CONVERSION OF DETERMINISTIC FINITE AUTOMATA TO REGULAR EXPRESSION USING BRIDGE STATE

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Submitted By
Kulwinder Singh
(800932012)

Under the supervision of:
Mr. Ajay Kumar Loura
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2011

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Conversion of Deterministic Finite Automata to Regular Expression using Bridge State"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Mr. Ajay Kumar Loura** and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature: Kulwinder Singh 15/6/11

(Kulwinder Singh)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

A Kumar 15/6/11

(Mr. Ajay Kumar Loura)
Assistant Professor
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

(Dr. Maninder Singh)
Head 15/6/11
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

No volume of words is enough to express my gratitude towards my guide, **Mr. Ajay Kumar Loura** Assistant. Professor, Computer Science and Engineering Department, Thapar University, who has been very concerned and has aided for all the material essential for the preparation of this thesis report. He has helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research-oriented venture.

I am also thankful to **Dr. Maninder Singh**, Head of Department, CSED and **Mr. Karun Verma**, P.G. Coordinator for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there in the need of the hour and provided with all the help and facilities, which I required, for the completion of my thesis.

Most importantly, I would like to thank my parents and the Almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

Kulwinder Singh
(800932012)

Abstract

Regular expressions are widely used in the field of compiler design, text editor, search for an email- address, grep filter of unix, train track switches, pattern matching ,context switching and in many areas of computer science. The demand of smaller regular expression motivates research into the conversion of deterministic finite automta to regular expression for obtaining smaller regular expression.

For conversion of deterministic finite automata to regular expression, several techniques like Transitive closure method, Brzozowski Algebraic method and state elimination method have been proposed. None of the above specified technique is able to find smallest regular expression. Our purpose is to find the smallest regular expression equivalent to given deterministic finite automata. State elimination approach is the most widely used and efficient approach for converting deterministic finite automata to regular expression. In order to choose optimal removing sequence of states in state elimination method for obtaining smaller regular expression, some heuristics like Delgado and Morais's state weight heuristic and Yo-Sub Han and Derick Wood's concept of bridge state, vertical chopping and horizontal chopping have been proposed.

The presented thesis investigates and compares different techniques used for converting deterministic finite automata to regular expression. Brief comparisons amongst different techniques are presented and several heuristics are explored for obtaining smaller regular expression using state elimination approach. In order to obtain smaller regular expression using state elimination method optimal removal sequence of states should be chosen. For choosing optimal removal sequence of states new heuristics have been proposed in this thesis work. The software has been designed for conversion of deterministic finite automata to regular expression using newly developed heuristics.

Table of Contents

List of Keywords

FA	Finite Automata
DFA	Deterministic Finite Automata
NFA	Non-deterministic Finite Automata
RE	Regular Expression
GUI	Graphical User Interface

List of Figures and Tables

Table No.	**Table Title**	**Page No.**

Chapter 1
Introduction

This chapter gives an introduction to automata and regular expression. Some basic definitions, notations and organization of thesis along with brief idea about the contents of each of the following chapters have also given in this chapter.

1.1 What is Automata?

The theory of computation [31] is the branch of computer science and mathematics that deals with whether and how efficiently problems can be solved on a model of computation using an algorithm. In theoretical computer science, automata theory is the study of abstract machines and the computational problems that can be solved using these abstract machines. These abstract machines are called automata.

Automata theory [16, 31] is closely related to formal language theory as the automata are often classified by the class of formal languages they are able to recognize. Study of automata is an important part of core of Computer Science. Automata is used in designing and checking the behaviour of digital circuits, in lexical analysis phase of compiler construction, in software for scanning large bodies of text and in software for verifying systems of all types that have finite number of distinct states, such as communications protocols [31]. Finite automata have finite number of states. For example, following figure 1.1 shows finite automata model for switch.

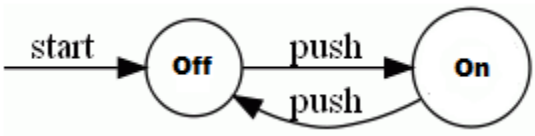


Figure 1.1: A finite automata model for on/off switch [31]

A finite automata shown in figure 1.1 has two states, ‘off’ and ‘on’, represented by a circle. Arcs between the states are labelled by an input symbol. Here, both the arcs have input symbol push, which represent a user pushing the button. When push input is given to state ‘off’, control goes to state ‘on’. An arrow leading to state ‘off’ indicates that it is start state of finite automata.

It is necessary to indicate final or accepting state of finite automata, so that it can be checked which particular sequence of input symbols is accepted by finite automta. For example, following automata accept the string "start".

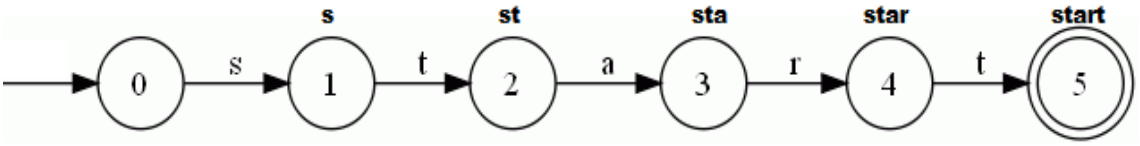


Figure 1.2: A finite automata for recognition of string "start"[31]

As shown in figure 1.2, a finite automata has six states and there is a transition from one state to other with particular input symbol. This finite automata recognizes the string "start" and each state of finite automata represents the different position of string that has been reached so far. Finite automata can be deterministic and non-deterministic. Every regular language that is described by non-deterministic finite automata can also be described by deterministic finite automata.

Regular expressions [31] also denote regular languages, which consists of strings of particular type. The patterns of strings described by regular expression are exactly same as what can be described by finite automata. It means every formal language defined by any finite automata is also defined by a regular expression.

1.2 Definitions and Notations

1.2.1 Alphabet and Language: Alphabet is defined as finite non-empty set of symbols on which the language is defined. Alphabets are denoted by $\sum$. Language [16, 23] is defined as a subset of $\sum^*$. Empty string and null language are denoted by ϵ and ϕ respectively. Various kinds of formal languages can be classified as regular, context free, context sensitive and recursive language. Regular language can be described by regular expression, finite automata (Deterministic or Non-deterministic). A language over 0 and 1 that will include all strings having length less than 2 is $L=\{ \epsilon,0,1,00,01,10,11\}$.

Operations on languages

Following are the operations that can be performed on languages.[31]

1. **Union**: Union of two languages L_1 and L_2 is set of all the strings that are in either L_1 or L_2, or both. For example, if L_1={001,10,111} and L_2={ϵ,01}, then L_1UL_2={ϵ,01,001,10,111}.
2. **Concatenation**: Concatenation of two languages L_1 and L_2 is set of all the strings that can be formed by taking any string in L_1 and concatenating it with any string in L_2. Regular Language can be expressed by regular expression or DFA. i.e. if L_1={001,10,111} and L_2={ϵ,01}, then L_1L_2={001,10,111,00101,1001,11101}.
3. **Kleene Closure**: Kleene closure of a language L represent the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (same string can be selected more than once) and concatenating all of them. Kleene closure of a language L is denoted by L^*. For example if L={0,1}, then kleene closure of L is all strings of 0's and 1's i.e. L^* ={ϵ,0,1,01,001,0110,111010.....}.

1.2.2 Deterministic Finite Automata: Deterministic finite automaton (DFA) [16, 31] is a finite state machine accepting finite strings of symbols. For each state, there is a transition arrow leading out to a next state for each symbol.

Deterministic finite automata (DFA) can be defined by 5-tuples (Q, Σ, δ, q_0, F), where

Q is a finite set of states

Σ is a finite set of symbols

δ is the transition function, that is, δ: Q × Σ → Q.

q_0 is the start state

F is a set of states of Q (i.e. F⊆Q) called accept states.

Transition functions can also be represented by transition table as shown in table 1.1.

Example 1.1**:** A finite automata is represented by ({0, 1, 2}, {a}, δ, {0}, {2}) where, δ is shown in the following table.

Table 1.1: Transition Table representing transition function of DFA

State (Q)	Next State δ(q,a)
0	1
1	2
2	2

Transition function can also be represented by transition diagram as shown below in figure 1.3.

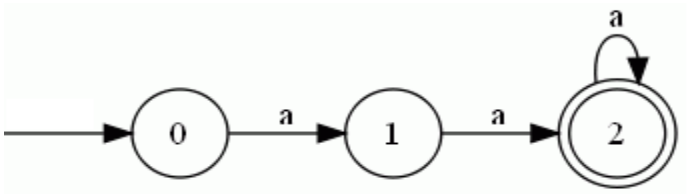


Figure 1.3: Deterministic finite automata corresponding to table 1.1.

1.2.3 Non-deterministic Finite Automata: A Non-deterministic finite automata (NFA) is same as DFA except the transition function. Transition function in NFA is defined as: $Q \times \Sigma \rightarrow 2^Q$. A Non-deterministic finite automaton (NFA) [16, 31] is a finite state machine where for each pair of state and input symbol there may be more than one next states. Following figure 1.4 shows non-deterministic finite automata accepting all the strings terminating with 01 and in which state A has two transitions for same input symbol 0.

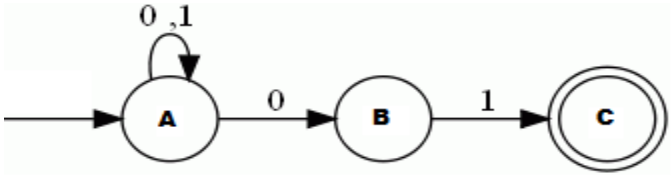


Figure 1.4: An example of non-deterministic finite automata.

1.2.4 Regular Expression: A regular expression (RE) [31] is a pattern that describes some set of strings. Regular expression over a language can be defined as:

1) Regular expression for each alphabet will be represented by itself. The empty string (ϵ) and null language (ϕ) are regular expression denoting the language $\{\epsilon\}$ and $\{\phi\}$ respectively.
2) If E and F are regular expressions denoting the languages L(E) and L(F) respectively, then following rules can be applied recursively
    a) Union of E and F will be denoted by regular expression E+F and representing language L(E) U L(F).
    b) Concatenation of E and F denoted by EF and representing language L(E*F) = L(E) * L(F).
    c) Kleene closure will be denoted by E^* and represent language $(L(E))^*$.

3) Any regular expression can be formed using 1-2 rules only.

For example, the set containing the three strings "Handel", "Händel", and "Haendel" can be described by the pattern H(ä|ae?)ndel. Alternatively, pattern matches each of the three strings.

1.3 Thesis Outline

Thesis is organized into 5 chapters. Chapter 1 describes background information relating to deterministic finite automata, regular expression. Chapter 2 describes different approaches used for converting deterministic finite automata to regular expression. Chapter 3 describes motivation behind the thesis, discusses the problem statement and compares different approaches used for conversion of deterministic finite automata to regular expression. Chapter 3 also describes newly proposed heuristics in this thesis work for obtaining smaller regular expression. Chapter 4 includes the implementation detail of proposed heuristics; Chapter 5 summarizes the conclusions drawn in the thesis along with the future research directions.

Chapter 2
Conversion of DFA to RE

This chapter describes different techniques used for converting deterministic finite automata to regular expression. Heuristics like bridge state concept, horizontal chopping, vertical chopping and Delgado and Morais's state weight approach for choosing optimal removal sequence of states in order to obtain smaller regular expression using state elimination method has also been described in this chapter.

2.1 Conversion of DFA to RE

Kleene proves that every RE has equivalent DFA and vice versa. On the basis of this theoretical result, it is clear that DFA can be converted into RE and vice versa using some algorithms or techniques. For converting RE to DFA, first we convert RE to NFA(Thomson Construction) and then NFA is converted into DFA(Subset construction).For conversion of DFA to regular expression, following methods have been introduced.[2, 22, 10]

- Transitive closure method
- Brzozowski Algebraic method
- State elimination method

2.2 Transitive Closure Method

Kleene's transitive closure method [2, 22] defines regular expressions and proves that there is equivalent RE corresponding to a DFA. Transitive closure is the first mathematical technique, for converting DFAs to regular expressions. It is based on the dynamic programming technique. In this method we use R_{ij}^{k} which denotes set of all the strings in $\sum^{*}$ that take the DFA from the state q_i to q_j without entering or leaving any state higher than q_k. There are finite sets of R_{ij}^{k} so that each of them is generated by a simple regular expression that lists out all the strings.

Let regular expression R_{ij} represents the set of all strings that take the DFA from state q_i to q_j. R_{ij} can be constructed by successively constructing $R_{ij}^1, R_{ij}^2, R_{ij}^3, \ldots\ldots\ldots R_{ij}^m$.

R_{ij}^k is recursively defined as: $R_{ij}^k = R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1} + R_{ij}^{k-1}$ ……(1)

Assuming we have initialized R_{ij}^0 to be:

$$R_{ij}^0 = \begin{cases} r & \text{if } i \neq j \text{ and r transitions from } q_i \text{ to } q_j \\ r + \epsilon & \text{if } i = j \text{ and r transitions from } q_i \text{ to } q_j \\ \phi & \text{otherwise} \end{cases}$$

Using (1) $R_{ij}^k = R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1} + R_{ij}^{k-1}$, we obtained regular expression R_{ij}.

This successive construction builds up regular expressions until we have R_{ij}. Then construct a regular expression representing DFA as the union of all $R_{sf,}$ where q_s is the starting state and q_f is the final state of DFA. The chief problem of the transitive closure approach is that it creates very large regular expressions.

Following Kleene's algorithm constructs a regular expression R from a deterministic finite automata 'A' such that L(R) = L(A). Suppose the states of 'A' are numbered 1,2.....,n and Kleene's algorithm constructs a set of regular expressions R[i,j,k] representing all paths from state i to state j with no intermediate node in the path numbered higher than k.

Kleene's Algorithm [1]

```
for i=1 to n
    for j=1 to n
        if (i != j) then
            if (there are transitions from state i to state j labeled a1, a2, ..., ak) then
                R[i,j,0] = a1 + a2 + ... + ak;
            else
                R[i,j,0] = ϕ;
            end if
        else if (i = = j) then
            if (there are transitions from state i to state i labeled a1, a2, ..., ak)
```

```
                R[i,i,0] = ϵ + a1 + a2 + ... + ak;
              else
                 R[i,i,0] = ϕ;
            end if
         end if
      end if
   end for
end for
 for k=1 to n
          for i=1 to n
              for j=1 to n
                   R[i, j, k] = R[i, j, k-1] + R[I, k, k-1] (R[k, k, k-1] )* R[k, j, k-1];
               end for
          end for
     end for
```

If start state of DFA is 1, then regular expression for the DFA is sum (union) of all expressions R[1,j,n] where j is a final state.

Consider the DFA given in fig. 2 and applying transitive closure method on it.

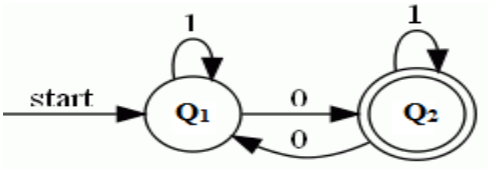


Figure 2.1: DFA for the language having odd number of 0's

$r_{11}^{0} = 1 + \epsilon \qquad r_{22}^{0} = 1 + \epsilon \qquad r_{12}^{0} = 0 \qquad r_{21}^{0} = 0$

$r_{12}^{1} = r_{12}^{0} + r_{11}^{0} (r_{11}^{0})^{*} r_{12}^{0} = 0 + (1+\epsilon)^{+} 0$

$r_{22}^{1} = r_{22}^{0} + r_{21}^{0} (r_{11}^{0})^{*} r_{12}^{0} = (1+\epsilon) + 0(1+\epsilon)^{*} 0$

$r_{12}^{2} = r_{12}^{1} + r_{12}^{1} (r_{22}^{1})^{*} r_{22}^{1} = (1+\epsilon)^{*} 0(1+\epsilon+01^{*}0)$

$r_{12}^{2} = (1)^{*} 0(1+ 01^{*}0)^{*}$

$L(M) = L(r_{12}^{2})$

2.3 Brzozwski Algebraic Method

Brzozowski method [10, 22] is a unique approach for converting deterministic finite automata to regular expressions. In this approach first characteristic equations for each state are created which represent regular expression for that state. Regular expression equivalent to deterministic finite automata is obtained after solving the equation of R_s (regular expression associated with starting state q_s).

If R_i is regular expression for state q_i and there is a transition on reading input symbol a from state q_i to q_j , then term aR_j is added in the equation for R_i . If q_i is final node then term ϵ (null index) is added in the equation. This leads to a system of equations in the form:

$R_1 = a_1R_1 + a_2R_2 + \ldots\ldots\ldots$

$R_2 = a_1R_1 + a_2R_2 + a_3R_3 + \ldots\ldots\ldots$

$R_m = a_1R_1 + a_2R_2 + \ldots + \ldots \epsilon$ ϵ is added if R_m is final node

Where $a_x = \emptyset$ if there is no transition from Ri to Rj.

Using Arden's Theorem [31] we solve the obtained equations. Arden's theorem states that if an equation is of the form $X = AX + B$, its solution is $X = A^* B$

Consider the DFA in the following figure 2.2:

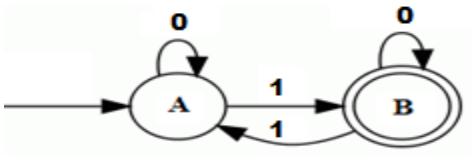


Figure 2.2: DFA for strings with an odd no of 1's.

Characteristics equations are as follow:

$A = 0A + 1B\ldots\ldots\ldots\ldots\ldots\ldots$ (2)

$B = 1A + 0B + \epsilon\ldots\ldots\ldots\ldots\ldots$ (3)

Solving these equations by Arden's theorem

$B = 1A + 0B + \epsilon = 0B + (1A + \epsilon) = 0^*(1A + \epsilon)$

$B = 0^*(1A) + 0^*(\epsilon) = 01^*A + 0^*\ldots\ldots\ldots\ldots\ldots\ldots$ (4)

Using (4) we obtain

$A = 0A + 1B = 0A + 1(0^*1A + 0^*) = 0A + 10^*1A + 10^*$

$A = (0 + 10^*1)^*(10^*)$ (Using Arden's rule)

2.4 State Elimination Method

The state removal approach [22, 31] is widely used approach for converting DFA to regular expression. In this approach, states of DFA are removed one by one until we left with only starting and final state, for each removed state regular expression is generated. This newly generated regular expression act as input for a state which is next to removed state. The advantage of this technique over the transitive closure method is that it is easier to visualize. This technique is described by Du and Ko [2], but a much simpler approach is given by Linz [23]. First, if there are multiple edges from one node to other node, then these are unified into a single edge that contains the union of inputs. Suppose from q_1 to q_2 there is an edge weighted 'a' and an edge weighted 'b', those would be unified into one edge from q_1 to q_2 that has the weight (a + b). If there are n accepting states, take union of n different regular expressions.

For example, DFA shown in figure 2.3 is converted to regular expression using state elimination method

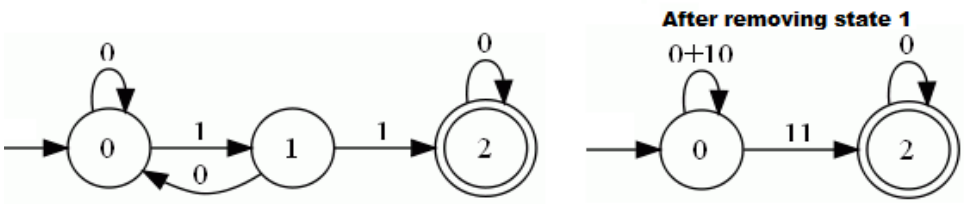


Figure 2.3: An example of state elimination method.

Equivalent regular expression corresponding to given DFA is (0+10)*11(0)*. But in state elimination method using different removal sequences of states, we obtain different regular expressions for the same language. Therefore depending on which removal sequence we choose, we may have a smaller regular expression for the same deterministic finite automata. We require a removal sequence which gives smaller regular expression.

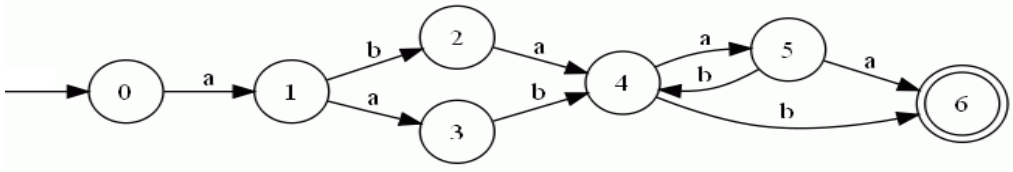


Figure 2.4: An example of different removal sequence of states gives different RE

Figure (2.5-2.9) show the DFA obtained after removing states of finite automata shown in figure 2.4 using removal sequence 2-3-5-1-4.

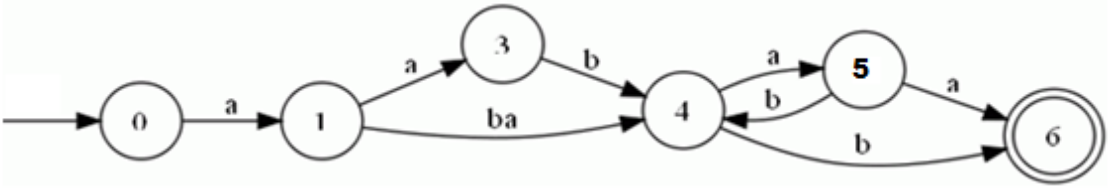


Figure 2.5: DFA after removing state 2

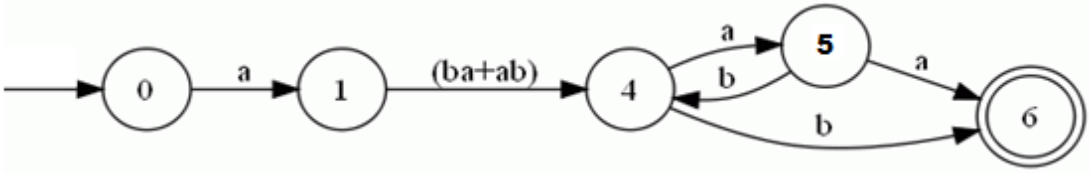


Figure 2.6: DFA after removing state 3

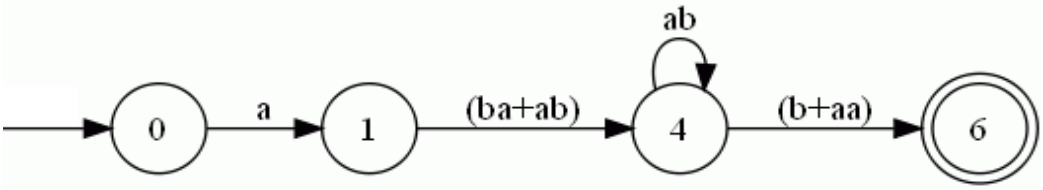


Figure 2.7: DFA after removing state 5

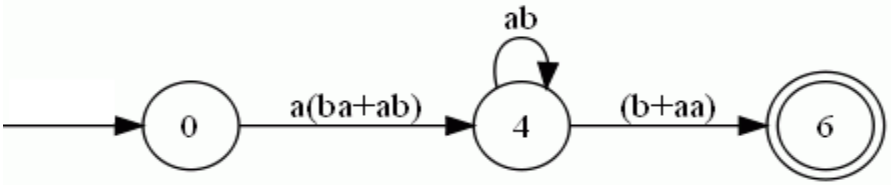


Figure 2.8: DFA after removing state 1

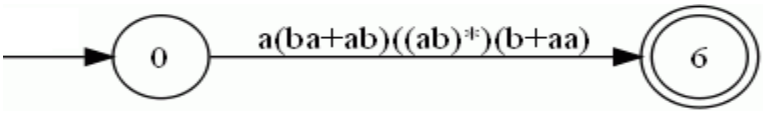


Figure 2.9: DFA after removing state 4

Regular expression R_1= a(ba+ab)(ab)*(b+aa) is obtained using state elimination sequence 2-3-5-1-4 .Similarly regular expression R_2=(ab(ab+aa(ba)*a) + aa(bb+ba(ba)*a)) and R_3=[(aba+aab)b + (aba+aab)a(ba)*a] are obtained using state elimination sequence 1-4-5-2-3 and 2-1-3-4-5 respectively.

For n-state deterministic finite automata excluding starting and final states, n! removal sequences are there. It is very difficult to try all the possible removal sequence for smaller regular expression. Instead, researcher introduced new heuristics using structural

properties of deterministic finite automata for state elimination method that can give smaller regular expression equivalent to deterministic finite automata.

Gruber and Holzer [5] proposed graph separator techniques for obtaining shorter regular expression. Delgado and Morais [21] proposed a strategy relied on weight of the state. Recently, Moreira and Reis presented an algorithm that obtains O(n) size regular expression from an n-state acyclic finite automata having complexity $O(n^2 \log n)$. Some heuristics for state elimination method run in exponential time. Since we intend to find the smaller regular expression from deterministic finite automata quickly, only polynomial running time heuristic can be considered for implementation. Concept of bridge state, vertical chopping, horizontal chopping given by Han and Wood [33] and state elimination using state weight given by Delgado and Morais run in polynomial time.

2.4.1 Bridge State

Han and Wood [33] proposed the concept of bridge state in the state elimination method. A state q_b of automata A is said to be bridge state, if it satisfies the following conditions:

1) State q_b is neither a start nor a final state.
2) For each string w ϵ L(A), its path in A must pass through q_b at least once.
3) Once string w's path passes through q_b for the first time, the path can never pass through any states that have been visited before apart from state q_b.

Consider the following deterministic finite automata shown in figure 2.10.

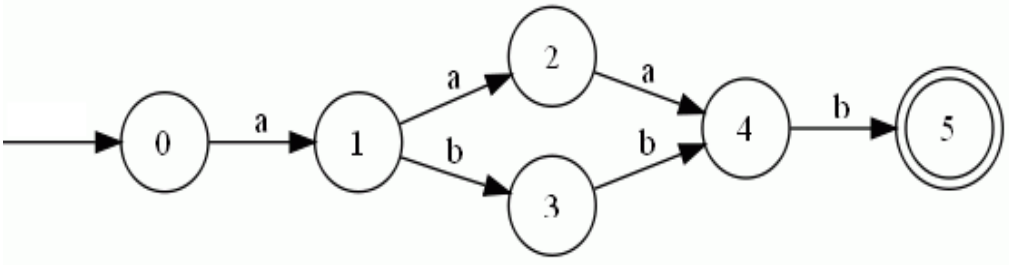


Figure 2.10: An example deterministic finite automata having bridge state

State 1 and state 4 satisfy all the conditions of bridge state introduced by Han and Wood. In figure 2.10, removal sequence 2-3-1-4 gives regular expression R_1= a(aa+bb)b as shown in figure 2.11 whereas removal sequence 1-4-2-3 and 1-2-4-3 gives regular expression R_2=(aaab+baaa). Note that $|R_1| < |R_2|$.

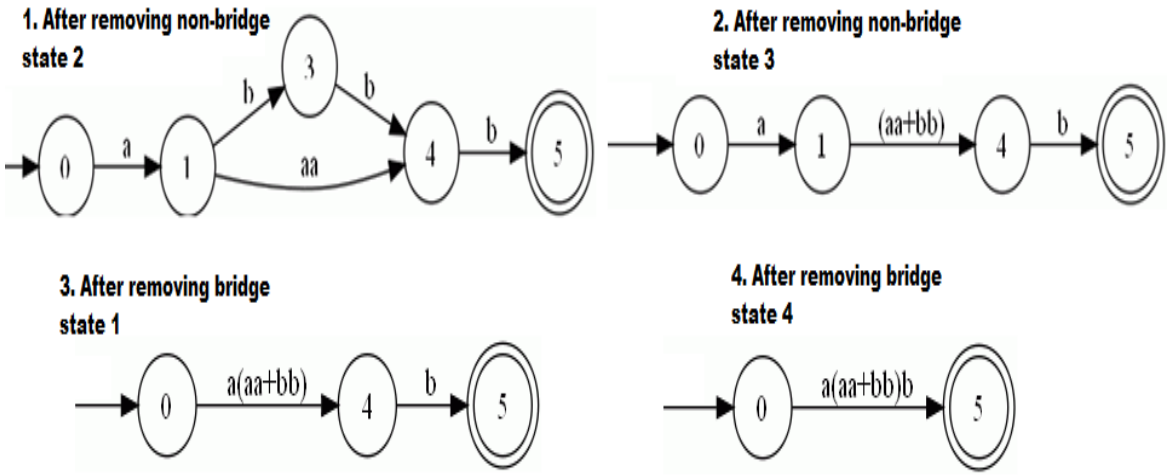


Figure 2.11: An example of elimination of non-bridge states before bridge states.

Also in figure 2.4, state 1 and state 4 satisfy all the conditions of bridge state and if we remove bridge states after removing all non-bridge states, regular expression obtained is R_1=a(ba+ab)(ab)*(aa+b). Regular expression R_2=(ab(ab+aa(ba)*a) + aa(bb+ba(ba)*a)) is obtained if bridge states are removed before removing non-bridge states of deterministic finite automata and | R_1|<| R_2|. In order to obtain smaller regular expression from deterministic finite automata bridge state should be removed after removing all non-bridge state [33].

2.4.2 Bridge State Determination

Han and Wood [33] presented an algorithm that finds bridge states in linear time of a given deterministic finite automata based on the depth first search algorithm. If q is a bridge state and P is a simple path from initial state to final state, then q must belongs to P (q ϵ P) .

We apply the algorithm given by Han and Wood for finding bridge state of the deterministic finite automata shown in figure 2.12. First a simple path P is found from initial to final state. Let S_B = {S, b_1, b_2, b_3, b_4, f} be the set of states present in the simple path P. In this approach, all the states that are presented in the simple path or set S_B, are considered as candidates of bridge state and if any state, present in simple path or set S_B, violates any requirement of bridge state that is removed from S_B. Finally, we are left with only those states that satisfy the requirement of bridge state in set S_B.

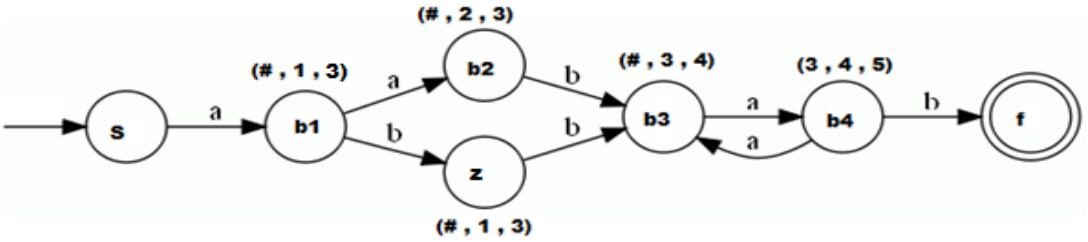


Figure 2.12: Bridge state determination using depth first search

Depth first search is applied to traverse deterministic finite automata as shown in figure 2.12.Three values are maintained for each state (except for start and final state) of the deterministic finite automata known as (min, anc, max).

anc: The index i of a state $b_i \in S_B$ such that there is a path from b_i to q and there is no path from $b_j \in S_B$ to q for $j > i$. The anc of b_i is i.

Basically anc value is index value of state present in simple path from start to final state. For example, states $\{b_1, b_2, b_3, b_4\}$, of DFA shown in figure 2.12, are present in the simple path and their anc values are {1, 2, 3, 4} respectively. For any state which is not present in simple path or S_B anc value will be same as anc value of its parent state. For example, state z shown in figure 2.12 is not present in simple path from start to final state and state b_1 is parent state of state z. So anc value of child state z will be same as anc value of its parent state b_1. i.e. 1.

min: The index i of a state $b_i \in S_B$ such that there is a path from q to b_i and there is no path from q to b_j for $j < i$.

min value of state q means there is a path from state q to another state (b_{min}) which has been traversed before apart from state q. For example, as shown in figure 2.12 state 3 is traversed before state 4 and there is a path from state 4 to state 3, so min value of state 4 is 3. If any state q has min value, it means there is a circle present in the automata and state q is also present in that circle. If there is no path from state q to state, that has been traversed before apart from state q, then symbol # is used for min value of state q.

For example in figure 2.12 states $\{b_1, b_2, b_3, z\}$ have min value # (null index) because no path is there from any state out of $\{b_1, b_2, b_3, z\}$ to any other state, that has already been traversed before that state. It means these states are not present in any circle. But state b_4

has min value 3 and it is present in circle b_3-b_4-b_3. Now any state q of DFA has two types of child states: one that are present in simple path or S_B and other that are not present in simple path or S_B. min value of state q of deterministic finite automata, after exploring all its child states, can be updated as follows.

$$\text{min} = \min_{q \epsilon T_1}(\text{min}(q.\text{anc}), \min_{q \in T_2}(q.\text{min}))$$

Where, T_1 is set of states present in simple path and T_2 is set of states which are not present in simple path.

max: The index i of a state $b_i \in S_B$ such that there is a path from q to b_i and there is no path from q to b_j for $i < j$.

The max value of state q means there is path from state q to another state, whose index value is same as max value of state q. For example, as shown in figure 2.12 there is path from state b_2 to state b_3 and max value of b_2 is 3. If state b_i has max value, where max ≠ i+1. Then it means there exists another simple path from state b_i to b_{max} without passing through state b_{i+1}. As shown in figure 2.12 state b_1 has max value 3 because another path is there from state b_1 to b_3 without passing through state b_2.

max value of any state of deterministic finite automata, after exploring all its child states, can be updated as follows.

$$\text{max} = \max_{q \epsilon T_1}(\text{max}(q.\text{anc}), \max_{q \in T_2}(q.\text{max}))$$

If state does not have any out transition except for a transition to final state ($f \in S_B$), it means depth-first-search is complete. After applying depth-first-search and calculating all three values (min, anc, max) for each state of the finite automata, states which do not satisfy any condition of bridge state are removed from set S_B. Suppose three values (min, anc, max) of state $b_y \in S_B$ are (x,y,z), where $x<y<z$ and $x \neq \#$. If $x = \#$ then there is no need to remove any state from set S_B. Now $x<y$ and $x \neq \#$, it means there is a circle present in the finite automata and there exist a path from state b_y to state b_x. Now state b_x is already traversed before state b_y and there is path present from state b_y to b_x. It violates the third condition of bridge state. So states $b_{x+1}, b_{x+2} \ldots\ldots\ldots\ldots, b_y$ are removed from set S_B.

Now y<z , if z ≠ y+1 then it means there exist a path from b_y to b_z without passing through states b_{y+1}, b_{y+2}……., $b_{z-1.}$ Hence there is another simple path from b_y to final state (f). It violates the second condition of bridge state. So we remove states b_{y+1}, b_{y+2}……., $b_{z-1.}$from set S_B.

For example three values of state b_4 of Deterministic finite automata shown in figure 2.12 are (3,4,5). There is path from state b_4 to b_3 and state b_3 is already traversed before b_4. So, state b_4 is removed from set S_B because it violates third conditions of bridge state.

State b_1 has (#, 1, 3) values after depth-first-search is complete. It means there is direct path from state b_1 to b_3 and there exist another simple path from sate b_1 to final state (f) without visiting through state b_2. State b_2 is removed from set S_B because it violates second condition of bridge state. Finally, we are left with only two states $\{b_{1,} b_3\}$ in set S_B which satisfy all the conditions of bridge state and these states have values (#, 1, 3) and (#, 3, 4) respectively.

In this way, using depth-first-search bridge states of deterministic finite automata can be found. It takes constant time to compute three values for each state and depth-first-search runs in linear time in size of deterministic finite automata. So bridge state of finite automata can be found in O(|Q| + |δ|) time using depth-first-search.

Bridge states should be removed after removing all non-bridge states of deterministic finite automata to obtain smaller regular expression. If all the states(except start and final states) of deterministic finite automata satisfy all the conditions of bridge state, then removal of these states in any sequence gives same regular expression [33]. For example, in the following figure 2.13: state 1, state 2 and state 3 are bridge states.

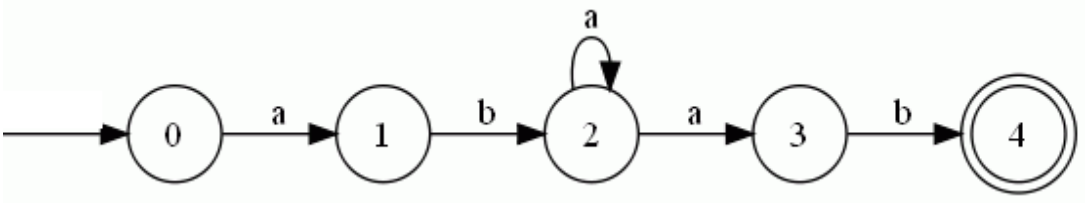


Figure 2.13: DFA having all states(except for starting and final state) as bridge states

If states of finite automata are removed in order of 2-1-3, 1-2-3, 3-2-1 and 3-1-2 respectively, every time same regular expression $ab(a)^*ab$ is obtained.

Han and Wood [21,33] also suggested two chopping techniques, based on the structural properties of a given deterministic finite automata, known as vertical chopping and horizontal chopping.

2.4.3 Vertical chopping

Vertical chopping always gives smaller regular expression using state elimination method. In vertical chopping, deterministic finite automata 'A' is divided into sub-automata A_1 and sub-automata A_2 vertically, such that $L(A) = L(A_1).L(A_2)$. Vertical chopping is done at the bridge state. All the in-transitions of bridge state appear in sub-automata A_1 and all the out transitions of bridge state appear only in A_2. After chopping regular expressions are created for sub-automata A_1 and sub-automata A_2 and that bridge state becomes final state of sub-automata A_1 and start state of sub-automata A_2. Final regular expression is obtained by concatenating these two regular expressions obtained from sub-automata A_1 and sub-automata A_2. There is possibility of duplicate states and transitions in both sub-automata A_1 and A_2 and it may not give smaller sub-automata. For example in following figure 2.14, state 1 and state 3 are bridge states.

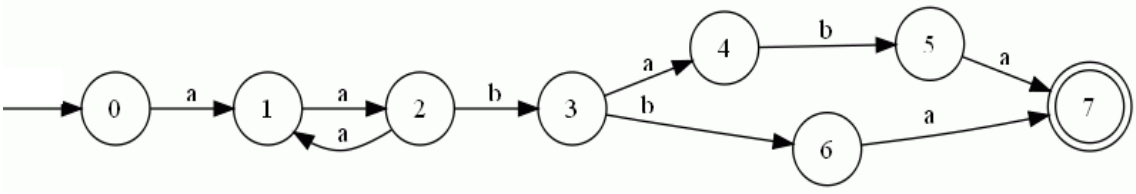
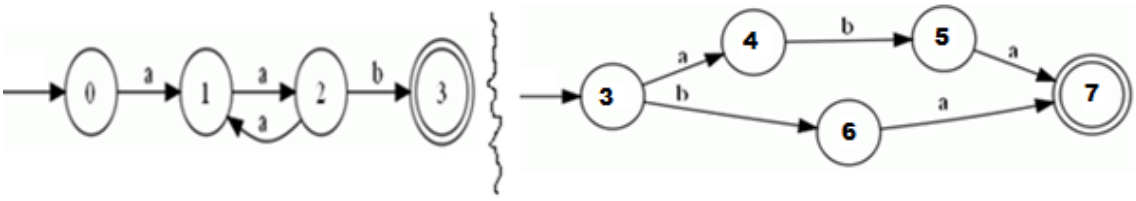


Figure 2.14: Deterministic finite automata having bridge states.

Now, vertical chopping can be done on state 1 and state 3. Vertical chopping at bridge state 3 results in the following two sub-automata.



Sub-automata A_1 Sub-automata A_2

Figure 2.15: Vertical chopping at bridge state 3.

Now in sub-automata A_1 state 1 is bridge state, so states of sub-automata are removed in sequence of 2-1.

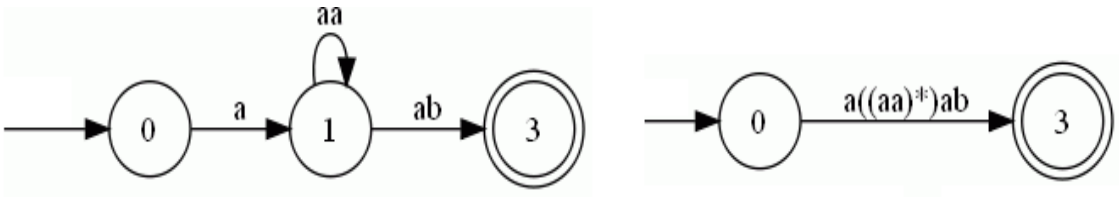


Figure 2.16: Sub-automata A_1 after removing state 2 and state 1 respectively.

Regular expression corresponding to sub-automata A_1 is a(aa)*ab.

In sub-automata A_2, no state satisfies the conditions of bridge state. States of sub-automata are removed in sequence of 5-4.

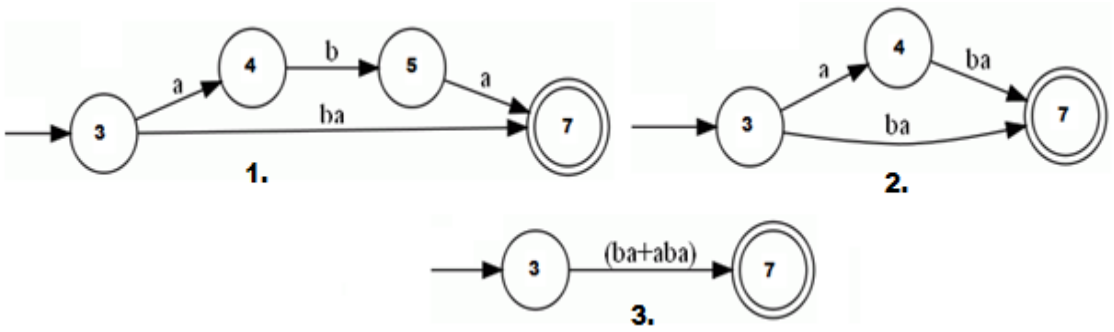


Figure 2.17: Sub-automata A_2 after removing state 6, state 5 and state 4 respectively.

Regular expression corresponding to sub-automata A_2 is (ba+aba).

Final regular expression equivalent to deterministic finite automata shown in figure 2.14 can be obtained by concatenating the regular expression of sub-automata A_1 and regular expression of sub-automata A_2. Regular expression equivalent to deterministic finite automata shown in figure 2.14 is $a(aa)^*ab(ba+aba)$.

2.4.4 Horizontal chopping

In horizontal chopping, automata is decomposed into two sub-automata A_U and A_L horizontally. Horizontal chopping is also used in state elimination method for obtaining smaller regular expression and removal sequence of each sub-automata A_u does not influence any other removal sequence of A_L. Final regular expression is obtained by taking union of regular expressions obtained from sub-automata A_U and A_L. Also, deterministic finite automata can be decomposed into several horizontally disjoint sub-automata and finally, regular expression can be obtained by taking union of corresponding regular expressions obtained from all the sub-automata.

Vertical chopping on deterministic finite automata shown in figure 2.14 resulted in two sub-automata A_1 and A_2 shown in figure 2.15. Sub-automata A_2 has one start state and one final state and no state satisfies the conditions of bridge state. Sub-automata A_2 can be chopped horizontally into sub-automata A_U and A_L as shown in figure 2.18.

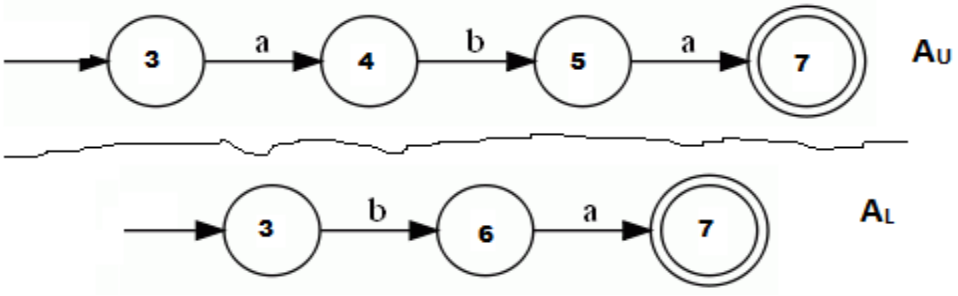


Figure 2.18: Horizontal chopping on sub-automata A_2

Regular expression corresponding to sub-automata A_2 shown in figure 2.15 can be obtained by taking union of regular exprressions of sub-automata A_U and A_L. Removal sequences 4-5-6 and 4-6-5 give same regular expression. It means removal sequence of sub-automata A_U does not influence the removal sequence of sub-automata A_L. Regular expression aba is obtained from sub-automata A_U after removing state 5 and state 4 as shown below in figure 2.19. State 4 and state 5 become the bridge states of sub-automata A_U and concept of bridge state can be applied for removing states of sub-automata A_U. It means, if possible, vertical and horizontal chopping can be repeated again in sub-automata. Choppping can be continued until no further chopping is possible and after this, optimal removal sequence for sub-automata is found.

Figure 2.19: Sub-automata A_U after removing states 5 and state 4 respectively.

Similarly regular expression ba is obtained after removing sate 6 of sub-automata A_L as shown in figure 2.20.

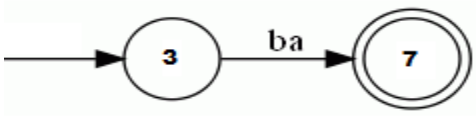


Figure 2.20: Sub-automata A_L after removing state 4.

Final regular expression corresponding to sub-automata A_2 shown in figure 2.15 will be the union of aba and ba i.e (aba+ba). All the sub-automata that are disjoint from each other except for starting and final state can be found in O(|Q|+|δ|) using depth-first-search, where Q is set of states of finite automata.

2.4.5 Delgado and Morais's heuristic

According to Delgado and Morais [21] in each step of state elimination, weight of each state of deterministic finite automata is calculated using following formula and state with lowest weight is deleted first for obtaining shorter regular expression corresponding to that deterministic finite automata.

$$W(q) = (in - \deg ree) \sum_{n=1}^{out-\deg ree} (L_{on}) + (out - \deg ree) \sum_{n=1}^{in-\deg ree} (L_{in}) + (in - \deg ree)(out - \deg ree) L_{loop}$$

where L_{on} is length of transition label on n^{th} out-transition of state q, L_{in} is length of transition label on n^{th} in-transition of state q and L_{loop} is length of self loop transition label.Running time complexity of this heuristic is $O(n^2)$. Consider the following deterministic finite automata shown in figure 2.21 and eliminates states of finite automata using Delgado and Morais heuristic.

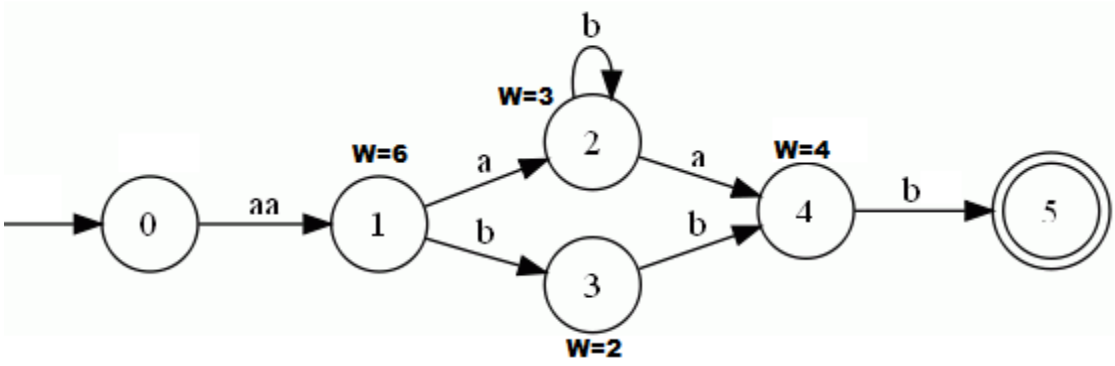


Figure 2.21: Weight calculation using Delgado and Morais's heuristic

W(1) = (1) x (2) + (2) x (2) = 4, W(2) = (1) x (1) + (1) x (1) + (1) x (1) x (1) = 3

W(3) = (1) x (1) + (1) x (1) = 2, W(4) = (2) x (1) + (1) x (2) = 4

Now state 3 has lowest weight as compared to all other states. So state 3 of finite automata will be removed first using state elimination approach as shown in figure 2.22.

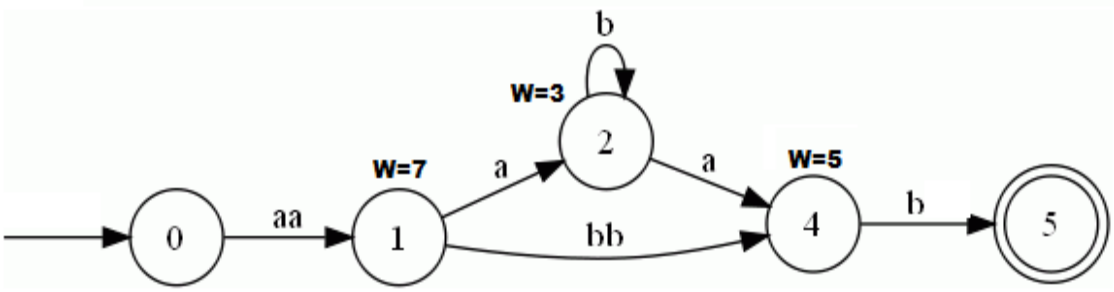


Figure 2.22: DFA after removing state 3.

State 2 of finite automata shown above in figure 2.22 has lowest weight after removing state 3.

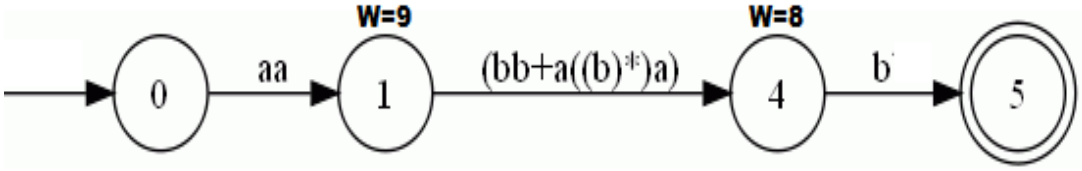


Figure 2.23: DFA after removing state 2

Now DFA has state 1 and state 4 left for deletion with weight 9 and 8 respectively. So state 4 is removed before state 1 according to Delgado and Morais heuristic as shown below in figure 2.24. Regular expression aa(bb+a(b)*a)b is generated after removing all the states(except start and final state).

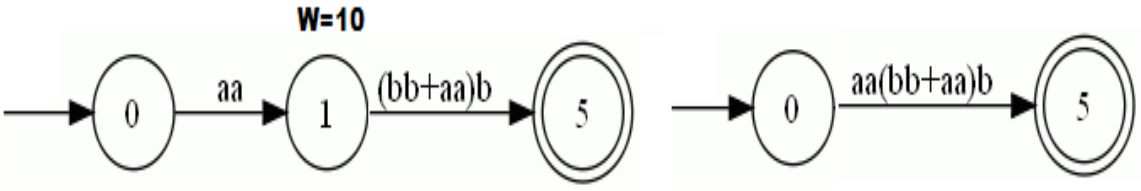


Figure 2.24: DFA after removing state 4 and state 1.

Delgado and Morais heuristic does not guarantee to generate smaller regular expression from deterministic finite automata. Sometimes, removal sequence chosen by this heuristic gives longer regular expression as compared to other optimal removal sequence. For example, we apply delgado amd Morais heuristic on following finite automata shown in figure 2.25.

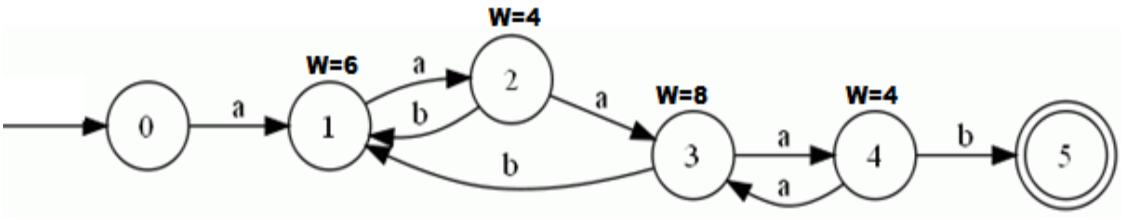


Figure 2.25: DFA with weight of states, calculated using Delgado and Morais heuristic.

Now after removing states of finite automata shown above in sequence of 2-4-1-3 using Delgado and Morais heuristic, regular expression R_1= a(ab)*aa(aa+b(ab)*aa)*ab is obtained equivalent to finite automata.

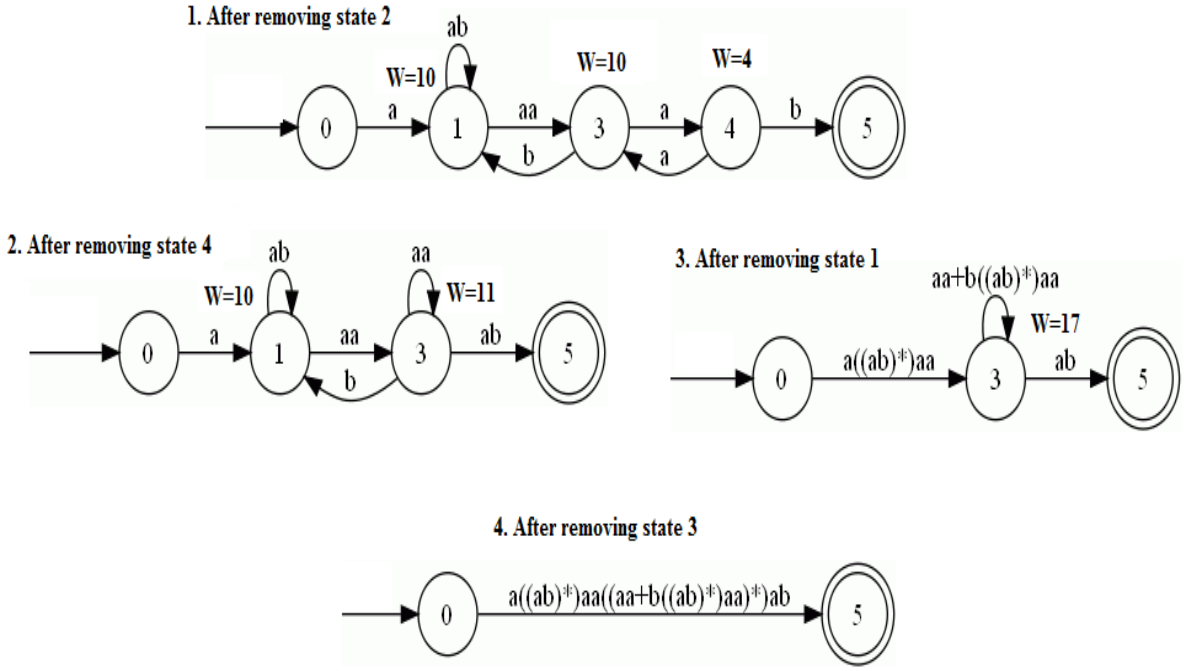


Figure 2.26: Eliminating states of DFA using Delgado and Morais heuristic

But regular expression R_2= aa(b+a(aa)*b)*aa(aa)*ba is obtained if states are removed in sequence of 4-3-2-1 and $|R_1|>|R_2|$.So Delgado and Morais heuristic does not always gives optimal removal sequence for obtaining smaller regular expression.

Chapter 3

Problem Statement and Proposed Solution

This chapter includes the motivation behind the thesis, problem statement followed by comparisons of different techniques used for conversion of DFA to RE. Finally some new heuristics are proposed for choosing optimal removal sequence in state elimination method.

3.1 Motivation

Kleene proves that there is equivalent regular expression corresponding to deterministic finite automata [26]. For converting deterministic finite automata to regular expression, transitive closure approach for conversion of DFA to RE is given by Kleene. Brzozowski extend transitive closure approach, called Brzozowski algebraic approach [10]. Brzozowski's approach is a recursive approach and also uses Arden's Rule [31] for generating regular expression from deterministic finite automata. In this approach a system of equations is generated and regular expression is obtained by solving these equations. State elimination method [2, 23] is the most widely used approach for converting deterministic finite automata to regular expression. In state elimination method, states of DFA are removed one by one except starting and final state and finally regular expression is generated corresponding to given deterministic finite automata. For obtaining smaller RE equivalent to DFA, optimal removal sequence of states should be chosen. In order to choose optimal removing sequence of states in state elimination method, some heuristics are proposed by researchers. Delgado and Morais [21] proposed a strategy that in each step, eliminates a state with lowest weight. Weight of state can be calculated using in-degree, out-degree and loop of that state. Yo-Sub Han and Derick Wood introduced the concept of bridge state, vertical chopping and horizontal chopping [33]. They proved that in order to obtain smaller regular expression using state elimination method, bridge states should be removed after removing all non-bridge states.

The analysis of different techniques for converting deterministic finite automata to regular expression is done in this thesis. Efficiency of state elimination method, using

heuristics proposed by researchers for choosing optimal removal sequence of states, is tested in this thesis work. In order to obtain smaller regular expression, we have proposed some new heuristic for choosing optimal removal sequence of states of deterministic finite automata.

3.2 Problem Statement

The problem of finding the minimal regular expression is NP-complete for DFA. Different techniques or approaches are available for converting deterministic finite automata to regular expression. But our purpose is to find out technique which is efficient and gives smaller regular expression. So we have to carry out comparison between them, design some new heuristics and also there is a need of software for carrying out conversion.

3.3 Objectives and Methodology

1. Analysis and comparison of different approaches used for conversion of DFA to RE.
2. Development of software for converting DFA to RE.
3. To design new heuristics for obtaining smaller regular expression corresponding to given DFA.
4. Verification and validation of the software using newly proposed heuristics.

Analysis and comparison of different techniques used for converting deterministic finite automata to regular expression is carried out by studying research papers and technical notes of various researchers. Implementation of software for conversion of DFA to RE is done in java. New heuristics are proposed using structural properties of DFA and verification of software is also done for the same.

3.4 Comparisons of various approaches used for conversion of DFA to RE

We will apply all the approaches discussed above on deterministic finite automata, which accepts the string having even number of 0's and 1's shown in figure 3.1.

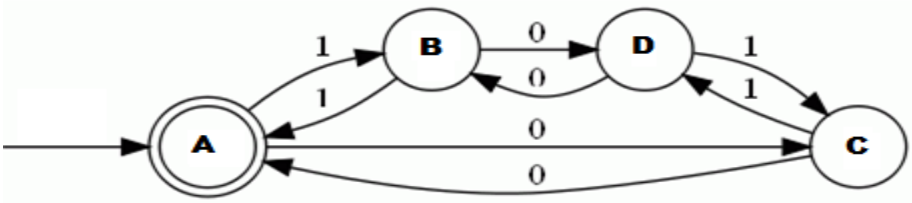


Figure 3.1: DFA for strings having even number of 0's and 1's

1) Transitive closure approach

Applying transitive closure approach on deterministic finite automata shown in figure3.1:

System of equations for deterministic finite automata shown above, are:

R = $R^c_{aa} = R^d_{aa} + R^d_{ac}(R^d_{cc})^* R^d_{ca}$ ..... (5)

$R^d_{aa} = R^b_{aa} + R^b_{ad}(R^b_{dd})^* R^b_{da}$

$R^b_{aa} = R^a_{aa} + R^a_{ab}(R^a_{bb})^* R^a_{ba}$

$R^a_{aa} = R^0_{aa} + R^0_{aa}(R^0_{aa})^* R^0_{aa} = \epsilon + \epsilon\ (\epsilon)^* \epsilon = \epsilon$

$R^a_{ab} = R^0_{ab} + R^0_{aa}(R^0_{aa})^* R^0_{ab} = 1 + \epsilon\ (\epsilon)^* 1 = 1$

Similarly we will get

$R^a_{bb} = 1$, $R^a_{ba} = 1$, $R^b_{aa} = 1\ (1)^* 1$, $R^b_{ad} = 1\ (1)^* 0$

$R^d_{aa} = [1\ (1)^* 1] + [\ 1\ (1)^* 0]\ [\ 0\ (1)^* 0]\ [0\ (1)^*\ ]$

$R^d_{ac} = \ [\ 0+1(1)^*0]+[1(1)^*0][\ 0\ (1)^*0\ ]\ [1 + 0\ (1)^*1]$

$R^d_{cc} = \ [\ 00+01(1)^*10]+(1)\ [\ 0\ (1)^*0]\ [\ 0 + 0\ (1)^*10\ ]$

$R^d_{ca} = 0\ [\ (0 + 00) + 111\ ] + 1\ [\ 0\ (1)^*\ 0\ ]\ [\ 0\ (1)^*\ 1\ ]$

By Putting these values in (5)

R= $[1(1)^*1] + [\ 1(1)^*0\ ]\ [\ 0(1)^*0\ ]\ [\ 0(1)^*\ ] + [\ 0 + 1(1)^*0\ ] + [\ 1(\ 1)^*0\ ]\ [\ 0\ (1)^*0\ ]$ $[1+0(1)^*1]\ (\ [\ 00 + 01(1)^*10\ ] + (1)\ [\ 0(1)^*0\ ]\ [\ 0 + 0(1)^*10]\ )^*0\ [\ (0+00) + 111\ ] + 1[0\ (1)^*$ $0]\ [0\ (1)^*1\ ]$

2) Brzozowski Algebraic method

Set of equations for generating regular expression for DFA shown in fig..5.

A = 0C + 1B +ϵ ……..(6)

B = 0D + 1A ……...(7)

C = 0A + 1D……(8)

D = 0B + 1C…… (9)

Using (7)

A = 0C + 10D + 11A + ϵ D = 00D + 01A + 1C

Using (8)

A = 00A + 01D + 10D + 11A + ϵ

A = (00 + 11)A + (01 + 10)D + ϵ

D = (00 + 11)D + (01 + 10)A

D= $(00 + 11)^*(01 + 10)A$ (using Arden's rule) ……(10)

Using (10)

A = (00 + 11)A + (01 + 10)$(00 + 11)^*(01 + 10)A$ + ϵ

A= [00 + 11 + (01 + 10)$(00 + 11)^*(01 + 10)$]A + ϵ

A = [(00 + 11) + (01 + 10)$(00 + 11)^*(01 + 10)]^*$ (using Arden's rule)

Regular expression obtained using Brzozowski algebraic method corresponding to deterministic finite automata shown in figure 3.1 is [(00+11) + (01+10)$(11+00)^*$(01+10)]*

3) State Elimination Method

There is no bridge state in deterministic finite automata shown in figure 3.1 and each state has same weight 8, calculated using formula given by Delgado and Morais. So any state out of B, C and D can be deleted first.

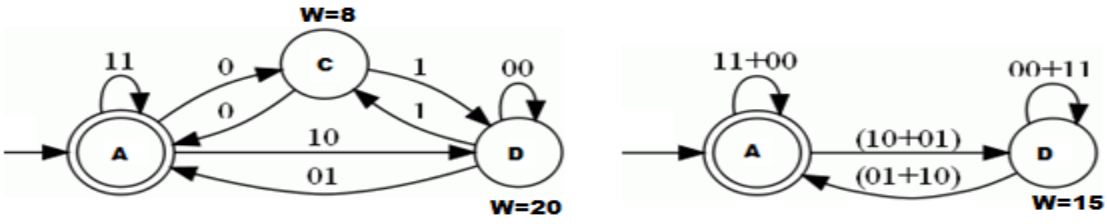


Figure 3.2: DFA after eliminating state B and state C respectively

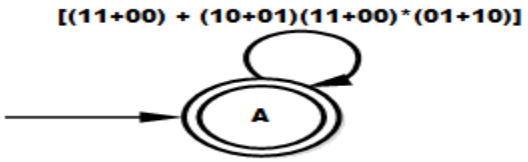


Figure 3.3: Final DFA after eliminating state D

Regular expression obtained corresponding to deterministic finite automata shown in figure 3.1 is $[(11+00) + (10+01)(11+00)^*(01+10)]^*$ using state elimination method.

Now applying all the three methods on the DFA, which accepts all the strings with sub-string "aa".

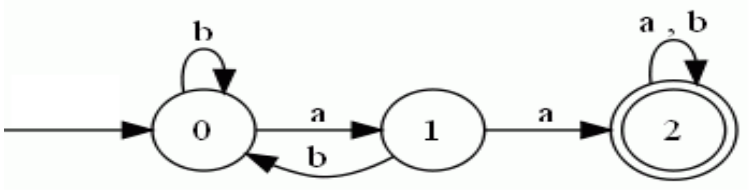


Figure 3.4: An example of DFA accepting all the strings having sub-string "aa"

1) Transitive Closure Approach

System of equations for DFA shown in figure 3.4

R = $R^2_{02} = R^1_{02} + R^1_{02}(R^1_{22})^* R^1_{22}$

$R^1_{02} = R^0_{02} + R^0_{01}(R^0_{11})^* R^0_{12}$

$R^1_{01} = R^0_{01} + R^0_{01}(R^0_{11})^* R^0_{11}$

$R^1_{22} = R^0_{22} + R^0_{21}(R^0_{11})^* R^0_{12}$

$R^0_{02} = \phi$ $R^0_{01} = a$ $R^0_{00} = b+\epsilon$ $R^0_{12} = a$

$R^0_{11} = \epsilon$ $R^0_{22} = a+b+\epsilon$ $R^0_{21} = \phi$

$R^1_{02} = a(\epsilon)^* a$ $R^1_{01} = a + a$ $R^1_{22} = (a+b+\epsilon)$

R = R^2_{02} = aa + aa $(a+b)^*(a+b)$.

2) Brzozowski Algebraic Method

System of equations for DFA shown in figure 3.4

0 = b0 + a1 1 = a2 + b0 2 = (a+b)2 + ϵ

$0 = b^*(a1)$ …….. (using Arden's Theorem)

$2 = (a+b)^* + \epsilon$ …… (using Arden's Theorem)

$1 = a((a+b)^* + \epsilon) + b\,(b^*a1)$

$1 = (b(b^*)a)^*\, a((a+b)^* + a)$

R = $(b(b^*)a)^*\, a((a+b)^* + a)$

3) State Elimination Method

States of DFA shown in figure 3.4 do not satisfy the conditions of bridge state. Regular expression corresponding to DFA can be obtained after removing state 1

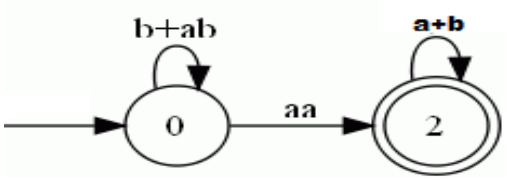


Figure 3.5: DFA after removing state 1

Regular expression (b+ab)aa(a+b)* will accept all the strings having substring "aa"

Due to repeated union of concatenated terms, transitive closure method is very complex [22] and gives very long regular expression as compared to Brzozowski algebraic method and state elimination method. Brzozowski algebraic method is recursive approach and generates reasonably compact regular expressions. For recursion oriented languages, like functional languages, transitive closure method is difficult to implement. In that case, Brzozowski algebraic method is used. But in Brzozowski's approach, also a system of equations is generated.

Brzozowski method takes more time as compared to state elimination method, in which regular expression is obtained by manual inspection. State elimination method, using bridge state concept given by Han and Wood and state weight heuristic by Delgado and Morais, gives shorter regular expression as compared to Brzozowski algebraic method. State elimination method is efficient approach as compared to other approaches and smaller regular expression can be obtained corresponding to given deterministic finite automata using heuristics discussed above.

3.5 Finding Circle and Sub-circle of the DFA

In this section we give definition of circle and sub-circle and an algorithm to find circle of a DFA.

Circle of a DFA: We call a circle exists in a DFA 'A', if it satisfies following conditions:

1. The sub-string 'x' processed by the circle is a sub-string of 'y' ϵ L(A).
2. 'x' may or may not belong to L(A).

3. Path traversed by ‘x’ is such that it will visit the previously visited state q_v again and Q' contain all these states that are traversed during processing of x from first q_v to second q_v.
4. $Q \subseteq Q$(set of states of DFA).

Sub-circle of a Circle in DFA: Any circle with set of states Q" is called a sub-circle of a circle ‘C’ with set of states Q, if it satisfies following conditions:

1. Q" is a proper subset of Q. i.e. $Q'' \subset Q$
2. Q" also satisfy the properties of a circle of a DFA.

If a sub-circle C’ exists in a circle C, we call C as Outer Circle. Based on structural properties of DFA sub-circle of a circle present in DFA can be found. For example DFA shown below has two circles 1→2→3→4→1 and 2→3→2 and these circles satisfy all the conditions of a circle.

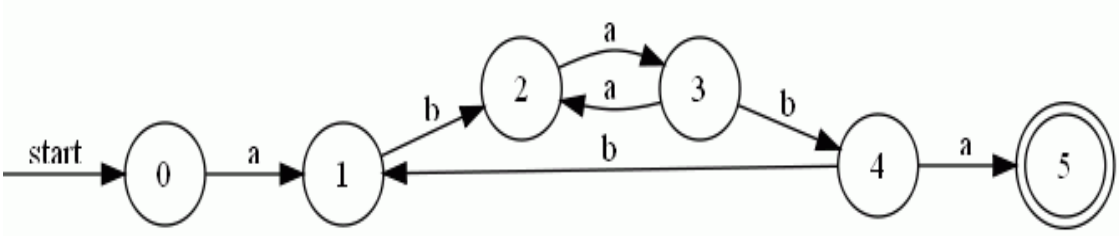


Figure 3.6: An example of circle and sub-circle.

Circle 2→3→2 has set of state Q"={2,3} and circle 1→2→3→4→1 has set of states Q={1,2,3,4} i.e. $Q'' \subset Q$. So circle 2→3→2 is sub-circle of circle 1→2→3→4→1. If circle ‘C’ is present in a DFA and any node present in that circle has self loop then loop is also consider as sub-circle of circle ‘C’ as shown below.

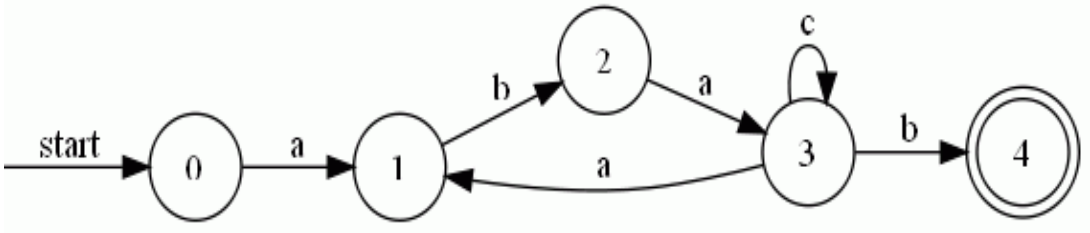


Figure 3.7: Sub-circle of a state with self-loop.

DFA (in fig. 3.7) has two circles 1→2→3→1 and 3→3. Circle 3→3 is sub-circle of 1→2→3→1.

3.5.1 Tarjan's algorithm for finding Circle

For finding circles present in the DFA, Tarjan's algorithm [29] is used. This algorithm uses the concept of strongly connected components of a graph.

Strongly connected component of a graph:- A state 1 of the DFA is said to be strongly connected to state 2 if there exist two paths, one from state 1 to state 2 and another from state 2 to state 1. It means circle 1→2→1 is present in the DFA.

Tarjan's Algorithm [29]

```
Input: Graph G = (V, E)
index = 0                                          // DFS node number counter
S = empty                                          // An empty stack of nodes
for all v in V do
    if (v.index is undefined)                      // Start a DFS at each node
       tarjan(v)                                   // we haven't visited yet
    end if
end for
end of procedure

Procedure tarjan(v)
 v.index = index                                   // Set the depth index for v
 v.lowlink = index
 index = index + 1
 S.push(v)                                         // Push v on the stack
 for all (v, v') in E do                           // Consider successors of v
       if (v'.index is undefined)                  // Was successor v' visited?
          tarjan(v')                               // Recursion
          v.lowlink = min(v.lowlink, v'.lowlink)
     else if (v' is in S)                          // Was successor v' in stack S?
       v.lowlink = min(v.lowlink, v'.index)        // v' is in stack but it isn't in the dfs tree
      end if
  end if
 end for
```

```
 if (v.lowlink == v.index)                    // Is v the root of an SCC?
      print "SCC:"
       v' = S.pop
       print v'
    until (v' == v)
  end if
```

end of procedure

The Tarjan procedure is called once for each state of DFA. The algorithm's running time complexity is linear in the number of transitions (edges) in the given DFA, i.e. O(|V|+*E*).

3.6 New heuristics for choosing optimal removal sequence in State Elimination Method

If more than one, circles are presented in a DFA, then the general procedure is to remove uncommon states (which are not present in more than one, circles) before common states (which are part of more than one, circles). But based on the structural properties of DFA following different cases are possible that must be considered.

1. DFA having no sub-circle of any circle present in the DFA.
2. DFA having sub-circle of any circle present in the DFA. Following sub-cases can also exist.
   a) Circle and its sub-circle with same starting state.
   b) Circle and its sub-circle with different starting state. Following sub-cases can also exist.
      i) DFA having direct transition from starting state of outer-circle to final state of the DFA.
      ii) DFA having no direct transition from starting state of outer-circle to final state of the DFA.

3.6.1. Heuristic for DFA having no sub-circle of any circle present in the DFA

If more than one, circles are presented in a deterministic finite automata and no circle satisfies conditions of sub-circle. Then in each step of state elimination, common states (states, present in more than one circle) are removed after un-common states (states,

present in only one circle). Removal sequence of un-common and common states should be chosen using state weight heuristics.

For example following automata has two circles 1→2→1 and 2→3→2 and state 2 is common between these circles.

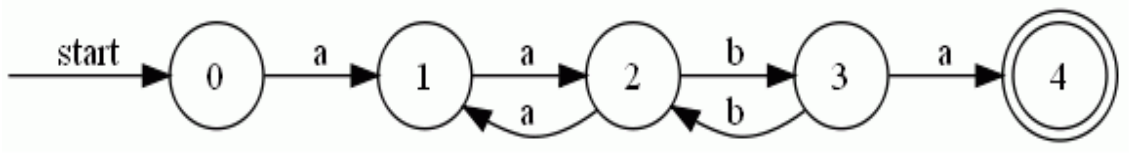


Figure 3.8: DFA having two circles

State 1 satisfies all the conditions of bridge state but it is un-common state. So in order to obtain smaller regular expression state 1 should be removed before removing common state 2.

Case 1:- Removing common states before removing all un-common states present in circles using removal sequence of 3→2→1

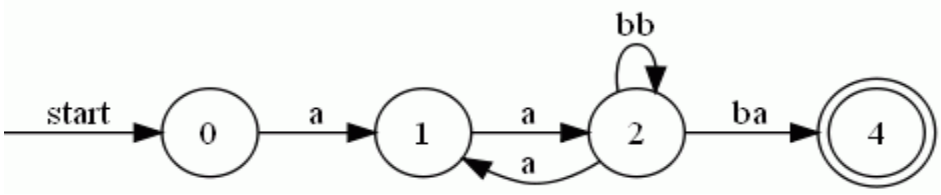


Figure 3.9: DFA after removing state 3

Now 2→2 is sub-circle of circle 1→2→1 as shown above in figure 3.9

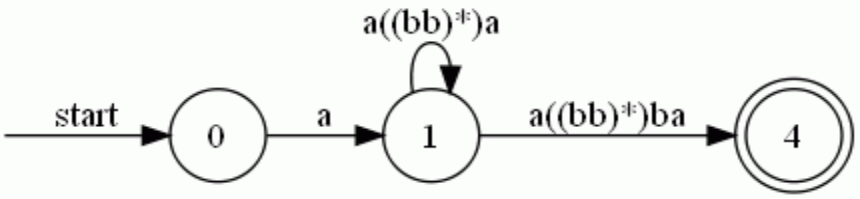


Figure 3.10: DFA after removing state 2

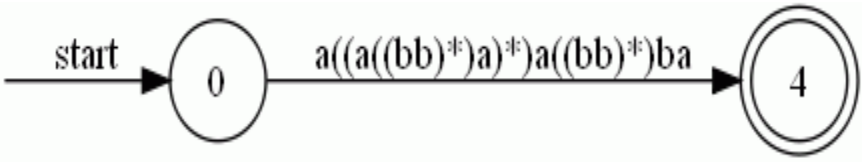


Figure 3.11: DFA after removing state 1

Regular expression $R_1 = a(a(bb)^*a)^*a(bb)^*ba$ is obtained using removal sequence 3→2→1.

Case 2:- Removing all the un-common states before removing common states present in circles, using removal sequence of 3→2→1.

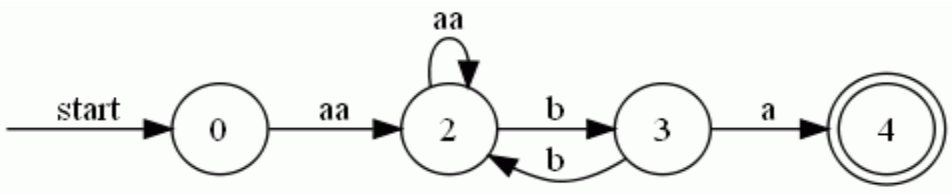


Figure 3.12: DFA after removing state 1

Circle 2→2 is sub-circle circle of 2→3→2(outer circle) present in DFA shown above. So state 3 is un-common state and for obtaining smaller regular expression, it should be removed before state 2.

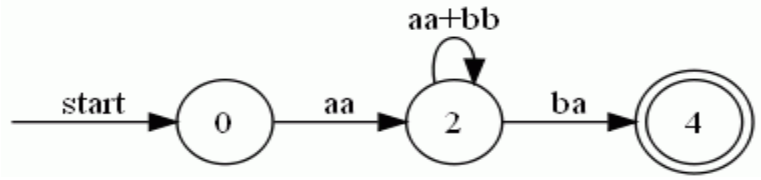


Figure 3.13: DFA after removing state 3

After removing all the un-common nodes, DFA is left with only one state, which is common between circles. So, after removing state 2, following regular expression is obtained shown in figure 3.14.

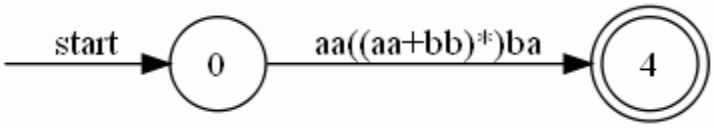


Figure 3.14: DFA after removing state 2

Regular expression $R_2 = aa(aa+bb)^*ba$ is obtained after using removal sequence 1→3→2. Now $|R_1| > |R_2|$, it means smaller regular expression is obtained if un-common states of circles present in DFA are removed before removing common states of circles using state elimination method and if any un-common state satisfies conditions of bridge state that should be removed before non-bridge states.

3.6.2 Heuristic for DFA having sub-circle of any circle present in the DFA.

If circle and its sub-circle are presented in deterministic finite automata, then following two cases must be considered.

3.6.2.1 Circle and sub-circle with same starting state

If circle and its sub-circles, present in a DFA, have same start or finish state, then rule for choosing removal sequence of states is same as rule for DFA having no sub-circle of any circle i.e. removal of un-common states before removing common states. Following figure 3.15 shows a DFA having circle and its sub-circle with same starting state.

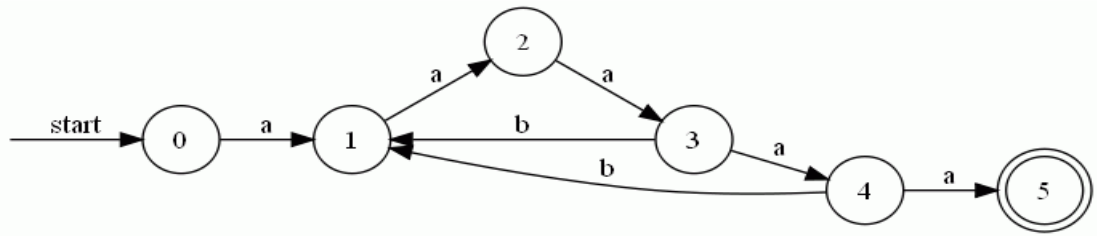


Figure 3.15: DFA having circle and its sub-circle with same starting state.

Circle 1→2→3→1 is sub-circle of circle 1→2→3→4→1 and state 1 is a bridge state of DFA shown in figure 3.15. States {1,2,3} of DFA are common states of two circles and state 4 is un-common state of circles, present only in outer circle.

Case 1:- Removing un-common states before common states using removal sequence 4→2→3→1

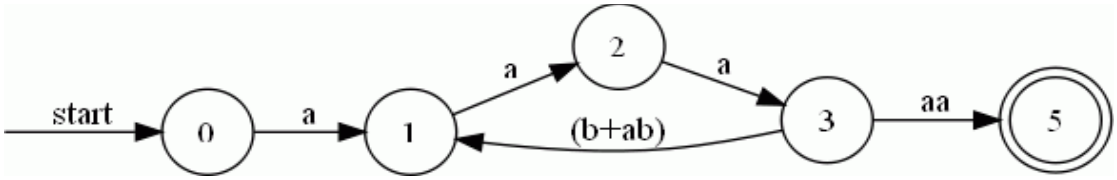


Figure 3.16: DFA after removing 4

Now state 1 is bridge state and only one circle is there after removing state 2. So bridge state should be removed after removal of all non-bridge states.

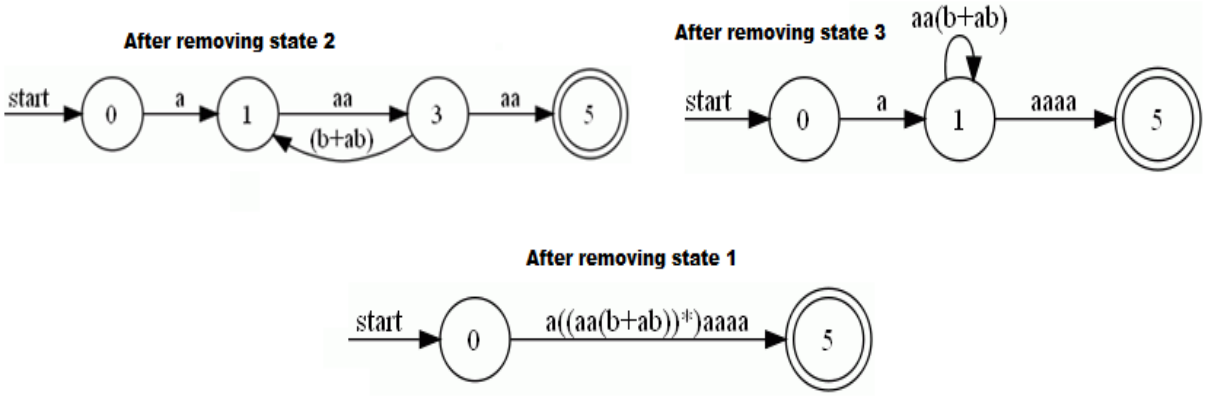


Figure 3.17: DFA after state removal sequence 2→3→1.

Regular expression R_1 = a(aa(b+ab))*aaaa is obtained.

Case 2:- Removing common states before un-common states of circles using removal sequence 2→3→4→1 as shown below.

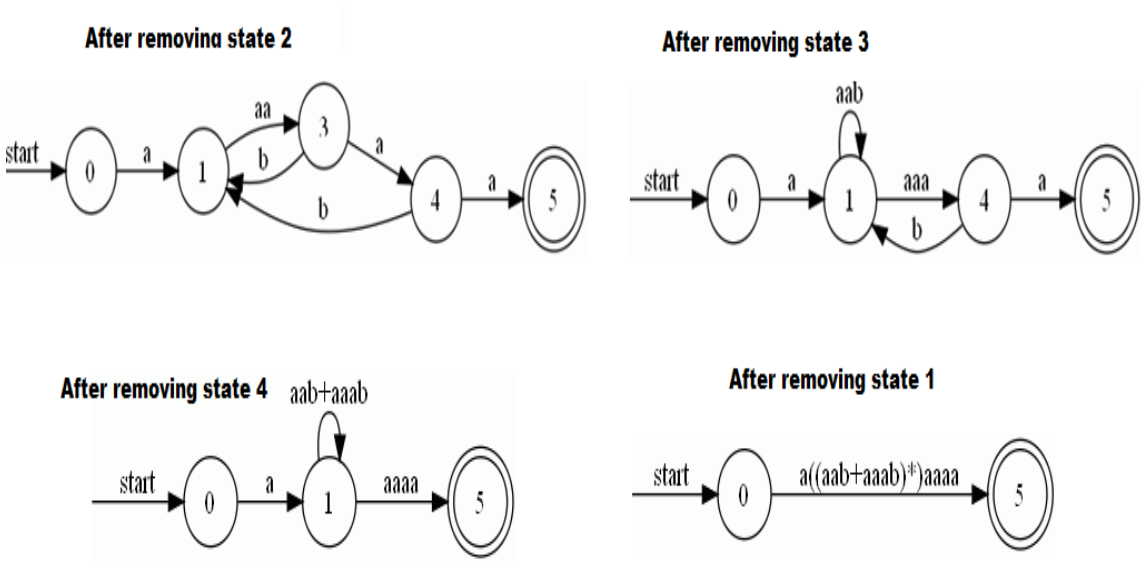


Figure 3.18: DFA after state removal sequence 2→3→4→1

Regular expression R_2 = a(aab+aaab)*aaaa is obtained using removal sequence 2→3→4→1. Now $|R_2| > |R_2|$, Hence removal of un-common states before common states gives smaller regular expression. In this case, starting state of two circles satisfies all the conditions of bridge state and also it is common state of circles. So it should be removed at last. In this case, Han and Wood's heuristic [33] of removing bridge state after removing all non-bridge state is also considered.

Following figure 3.19 shows another example of DFA having circle and its sub-circle with same starting state. Circle 2→3→4→2 is sub-circle of 2→3→4→5→6→2 and starting state of these circles is not a bridge state.

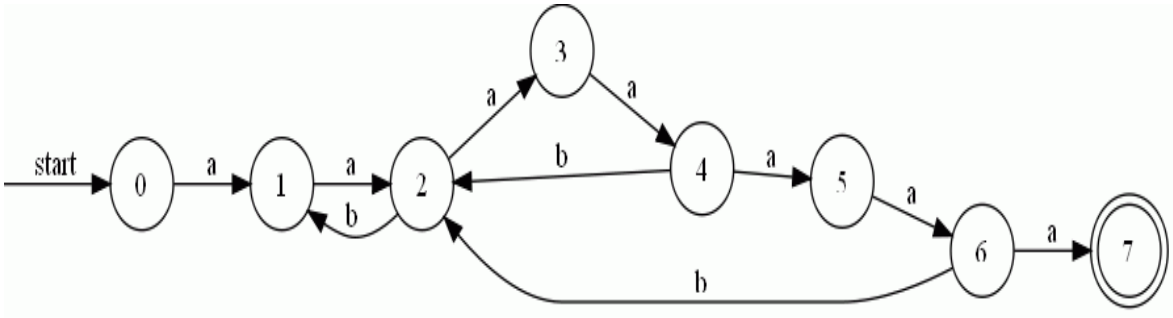


Figure 3.19: Another example of circle and its sub-circle with same starting state

States {2, 3, 4} are common states of circle 2→3→4→5→6→2 and its sub-circle 2→3→4→2. States {5, 6} are un-common states of these circles. State 2 is common between circles 1→2→1 and 2→3→4→2. Also it is common between 2→3→4→5→6→2 and 1→2→1. State 1 of circle 1→2→1 satisfies all the conditions of bridge state. But state 1 is un-common state, so according to heuristic 1 bridge state will be deleted before non-bridge states for obtaining smaller regular expression.

Case 1:- Removing un-common states before un-common states of circles using removal sequence 1→5→6→3→4→2.

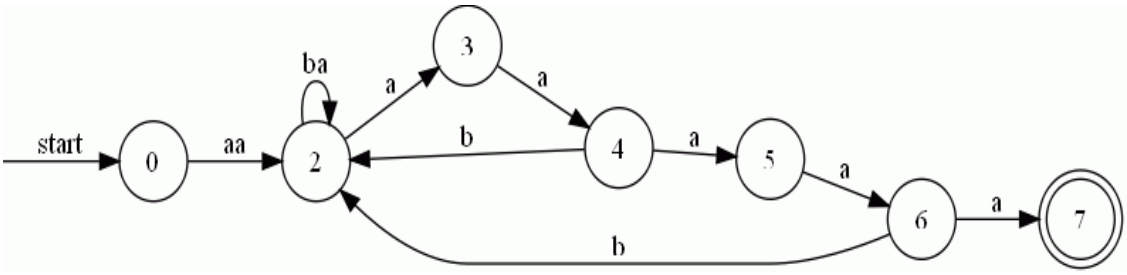


Figure 3.20: DFA after removing state 1

Now circle 2→2 is also sub-circle of circles 2→3→4→2 and 2→3→4→5→6→2. State 5 and state 6 are un-common states. Regular expression R_1 = aa(ba+aa(b+aab))*aaaaa is obtained using this removal sequence as shown in figure 3.21.

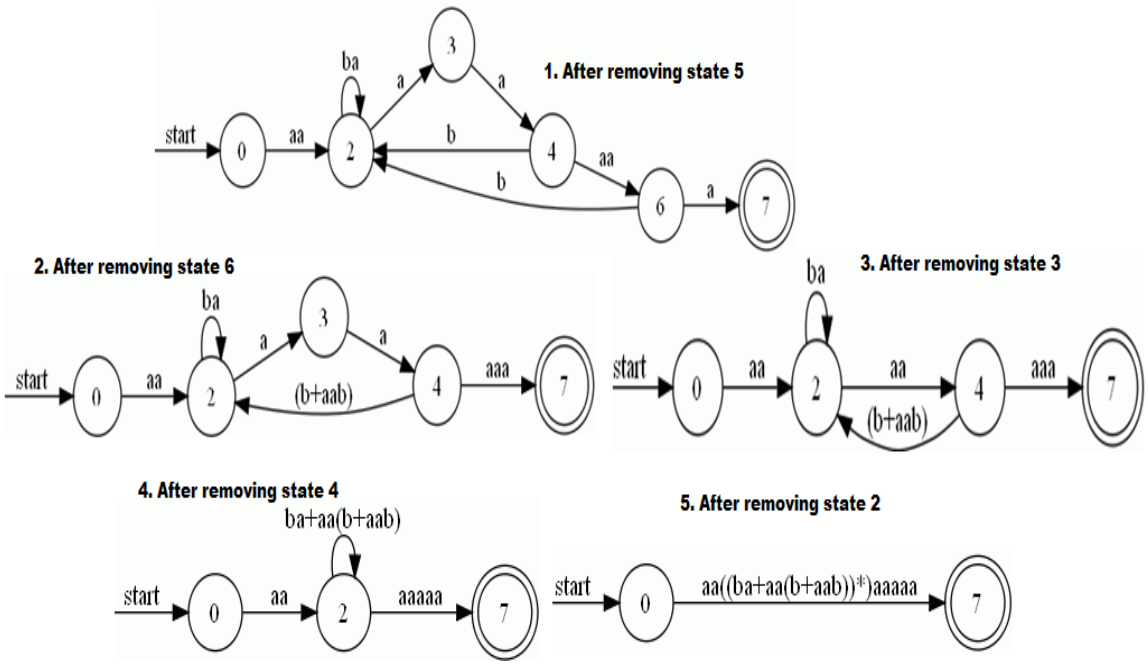


Figure 3.21: DFA after removal sequence 5→6→3→4→2.

Case 2:- If common states are removed before un-common states using removal sequence 1→3→4→5→6→2, then regular expression R_2 = aa(ba+aab+aaaab)*aaaaa is obtained and if bridge state is removed after removing non-bridge states regular expression obtained equivalent to DFA is R_3 = aa(aa(b+aab))*ba(aa(b+aab))*aaaaa. | R_1 | < | R_2 | and | R_1 | < | R_3 |. So in this case Han and Wood's [33] heuristics of bridge state is not applicable.

3.6.2.2 Circle and sub-circle with different starting state

If circle and its sub-circle, present in deterministic finite automata, have different starting states then following two cases must be considered.

1. DFA having direct transition from starting state of outer circle to final state of DFA.

If circle and its sub-circle with different starting state are presented in a DFA and also direct transition is there from starting state of outer circle 'C' to final state of DFA. Then in each step of state elimination, all other un-common states of the circles except starting state of circle 'C' should be removed before removing common states of these circles. For example following figure 3.22 shows a DFA having circle 1→2→3→4→1 and its sub-circle 2→3→2. There is direct transition from starting state 1 of circle 1→2→3→4→1 to final state 5 of DFA.

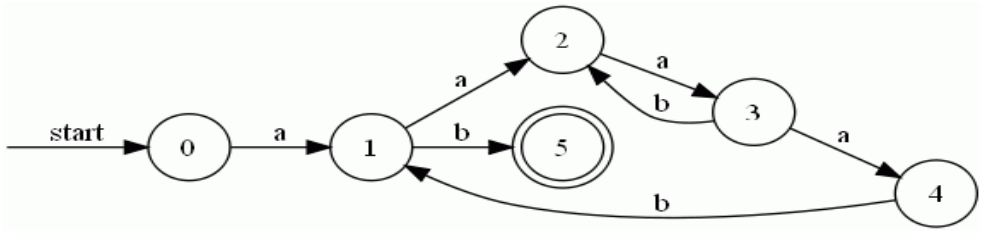


Figure 3.22: DFA with direct transition from starting state of outer circle to final state.

States {2, 3} nodes are common states of the circles and states {1, 4} are un-common states. State 1 is a bridge state. In this case bridge state or starting state of outer circle 'state 1' should be removed after removing all other un-common states then common states of the circles. State 4 is un-common state of circles, so it should be removed first as shown below.

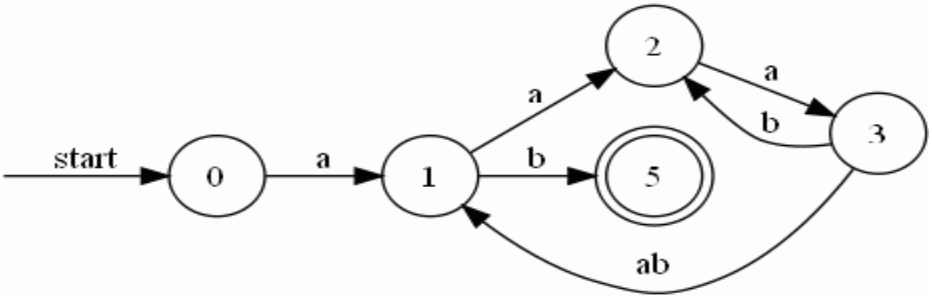


Figure 3.23: DFA after removing state 4.

Now after removing state 4, circle 2→3→2 becomes sub-circle circle of 1→2→3→1 and state 1 is the only un-common state present in these circles. But due to direct path from state 1 to final state of DFA, it should be removed at last in this case.

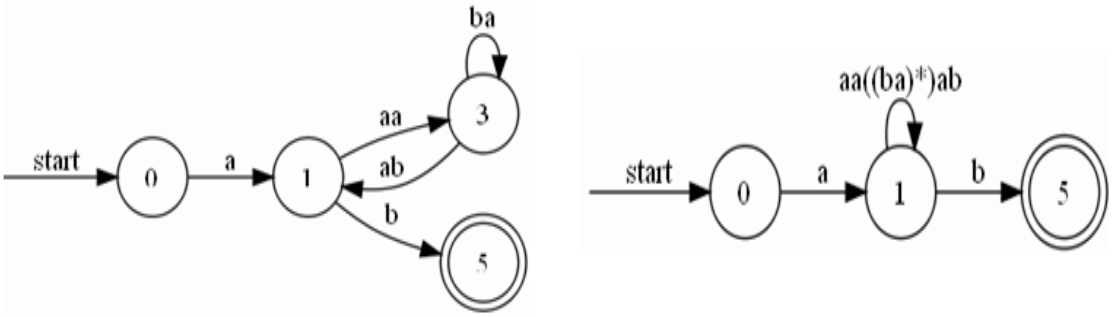


Figure 3.24: DFA after removing states 2 and state 3 respectively.

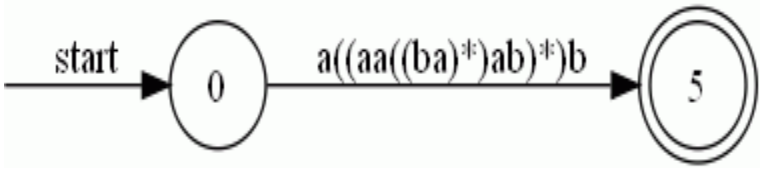


Figure 3.25: DFA after removing state 1.

Regular expression R_1 = a((aa((ba)*)ab)*)b is obtained using removal sequence 4→2→3→1. In this case first removal of un-common states, followed by common states elimination of circles without considering the direct path from starting state of outer-circle to final state of DFA, does not give smaller regular expression as shown below.

Removing all the un-common states before removing common states of circles, present in DFA, shown in figure 3.26 using removal sequence 1-4-3-2.

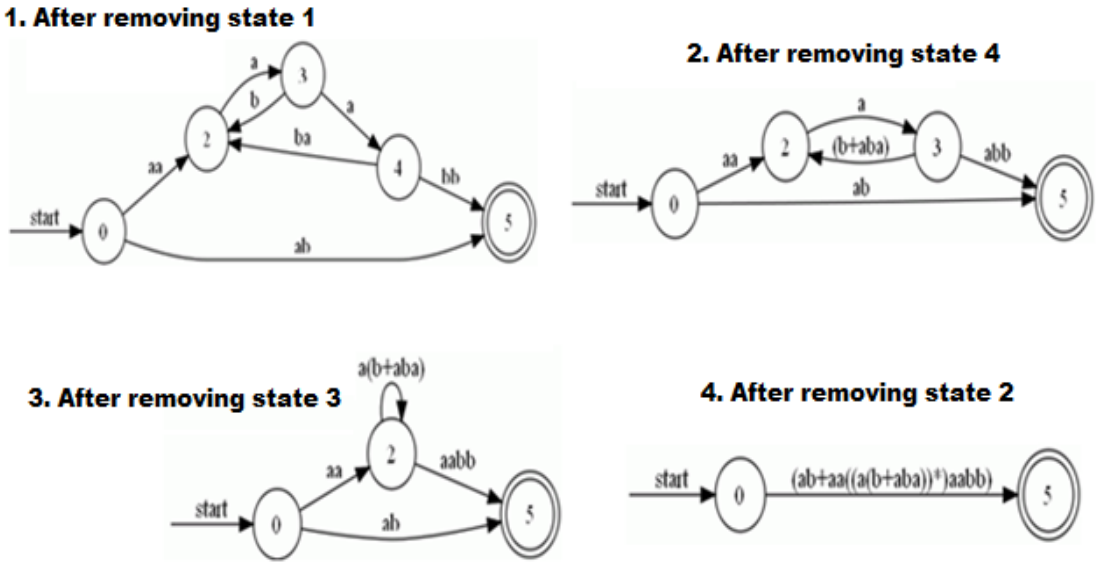


Figure 3.26: An example of eliminating states using removal sequence 1→4→3→2.

Regular expression R_2 = ab+aa(a(b+aba))*aabb is obtained from removal sequence 1→4→3→2. Now $|R_2|>|R_1|$, so in this case starting state of outer circle should be removed after removing all other states of circles.

Following figure 3.27 shows another example of the same case that has been discussed above. In this DFA, three circles are there. 3→4→3 is sub-circle of 2→3→4→5→2. State 2 is starting state of circle 2→3→4→5→2 and there is direct transition from state 2 to final state 6 of DFA. In this case, if removal sequence 1→5→2→4→3 is used, i.e. first remove all un-common states, followed by common states of circles without considering direct transition from state 2 to state 6, then regular expression R_1=aa(aa)*c+aa(aa)*b(b(b+aa(aa)*b)*)*baa(aa)*c is obtained equivalent to DFA.

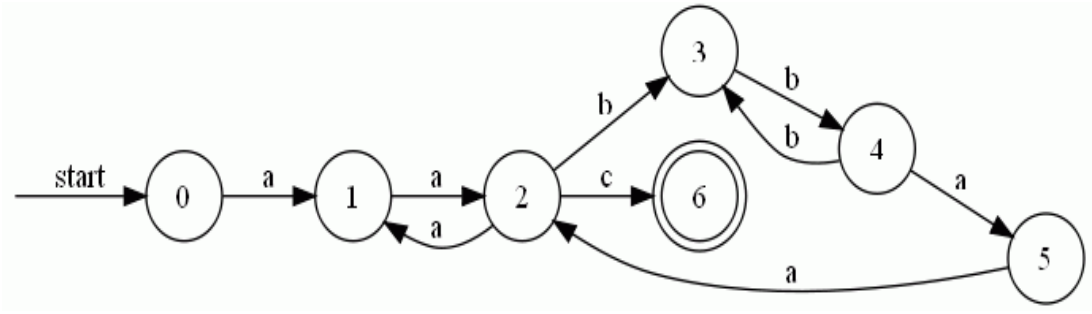


Figure 3.27: Another example of DFA with direct transition from starting state of outer circle to final state

If removal sequence 1→5→4→3→2 is used, i.e. removing starting state of circle 2→3→4→5→2 after removing all other states of circles using concept of sub-circle, then regular expression R_2=aa (aa + b (bb)*baa)*c is obtained and $|R_2|<|R_1|$.

2. DFA having no direct transition from starting state of outer-circle to final state

If circle and its sub-circle with different starting states are presented in a DFA and no direct transition is there from starting state of outer-circle to final state of DFA. Then the removal of un-common states before removing common states gives smaller regular expression. For example following figure 3.28 shows DFA having circles 1→2→3→4→1 and its sub-circle 2-3-2 and no direct transition from starting state of circle 1-2-3-4-1 to final state of DFA.

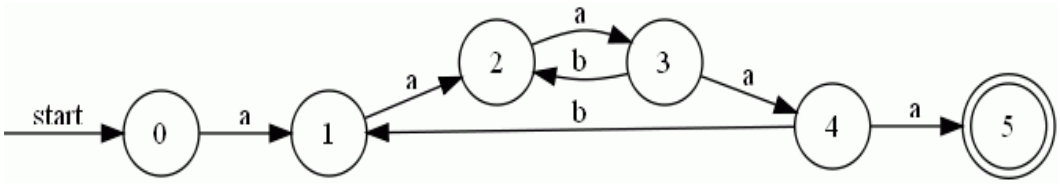


Figure 3.28: DFA without any direct transition from starting state of outer circle to final state.

State 1 is bridge state of given DFA, present in circle 1→2→3→4→1 and states {2, 3} are common states of circles. States {1, 4} are un-common states of these circles. So in this case bridge state 1 will be removed before removing non-bridge states of circles using concept discussed above for obtaining smaller regular expression corresponding to DFA.

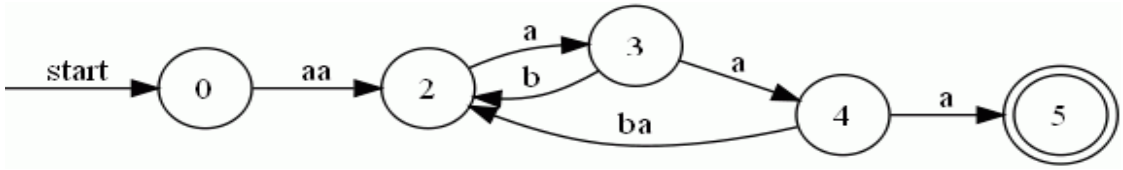


Figure 3.29: DFA after removing state 1

Now state 4 is un-common state and should be removed first

Figure 3.30: DFA after removing state 4

After removing state 4 DFA has left with only two states which are common between the circles shown in figure 3.28.

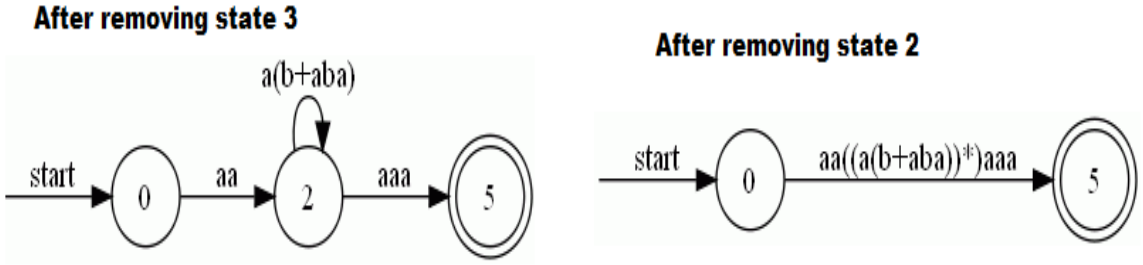


Figure 3.31: DFA after removing state 3 and state 2 respectively.

Regular expression R_1 = aa(a(b+aba))*aaa is obtained equivalent to DFA as shown in figure 3.31.

Removing common states of circles, present in the DFA shown in figure 3.28, before removing all un-common states using removal sequence 4→2→3→1 as shown below.

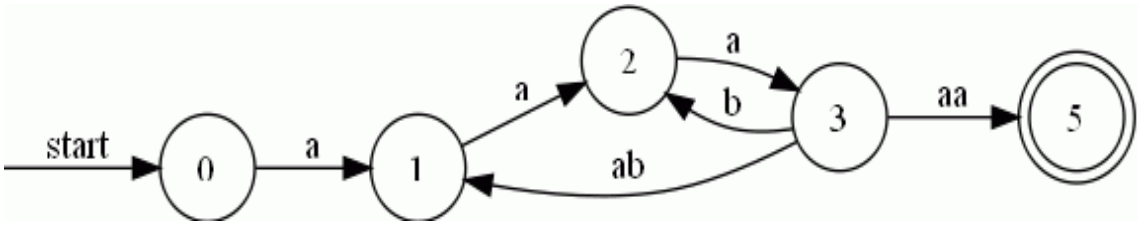


Figure 3.32: DFA after removing state 4

Now state 1 is un-common state. It should be removed before any other state of circles. But first removal of state 2 then state 1 gives following DFA.

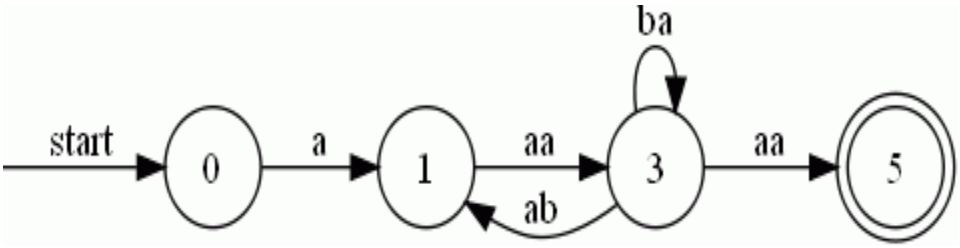


Figure 3.33: DFA after removing state 2

State 3 is common between circles 1→3→1 and 3→3 shown in figures 3.33. Removal of state 3 gives following DFA.

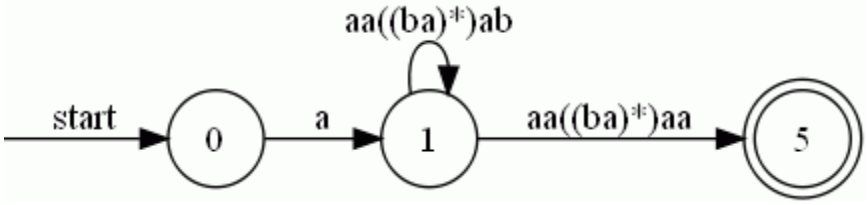


Figure 3.34: DFA after removing common state 3

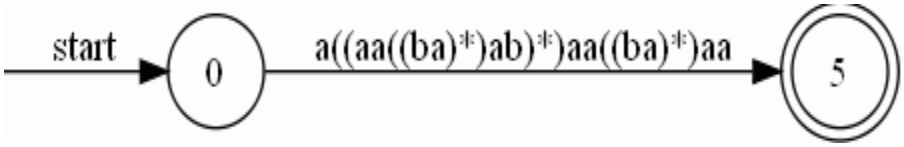


Figure 3.35: DFA after removing node 1

Regular expression R_2 = a(aa(ba)*ab)*aa(ba)*aa is obtained using removal sequence 4→2→3→1 for given DFA. $|R_2|>|R_1|$, so in each step of state elimination all un-common states should be removed before removing common states of the circles in this kind of cases.

Following figure 3.36 shows another example of the same case that has been discussed above in section 2.2.2. In this case three circles, 1→2→1, 2→3→4→5→2 and 3→4→3, are presented in the DFA. Circle 3→4→3 is sub-circle of circle 2→3→4→5→2. State 1 is bridge state of DFA. State 2 is common between circles 2→3→4→5→2 and 1→2→1. So, according to heuristic 1 bridge state 1 should be removed first for obtaining smaller regular expression.

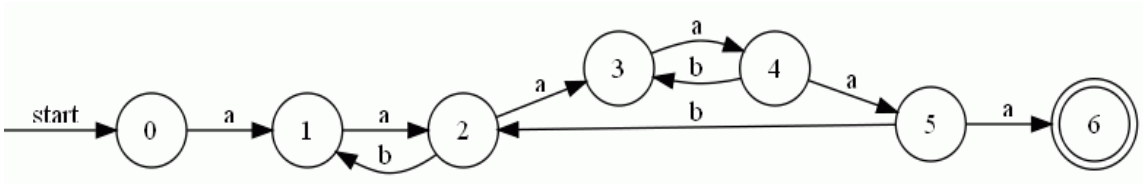


Figure 3.36: Another example of DFA without any direct transition from starting state of outer circle to final state.

Circle 3→4→3 is sub-circle of circle 2→3→4→5→2 of DFA shown in figure 3.36. States {3, 4} are common states of these circles. States {2, 5} are un-common states the circles. So in order to obtain smaller regular expression optimal removing sequences are 1→2→5→3→4 or 5→1→2→3→4.

In each step of state elimination, removal of all un-common nodes before removal of common states of the circles gives regular expression R_1=aa((ba)*)aa(((b+ab((ba)*)a)a)*)aa corresponding to DFA shown in figure 3.36. If removal sequence 1→3→4→2→5 is used (i.e. removing common states of circles before removing all un-common states) then regular expression R_2=aa(ba)*aa(ba)*a(b(ba)*aa(ba)*a)*a is obtained and $|R_2|>|R_1|$. If removal sequence 5→2→3→4→1 is used for removing bridge state at last then regular expression R_3=a((ab+aaa(((b+aba)a)*)abb)*)aaa(((b+aba)a)*)aa is obtained and $|R_3|> |R_2|>|R_1|$.

These newly designed heuristics will help us to find smaller regular expression from DFA. These heuristics will be useful in the field where conversion from DFA to RE is required.

Chapter 4

Experimental Results and Software validation

This chapter gives the description of the software developed for the conversion of deterministic finite automata to regular expression using state elimination method with heuristics proposed by various researchers for choosing optimal removal sequence. Verification and Validation of the software for the newly proposed heuristics have also been carried out.

4.1 Software for Conversion of DFA to RE using State Elimination Approach

Software for conversion of DFA to RE has been developed in java. After execution of software java applet or GUI is created that takes deterministic finite automata as input and gives regular expression equivalent to input DFA. It also shows the DFA after removal of every state of DFA i.e. shows intermediate results. Following steps are carried out to convert DFA to RE.

1. Select the operation 'Add Node' from combo box.
2. Create the states of DFA by left click the mouse button on the panel as shown below.

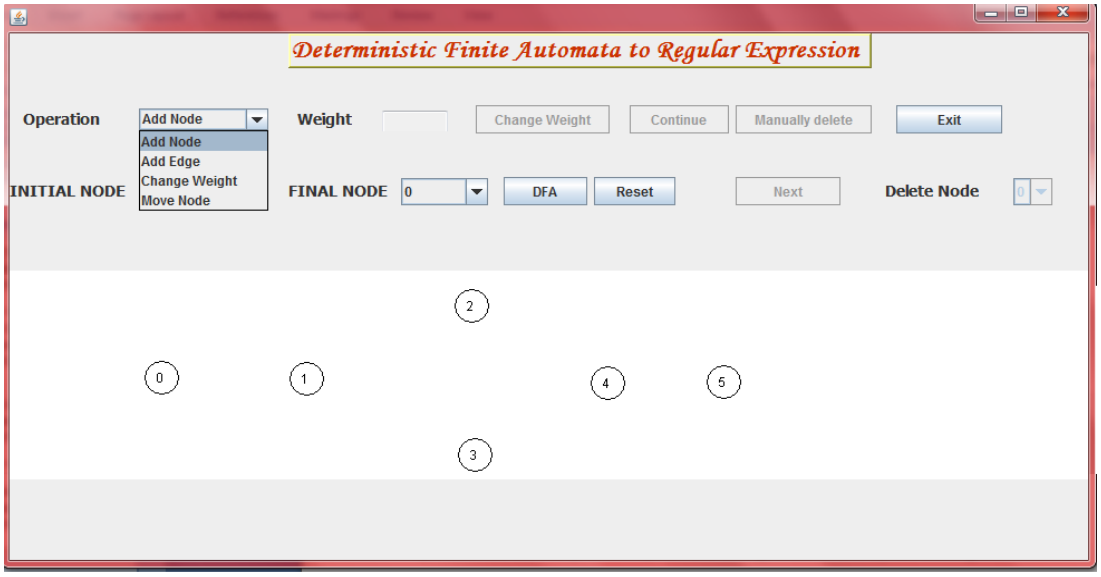


Figure 4.1: Operation Add Node

3. After creating states of DFA, select the operation add edge from the combo box. To create an edge between two states, first left click on source state then left click on destination state of edge. Make sure that weight of edge has been entered into the text box named weight.

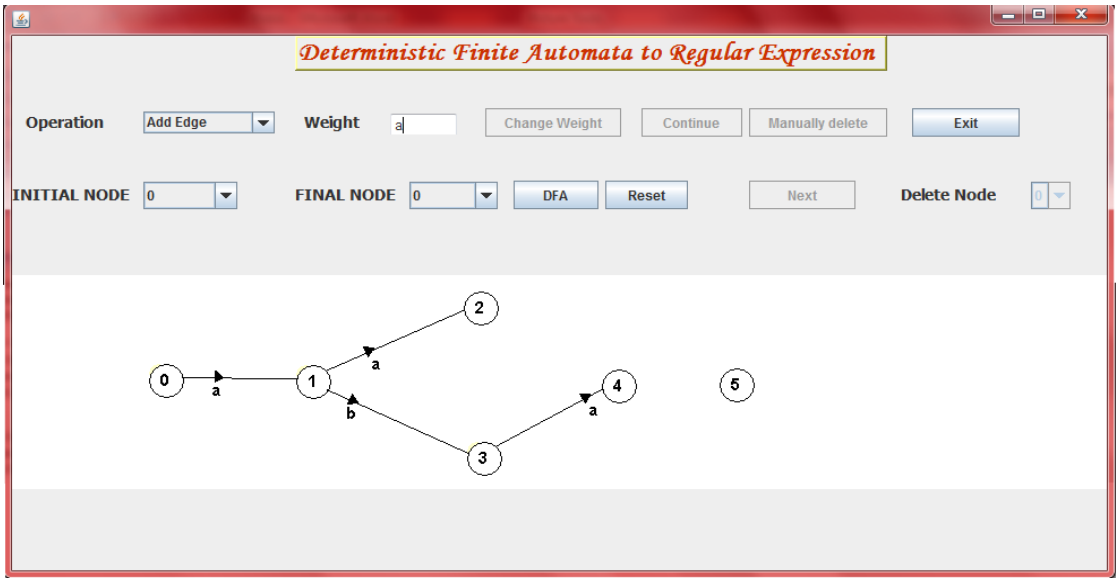


Figure 4.2: Operation Add Edge

4. After connecting all the states of DFA, select the starting state and final state of the DFA from combo box named initial node and final node respectively.

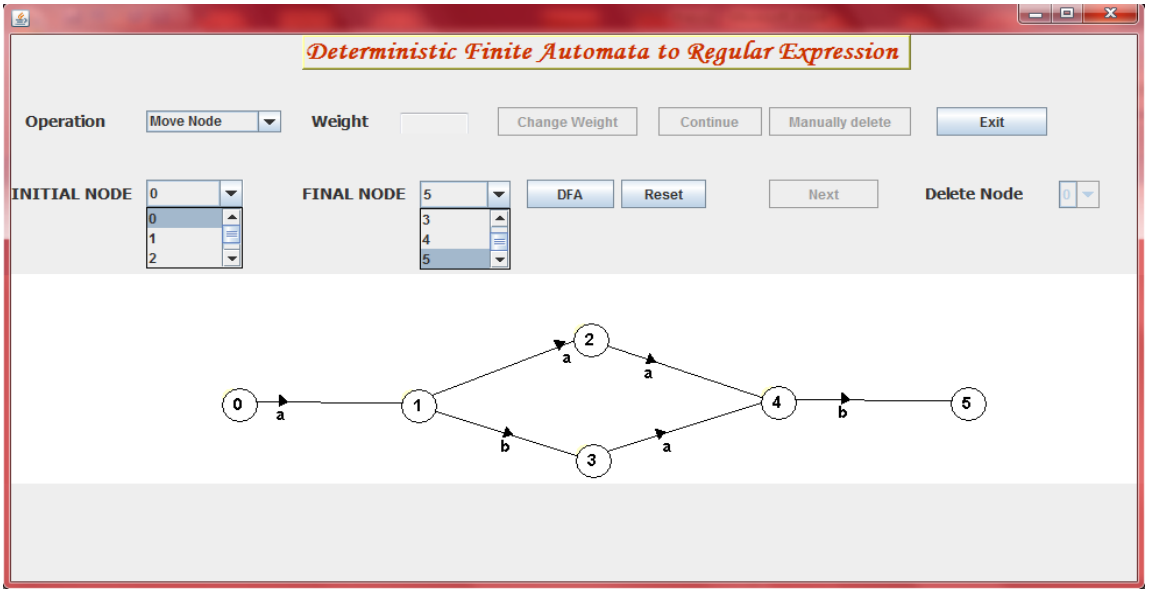


Figure 4.3: Starting and final state selection

5. Weight of edge can be changed by left clicking mouse on the arrow of the edge and then new weight can be updated in the text box named weight. After this click the change weight button.

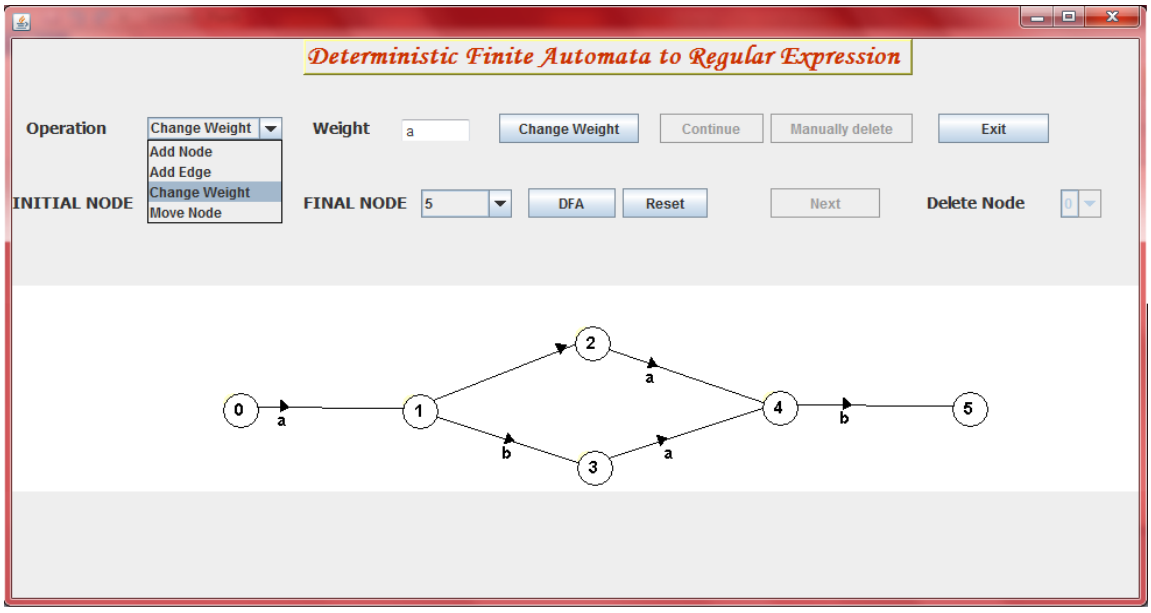


Figure 4.4: Operation Change weight

6. After selecting starting and final state of the DFA, click on button DFA to create a DFA. It also finds the bridge states that are shown below the panel.

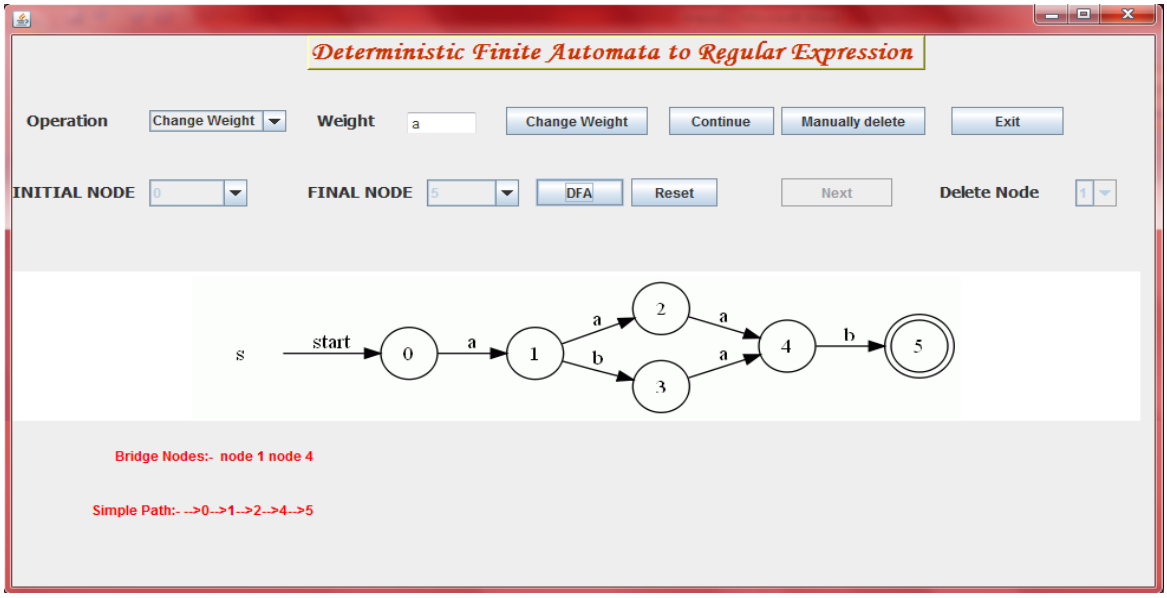


Figure 4.5: creating DFA and generating regular expression equivalent to DFA.

7. Now states of DFA can be removed by two ways

   1) By clicking continue button 2) By clicking manually delete button.

   1) If continue button is pressed then all the states of DFA can be removed automatically and for choosing optimal removal sequence of states for smaller regular expression heuristics, that has been discussed in previous chapter, are used.

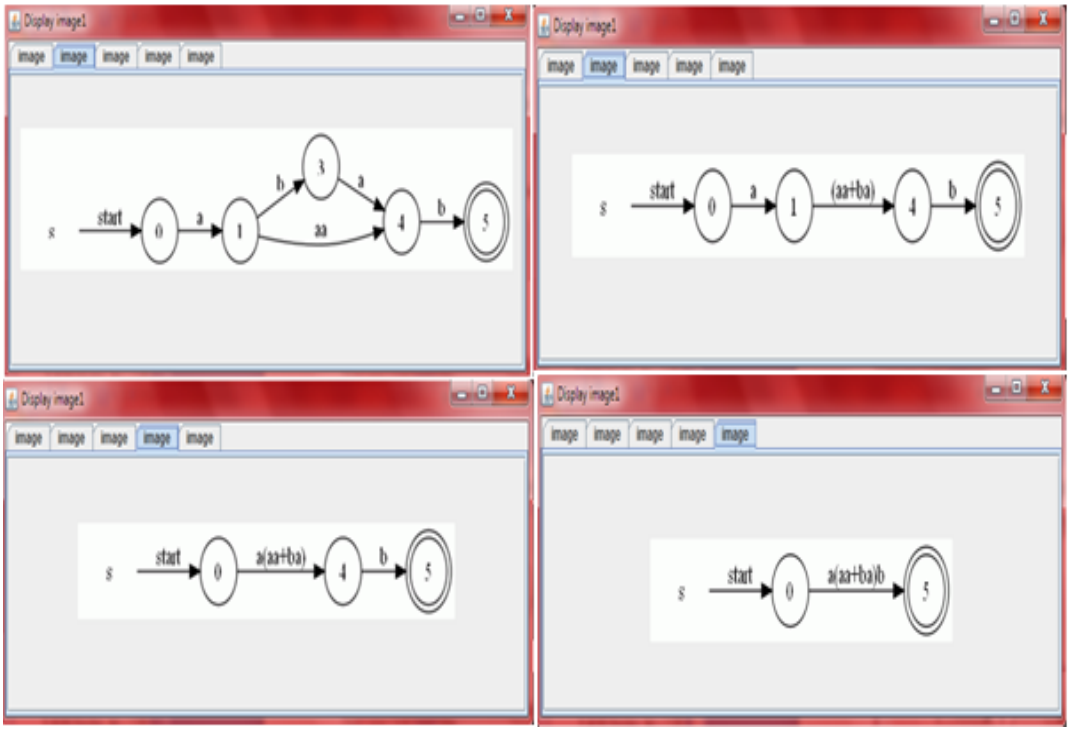


Figure 4.6: DFA after each step of state elimination.

It also shows intermediate stage after removal of every state and gives regular expression equivalent to input DFA. As shown in above figures state of DFA are removed using heuristics discussed in this thesis. State 1 and state 4 are bridge states and these states should be removed at last. So regular expression a(ba+ab)a is obtained and it is also shown below the panel of main applet.

2) If manually delete button is pressed then nodes of DFA are deleted one by one after selecting particular state you want to delete from combo box named Delete Node and then pressing next button. After removing all the nodes except starting and final nodes of DFA, regular expression equivalent to input DFA is generated.

4.2 Software Validation using proposed heuristics

Software designed for conversion of deterministic finite automata to regular expression has been validated using proposed heuristics. Removal sequence of states for conversion of DFA to RE is automatically chosen by software according to proposed heuristics. For example if DFA with circle and its sub-circle, is given as input to software as shown in figure 4.7 then after pressing continue button states are removed automatically according to heuristics proposed in chapter 3. There are two circles, 2→3→2 and 1→2→3→4→1, present in the DFA. Circle 2→3→2 is sub-circle circle of 1→2→3→4→1.

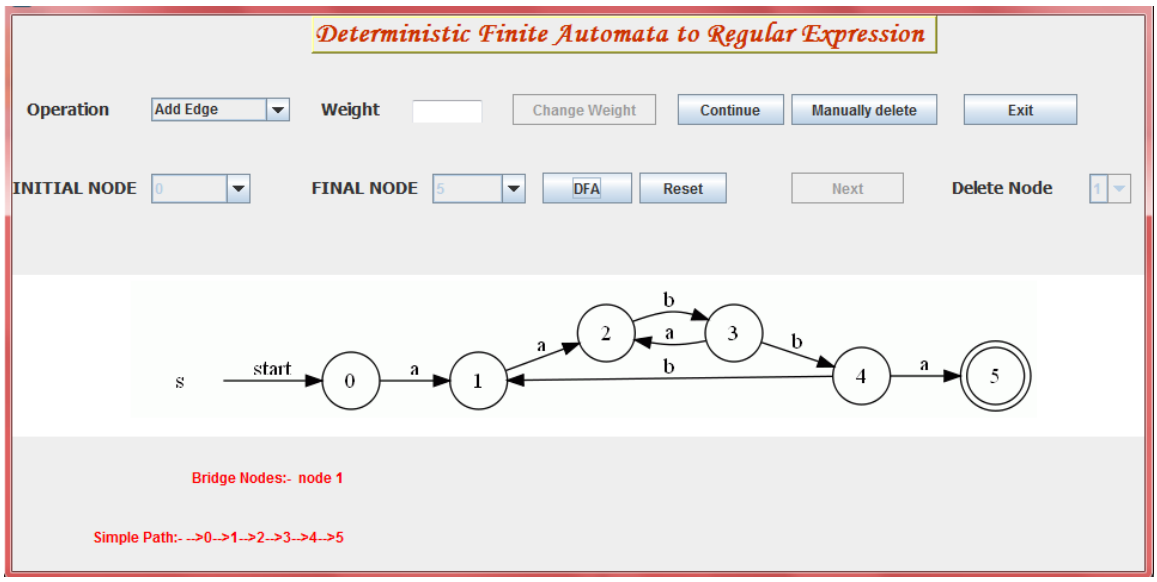


Figure 4.7: DFA with circle and its sub-circle, input to software

So according to heuristics proposed in chapter 3, optimal removal sequence should be 1→4→2→3 for obtaining smaller regular expression equivalent to input DFA.

Figure 4.8 shows elimination of states of DFA according to newly proposed heuristics and regular expression aab((a+bba)b)*ba is obtained equivalent to DFA.

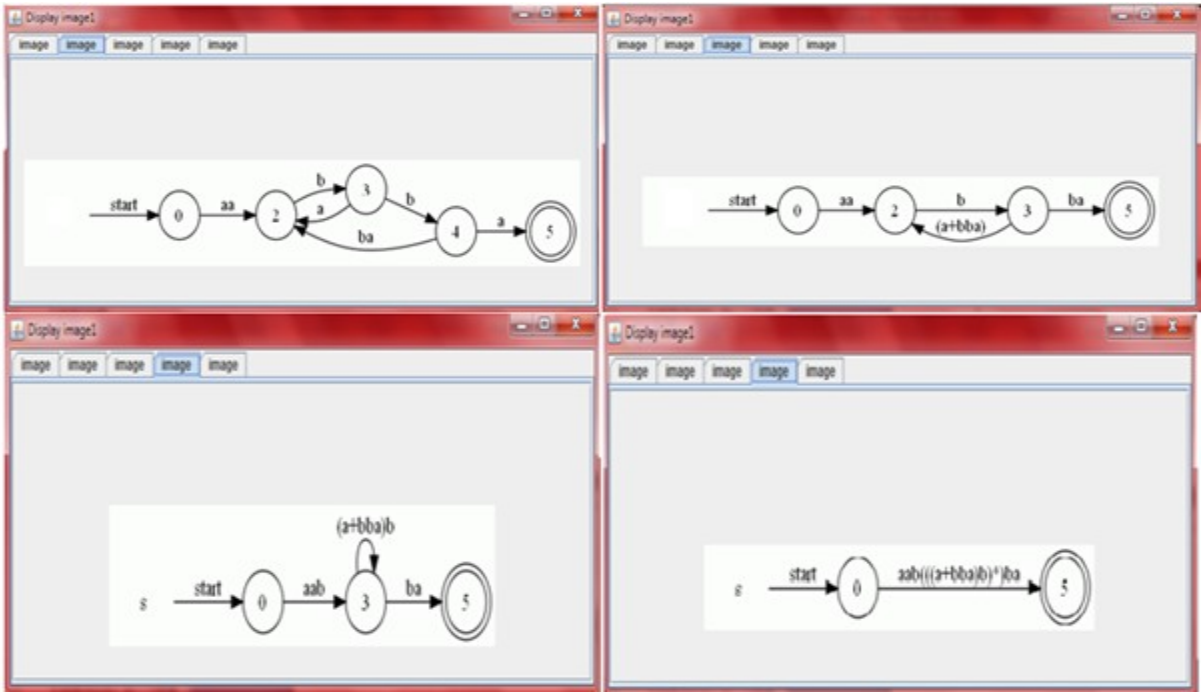


Figure 4.8: State elimination according to proposed heuristics

Similarly, figure 4.9 shows another example of DFA with circle and its sub-circle for testing, removal sequence chosen by software. In order to obtain smaller regular expression optimal removal sequence of states for this DFA is 1→5→6→3→4→2.

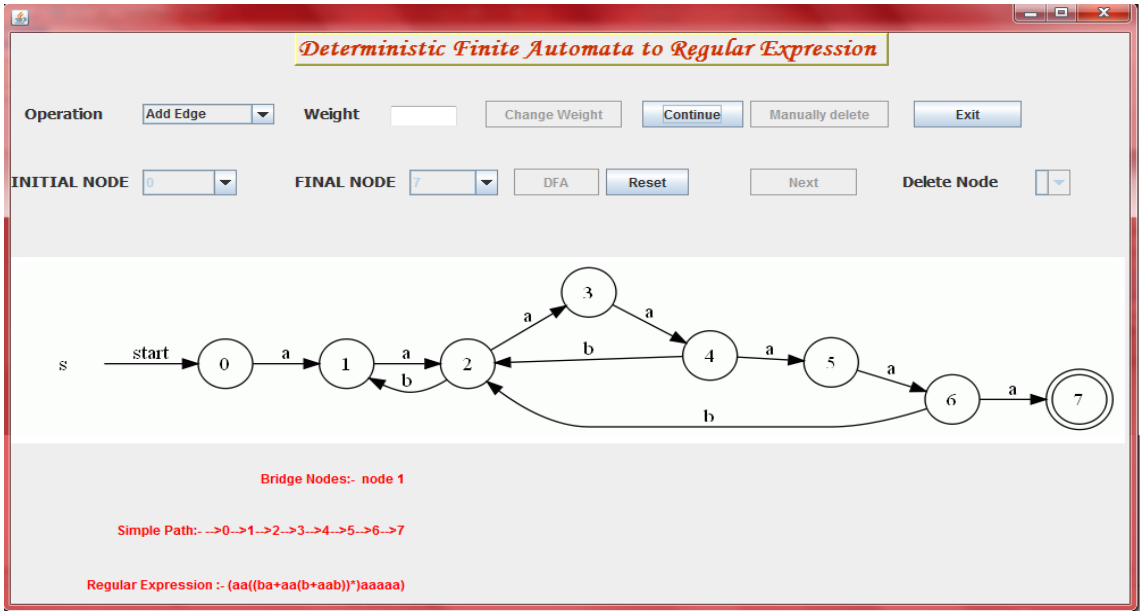


Figure 4.9: Another example of DFA with circle and its sub-circle input to software

After using this removal sequence regular expression R_1 = aa(ba+aa(b+aab))*aaaaa is obtained as shown in figure 4.11.

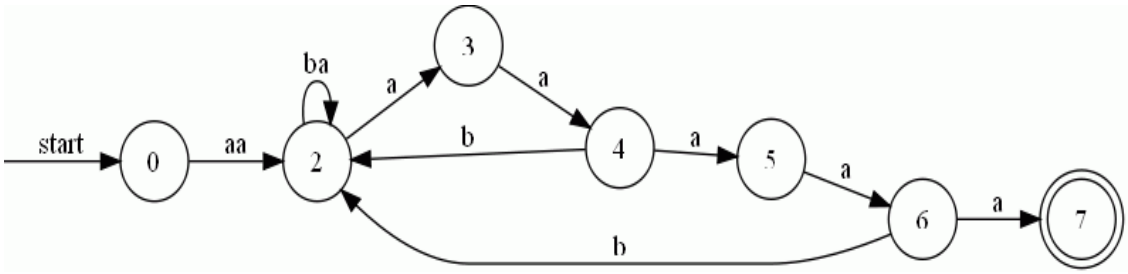


Figure 4.10: DFA after removing state 1

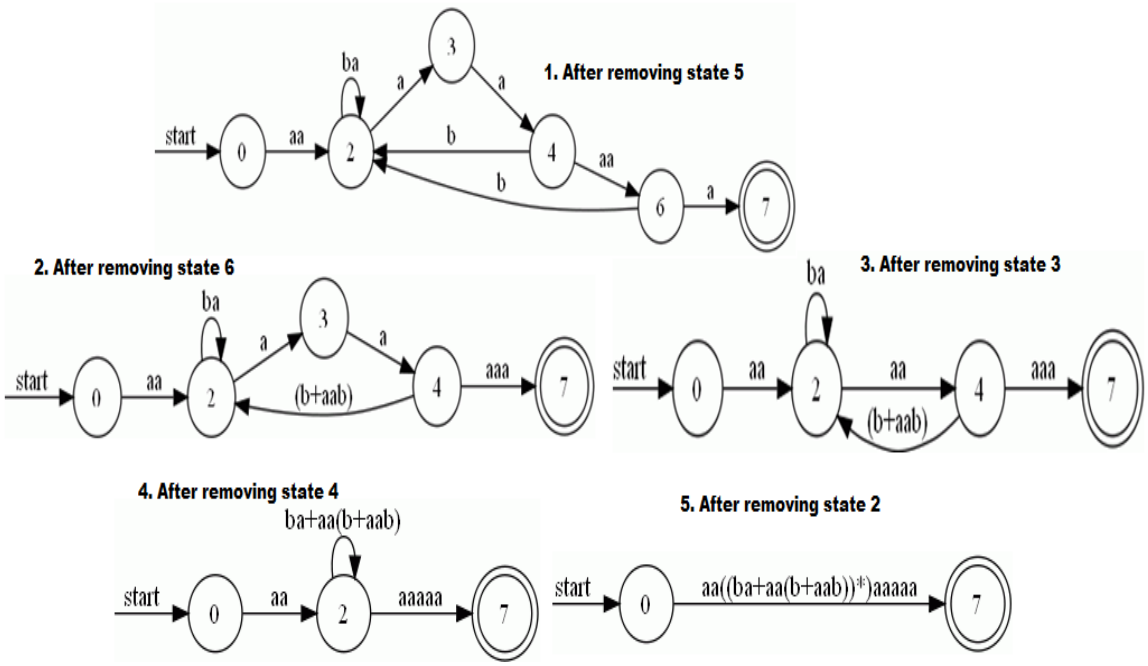


Figure 4.11: DFA after removal sequence 5→6→3→4→2.

Regular expression generated by software is same as regular expression R_1. Hence we can say for generating regular expression equivalent to input DFA, this software eliminates state of DFA according to newly proposed heuristics.

Chapter 5

Conclusions and Future work

This thesis work provides an insight into the various approaches used for conversion of deterministic finite automata to regular expression. Comparisons between different techniques for conversion of DFA to RE are carried out. New heuristics are proposed for choosing optimal removal sequence of states in state elimination method. Software is designed for conversion of DFA to RE using state elimination approaches along with newly designed heuristics for choosing optimal removal sequence of states.

5.1 Conclusions

State elimination method takes less time, and generally gives smaller regular expression as compared to Transitive closure and Brzozowski algebraic method. In state elimination method for a DFA having n states excluding starting and final state, n! removal sequences are possible and the different removal sequences of states give us different regular expression equivalent to same DFA. By choosing a good removal sequence, we can obtain a shorter regular expression. Concept of bridge state and state weight approach, have been proposed by researchers to generate smaller regular expression. Using structural properties of given DFA, smaller regular expression can be generated using optimal removal sequence of states in state elimination method.

Based on structural properties of DFA, new heuristics are proposed in this thesis work using concept of sub-circle and circle presented in the DFA. Using these new heuristics, smaller regular expression can be generated.

5.2 Future Scope

Conversion of DFA to RE is NP-complete problem. Newly designed heuristics can be validated on some real time applications. Still, there is a scope of generating smaller RE equivalent to a DFA. New heuristics or approaches can be designed to obtain smaller regular expression.

References

[1] Alfred V. Aho, "Constructing a Regular Expression from a DFA", Lecture notes in Computer Science Theory, September 27, 2010, Available at http://www.cs.columbia.edu/~aho/cs3261/lectures.

[2] Ding-Shu Du and Ker-I Ko, "Problem Solving in Automata, Languages, and Complexity", John Wiley & Sons, New York, NY, 2001.

[3] Gelade, W., Neven, F., "Succinctness of the complement and intersection of regular expressions", Symposium on Theoretical Aspects of Computer Science. Dagstuhl Seminar Proceedings, vol. 08001, pages 325–336. IBFI (2008).

[4] Gruber H. and Gulan, S. (2009), "Simplifying regular expressions: A quantitative perspective", IFIG Research Report 0904.

[5] Gruber H. and Holzer, M., "Provably shorter regular expressions from deterministic finite automata", LNCS, vol. 5257, pages 383–395. Springer, Heidelberg (2008).

[6] Gulan, S. and Fernau H., "Local elimination-strategies in automata for shorter regular expressions", In Proceedings of SOFSEM 2008, pages 46–57 (2008).

[7] H. Gruber and M. Holzer, "Finite automata, digraph connectivity, and regular expression size", In Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Iceland, July 2008. Springer.

[8] H. Gruber and J. Johannsen, "Optimal lower bounds on regular expression size using communication complexity", In Proceedings of the 11th International Conference Foundations of Software Science and Computation Structures, volume 4962 of LNCS, pages 273–286, Budapest, Hungary, March–April 2008. Springer.

[9] H. Hosoya, "Regular expression pattern matching – a simpler design", Technical Report 1397, RIMS, Kyoto University, 2003.

[10] Janusz A. Brzozowski, "Derivatives of regular expressions", J. ACM,11(4) pages 481-494, 1964.

[11] J. Brzozowski and E. McCluskey Jr., “Signal flow graph techniques for sequential circuit state diagrams”, IEEE Transactions on Electronic Computers EC-12 (1963) pages 67–76.

[12] J. J. Morais, N. Moreira, and R. Reis, “Acyclic automata with easy-to-find short regular expressions”, In 10th Conference on Implementation and Application of Automata, volume 3845 of LNCS, pages 349–350, France, June 2005. Springer.

[13] K. Ellul, B. Krawetz, J. Shallit, and M.Wang, “Regular expressions: New results and open problems”, Journal of Automata, Languages and Combinatorics, 10(4):pages 407– 437, 2005.

[14] Larkin, H., “Object oriented regular expressions”, 8th IEEE International Conference on Computer and Information Technology , vol., no., pages 491-496,8-11 July,2008

[15] McNaughton R. and Yamada H., “Regular expressions and state graphs for automata”. IEEE Transactions on Electronic Computers 9, pages 39–47 (1960)

[16] Mishra K.L.P.& N. Chandrasekaran, “Theory of Computer Science (Automata Language and. Computation)”, PHI, Second edition, 1998

[17] Moreira N. and Reis R., “Series-parallel automata and short regular expressions”, Fundamenta Informaticae, v.91 n.3-4, pages 611-629, August 2009.

[18] M. Delgado and J. Morais, “Approximation to the smallest regular expression for a given regular language”, In Proceedings of CIAA’04, Lecture Notes in Computer Science, vol. 3317, 2004, pages 312–314.

[19] M. Lesk and E. Schmidt, “Lex - A Lexical Analyzer Generator”, Computing Science Technical Report No. 39, Bell Laboratories, USA, 1975.

[20] Mulder M. and Nezlek G.S., “Creating protein sequence patterns using efficient regular expressions in bioinformatics research”, 28th International Conference 2006 , Pages 207 – 212.

[21] Nelma Moreira, Davide Nabais and Rog´erio Reis, “State Elimination Ordering Strategies: Some Experimental Results”, DCC-FC & LIACC, Universidade do Porto,R. do Campo Alegre 1021/1055, 4169-007 Porto, Portugal.

[22] Neumann Christoph, Converting Deterministic Finite Automata to Regular Expressions, Mar 16, 2005

[23] Peter Linz, "An introduction to Formal Languages and Automata", Jones and Bartlett Publishers, Sudbury, MA, third edition, 2001.

[24] R. McNaughton and H. Yamada, "Regular expressions and state graphs for automata", IEEE Transactions on Electronic Computers 9 (1960) pages 39–47.

[25] R. Tarjan, "Depth-first search and linear graph algorithms", SIAM Journal on Computing 1 (2) (1972) pages 146–160.

[26] S. C. Kleene, "Representation of events in nerve nets and finite automata". 40th Ann. of Math, Studies No. 34, Princeton University Press, Princeton, NJ, 1956.

[27] Sakarovitch, J., "The language, the expression, and the (small) automaton", CIAA 2005. LNCS, vol. 3845, pages 15-30. Springer, Heidelberg (2006).

[28] S. Saunders and T. Takaoka, "Improved shortest path algorithms for nearly acyclic graphs", Theoretical Computer Science 293 (3) (2003) pages 535–556.

[29] Tarjan R. E., "Depth First Search and Linear Graph Algorithms," SIAM J. Computing 1:2, pages 146-160, 1972.

[30] Ulman, J., A. V. Aho and R. Sethi, "Compiler Design: Principles, Tools, and Techniques", Pearson Education Inc, ISBN 0-201-10088-6,1986.

[31] Ullman, J., J. E. Hopcroft and R. Motwani, "Introduction to Automata Theory, Languages, and Computation". Pearson Education Inc, ISBN 0-201-44124-1. Addison Wesley, 2001.

[32] Will Drewry and Tavis Ormandy, Google, Inc., Insecure Context Switching: Inoculating regular expressions for survivability

[33] Yo-Sub Han and Derick Wood, Obtaining shorter regular expressions from finite-state automata , Theoretical Computer Sciecce 370(1-3) (2007): pages 110-120.

[34] Y.-S. Han and D. Wood, "The generalization of generalized automata: Expression automata", International Journal of Foundations of Computer Science 16 (3) (2005) pages 499–510.

[35] Zeiger H. P. and Ehrenfeucht A., "Complexity measures for regular expressions", Journal of Computer and Systems Sciences 12(2): pages 134–146, 1976.

List of Publications

ACCEPTED

- Kulwinder Singh and Ajay Kumar Loura, "Comparisons amongst different techniques for Conversion of Deterministic Finite Automata to Regular Expression", Paper ID 1238, International Journal of Advanced Research in Computer Science

COMMUNICATED

- Kulwinder Singh and Ajay Kumar Loura, "New heuristics for conversion of Deterministic Finite Automata to Regular Expression",