CONVERSION OF DETERMINISTIC FINITE AUTOMATA TO REGULAR EXPRESSION

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

Master of Engineering
in
Computer Science and Engineering

Submitted By
Tamanna Chhabra
Roll No: 801032030

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

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2012

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*Conversion of Deterministic finite automata to Regular expression*", 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 *Ajay Kuman 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:

(Tamanna Chhabra)

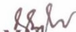
This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mr. Ajay Kumar Loura)

Assistant Professor
Computer Science and
Engineering Department
Thapar University
Patiala

Countersigned by

(Dr. Maninder Singh)
Head
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.

Tamanna Chhabra

801032030

Abstract

Regular expressions are well known in the field of computer science. They are widely used in the field of compilers, programming languages, pattern recognition, protocol conformance testing etc.

There are various methods for the conversion of deterministic finite automata to regular expression like Transitive closure method, Brzozowski Algebraic method and state elimination method. State elimination method is the most widely used approach for converting deterministic finite automata to regular expression. There are many heuristics proposed by the researchers on the basis of which state elimination gives shorter regular expression. Yo Sub Han and Derick Wood introduced the concept of bridge state, vertical chopping and horizontal chopping for smaller regular expressions. We intend to find the smallest regular expression equivalent to a given deterministic finite automata.

The thesis analyzes and compares different approaches used for conversion of DFA to RE and new heuristics are proposed for obtaining smaller regular expression corresponding to a given DFA. Researchers have proposed that removal of bridge states using state elimination method after the removal of non bridge states leads to shorter regular expression. The concept of bridge state is mathematically given but no formal algorithm is given. A formal algorithm for bridge state is proposed by which DFA can be converted into regular expression and the relation between the size of a regular expression and the size of the DFA is estimated.

Table of Contents

Certificate	i
Acknowledgement	ii
Abstract	iii
Table of Contents	iv
List of Figures and Tables	vii
Chapter 1: Introduction	1-7
1.1 Formal Language	1
1.2 Automata	2
1.2.1 Deterministic Finite Automata	2
1.2.2 Non-deterministic finite automata	3
1.3 Language accepted by a DFA	4
1.4 Operation on languages	5
1.5 Regular expression	6
1.5.1 Applications of regular expressions	6
1.6 Thesis Outline	7
Chapter 2: Conversion of DFA to RE	8-19
2.1 Conversion of DFA to RE	8
2.2 Transitive Closure Method	8
2.3 Brzozowski Algebraic Method	11
2.4 State Elimination Method	12
2.4.1 Bridge State	12
2.4.2 Vertical Chopping	13

2.4.3 Horizontal Chopping 17
Chapter 3: Problem Statement 20-23
3.1 Problem Statement 20
3.2 Objectives and Methodology 20
3.3 Motivation 20
3.4 State Elimination 21
Chapter 4: Proposed Solution 24
4.1 Comparisons of various approaches used for conversion of DFA to RE 24
4.2 Algorithm to determine bridge state 27
4.3 New heuristics for choosing optimal removal sequence 33
4.3.1 Cycle detection algorithm 33
4.4 Relation between the size of the DFA and regular expression 44
Chapter 5: Conclusions and Future Work 47
5.1 Conclusions 47
5.2 Summary of Contributions 47
5.3 Future Scope 47
Annexures
References 48-52
List of Publications 55

Chapter 3: Problem Statement 23-43

3.1 Problem Statement 24

3.2 Objectives and Methodology 24

3.3 Motivation

3.4 State Elimination

Chapter 4: Proposed Solution 44-50

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

4.2 Algorithm to determine bridge state 48

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

4.3.1 Cycle detection algorithm

4.4 Relation between the size of the dfa and regular expression

Chapter 5: Conclusions and Future Work 51

5.1 Conclusions 51

5.2 Future Scope 51

Annexures

References 52

List of Publications 55

List of Figures and Tables

Figure No.	Figure Title	Page No.
Figure 1.1	A finite automata model	1
Figure 1.2	A finite automata for recognition of string “abab”	2
Figure 1.3	Deterministic finite automata corresponding to table 1.1	2
Figure 1.4	Non deterministic finite automata corresponding to table 1.2	4
Figure 1.5	Null language DFA	4
Figure 1.6	Another example of language accepted by DFA	5
Figure 2.1	Example of transitive closure	8
Figure 2.2	DFA for finding RE using transitive closure	9
Figure 2.3	DFA for the language having odd number of a’s	10
Figure 2.4	DFA accepting all the strings with an odd no of 1‟s	11
Figure 2.5	Example of state elimination	12
Figure 2.6	Elimination of state 1	12
Figure 2.7	Example of bridge state	13
Figure 2.8	Example of vertical chopping	14
Figure 2.9	An example of vertical chopping of DFA at bridge state 7	14
Figure 2.10	Vertical chopping of sub automata A_1 at the bridge state 1	15
Figure 2.11	After removing state 5	15
Figure 2.12	After removing state 4	16
Figure 2.13	After removing state 6	16
Figure 2.14	After removing state 3	16
Figure 2.15	After removing state 2	17
Figure 2.16	Sub-automata A_2	17
Figure 2.17	After removing state 8	17
Figure 2.18	Sub Automata A_U	18
Figure 2.19	Sub Automata A_L	18
Figure 2.20	After removing state 5	18
Figure 2.21	After removing state 2	18
Figure 2.22	After removing state 4	19

Figure 2.23	After removing state 6	19
Figure 2.24	After removing state 3	19
Figure 3.1	Example of state elimination	21
Figure 3.2	Another example of state elimination	22
Figure 3.3	DFA with starting state q0 and final state q4	22
Figure 3.4	After removing state q1	22
Figure 3.5	After removing state q2	22
Figure 3.6	After removing state q3	23
Figure 3.7	After removing state q3	23
Figure 3.8	After removing state q1	23
Figure 3.9	After removing state q2	23
Figure 4.1	DFA accepting all the strings having odd number of 1's	24
Figure 4.2	RE obtained by state elimination	25
Figure 4.3	DFA accepting all the strings having sub-string "bb"	26
Figure 4.4	After removing state	27
Figure 4.5	DFA with q0 as the start state and q6 as the final state	31
Figure 4.6	Bridge state in DFA	32
Figure 4.7	After removing state q2	32
Figure 4.8	After removing state q3	32
Figure 4.9	After removing state q1	32
Figure 4.10	After removing state q4	33
Figure 4.11	After removing state q1	33
Figure 4.12	After removing state q4	33
Figure 4.13	After removing state q2	34
Figure 4.14	After removing state q3	34
Figure 4.15	Example of DFA where all the states are bridge states	34
Figure 4.16(a)	After removing state q1	34
Figure 4.16(b)	After removing state q2	35
Figure 4.16	Removal of the states in the order q1-q2	35
Figure 4.17(a)	After removing state q2	35
Figure 4.17(b)	After removing state q1	35

Figure 4.17	Removal of the states in the order q2-q1	35
Figure 4.18	Example of circles in DFA	37
Figure 4.19	DFA with two circles	37
Figure 4.20	After removing state state 4	38
Figure 4.21	After removing state 2	38
Figure 4.22	After removing state 3	38
Figure 4.23	After removing state 1	38
Figure 4.24(a)	After removing state 2	39
Figure 4.24(b)	After removing state 3	39
Figure 4.24(c)	After removing state 4	39
Figure 4.24(d)	After removing state 1	39
Figure 4.24	Removing the states in the order 2,3,4 and 1	39
Figure 4.25	DFA with two circles, one circle and one self loop	40
Figure 4.26(a)	After removing state 3	40
Figure 4.26(b)	After removing state 2	40
Figure 4.26(c)	After removing state 1	40
Figure 4.26	Removing the states in the order 3,2 and 1	40
Figure 4.27(a)	After removing state 1	40
Figure 4.27(b)	After removing state 2	41
Figure 4.27(c)	After removing state 3	41
Figure 4.27	Removing the states in the order 1,2 and 3	41
Figure 4.28	DFA having three circles	41
Figure 4.29(a)	After removing state 3	41
Figure 4.29(b)	After removing state 5	42
Figure 4.29(c)	After removing state 2	42
Figure 4.29(d)	After removing state 1	42
Figure 4.29	Removing the states in the order 3,5,2 and 1	42
Figure 4.30(a)	After removing state 1	43
Figure 4.30(b)	After removing state 5	43
Figure 4.30(c)	After removing state 2	43
Figure 4.30(d)	After removing state 3	43

Figure 4.30	Removing the states in the order 1,5 2 and 3	43
Figure 4.31	Another example of DFA with three circles	44
Figure 4.32(a)	After removing state q1	44
Figure 4.32(b)	After removing state q2	44
Figure 4.32(c)	After removing state q3	44
Figure 4.32	Removing the states in the order q1,q2 and q3	44
Figure 4.33(a)	After removing state q3	45
Figure 4.33(b)	After removing state q2	45
Figure 4.33(c)	After removing state q1	45
Figure 4.33	Removing the states in the order q3,q2 and q1	45
Figure 4.34	DFA with three self loops	45
Figure 4.35(a)	After removing state q1	46
Figure 4.35(b)	After removing state q2	46
Figure 4.35(c)	After removing state q3	46
Figure 4.35	Removing the states in the order q1,q2 and q3	46
Figure 4.36(a)	After removing state q3	46
Figure 4.36(b)	After removing state q2	46
Figure 4.36(c)	After removing state q1	46
Figure 4.36	Removing the states in the order q3,q2 and q1	46
Figure 4.37	DFA accepting all the strings having odd number of a's	47
Figure 4.38	DFA with eight states	47
Figure 4.39	DFA accepting all the strings starting with aa	48
Figure 4.40	DFA accepting all the strings with substring aa	48
Figure 4.41	DFA with four states	48

Table No.	**Table Title**	**Page No.**
Table 1.1	Transition table representing transition function of DFA	3
Table 1.2	Transition table representing transition function of DFA	4

List of Keywords

FA	Finite Automata
DFA	Deterministic Finite Automata
NFA	Non-deterministic Finite Automata
RE	Regular Expression

Chapter 1
Introduction

This chapter gives an introduction to formal languages, automata and regular expressions.

1.1 Formal Language

Finite non-empty set of symbols is called the alphabet [1]. Finite sequence of symbols from the alphabet is called a string or a word. For example if alphabet Σ ={a,b}, then ababbb and aaababb are strings on Σ .A language [1] is defined as a subset of Σ^*. Empty string and null language are denoted by ϵ and $\emptyset$ respectively. Formal languages can be classified as regular, context free, context sensitive and recursive languages. For instance a language over 0 and 1 that will include all strings having length less than or equal to 2 is L={ ϵ,0,1,00,01,10,11}.

1.2 Automata

An automata is an abstract model of a digital computer [1]. It consists of a finite set of states. It also has a start state and a set of final states. An automata runs on given sequence of inputs in discrete time. At each step, an automata gets one input symbol from finite set of symbol called an alphabet. The symbols which are already fed to an automata form a sequence called a word. At each step the automata is in one of its states and when it reads a symbol, it jumps or transits to a next state that is decided by a function that takes current state and the symbol currently read as parameters. This function is called transition function. The automata reads the symbols of the input word one after another and transits from state to state according to the transition function until the word is read completely. When it has read the complete input word, it stops. If the state at which the automata stops is one of the final states then the word is accepted by the automata otherwise it is rejected. The set of all the words accepted by an automata is called the language recognized by the automata.

Figure 1.1: A finite automata model

Figure 1.1 illustrates a finite automata model. This automata consists of states (represented by circles), and transitions (represented by edges) and $\Sigma = \{a\}$.q_0 is the initial state and q_1 is the final state denoted by double circle. When the automata reads the input symbol a, while in q0 state, it makes a transition to state q1.When state q1 reads a symbol a, it remains at q1 only. Due to self loop on q1, it can read the input alphabet a multiple times.

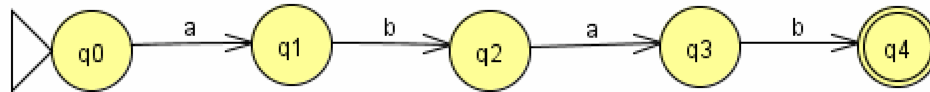


Figure 1.2: A finite automata for recognition of string "abab"

The automata in the figure 1.2 has five states and q0 is the initial state and q4 is the final state. It accepts the string "abab".

Finite automata can be classified into deterministic finite automata and non-deterministic finite automata.

1.2.1 Deterministic finite automata

A deterministic finite automata (DFA)[16,23] is a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$, consisting of

a finite set of states(Q)

a finite set of input symbols called the alphabet (Σ)

a transition function ($\delta : Q \times \Sigma \rightarrow Q$)

a start state ($q_0 \in Q$)

a set of final states ($F \subseteq Q$)

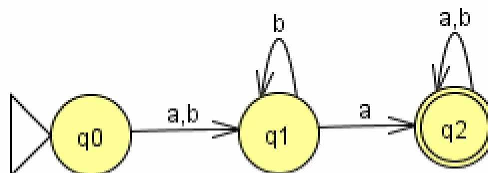


Figure 1.3: Deterministic finite automata corresponding to table 1.1

Example 1.1: The DFA(in figure 1.3) can be represented by ({q0, q1, q2}, {a,b}, δ, {q0}, {q2}) where δ is shown in the table 1.1.

Table 1.1: Transition Table representing transition function of DFA**.**

State(q)	Input symbol	Next State(δ)
q0	a	q1
q0	b	q1
q1	a	q2
q1	b	q1
q2	a	q2
q2	b	q2

The transition function takes input symbol and the next state as the input parameters.Depending on the input symbol and the next state it decides the next state of the automata.

1.2.2 Non-deterministic finite automata

Non determinism means a choice of moves for an automata. Rather than moving in a unique direction at each step, it allows a set of possible moves. A non-deterministic finite automata (NFA) is same as DFA except the transition function. Transition function of a NFA is defined by $Q \times \Sigma \rightarrow 2^Q$. A non-deterministic finite automata (NFA) [16, 23] is a finite state automata where for each pair of state and input symbol, ther e may be more than one next states. A non-deterministic finite automata (NFA)[16,23] is a 5-tuple, (Q, Σ, δ, q_0, F), consisting of

1. A finite set of states Q
2. A finite set of input symbols Σ
3. A transition function $\delta : Q \times \{\Sigma \cup \epsilon\} \rightarrow 2^Q$
4. An initial (or start) state $q_0 \in Q$
5. A set of final states F ($F \subseteq Q$)

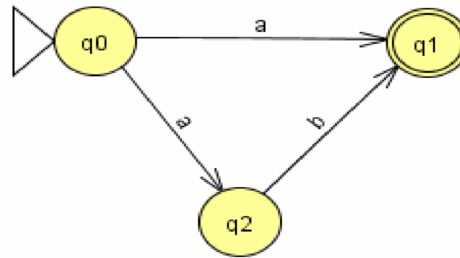


Figure 1.4: Non deterministic finite automata corresponding to table 1.2

Example1.2: The NFA (in figure 1.4) is represented by ({q0, q1, q2}, {a,b}, δ, {q0}, {q1}) where δ is shown in the table 1.2.

Table 1.2: Transition Table representing transition function of NFA

State(q)	Input	Next State(δ)
q0	a	{q1,q2}
q0	b	∅
q1	a	∅
q1	b	{q2}
q2	a	∅
q2	b	∅

1.3 Language accepted by a DFA:

A language L is accepted by a DFA $(Q, \Sigma, \delta, q_0, F)$, [23] if and only if $L = \{ w \mid \delta^*(q0, w) \in F \}$ .The language accepted by a DFA is the set of all the strings accepted by the DFA. The language accepted by a DFA is a regular language and can be represented by a regular expression.

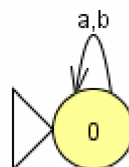


Figure 1.5: Null language DFA

The DFA in figure 1.5 does not accept any string because it has no accepting state. Thus the language it accepts is the empty set ∅ .

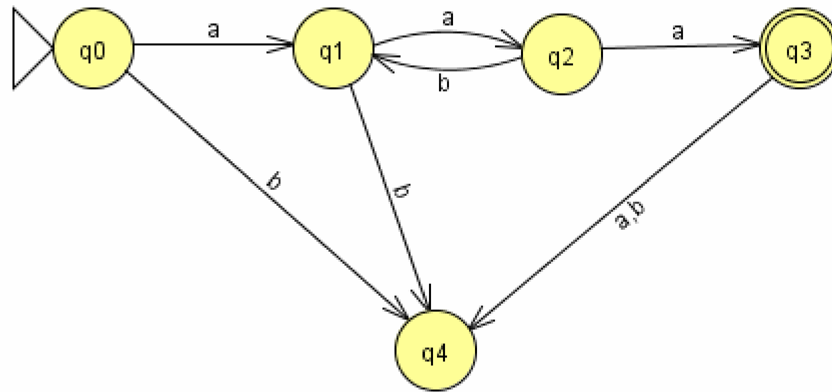


Figure 1.6: Another example of language accepted by DFA

The DFA in the figure 1.6 has a circle 1 - 2 - 1 and it can go through this circle any number of times by reading substring ab repeatedly. To find the language accepted by the DFA first from the initial state go to state 1 by reading a. Then from state 1 go through the circle 1 - 2 - 1 any number of times by reading substring ab any number of times to come back to state 1. This is represented by $(ab)^*$. Then from state 1 go to state 2 and then to state 3 by reading aa. Thus the language accepted by this DFA is $a(ab)^*aa$. In this DFA state 4 is a trap state because no other state in the DFA is reachable from state 4.

1.4 Operations on languages

Following are the operations that can be performed on languages.[3,23,31]

1. Union: Union of two languages L1 and L2 is a set of all the strings that are either in L1 or L2 or both. For example, if L1={001,10,111} and L2={ϵ,01}, then L1UL2={ϵ,01,001,10,111}.
2. Concatenation: Concatenation of two languages L1 and L2 is a set of all the strings that are formed by taking any string in L1 and concatenating it with any string in L2. For example, if L1={001,10,111} and L2={ϵ,01}, then L1.L2={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 the set of all the strings of 0's and 1's i.e. L* ={ϵ,0,1,01,001,0110,111010................}.

1.5 Regular Expression

A regular expression (RE) [31] is a pattern that describes a set of strings. Regular expressions denote regular languages and consists of strings of a particular type. The pattern of strings described by regular expression are same as described by finite automata. Every formal language can be represented by a finite automata or a regular expression. Regular expression over a language can be defined as:

1. Regular expression for each alphabet is represented by itself. The empty string (ϵ) and null language (∅) are regular expression denoting the language ϵ and ∅ 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 is denoted by regular expression E+F and represents language L(E) U L(F).
    b) Concatenation of E and F is denoted by EF and represents language L(E*F) = L(E) * L(F).
    c) Kleene closure is denoted by E* and represents language (L(E))*.
3. Any regular expression can be formed using 1-2 rules.

1.5.1 Applications of regular expressions

Regular expressions have a wide variety of applications [16,30]. They are used in web search engines, in software engineering, lexical analysis, databases etc.

1. Lexical analyzers: The tokens of the programming language can be expressed using regular expressions. The lexical analyzer scans the input program and separates the tokens. For example, identifier can be expressed as a regular expression :(letter)(letter+digit)* .If anything in the source language matches with this regular expression then it is recognized as an identifier. The letter in RE is {A,B,C,………..Z,a,b,c….z} and digit is {0,1,…9}.Thus regular expression identifies tokens in a language.

2. Text editors: These are the programs used for processing the text. In UNIX text editors any regular expression is converted to a NFA with ϵ transitions and this NFA can be then simulated directly.

1.6 Thesis Outline

Thesis is organized into 5 chapters. Chapter 1 describes automata, regular expression and languages. Chapter 2 describes different approaches used for conversion of deterministic finite automata to regular expression. Chapter 3 discusses the problem statement, objectives and methodology, motivation behind the thesis and the method of state elimination. Chapter 4 compares different approaches used for the conversion of deterministic finite automata to regular expression. It also discusses the algorithm of bridge state and the relation between the size of the DFA and regular expression. Chapter 5 summarizes the conclusions drawn in the thesis along with the future directions.

Chapter 2
Conversion of DFA to RE

This chapter describes different techniques used for conversion of deterministic finite automata to regular expression. In state elimination method concepts like bridge state, horizontal chopping and vertical chopping are used to generate smaller regular expression from deterministic finite automata.

2.1 Conversion of DFA to RE

Kleene proved that every RE has an equivalent DFA and vice versa[22].Conversion of DFA to regular expression are carried out using following techniques.[2, 10,22]

1. Transitive closure method
2. Brzozowski Algebraic method
3. State elimination method

2.2 Transitive Closure Method

The transitive closure method [2,10] was proposed by Kleene.The DFA given in the figure 2.1 can be represented as a regular expression.

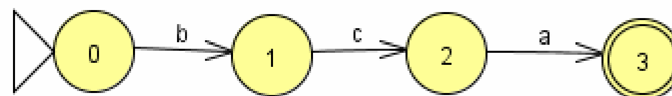


Figure 2.1: Example of transitive closure

The input for edge in the DFA is a regular expression. The regular expression for the transition from state 0 to 1 is b, from 1 to 2 is c and 2 to 3 is a. The regular expression representing the transition from state 0 to 2 is the concatenation of the regular expressions from 0 to 2 and RE bc is formed. The regular expression for the DFA is the concatenation of all the regular expressions from the starting state 0 to the final state 3 and RE bca is formed. For a path from q_a to q_f, the concatenation of the regular expression for each transition in the path forms a regular expression that represents the same string as the path from q_a to q_f in the DFA.

Consider transitive closure for figure 2.2

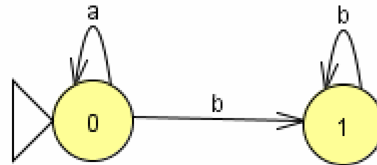


Figure 2.2 : DFA for finding RE using transitive closure

In figure 2.2 multiple paths exist between the two states and therefore a simple regular expression cannot be constructed to represent the DFA. The RE can be built using other operators such as union and iteration. Kleene's transitive closure[10] can be used in converting such DFA to RE. In this method we use R_{ij}^{k} which denotes the set of all the strings in Σ^* that takes the DFA from the state q_i to q_j without entering or leaving any state higher than q_k.

Let regular expression R_{ij} represent 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}^{m}.

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

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

R_{ij}^{k} is recursively defined as $R_{ij}^{k}= R_{ij}^{k-1(}R_{ij}^{k-1)*}\, R_{ij}^{k-1}+ R_{ij}^{k-1}$

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

Kleene gave an algorithm for transitive closure method.

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] = ∅ ;

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] = ∅;

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

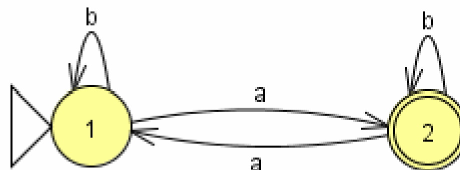


Figure 2.3: DFA for the language having odd number of a's

Applying transitive closure approach on deterministic finite automata shown in figure 2.3:

$r^0_{11} = b + \epsilon \quad r^0_{22} = b + \epsilon \quad r^0_{12} = a \quad r^0_{21} = a$

$r^1_{21} = r^0_{21} + r^0_{11} (r^0_{11})^* r^0_{21} = a + (b+ \epsilon) + a$

$r^1_{22}= r^0_{22}+ r^0_{21} (r^0_{11})^* r^0_{21} = (b+\epsilon) + a(b+\epsilon)^*a$

$r^2_{12}= r^1_{12} + r^1_{12} (r^1_{22})^* r^1_{22} = (b+\epsilon)^*a(b+\epsilon+ab^*a)$

$r^2_{12} = (b)^*a(b+ ab^*a)^*$

Regular expression obtained using transitive closure approach corresponding to deterministic finite automata shown in figure 2.3 is (b)*a(b+ab*a)*.

2.3 Brzozowski Algebraic Method

Brzozowski method [10,22] takes a unique approach to generate regular expression. A system of regular expressions for each state in the DFA M is formed. We solve the equations for R_a where R_a is the regular expression associated with the starting state q_a. For each state q_t in M the equation for R_t is a union of terms. This leads to a system of equations in the form of :

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

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

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

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

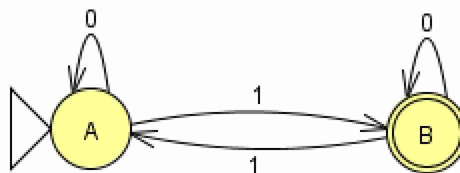


Figure 2.4: DFA accepting all the strings having odd number of 1's

Applying Brzozowski method on deterministic finite automata sh,own in figure 2.4.The equations generated by this method are as follows:

$A = 0A + 1B$

$B = 1A + 0B + \epsilon$

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

Using Arden's rule $B=0^* (1A + \epsilon)$

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

Using (1) we obtain

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

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

Regular expression obtained using Brzozowski method corresponding to deterministic finite automata shown in figure 2.4 is $(0 + 10^*1)^* (10^*)$.

2.4 State Elimination Method

State elimination came into use in 1960s by Brzozowski and McCluskey Jr. [22] and was formulated by Wood [2].In this method we keep removing states of the DFA except the start and final state while maintaining the transition information of the DFA until there are no more states to be eliminated. State elimination is shown in Fig 2.5.

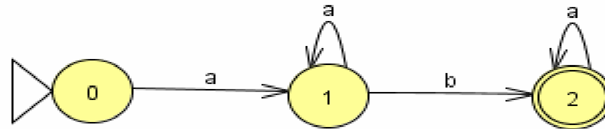


Figure 2.5: Example of state elimination

After state elimination

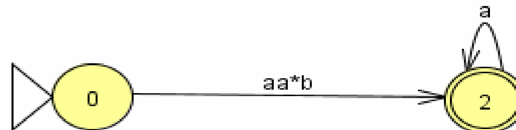


Figure 2.6: Elimination of state 1

After the state elimination we get the regular expression as aa*ba*.

For n-state deterministic finite automata excluding start and accepting state, n! removal sequences are there. It is very difficult to try all the possible removal sequence for smaller regular expression. Instead, researchers have introduced new heuristics using structural properties of deterministic finite automata for state elimination method that can give smaller regular expression equivalent to DFA. Gruber and Holzer [5] proposed graph separator techniques for obtaining shorter regular expression. Some heuristics for state elimination method run in exponential time. We intend to find smaller regular expression from deterministic finite automata quickly, so only polynomial running time heuristic are considered for implementation. Concept of bridge state, vertical chopping, horizontal chopping under state elimination given by Han and Wood [15] run in polynomial time.

2.4.1 Bridge State

Han and Wood[32] suggested horizontal decomposition and vertical decomposition based on the structural properties of a given DFA. First, the vertical decomposition is used since it always guarantee the shortest regular expression by state elimination and then if

possible horizontal decomposition is used. For the vertical decomposition first bridge states are identified.

A state q in a DFA A is said to be a bridge state [33] if it satisfies the following conditions:

1. State q is neither a start nor a final state.
2. For each string w ∊ L(A), its path in A must pass through q at least once.
3. State q is not in any circle except for the self-loop.

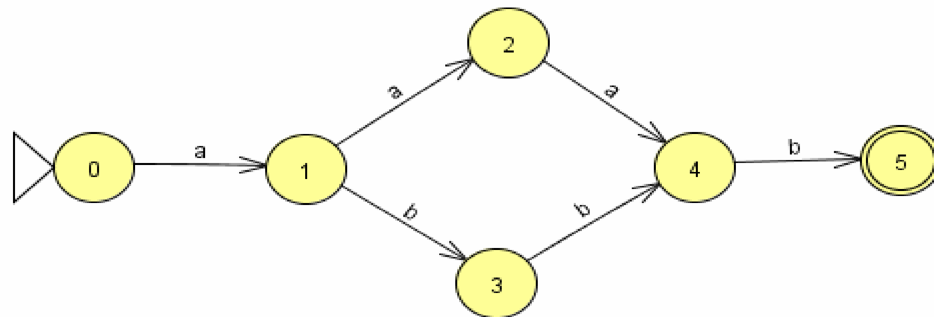


Figure 2.7: Example of bridge state

In the figure 2.7 state 1 and 4 satisfy all the conditions of bridge state introduced by Han and Wood. So state 1 and 4 are the bridge states. It is proved by the researchers that removal of non bridge states before the bridge states gives us smaller regular expression and if all the states in a DFA are bridge states, then any removal sequence gives the same regular expression.

2.4.2 Vertical chopping

In vertical chopping [21], DFA A is decomposed into two sub automata A_1 and A_2 such that $L(A) = L(A_1) \cdot L(A_2)$ .The decomposition is done at the bridge state. We perform vertical chopping on the DFA shown in figure 2.8. State 1 and 7 are the bridge states and first it is chopped vertically at the bridge state 7. After vertical chopping, we get two sub automata A_1 and A_2 as shown in the figure 2.9.

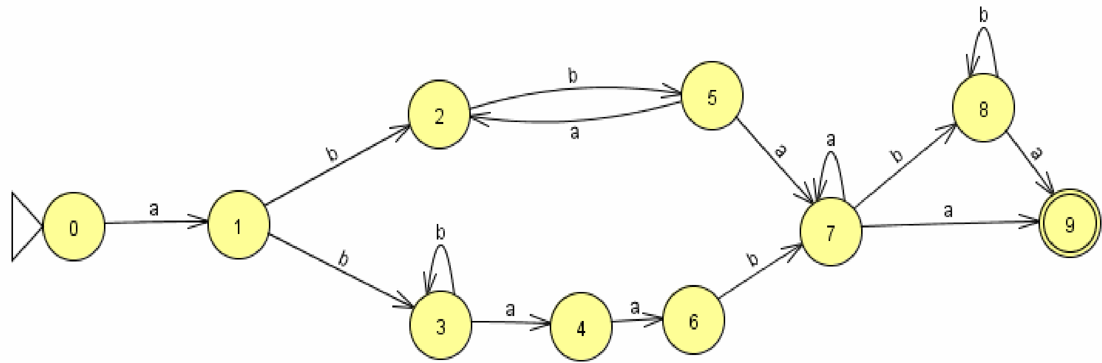
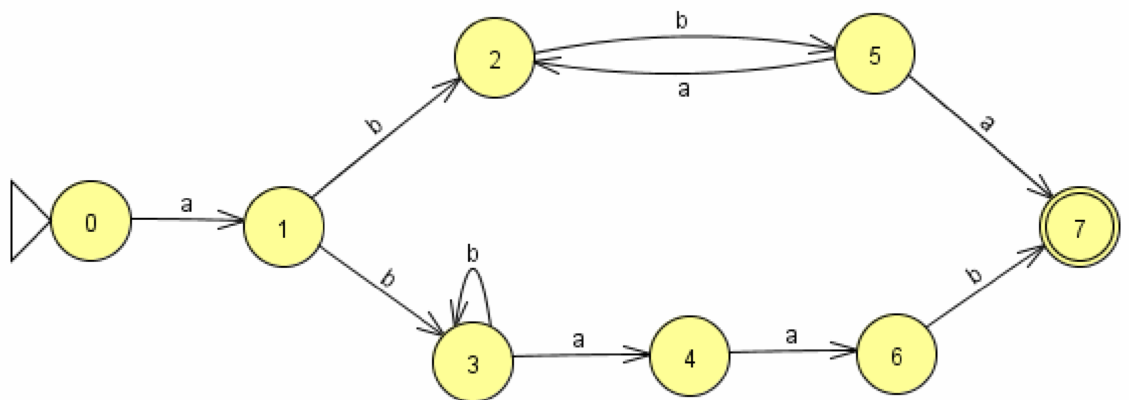
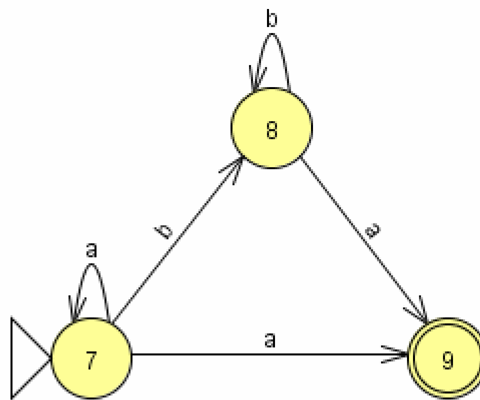


Fig 2.8: Example of vertical chopping



Sub automata A_1

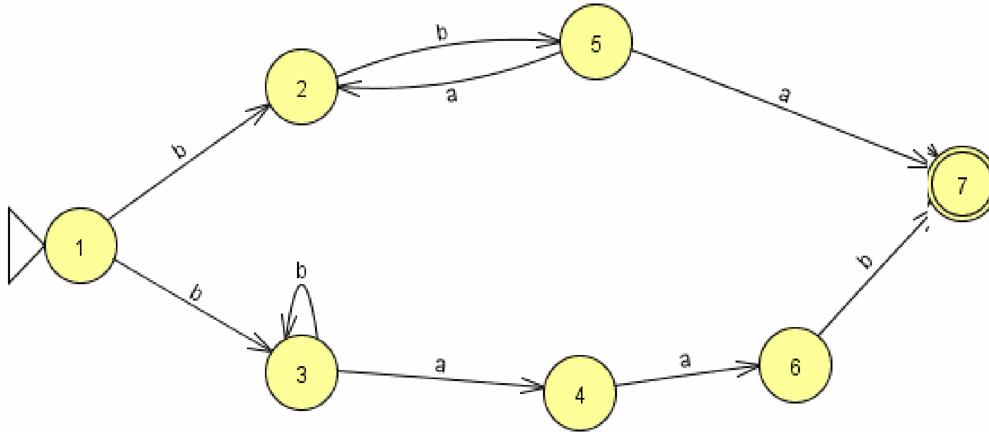


Sub automata A_2

Figure 2.9: An example of vertical chopping of DFA at bridge state 7

In sub automata A_1 state 1 is the bridge state and again the chopping can be done vertically at the bridge state 1 and we get two sub automata AA_1 and AA_2.

Sub automata AA_1



Sub automata AA_2

Figure 2.10: Vertical chopping of sub automata A_1 at the bridge state 1

Regular expression corresponding to sub-automata AA_1 is a. In sub-automata AA_2, no state satisfies the conditions of bridge state. To form the regular expression of AA_2 states of sub automata are removed in the sequence 5-4-6-3-2 as shown in the figure 2.11 to 2.15.

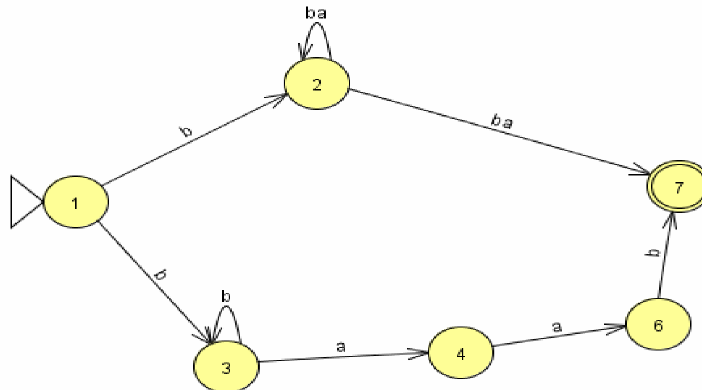


Figure 2.11: After removing state 5

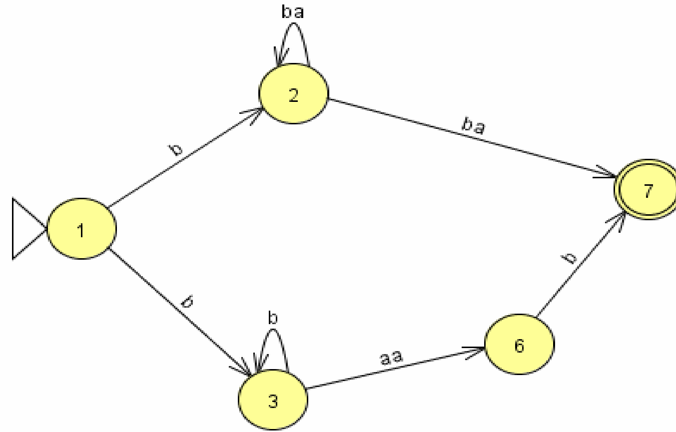


Figure 2.12: After removing state 4

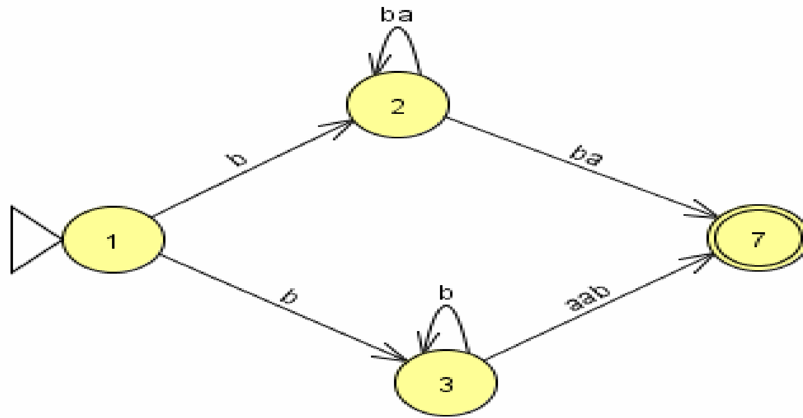


Figure 2.13: After removing state 6

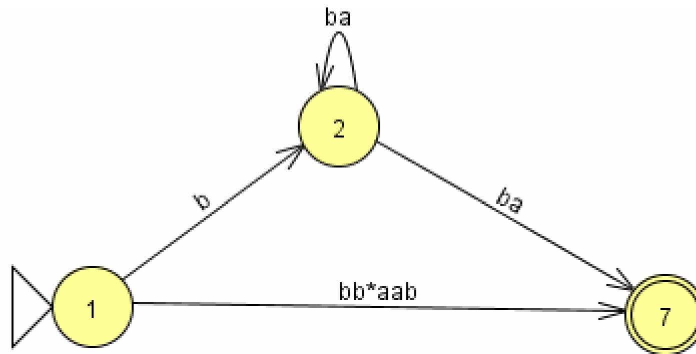


Figure 2.14: After removing state 3

Figure 2.15: After removing state 2

RE of AA_2=(b(ba) *ba+bb*aab) *.Combining the regular expression of AA_1 and AA_2 we get RE of sub automata A_1=a(b(ba) *ba+bb*aab) *.

In sub automata A_2 no state satisfies the condition of bridge state. The RE is formed by the state elimination method.

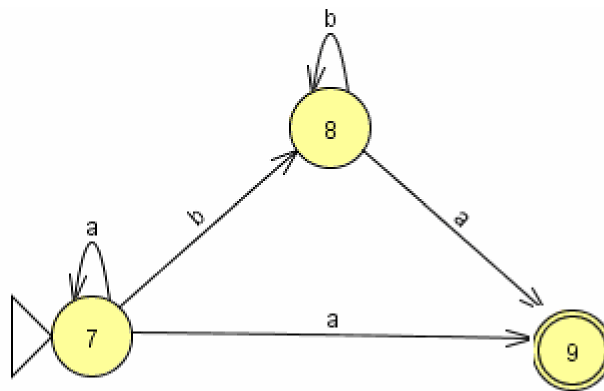


Figure 2.16: Sub automata A_2

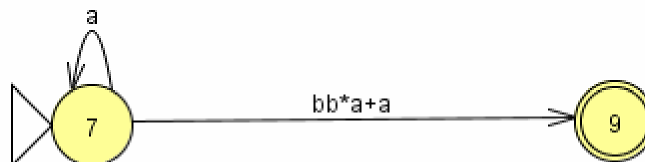


Figure 2.17: After removing state 8

RE of A_2=a* (bb*a+a)

The regular expression in case of vertical chopping is L(A)=L(A_1).L(A_2).We get the regular expression as a(b(ba) *ba+bb*aab)a* (bb*a+a)

2.4.3 Horizontal chopping

Horizontal chopping is also used in state elimination method for obtaining smaller regular expression. In horizontal chopping [32,21], DFA A is decomposed into two sub-automata A_U and A_L horizontally such that L(A) = L(A_U) + L(A_L). Deterministic finite automata can be decomposed into several horizontally disjoint sub automata and regular expression

can be obtained by taking union of corresponding regular expressions obtained from all the sub-automata. After applying vertical decomposition if we have a DFA A without any bridge states then horizontal chopping can be applied. For example, we can partition sub automata AA_2, shown in Figure 2.10 into two sub automata A_U and A_L . We can compute corresponding regular expressions e_u and e_l for A_U and A_L respectively. Then regular expression for A is $e_u + e_l$.

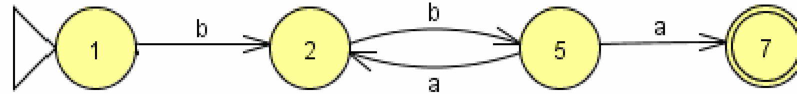


Figure 2.18: Sub Automata A_U

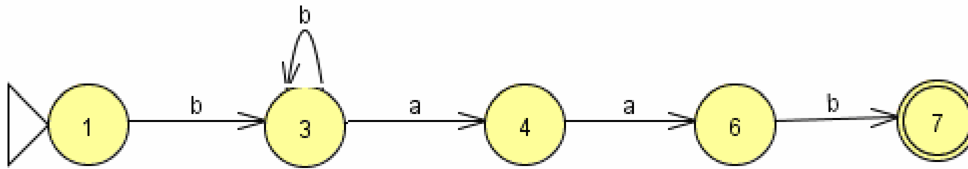


Figure 2.19: Sub automata A_L

Regular expression corresponding to sub-automata AA_2 shown in figure 2.10 can be obtained by taking union of regular expressions of sub-automata A_U and A_L. Removal sequences 5-2 in sub automata A_U gives regular expression b(ba)*ba.

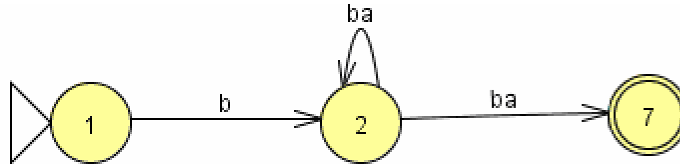


Figure 2.20: After removing state 5

Figure 2.21: After removing state 2

RE of A_u is $e_u=b(ba)^*ba$

In sub automata A_L,all the states are bridge state, so we can remove the states in any order and the order 4-6-3 gives the regular expression bb^*aab. Chopping can be continued

until no further chopping is possible and after this optimal removal sequence for sub automata is found.

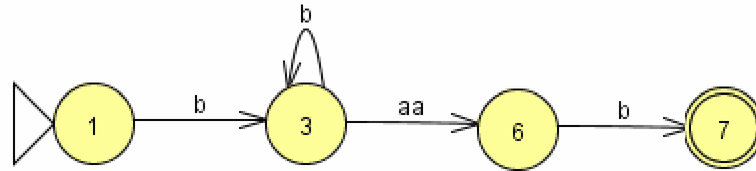


Figure 2.22: After removing state 4

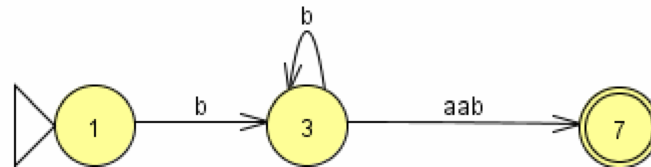


Figure 2.23: After removing state 6

Figure 2.24: After removing state 3

RE of A_L is $e_l = bb^*aab$

So the final regular expression of the sub automate $AA_2 = b(ba)^*ba + bb^*aab$

This approach is also called a divide-and-rule approach. Since we partition a DFA horizontally, it is called horizontal chopping.Once we partition a DFA A horizontally, some states become bridge states of sub automata. For example, state 2 is a bridge state of Au and states 3, 4 and 6 are bridge states of A_L in Fig 2.18 and 2.19 respectively. These states are not bridge states of the original DFA A. We can compute bridge states for each sub automata and perform vertical chopping if possible and then again we can repeat horizontal chopping. We continue chopping until no further chopping is possible and then compute a removal sequence.

Chapter 3
Problem Statement

This chapter includes the problem statement followed by the motivation behind the thesis and state elimination method.

3.1 Problem Statement

State elimination method is an efficient approach as compared to other approaches and smaller regular expression can be obtained corresponding to a given deterministic finite automata. Researchers have proposed that removal of bridge states using state elimination method after the removal of non bridge states leads to shorter regular expression. The concept of bridge state is mathematically given but there is no formal algorithm for it. Our purpose is to propose a 'formal algorithm for finding bridge state of the DFA' which is useful in converting DFA into regular expression. Size relationship between DFA and regular expression is explained. New heuristics are designed which are useful in generating smaller regular expression from DFA.

3.2 Objectives and Methodology

Following are the objectives of the thesis work:

1. Analysis and comparison of different approaches used for conversion of DFA to RE.
2. To propose a formal algorithm for finding bridge state which are useful in generating smaller regular expression from DFA.
3. To design new heuristics for obtaining smaller regular expression corresponding to a given DFA.
4. Estimating the relation between the size of a regular expression and the size of the DFA.

3.3 Motivation

There are various methods for conversion of DFA to regular expression. Transitive closure approach is proposed by Kleene [26]. Brzozowski algebraic approach [10] is a recursive approach and it uses the Arden's theorem [31] to generate regular expression from DFA. State elimination method [2, 23] is the most widely used approach for conversion

of deterministic finite automata to regular expression. In state elimination method, states of the DFA are removed one by one except the starting and final state and finally regular expression is generated corresponding to given deterministic finite automata. Although state elimination method is an intuitive method for computing regular expressions from DFA, the resulting regular expressions are sometimes very long and complicated. There are many heuristics proposed by the researchers on the basis of which state elimination gives shorter regular expression. Yo Sub Han and Derick Wood [32] introduced the concept of bridge state, vertical chopping and horizontal chopping. 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. Approaches such as vertical chopping based on bridge states and horizontal chopping based on the structural properties of DFA gives shorter regular expression. In this thesis, we have proposed a formal algorithm for finding bridge state. New.heuristics are proposed that lead to shorter regular expressions from DFA. Relationship between the size of regular expression and the size of DFA is also discussed.

3.4 State Elimination

In state elimination [5,22] states of the DFA are eliminated and the edges are replaced with regular expressions that includes the behavior of the eliminated states. At the end we get a DFA with a start and final node. The advantage of this technique over other methods is that it is easier to visualize. This technique was described by Du and Ko [2], but a much simpler approach was given by Linz [23].

In state elimination method, after eliminating the states from the original DFA if we get a DFA with different start and final state, then the DFA will look like the following:

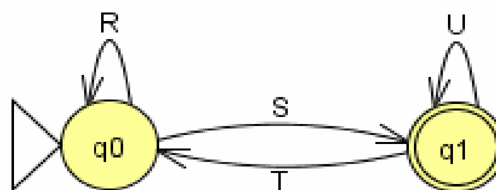


Figure 3.1: Example of state elimination

The DFA in the figure 3.1 is described as: (R | SU*T)*SU*

But if after state elimination, we obtain a single state that is starting as well as the final state,then the DFA will look like figure 3.2:

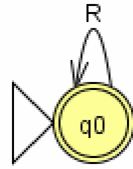


Figure 3.2: Another example of state elimination

The DFA in the figure 3.2 can be described as R^*.

If there are n accepting states of the DFA, the above steps are repeated for each accepting state to get n different regular expressions R_1, R_2, … R_n. The regular expression for the DFA is the union of n regular expressions R_1 U R_2… U R_n. The problem with state elimination is that different removal sequences give different regular expressions for the same language. In state elimination method, we can obtain smaller regular expression by choosing a better removal sequence as shown in figure 3.3.

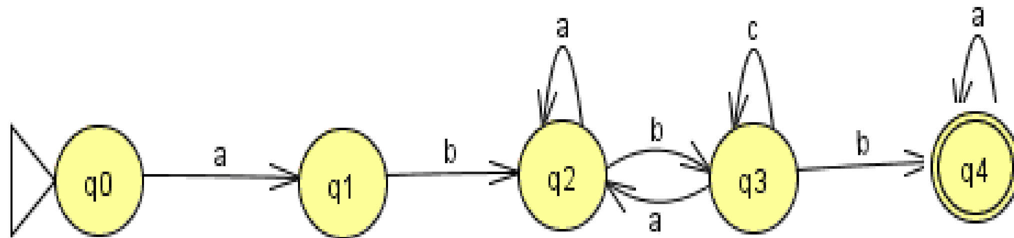


Figure 3.3: DFA with starting state q0 and final state q4

Case 1: We use the removal sequence q1-q2-q3.

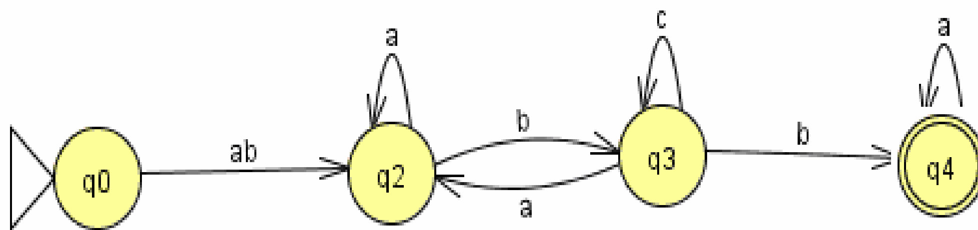


Figure 3.4: After removing state q1

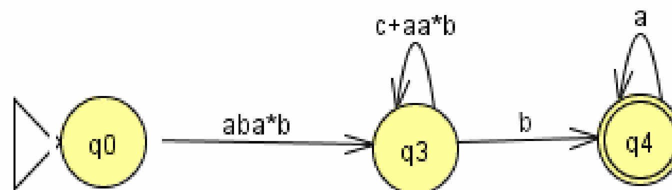


Figure 3.5: After removing state q2

Figure 3.6: After removing state q3

RE1=aba*b(c+aa*b)*ba*

Case 2: Removal sequence q3-q1-q2

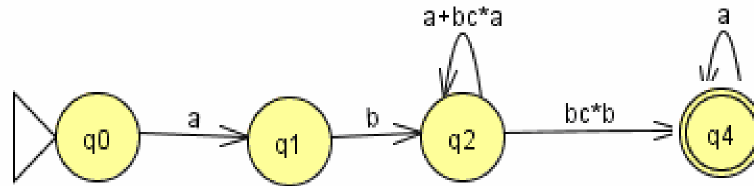


Figure 3.7: After removing state q3

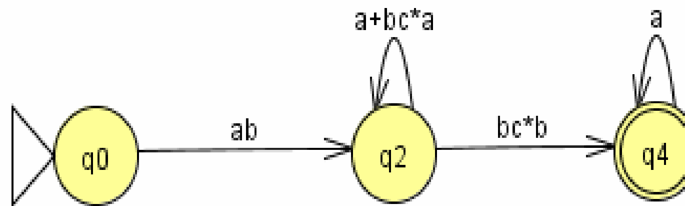


Figure 3.8: After removing state q1

Figure 3.9: After removing state q2

RE2=ab(a+bc*a)*bc*ba

We can see that different removal sequences give different regular expressions. Moreover |RE2|>|RE1|.It means choosing better removal sequence gives us smaller regular expression. We need to develop heuristics which can give us shorter regular expression.

Chapter 4
Proposed Solution

This chapter includes the analysis and comparison of different approaches used for conversion of DFA to RE and some new heuristics are proposed for choosing optimal removal sequence in state elimination method corresponding to a given DFA .In this chapter, we also propose a formal algorithm for determination of bridge state and estimate the relation between the size of regular expression and the size of DFA.

4.1 Comparison of various approaches used for conversion of DFA to RE

Due to repeated union of concatenated terms, transitive closure method is very complex [10] and gives very long regular expression as compared to Brzozowski algebraic method and state elimination method. Brzozowski algebraic method is a recursive approach and gives compact regular expressions. This method takes more time as compared to state elimination method. State elimination method using bridge state concept given by Han and Wood [32], gives shorter regular expression as compared to Brzozowski algebraic method [22] and transitive closure method. State elimination method is an efficient approach as compared to other approaches and smaller regular expression can be obtained using heuristics like vertical chopping and horizontal chopping.

Example 4.1: We will discuss the various approaches of converting DFA to RE on deterministic finite automata which accepts odd number of 1's.

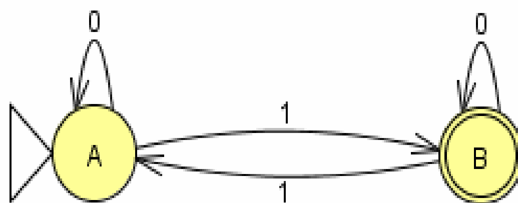


Figure 4.1: DFA accepting all the strings having odd number of 1's

1.Transitive closure approach

Applying transitive closure approach on deterministic finite automata shown in the figure 4.1 :

$r^2_{AB} = r^1_{AB} + r^1_{AB} (r^1_{BB})^* r^1_{BB}$

$r^1_{BA} = r^0_{BA} + r^0_{AA} (r^0_{AA})^* r^0_{BA}$

$r^1_{BB} = r^0_{BB} + r^0_{BA} (r^0_{AA})^* r^0_{BA}$

$r^0_{AA} = 0 + \epsilon \quad r^0_{BB} = 0 + \epsilon \quad r^0_{AB} = 1 \quad r^0_{BA} = 1$

$r^1_{BA} = r^0_{BA} + r^0_{AA} (r^0_{AA})^* r^0_{BA} = 1 + (0+\epsilon)^+ 1$

$r^1_{BB} = r^0_{BB} + r^0_{BA} (r^0_{AA})^* r^0_{BA} = (0+\epsilon) + 1(0+\epsilon)^* 1$

$r^2_{AB} = r^1_{AB} + r^1_{AB} (r^1_{BB})^* r^1_{BB} = (1+\epsilon)^* 1(0+\epsilon+10^* 1)$

$r^2_{AB} = (0)^* 1(0+ 10^* 1)^*$

Regular expression obtained using Transitive closure approach corresponding to deterministic finite automata shown in figure 4.1 is $(0)^* 1(0+ 10^* 1)^*$

2. Brzozowski Algebraic method

The equations generated by this method for the DFA in the figure 4.1 are as follow:

A = 0A + 1B

B = 1A + 0B + ϵ

B = 1A + 0B + ϵ= 0B + (1A + ϵ)

B=0*(1A + ϵ) ………….By Arden's rule

B= 0*(1A) + 0*(ϵ)= 0*1A + 0*…………………….. (1)

Using (1) we obtain

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

A= (0 + 10*1)*(10*) ………………….. By Arden's rule

Regular expression obtained using Brzozowski algebraic method corresponding to deterministic finite automata shown in figure 4.1 is $(0 + 10^* 1)^* (10^*)$.

3. State Elimination Method

Figure 4.2: RE obtained by state elimination

Regular expression obtained corresponding to deterministic finite automata shown in figure 4.1 is $[0^*1(10^*1+0)]^*$ using state elimination method.

Example 4.2: Now applying all the three methods on the DFA, which accepts all the strings with sub-string "bb".

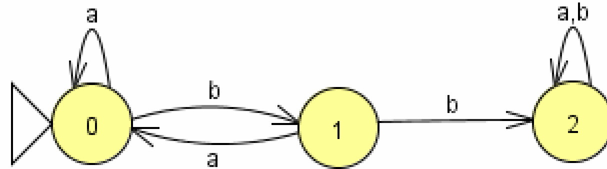


Figure 4.3: DFA accepting all the strings having sub-string "bb"

1.Transitive Closure Approach

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

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

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

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

$r^0_{11} = \epsilon \quad r^0_{22} = a+b+ \epsilon \quad r^0_{12} = b \quad r^0_{21} = \emptyset$

$r^0_{02} = \emptyset \qquad r^0_{01} = b \qquad r^0_{00}= a+\epsilon$

$r^1_{02} = \emptyset + b(\epsilon)^* b=bb$

$r^1_{01}= b + b(\epsilon)^*\epsilon=b+b$

$r^1_{22}= (a+b+\epsilon)+ \emptyset (\epsilon)^*b= (a+b+\epsilon)$

$r^2_{02} = bb+bb(a+b)^* (a+ b)$

Regular expression obtained using Transitive closure approach corresponding to deterministic finite automata shown in figure 4.3 is $bb+bb(a+b)^*(a+ b)$.

2. Brzozowski Algebraic Method

The equations generated by this method for the DFA in the figure 4.3 are as follow:

0=0a+b1

1=b2+a0

2=(a+b)2+ϵ

0=a*(b1)……………. By Arden's rule

2=(a+b)*+ ϵ……………….. By Arden's rule

1=b[(a+b)*+ϵ]+a(a*b1)

1=(a(a*b))*(b(a+b)*+b)................ By Arden's rule

R=(a(a*b))*(b(a+b)*+b)

Regular expression obtained using Brzozowski Algebraic Method corresponding to deterministic finite automata shown in figure 4.3 is $(a(a^{*}b))^{*}(b(a+b)^{*}+b)$.

3) State Elimination Method

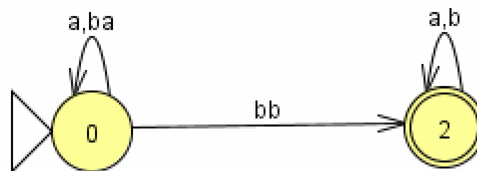


Figure 4.4: After removing state 1

Regular expression obtained using state elimination method corresponding to deterministic finite automata shown in figure 4.3 is (a+ba)*bb(a+b)*.

State elimination method gives us shorter regular expressions as we can see from the above examples.

4.2 Algorithm to determine bridge state

State elimination method is an efficient approach as compared to other approaches and smaller regular expression can be obtained corresponding to a given deterministic finite automata. Researchers have proposed that the removal of non bridge states before the removal of bridge states gives us shorter regular expression. And we achieve this by first detecting all the bridge states of the DFA and then we remove all the non bridge states before the bridge state to get smaller regular expression. It is mathematically given but no formal algorithm is given. So our purpose is to propose a formal algorithm for bridge state by which DFA can be converted into regular expression.

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.

As already discussed in chapter 2 a state q in a DFA A is said to be a bridge state [33], if it satisfies the following conditions:

1. State q is neither a start nor a final state.
2. For each string w ∊ L(A), its path in A must pass through q at least once.
3. State q is not in any circle except for the self-loop.

For finding the bridge states of the graph,we can consider DFA(with q_o as the start state and q_f as the final state) as a graph G(V, E) having V as set of vertex and E is set of edges in G. Let QB denote the set of bridges states. Initially, we take QB=Q

1. We cannot call initial and final state as bridge state. Hence QB=QB-{ q_o,q_f}
2. find_all_paths(graph, start, end, path)

   // The start and final state are not to be considered and the common vertices between the two paths are put in the stack.
3. circle_detection( )

   // It gives the root of all the strongly connected components of the graph and all those vertices which are not involved in any circle.
4. Find Intersection of the vertices which we obtain in the steps 2 and 3.

```
Procedure find_all_paths(graph, start, end,path[]) [25]    //path[] is an empty list
path = path + [start]
if(start = = end)
return [path]
if(not graph.has_key(start))   //if the DFA has no start state
return []                  //return empty set
path = []
for each node in graph
   {
     if(node not in path)
      newpaths = find_all_paths(graph, node, end, path)
      for newpath in newpaths
            {
                 paths.append(newpath)
                 return paths
            }
   }
```

From all the simple paths, the common vertices are put in the stack.

5. For the third condition all the circles of the DFA are found and for this the algorithm circle_detection is called.

```
circle_detection(G(V,E))[29]
bool graph[128][128] ;                  // adjacency matrix
int scc[128] ;
stack=empty;
int num scc = 0 ;                        // number o f  strongly connected component
int dfsnum [128] ;                   // when each  vertex is first visited
int num = 0 ;                            //   counter for dfsnum
int  low [128] ;                     // smallest  dfsnum  reachable from the subtree
int dfsnum=-1,scc=-1;               // dfsnum and scc are initialized to -1
void SCC(int u )
    {
        low [ u ] = dfsnum [ u ] = num++;
        stack.pushback ( u ) ;
        for(int v = 0 ; v < 128; v++ )
           {
              if(graph[u][v] && scc[v] = = -1)
                 {
                      if (dfsnum[v] = = -1)
                       SCC(v) ;
                       low [u] = min(low[u] , low [v] ) ;
                   }
            }
      if (low [u] = = dfsnum[u] )       / / root  of  a  strongly  connected  component
           {
             while(scc[u ] != num scc )
                   {
                    print scc;
                    scc [stack.back ( )] = num scc ;
                     stack. popback ( ) ;
                      }
                 ++num scc ;
```

```
            }
        }
For all scc in the stack
  {
       Check if there is a back_edge from the root of scc
        If(there is a back_edge from  root of scc)
           {
              Pop(scc);
            }
  }
For all v in V
{
Also push onto the stack all the vertices except those which are involved in the circles.
}
```

This algorithm is worked out on the DFA shown in figure 4.3.

Explanation: To determine all the bridge states of an algorithm, we consider DFA as a graph with vertex set V and edge set E. All the start and the final states are ignored as they can never be a bridge state. For the second condition, we determine all the simple paths of the graph from the starting to the final state. A simple path of a graph is a path between two nodes such that no vertex is repeated. After finding all the simple paths between the start and final state the common vertices between the paths are put in a stack because these are the vertices which are always visited whenever the graph is traversed from the start to the final state. Let this set of vertices be called as A. For the third condition all the circles of the graph are found and for this strongly connected component (SCC) of the graph are determined and the roots of SCC are put in the stack. It is checked if there is a back edge from the root of SCC. If there is a back edge then that root of the SCC is popped. We also push onto the stack all the vertices except those which are involved in the circles. Let this set of vertices be called as B. The bridge state is determined by finding the intersection of the set A and B.

Example 4.3

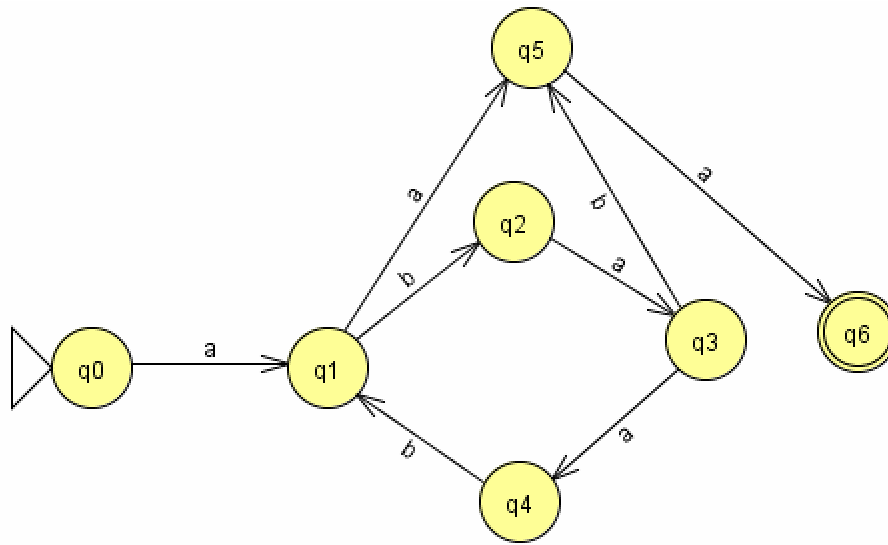


Figure 4.5: DFA with q0 as the start state and q6 as the final state

We consider the DFA as a graph with vertex set V={q0,q1,q2,q3,q4,q5,q6}.

1. The start state q0 and final state q6 are not considered.
2. All the simple paths between the state q0 and q6 are determined. The simple paths are q0-q1-q2-q3-q5-q6 and q0-q1-q5-q6.As the start and final state are not to be considered, the common vertices between the two paths are q1 and q5.These vertices are put in the stack.
3. All the circles are determined. There is one circle q1-q2-q3-q4-q1. The root q1 is pushed onto the another stack and all those vertices which are not involved in the circle are also pushed onto the stack.The state q0,q5 qnd q6 are not involved in the circle but only q5 is pushed onto the stack, as the start and final state are not to be considered .
4. In the first stack we have q1 and q5 and in the another stack also we have q1 and q5.The intersection gives us states q1 and q5 and these are the bridge states of the DFA.

Example 4.4: In the figure 4.6 states q1 and q4 are the bridge states. Removal sequence q2-q3-q1-q4,q1-q4-q2-q3 and q1-q2-q4-q3 give different regular expressions.

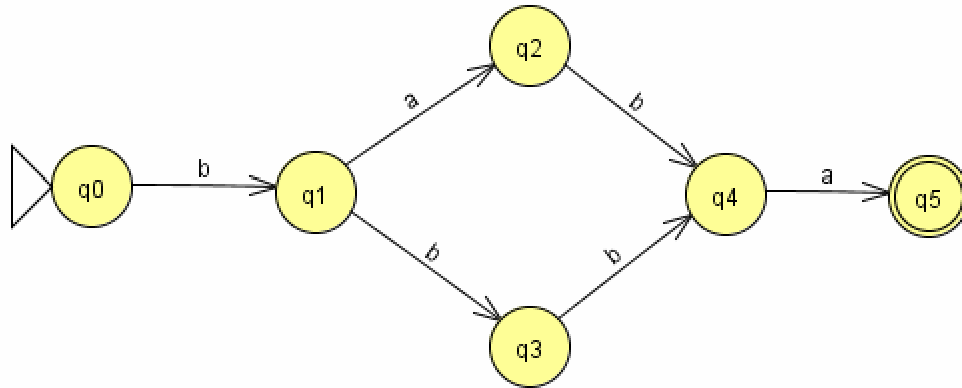


Figure 4.6: Bridge state in DFA

Case 1: Removal sequence q2-q3-q1-q4.Bridge states are eliminated at the end.

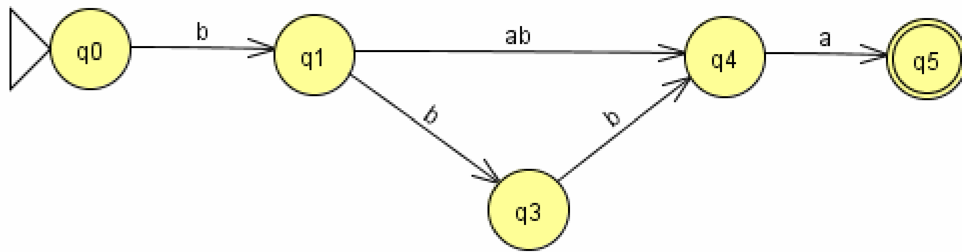


Figure 4.7: After removing state q2

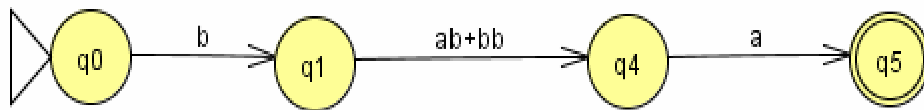


Figure 4.8: After removing state q3

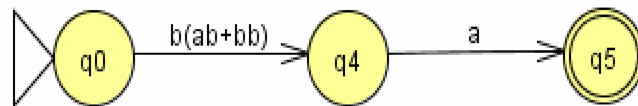


Figure 4.9: After removing state q1

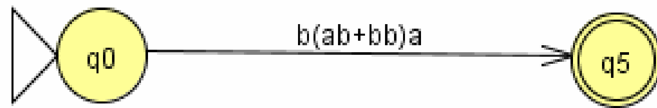


Figure 4.10: After removing state q4

This removal sequence gives us RE1=b(ab+bb)a as the regular expression.

Case 2: Removal sequence q1-q4-q2-q3

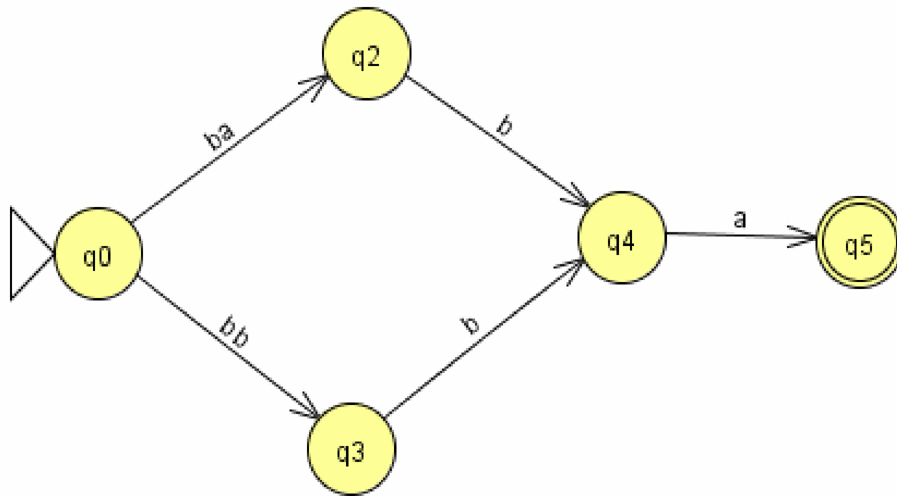


Figure 4.11: After removing state q1

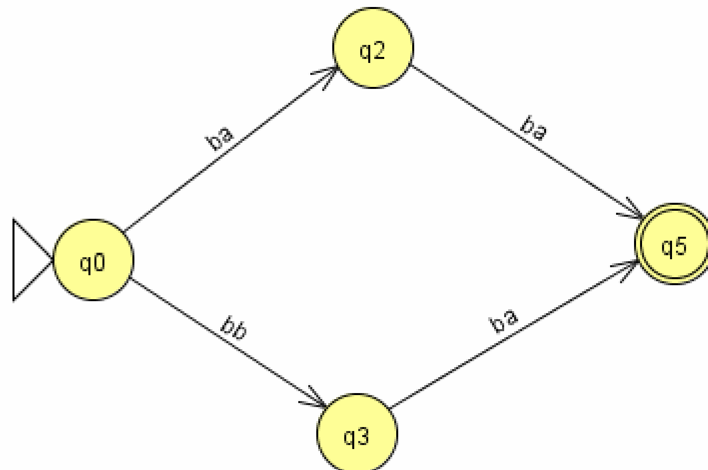


Figure 4.12: After removing state q4

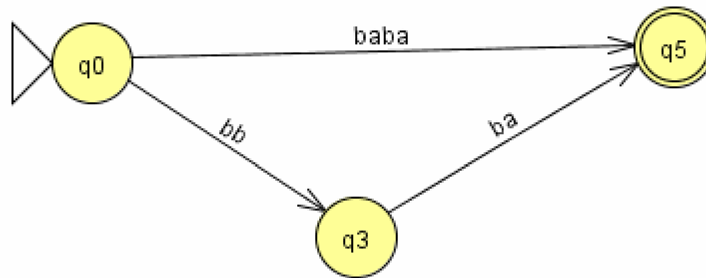


Figure 4.13: After removing state q2

Figure 4.14: After removing state q3

This removal sequence gives us RE2 = baba + bbba as the regular expression.

Similarly the removal sequence q1-q2-q4-q3 also gives us baba+bbba as the regular expression. Now |RE2|>| RE1|,so it is clear that the removal of non bridge states before the bridge states gives us smaller regular expression.

If all the states in the DFA are bridge states, then any removal sequence will give the same regular expression. This is shown in the figure 4.15.

Example 4.5

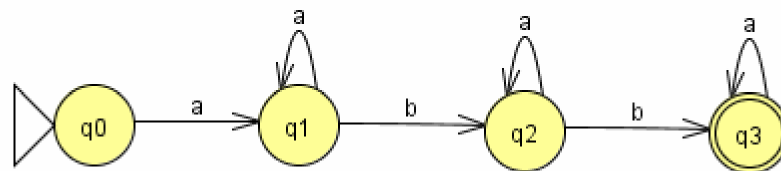


Figure 4.15: Example of DFA where all the states are bridge states

Case 1: Removal sequence q1-q2

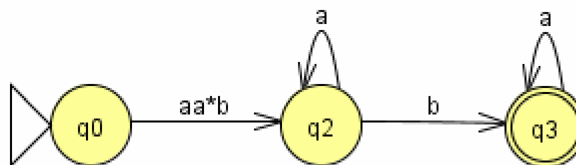


Figure 4.16(a): After removing state q1

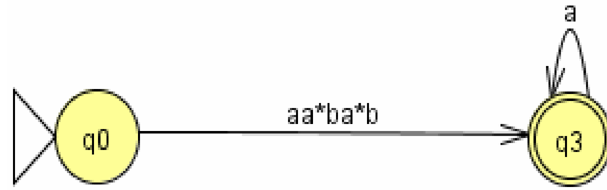


Figure 4.16(b): After removing state q2

Figure 4.16: Removal of the states in the order q1-q2

So the regular expression,RE1=$aa^*ba^*ba^*$

Case 2: Removal sequence q2-q1

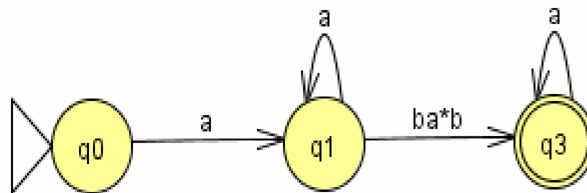


Figure 4.17(a): After removing state q2

Figure 4.17(b): After removing state q1

Figure 4.17: Removal of the states in the order q2-q1

The regular expression,RE2=aa*ba*ba*

In both the cases the regular expression is same. It can be concluded that if all the states in the DFA are bridge states then any removal sequence gives same regular expression.

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

We first find out all the circles in a DFA and then for each state leaving the initial and the final state, we count the number of circles in which that state is involved. Self loops are also counted as circles. We eliminate those states first which are involved in the minimum number of circles.

4.3.1 Circle detection algorithm [29]

```
bool graph[128][128] ;                              // adjacency matrix
int scc[128] ;                                      // strongly connected component
stack=empty;
int num scc = 0 ;                                   //number of strongly connected component
int dfsnum [128] ;                                  // when each  vertex is first visited
int num = 0 ;                                       //   counter for dfsnum
int  low [128] ;                                    // smallest  dfsnum  reachable from the subtree
int dfsnum=-1,scc=-1;                               // dfsnum and scc are initialized to -1
void SCC(int u)
   {
       low[u] = dfsnum[u] = num++;
       stack.push( u ) ;
       for( int v = 0 ; v < 12 8; v++ )
          {
             if ( graph[u][v] && scc[v] = = -1)
                {
                    i f ( dfsnum [v] = = -1)
                     SCC( v ) ;
                     low [ u ] = min ( low [ u ] , low [ v ] ) ;
                  }
          }
 if ( low [u] = = dfsnum[u] )       / / root  of  a  strongly  connected  component
         {
             whi l e (scc[u] != num scc)
                 {
                  print scc;
                  scc [stack .back ( )] = num scc ;
                   stack. pop ( ) ;
                   }
        ++num scc ;
```

}

}

Example 4.6: To find circles in a DFA.

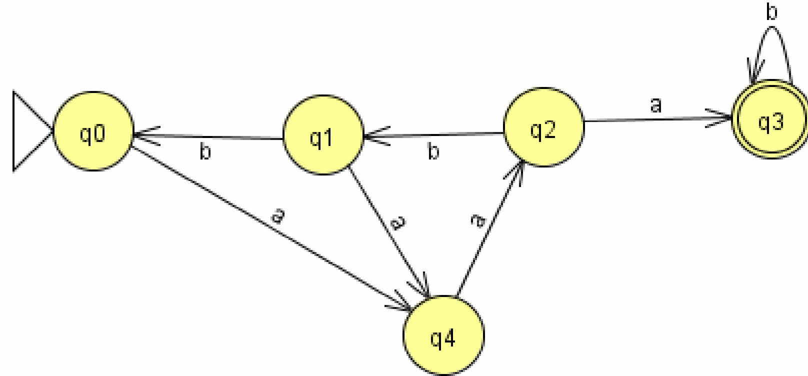


Figure 4.18: Example of circles in DFA

In the figure 4.18 the DFA has three circles. The circles are q0-q4-q2-q1-q0,q1-q4-q2-q1 and q3-q3.State q1 is involved in 2 circles, state q2 in 2 circles, state q3 in 1 circle and state q4 in 2 circles. So we will first eliminate state q3 and as states q1,q2 and q4 are involved in 2 circles, they can be removed in any order.

Example 4.7: The DFA in figure 4.19 has two circles. They are 1-2-3-1,1-2-3-4-1. State 1 is involved in both the circles. Similarly states 2 and 3 are also involved in both the circles. But state 4 is involved in one circle. So, we remove state 4 in the beginning and state 1 is a bridge state so we eliminate it at the end.

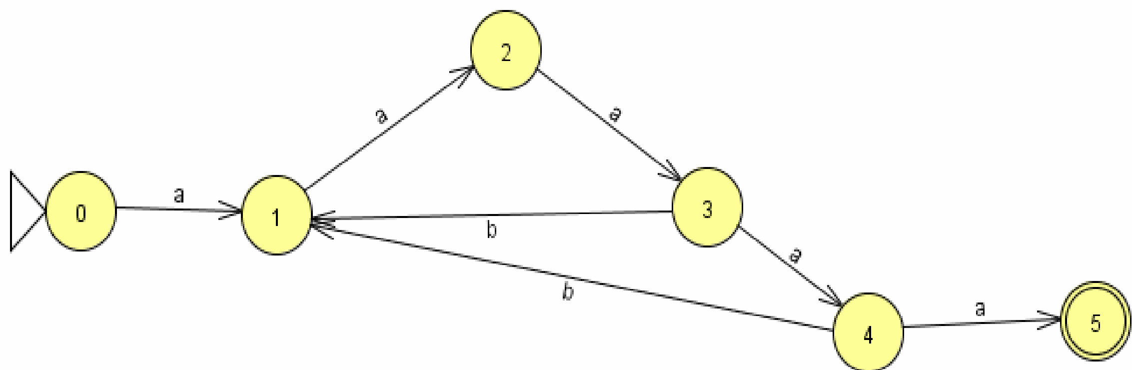


Figure 4.19: DFA with two circles

Case 1: Eliminate those states first which are involved in the minimum number of circles. The removal sequence is 4-2-3-1.

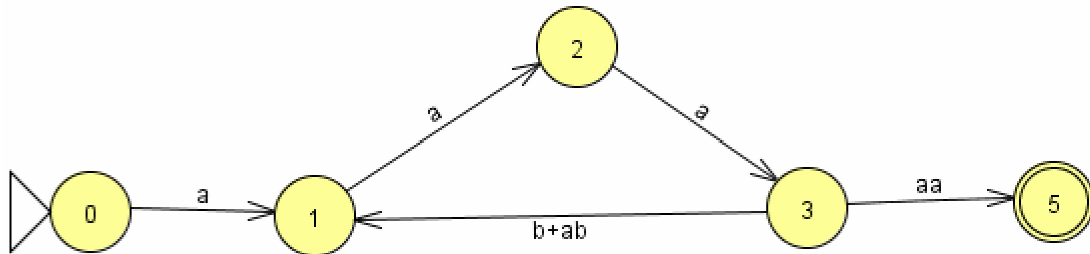


Figure 4.20: After removing state 4

Now again we count the circles. There is one circle 1-2-3-1.All the three states 1,2,3 are involved in one circle. So we can eliminate in any order.

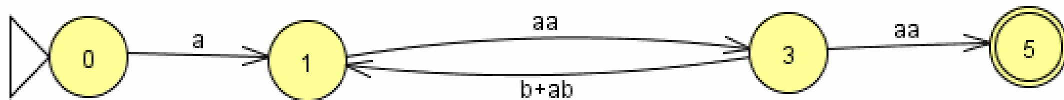


Figure 4.21: After removing state 2

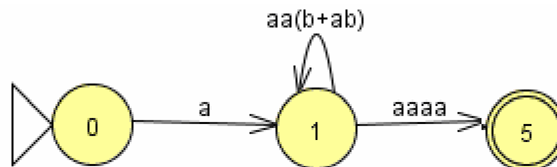


Figure 4.22: After removing state 3

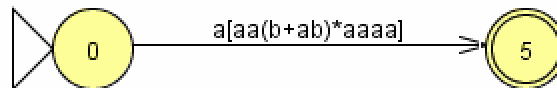


Figure 4.23: After removing state 1

Regular expression RE1 = a(aa(b+ab))*aaaa is obtained.

Case 2:- Removing those states first which are involved in the maximum number of circles using the sequence 2-3-4-1.

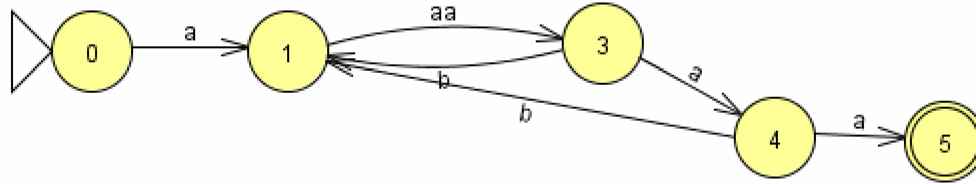


Figure 4.24(a): After removing state 2

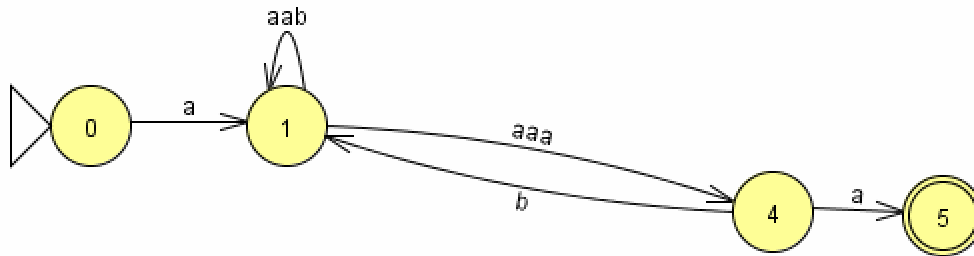


Figure 4.24(b): After removing state 3

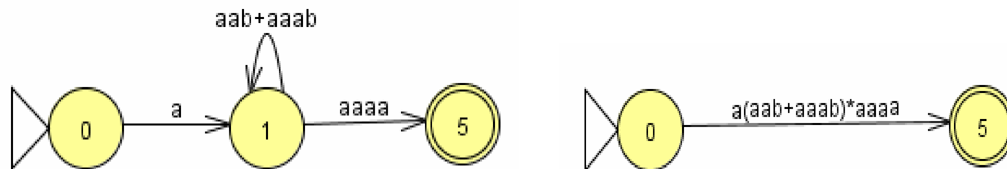


Figure 4.24(c): After removing state 4 **Figure 4.24(d):**After removing state 1

Figure 4.24: Removing the states in the order 2,3,4 and 1.

Regular expression RE2=a(aab+aaab)* is obtained using removal sequence 2,3,4,1. Now |RE2|> | RE1|, Hence removal of states which are involved in the minimum number of circles first gives us smaller regular expression.

Example 4.8: In the DFA in figure 4.25 also we first count the number of circles. There are two circles 1-2-1 and 1-1. State 1 is involved in two circles and state 2 in one circle and state 3 in zero cycle.

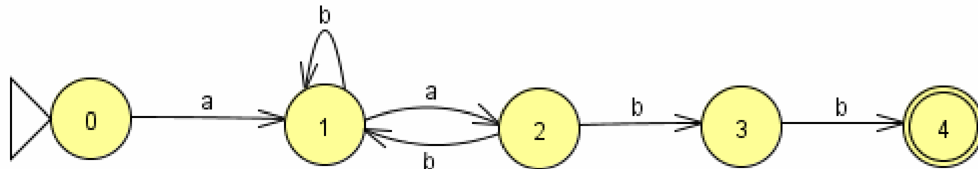
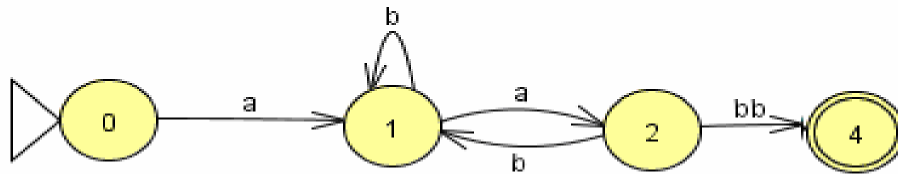


Figure 4.25: DFA with two circles,one circle and one self loop

Case 1: Eliminate those states first which are involved in the minimum number of circles. The removal sequence is 3-2-1.



. **Figure 4.26(a):** After removing state 3

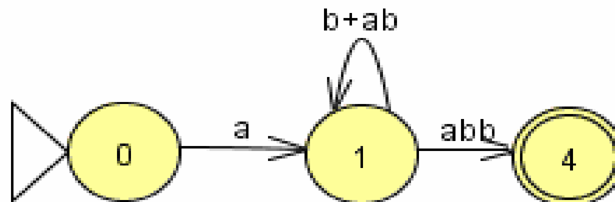


Figure 4.26(b): After removing state 2

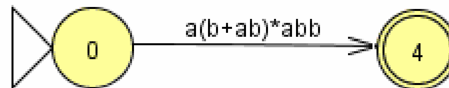


Figure 4.26(c): After removing state 1

Figure 4.26: Removing the states in the order 3,2 and 1.

The regular expression R1 is a(b+ab)*abb.

Case 2:- Removing those states first which are involved in the maximum number of circles using the sequence 1-2-3.

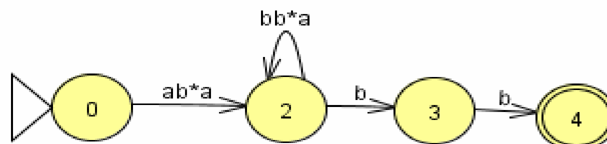


Figure 4.27(a): After removing state 1

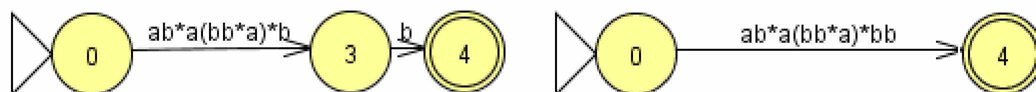


Figure 4.27(b): After removing state 2 **Figure 4.27(c):** After removing state 3

Figure 4.27: Removing the states in the order 1,2 and 3.

The regular expression R2=ab*a(bb*a)*bb. Now |R2|> | R1|, Hence removal of states which are involved in the minimum number of circles first gives us smaller regular expression.

Example 4.9: In the DFA in figure 4.28 also we first count the number of circles. There are three circles 1-2-1,0-1-0 and 0-5-2-1-0.State 1 is involved in three circles, state 2 in two circles, state 3 in zero circle, state 5 in one circle.

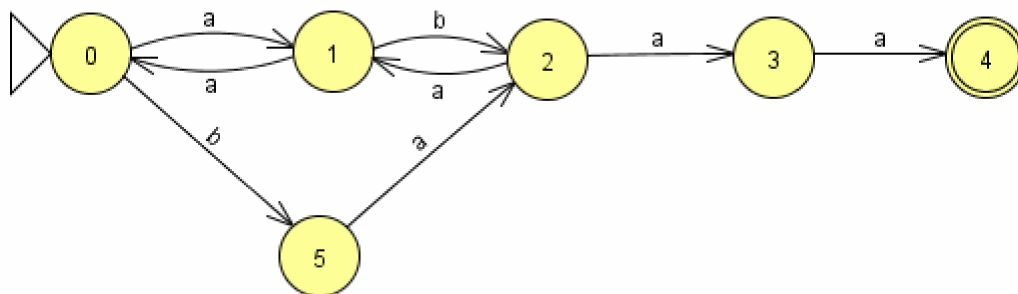


Figure 4.28: DFA having three circles

Case 1: Eliminate those states first which are involved in the minimum number of circles. The removal sequence is 3-5-2-1..

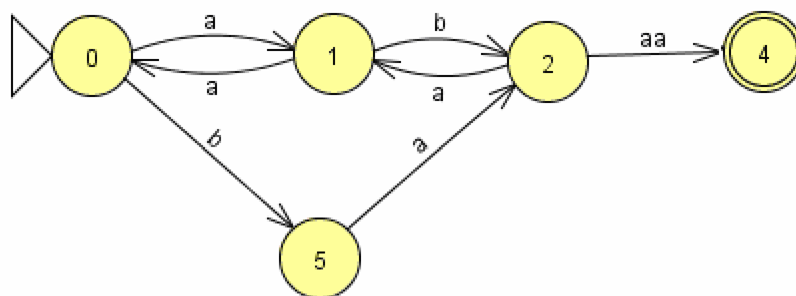


Figure 4.29(a): After removing state 3

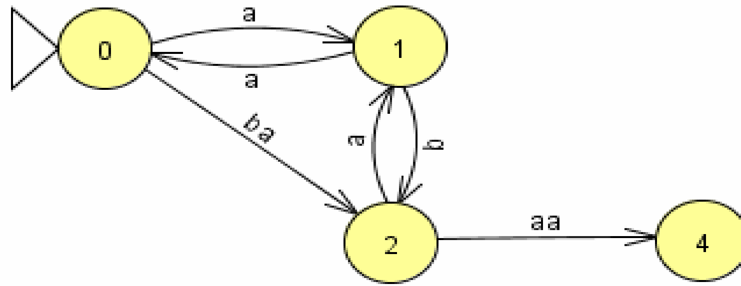


Figure 4.29(b): After removing state 5

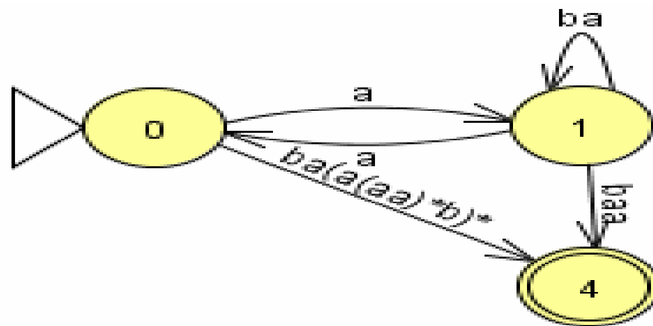


Figure 4.29(c): After removing state 2

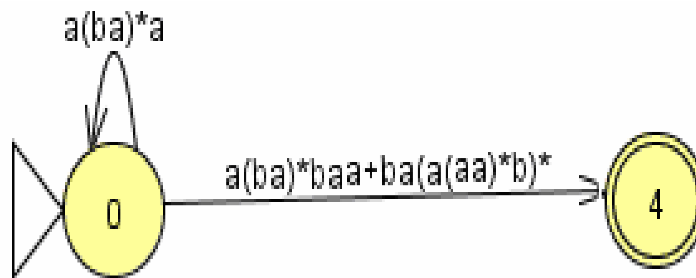


Figure 4.29(d): After removing state 1

Figure 4.29: Removing the states in the order 3,5,2 and 1.

In this case the RE,RE1=(a(ba)*a)*[a(ba)*baa+ba(a(aa)*b)*

Case 2: Removing those states first which are involved in the maximum number of circles using the sequence 1-5-2-3.

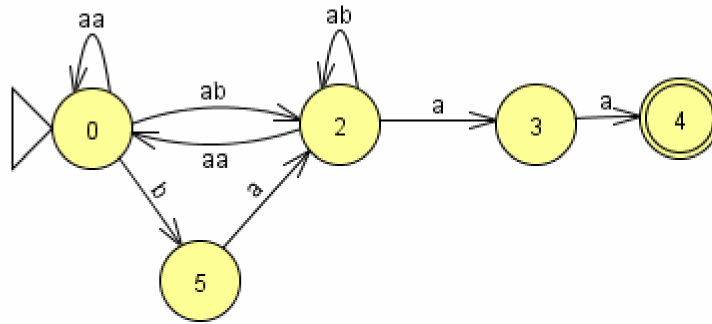


Figure 4.30(a): After removing state 1

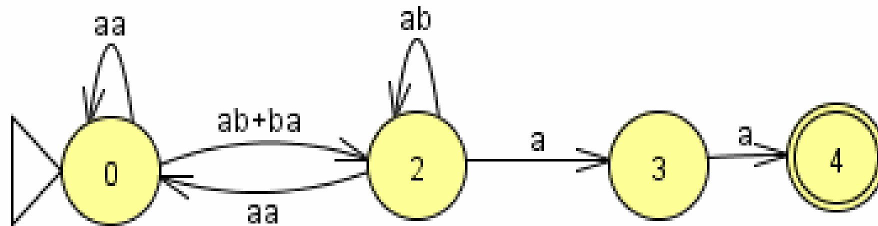


Figure 4.30(b): After removing state 5

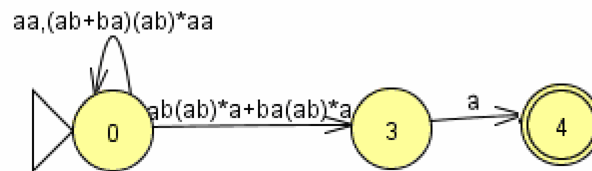


Figure 4.30(c): After removing state 2

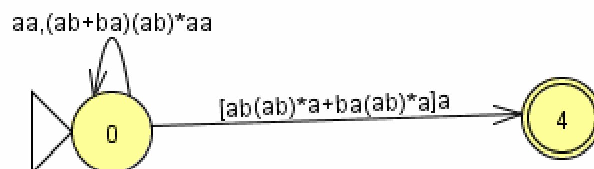


Figure 4.30(d): After removing state 3

Figure 4.30: Removing the states in the order 1,5 2 and 3.

In this case the RE,RE2=(aa+(ab+ba)(ab)*aa)*[ab(ab)*a+ba(ab)*a]a. Now |RE2|> | RE1|, Hence removal of states which are involved in the minimum number of circles first gives us smaller regular expression.

EXAMPLE 4.10: The number of circles in the DFA in figure 4.31 is three. The three circles are 1-1,2-2 and 2-3-2.

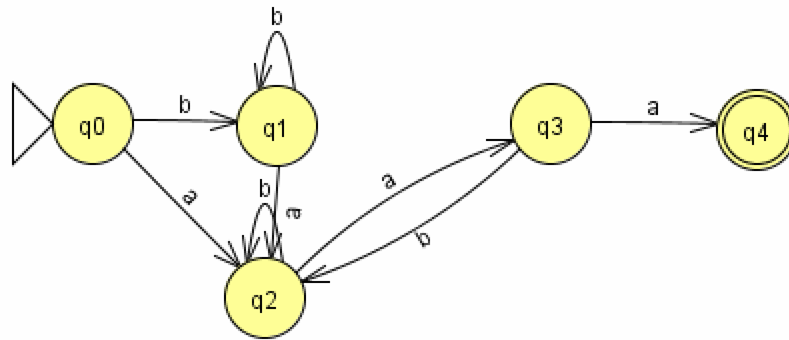


Figure 4.31: Another example of DFA with three circles

Case 1: Eliminate those states first which are involved in the minimum number of circles. The removal sequence is q1-q3-q2.

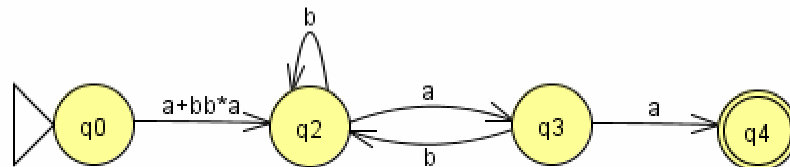


Figure 4.32(a): After removing state q1

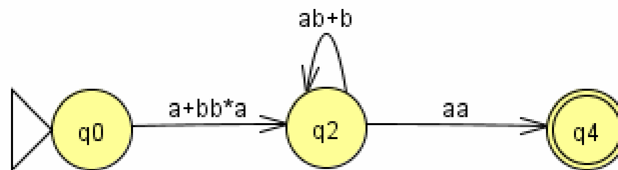


Figure 4.32(b): After removing state q3

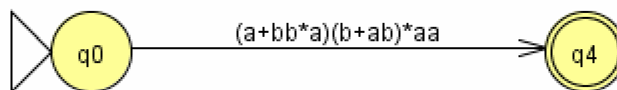


Figure 4.32(c): After removing state q2

Figure 4.32: Removing the states in the order q1,q3 and q2

In this case the RE, RE1=(bb*a+a)(b+ab)*aa

Case 2: Removing those states first which are involved in the maximum number of circles using the sequence q3-q2-q1.

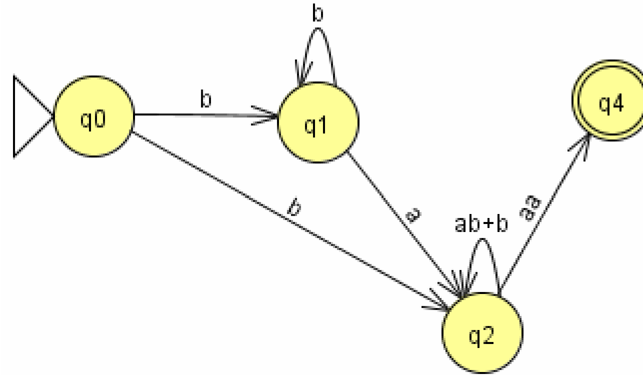


Figure 4.33(a): After removing state q3

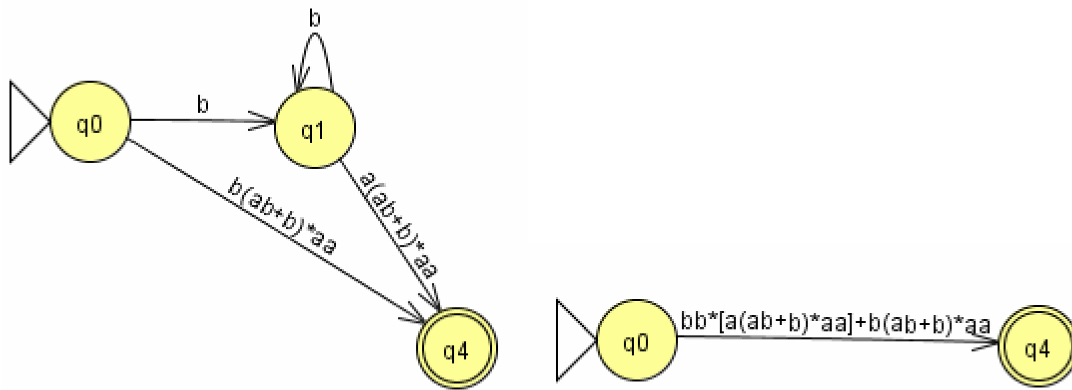


Figure 4.33(b): After removing state q2 **Figure 4.33(c):** After removing state q1

Figure 4.33: Removing the states in the order q3,q2 and q1.

In this case the RE, RE2=bb*[a(ab+b)*aa]+a(ab+b)*aa. |RE2|> | RE1|, hence removal of states which are involved in the minimum number of circles first gives us smaller regular expression.

Example 4.11: In the figure 4.34 there are 3 circles 1-1, 2-2 and 3-3.All the three states are involved in only one circle. So, any removal sequence gives us the same regular expression.

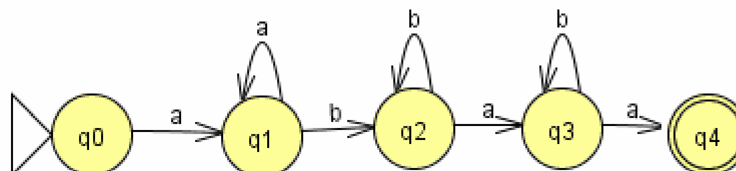


Figure: 4.34: DFA with three self loops and all the three states involved in one circle.

Case 1: Removal sequence q1-q2-q3

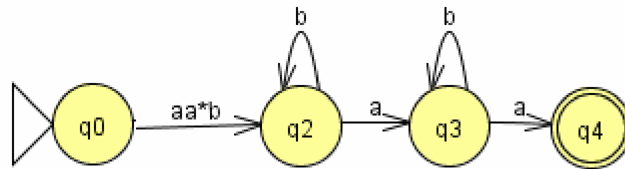


Figure 4.35(a): After removing state q1

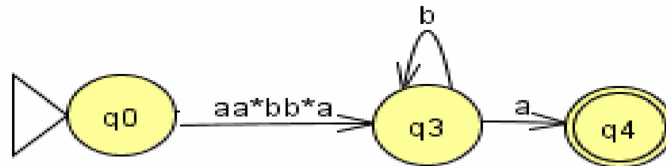


Figure 4.35(b): After removing state q2

Figure 4.35(c): After removing state q3

Figure 4.35: Removing the states in the order q1,q2 and q3.

RE1=aa*bb*ab*a

Case 2: Removal sequence q3-q2-q1.

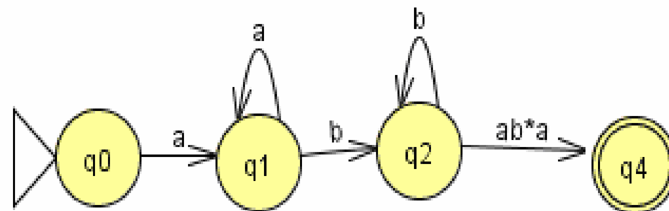


Figure 4.36(a): After removing state q3

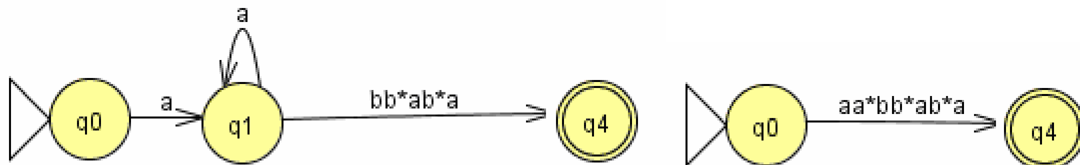


Figure 4.36(b): After removing state q2 **Figure 4.36(c):**After removing state q1

Figure 4.36: Removing the states in the order q3,q2 and q1.

RE2=aa*bb*ab*a

In both the cases the regular expression is same because all the three states are involved in only one circle.

4.4 Relation between the size of a DFA and regular expression

There are three types of operators used in the formation of regular expressions. They are union (+),star(*) and concatenation(.).The size of DFA can be determined by counting the number of states and the edges and the size of the regular expression can be determined by counting the number of operators and the alphabets. From the examples given below it can be seen that the relation between the size of DFA and the RE is linear. The size of RE increases linearly with the size of the DFA. Moreover,the size of the regular expression increases as the union and the concatenation operators increases.

Example 4.12

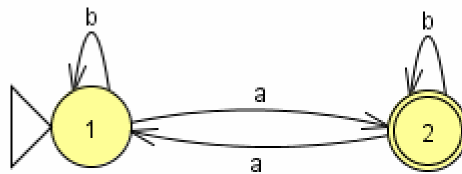


Figure 4.37: DFA accepting all the strings having odd number of a's

RE=b*a+(b+ab*a)*

Size of DFA=2+4=6

Size of RE=14

Example 4.13

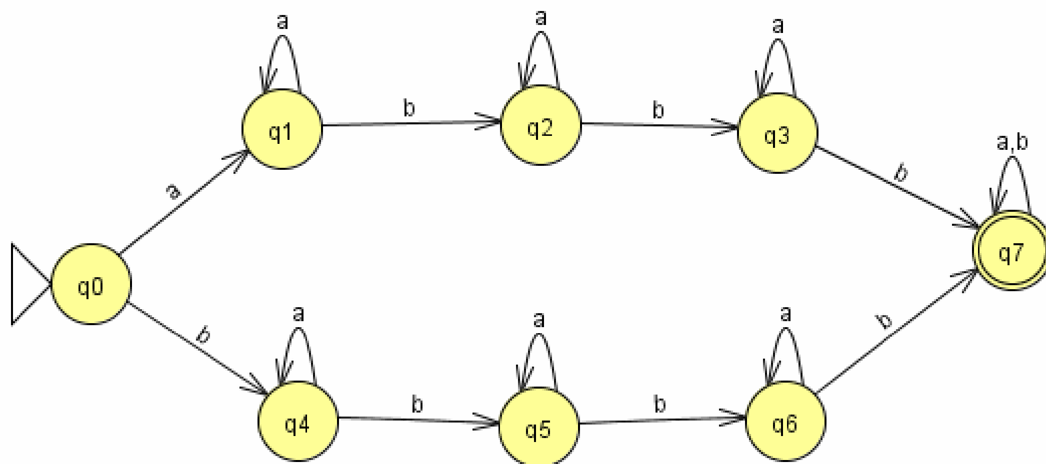


Figure 4.38: DFA with eight states

RE=[aa*ba*ba*b+ba*ba*ba*b](a+b)*

Size of DFA=8+15=23

Size of RE=38

Example 4.14

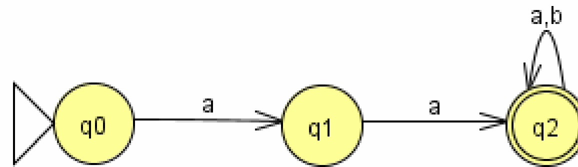


Figure 4.39: DFA accepting all the strings starting with aa

RE=aa(a+b)*

Size of DFA=3+3=6

Size of RE=8

Example 4.15

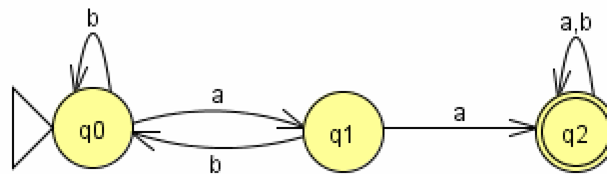


Figure 4.40: DFA accepting all the strings with substring aa

RE=(b+ab)*aa(a+b)*

Size of DFA=3+5=8

Size of RE=15

Example 4.16

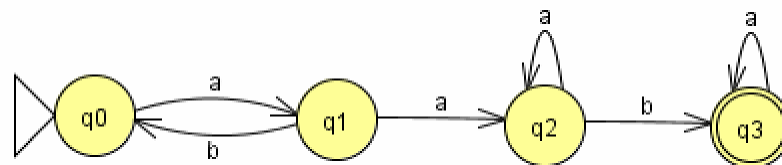


Figure 4.41: DFA with four states

RE=(ab)*aaa*ba*

Size of DFA=4+6=10

Size of RE=16

Chapter 5
Conclusions and Future work

This thesis work describes the various approaches used for conversion of deterministic finite automata to regular expression. The comparison of different approaches used for conversion of DFA to RE is being done and new heuristics are proposed for choosing optimal removal sequence using state elimination method corresponding to a given DFA . A formal algorithm for bridge state is proposed and the relation between the size of RE and DFA is estimated.

5.1 Conclusions

Due to repeated union of concatenated terms, transitive closure method is very complex and gives very long regular expression as compared to Brzozowski algebraic method and state elimination method. Brzozowski algebraic method is a recursive approach and gives compact regular expressions. This method takes more time as compared to state elimination method. State elimination method using bridge state concept given by Han and Wood, gives shorter regular expression as compared to Brzozowski algebraic method and transitive closure method. State elimination method is an efficient approach as compared to other approaches and smaller regular expression can be obtained using heuristics like bridge state, vertical chopping and horizontal chopping. Heuristics can be developed by which minimal RE can be obtained.

5.2 Summary of Contributions

New heuristics are proposed in this thesis work using concept of circle in the DFA. Using these new heuristics, smaller regular expression can be generated. A formal algorithm for bridge state is proposed and the relation between the size of DFA and RE is estimated.

5.3 Future Research

Still much smaller regular expression can be obtained from a given DFA. Therefore new heuristics can be developed 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, vol. 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,vol. 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 pages 67–76,1963.
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, vol. 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, vol. 10(4),pages 407– 437, 2005.
14. Larkin, H., "Object oriented regular expressions", 8th IEEE International Conference on Computer and Information Technology, pages 491-496, 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, 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, pages 312–314,2004.
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, pages 207 – 212,2006.
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 ,pages 39–47,1960.
25. R. Tarjan, “Depth-first search and linear graph algorithms”, SIAM Journal on Computing vol. 1(2), pages 146–160,1972.
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 vol. 293(3),pages 535–556,2003.
29. Tarjan R.E., ”Depth First Search and Linear Graph Algorithms,” SIAM J. Computing vol. 1(2), pages 146-160, 1972.
30. Ullman, J., A. V. Aho and R. Sethi, “Compiler Design: Principles, Tools, and Techniques”, Pearson Education Inc, ISBN 0-201-10088-6,198
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. Yo-Sub Han and Derick Wood, Obtaining shorter regular expressions from finite-state automata , Theoretical Computer Science vol. 370(1-3) ,pages 110-120,2007.
33. Y.-S. Han and D. Wood, “The generalization of generalized automata: Expression automata”, International Journal of Foundations of Computer Science vol. 16(3), pages 499–510,2005.