "Introduction to Automata Theory, Languages, and Computation", 12th Edn., Pearson Education, Singapore, **2005.**
4. P. D. Stotts and W. Pugh, "Parallel finite automata for modeling concurrent software systems", *J. Syst. Softw.*, **1994,** *27*, 27-43.
5. W. Gelade, "Succinctness of regular expressions with interleaving, intersection and counting", *Theor. Comput. Sci.*, **2010,** *411*, 2987-2998.
6. J. Clark and M. Murata, "RELAX NG Specifications", **2001**, http://www.oasis-open.org/committes/relax-ng/spec-20011203.html (Date of access: Dec. 2001).
7. L. Wall, T. Christiansen and J. Orwant , "Programming Perl",3rd Edn., O'Reilly Media, Sebastopol (CA), **2000**.
8. B. D. Estrade, "An investigation of equivalent serialized forms of parallel finite automata", *MS Thesis,* **2005**, University of Southern Mississippi, USA.
9. M. Nivat, "Behaviours of synchronized systems of processes", L.I.T.P. Report No. 81-64, University of Paris 7, France, **1981**.

Maejo Int. J. Sci. Technol. **2013**, *7*(03), 396-407

10. K. Iwama, "Unique decomposability of shuffled strings: A formal treatment of asynchronous time-multiplexed communication", Proceedings of 15th Annual ACM Symposium on Theory of Computing, **1983,** Boston, USA, pp.374-381.
11. J. C. M. Baeten and W. P. Weijland, "Process Algebra (Cambridge Tracts in Theoretical Computer Science)", Cambridge University Press, Cambridge, **1990**.
12. V. K. Garg, "Modeling of distributed systems by concurrent regular expressions", Proceedings of 2nd International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, **1989,** Vancouver, Canada, pp.313-327.
13. F. Biegler, M. Daley, M. Holzer and I. McQuillan, "On the uniqueness of shuffle on words and finite languages", *Theor. Comput. Sci.*, **2009,** *410*, 3711-3724.
14. D. Hovland, "The membership problem for regular expressions with unordered concatenation and numerical constraints", Proceedings of 6th International Conference on Language and Automata Theory and Applications, **2012**, Coruna, Spain, pp.313-324.
15. W. Gelade, W. Martens and F. Neven, "Optimizing schema languages for XML: Numerical constraints and interleaving", *SIAM J. Comput.*, **2009**, *38*, 2021-2043.
16. J. Berstel, L. Boasson, O. Carton, J.-E. Pin and A. Restivo, "The expressive power of the shuffle product", *Inform. Comput.*, **2010**, *208*, 1258-1272.
17. K. Thompson, "Regular expression search algorithm", *Commun. ACM,* **1968,** *11*, 419-422.
18. V. Antimirov, "Partial derivatives of regular expressions and finite automaton constructions", *Theor. Comput. Sci.*, **1996,** *155*, 291-319.
19. L. Ilie and S. Yu, "Follow automata", *Inform. Comput.*, **2003,** *186*, 140-162.
20. A. Brueggemann-Klein, "Regular expressions into finite automata", *Theor. Comput. Sci.*, **1993,** *120*, 197-213.