Closure Properties of Prefix-Free and Suffix-Free Regular Languages

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

Master of Technology
in
Computer Science and Applications

Submitted By
Meenu Lochan
(Roll No. 601003014)

Under the supervision of:
Mrs.Sunita Garhwal
Assistant Professor, SMCA

SCHOOL OF MATHEMATICS AND COMPUTER APPLICATIONS
THAPAR UNIVERSITY
PATIALA – 147004

June 2012

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Closure Properties of Prefix-Free and Suffix-Free Regular Languages*", in partial fulfillment of the requirements for the award of degree of Master of Technology in *Computer Science and Applications* submitted in School of Mathematics and Computer Applications of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mrs.Sunita Garhwal* 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: Meenu Lochan

(Meenu Lochan)

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

Sunita

(Mrs. Sunita Garhwal)
Assistant Professor
SMCA
Thapar University
Patiala

Countersigned by

(Dr. S.S.Bhatia)
Head
School of Mathematics and Computer Applications
Thapar University
Patiala

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

Acknowledgement

Words are inadequate to express my gratitude to my thesis supervisor, **Mrs.Sunita Garhwal** Assistant Professor, School of Mathematics and Computer Applications, Thapar University, who has been very concerned and has aided for all the material essential for the preparation of this thesis report. She 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 highly obliged to **Dr. S.S.Bhatia**, Head of Department, SMCA and **Mr. Singara Singh**, P.G. Coordinator for the motivation and inspiration that helped me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there in the need 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.

MeenuLochan

Meenu Lochan

(601003014)

ABSTRACT

Regular languages are specified by finite-state automata (FAs) or regular expressions. Regular languages are used in diverse fields of computer science like compilers, data compression, text processing, software engineering and pattern matching. Language can be categorized as regular, context free, context sensitive or recursive enumerable based on Chomsky hierarchy for languages.

Regular languages can be classified as infix free, prefix free, suffix free. These are used in various applications like pattern matching, computing forbidden words and text searching. The study of state complexity is also strongly motivated by applications of finite automata in software engineering, programming languages, natural language and speech processing and other practical areas. Since many of these applications use automata of large sizes, it is important to know the number of states of the automata.

A regular language is prefix-free if and only if its minimal DFA *M* has only one final state and the final state has no out transitions whose target state is not a sink state. A regular language is suffix-free if and only if its minimal DFA *M* have a exclusive sink state and its starting state does not have any in-transitions.This thesis specifies that in prefix-free and suffix-free regular languages under what operation languages are closed.

List of contents Page No.

Certificate i

Acknowledgement ii

Abstract iii

List of Contents iv

List of Figures and Tables vi

List of Abbreviations viii

Chapter 1: Regular Languages 1

1.1 Automata 1

1.1.1 Alphabets, words and languages 1

1.2 Finite Automata 1

1.2.1 Types of Finite Automata 3

1.3 Chomsky Hierarchy 6

1.4 Regular Expression 7

1.4.1 Applications for Regular Expressions 8

1.5 Regular languages 8

1.5.1 Forms of Regular Languages 9

1.6 Operations on Regular Languages 10

1.7 Thesis Outline 11

Chapter 2: Literature Review 12

2.1 Introduction 12

2.2 State Complexity 13

2.3 Closure properties of Regular Languages 18

Chapter 3: Problem Statement 19

3.1 Problem Statement 19

3.2 Thesis Objectives 19

3.3 Thesis Motivation 20

Chapter 4: Closure Properties of Prefix-Free and Suffix-Free Regular Languages 21

4.1 Closure properties of prefix-free regular languages 21
4.1.1 Union Operation on prefix-Free Regular Languages 21
4.1.2 Concatenation Operation on Prefix-Free Regular Languages 22
4.1.3 Complementation Operation on Prefix-Free Regular Languages 24
4.1.4 Intersection Operation on Prefix-Free Regular Languages 25
4.1.5 Kleene closure Operation on Prefix-Free Regular Languages 25
4.1.6 Reversal Operation on Prefix-Free Regular Languages 26
4.2 Closure properties of suffix-free regular languages 28
4.2.1 Union Operation on suffix-Free Regular Languages 28
4.2.2 Concatenation Operation on suffix-Free Regular Languages 29
4.2.3 Complementation Operation on suffix-Free Regular Languages 30
4.2.4 Intersection Operation on suffix-Free Regular Languages 32
4.2.5 Kleene closure Operation on suffix-Free Regular Languages 33
4.2.6 Reversal Operation on suffix-Free Regular Languages 34
Chapter 5: Conclusion and Future scope 36
References 37
List of Publications 41

List of Figures and Tables

Figure No. Figure Title	Page No.
Figure 1.1 Working of finite automata	2
Figure 1.2 Communication protocols	3
Figure 1.3 Operations on the machine	4
Figure 1.4 Transition from *q* to *p*	4
Figure 1.5 Deterministic Finite State Automata	5
Figure 1.6 Non- Deterministic Finite State Automata	5
Figure 1.7 Computation Tree	6
Figure 1.8 NFA for a string ending with 01	6
Figure 1.9 Hierarchy of languages	6
Figure 1.10 Relation between FA, RE AND RL	9
Figure 2.1 Automata recognizing (a^+),(b^+c) and $(c^?d^+)$	13
Figure 2.2 Types of Complexity	14
Figure 4.1 DFA for L_1 = *{ab}*	21
Figure 4.2 DFA for L_2 = *(aabb)*	22
Figure 4.3 Union property on L_1UL_2	22
Figure 4.4 Union after ε-transition	22
Figure 4.5 Language L_1 = *{ab}*	23
Figure 4.6 Language L_2 = *{ba}*	23
Figure 4.7 L_1. L_2 = *{abba}*	23
Figure 4.8 Final diagram of L_1. L_2 = *{abba}*	23
Figure 4.9 L = $\{ab^*a\}$	24
Figure 4.10 *L* with a dead state	24
Figure 4.11 L^c with a dead state	25
Figure 4.12 kleene closure L_1 = {ab}	25
Figure 4.13 DFA for L^*	26
Figure 4.14 *(*DFA with ε- Closure	26

Figure 4.15 L_1 = *(abc)* 27
Figure 4.16 Reversal of $(L_1)^R$ = *{cba}* 27
Figure 4.17 L_1 = *{ abc*}* 28
Figure 4.18 L_2 = $\{j^+k\}$ 28
Figure 4.19 Union of L_1 and L_2 29
Figure 4.20 L_1UL_2 = $\{abc^* U j^+k\}$ 29
Figure 4.21 L_1 = *{x | y}* 30
Figure 4.22 L_2 = *{z}* 30
Figure 4.23 $L_1 . L_2$ = (x | y)z 30
Figure 4.24 After simplification of (x | y)z 30
Figure 4.25 String that accepts only *aba* and *abb* 31
Figure 4.26 String that does not accepts only *aba* and *abb* 31
Figure 4.27 L_1 = $\{a^n b\}$ 32
Figure 4.28 L_1 = $\{a b^m\}$ 32
Figure 4.29 Suffix-free of $L_1 \cap L_2$ = *{ab}* 33
Figure 4.30 L_1 = *a* 33
Figure 4.31 L_2 = *(bc)* 33
Figure 4.32 Union of (*a U bc)* 34
Figure 4.33 Kleene closure of *(a U bc)** 34
Figure 4.34 *L* is the suffix-free language 35
Figure 4.35 $(L_1)^R$ is the suffix-free language 35

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

Table 1.1 Transition Functions 4
Table 1.2 Chomsky Hierarchy 7
Table 2.1 Syntactic complexities of prefix-free, suffix-free and bifix-free languages 15
Table 2.2 State Complexity of Basic Operations of Prefix-Free DFAs and NFAs 16
Table 2.3 State Complexity of Basic Operations of Suffix-Free DFAs and NFAs 17
Table 2.4 State Complexity of Basic Operations of Infix-Free DFAs and NFAs 17
Table 4.1 Closure Properties of Prefix-Free Regular Languages 27
Table 4.2 Closure Properties of Suffix-Free Regular Languages 35

List of Abbreviations

FA	Finite State Automata
DFA	Deterministic Finite Automata
NFA	Non-Deterministic Finite Automata
RE	Regular Expression
RL	Regular Languages
SC	State Complexity

Chapter 1

Regular Languages

This chapter contains basic introduction to automata and its types. This includes taxonomies and various forms of finite automaton and regular languages.

1.1 Automata

Theory of computation [7] 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. Automata theory is the study of abstract machines and the computational problems that can be solved using these abstract machines. An automaton is one in which information can be transformed and transmitted. An automaton can perform some functions without direct participation of human being [7, 14].

1.1.1 Alphabets, Words and Languages

An alphabet is a finite, nonempty set of symbols, denoted by $\sum$. Language is defined as a subset of $\sum^*$ (alphabet). The notation $\sum^*$ means the set containing all the finite strings whose symbols are taken from an alphabet [24].Strings is a finite sequences of letters called words. A word of length one is recognized with its only symbol. A special word is the empty word (or null word) having no symbols, denoted by λ, ^ or ε. The length of the word w is the number of symbols in it, denoted by $|w|$. The a-length of the word x is the number of times a particular letter *'a'* appears in x [14]. It is denoted by $|x|_a$. The concatenation of the words w_1 and w_2 is denoted by w_1w_2. e.g $w_1 = aacbba$, $w_2 = caac$, $w_1w_2 = aacbbacaac$, $w_1 = aacbba$, $w_2 = \lambda$, $w_1w_2 = w_1 = aacbba$, $w_1 = \lambda$, $w_2 = caac$, $w_1w_2 = w_2 = caac$. Concatenation is associative but not commutative.

1.2 Finite Automata

Formal language theory as a discipline is generally regarded as growing from the work of linguist Noam Chomsky [16] in the 1950s, when he attempted to give a precise characterization of the structure of natural languages. His goal was to define the syntax of languages using simple and precise mathematical rules. Later it was found that the syntax

of programming languages can be described using one of Chomsky's grammatical models called context-free grammars. After the advent of modern electronic computers, people realized that all forms of information whether numbers, names, pictures, or sound waves can be represented as strings. Then collections of strings known as languages became central to computer science. Every programming language from FORTRAN to JAVA can be precisely described by a grammar. The Grammar of the language allows us to write a computer program to determine whether a string of statements is syntactically correct in the programming language. Formal languages and grammars have many applications in other fields, including molecular biology [20, 14].

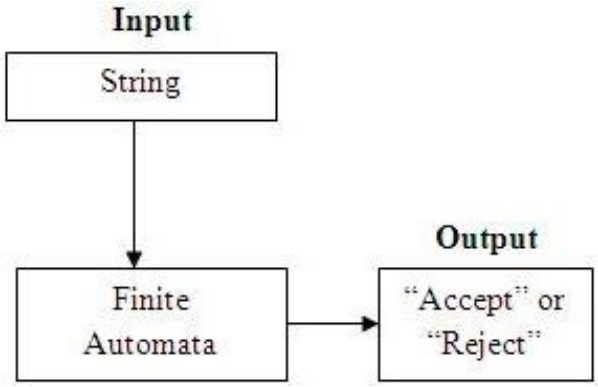

Figure 1.1: Working of Finite Automata [16]

Finite State Machines are used in real life applications such as Vending machines, Thermostats, Elevators, Train track switches, Lexical analyzers etc.

Finite automata are a useful model for many important kinds of software and hardware. Such as, software for designing and checking the behavior of digital circuits, the lexical analyzer of a typical compiler which breaks the input text into logical units, software for scanning large bodies of text, such as collections of Web pages, to find occurrences of words, phrases or other patterns and software for verifying systems of all types that have a finite number of distinct states, such as communications protocols of protocols for secure exchange information. Figure 1.2 shows a FSM which is used to describe reactive systems. A common example of such systems is communication protocols. Customer, Store, and Bank will be finite automata.

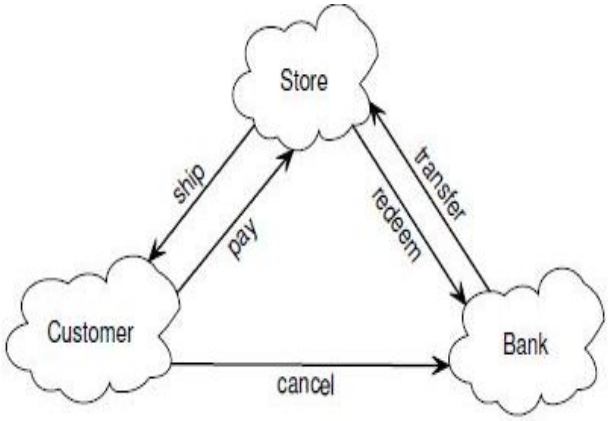

Figure 1.2: Communication Protocols [24]

1.2.1 Types of Finite Automata

Finite automata can be classified as deterministic finite-state automaton and Non-deterministic finite-state automaton.

1) Deterministic Finite State Automata

A deterministic finite-state automaton is a finite state machine accepting finite strings of symbols. DFA provide a simple way of describing languages. If input word is the language, DFA accepts it otherwise reject it. DFA scan the string from left to right. A DFA, $M = (Q, \sum, \delta, q0\ F)$ is defined as

1. **Finite State *Q***: *Q is* a finite set of internal states.
2. **Input Alphabet** ∑: The machine only operates on words over the alphabet∑.
3. **Transition Function δ**: The transition function describes how the machine changes its internal state. It is a function $\delta: QX\Sigma \rightarrow Q$ from (state, input letter) -pairs to states. If the machine is in state *q* and next input letter is a then the machine changes its internal state to $\delta\ (q; a)$ and moves to the next input letter.
4. **Initial State *q0***: $q0 \in Q$ is the internal state of the machine before any letters have been read.
5. **Final state *F***: Set $F \subseteq Q$ of final states specifies which states are accepting and which are rejecting [7]. If the internal state of the machine, after reading the whole input, is some state of *F* then the word is accepted, otherwise rejected. The language recognized (or accepted) by a DFA *M* consists of all words that *M* accepts. This language is denoted by L (*M*).

Example 1: Consider the DFA, $M = (\{p,q,r\},\{a,b\}, \delta,p,\{r\})$**,** Where the transition function *d* is given by the table 1.1

Table 1.1: Transition Functions [23]

Input / States	a	b
p	q	p
q	r	p
r	r	r

The operation of the machine on input word *w* = *abaab* is as follows:

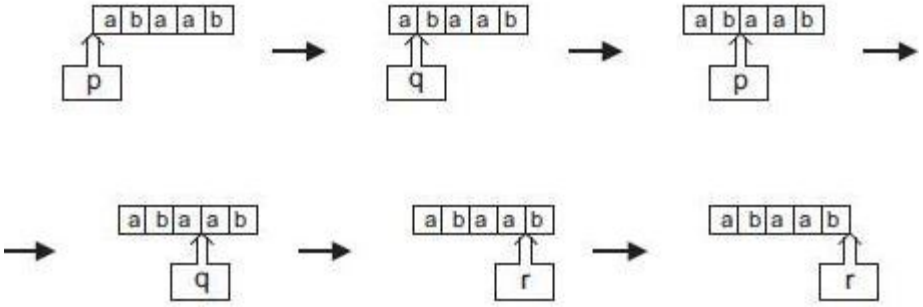

Figure 1.3: Operations on the machine [23]

The word *abaab* is accepted because *r* is a final state. A convenient way of displaying DFA is to use a transition diagram. It is a labeled directed graph where vertices are represented by different states of *Q*, and edges indicate the transitions with different input symbols. The edges are labeled with the input letters and the vertices are labeled with states. Transition $\delta\ (q,a) = p$ is represented by an arc labeled *a* going from vertex *q* into vertex *p*.

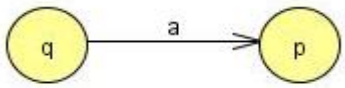

Figure 1.4: Transition from *q* to *p* [23]

Final states are indicated as double circles and the initial state is indicated by a short incoming arrow.

Example 2: Here's the transition diagram for the DFA *M* of Example 1

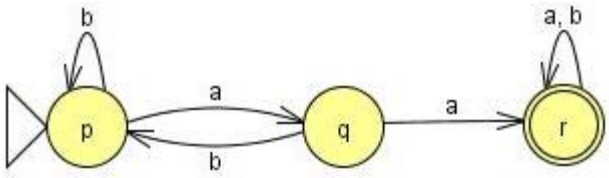

Figure 1.5: Deterministic Finite State Automata [23]

To determine whether a given word is accepted by *M* we need to follows the path labeled with the input letters, starting from the initial state. If the state where the path ends is a final state, the word is accepted. Otherwise it is rejected. In our example, path labeled with input word *w = abaab* leads to state r so the word is accepted. Input word *abba* is rejected since it leads to q which is not a final state. A DFA *M* is a finite description of the language *L(M)*. For example, the DFA in examples 1 and 2 represents the language of all words over the alphabet *{a, b}* that contain *aa* as a sub word.

2) Non-Deterministic Finite State Automaton

Non-deterministic finite automata are generalizations of DFA. Instead of exactly one outgoing transition from each state by every input letter, NFA allow several outgoing transitions at the same time. A word is accepted by the NFA if some choice of transitions takes the machine to a final state [21, 31]. Some other choices may lead to a non-final state, but the word is accepted as long as at least one accepting computation path in the automaton exists. A non-deterministic finite state automaton (NDFA) [14] is same as DFA except the transition function defined as $Q\,X\,\Sigma \text{->} 2^{Q}$.

Example 3: NFA can be represented by a transition diagram as shown in figure 1.6

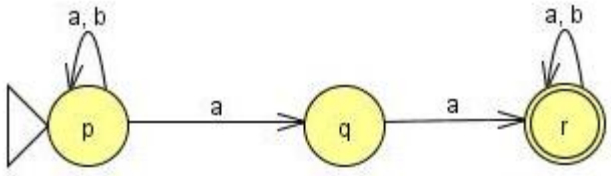

Figure 1.6: Non-Deterministic Finite State Automata [23, 31]

Note that there are two transitions from state *p* with input letter *a*, into states *p* and *q*. Note also that there are no transitions from state *q* with input letter *b*. The number of transitions may be zero, one or more. NFA may have several different computations for

the same input word. For string *w* = *abaa*, computation tree is represented in figure 1.7., Hence the string *abaa* is accepted, as one path leads to final state.

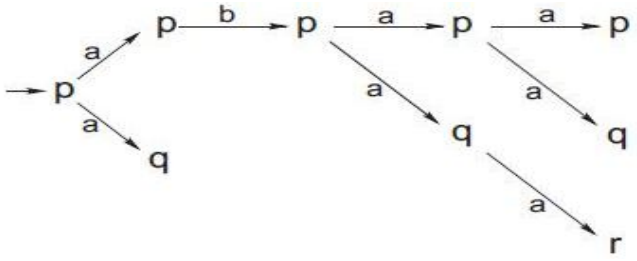

Figure 1.7: Computation Trees [23]

It is easy to see that the sample NFA accepts exactly those words that contain *aa* as a sub word. Two automata are called equivalent if they recognize the same language.

Example 4: An NFA accepting all strings that end in 01

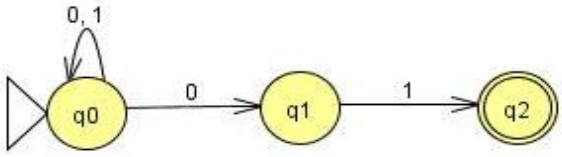

Figure 1.8: NFA for a string ending with 01 [23]

It is non-deterministic because input 0 in state *q0* can lead to both *q0* and *q1*.

1.3 Chomsky Hierarchy

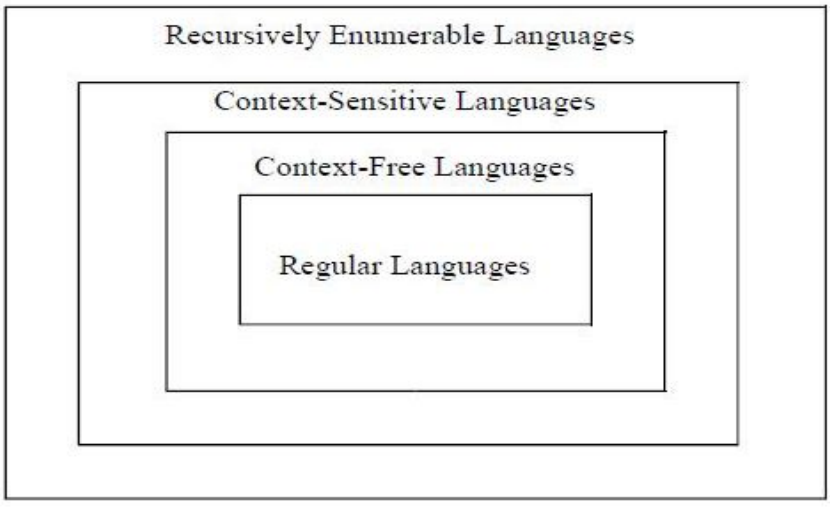

Figure 1.9: Hierarchy of languages [27]

The Chomsky hierarchy, named after linguist Noam Chomsky [22] describes a hierarchy of languages ranging from the very simple regular languages up through the very general recursively enumerable languages.

Table 1.2: Chomsky Hierarchy [22]

Languages	Grammars	Machines
Recursively Enumerable languages	Type 0: Unrestricted Grammars	Turing Machines(deterministic or nondeterministic)
Context-Sensitive Languages	Type 1: Context-Sensitive Grammars	Linear-Bounded Automata(nondeterministic)
Context-Free languages	Type 2: Context-Free Grammars	Pushdown Automata
Regular Languages	Type 3: Right-Linear Grammars	Finite Automata(deterministic or nondeterministic)

In Chomsky Hierarchy [22], different classes of languages can be defined by restrictions on the form of the productions of the grammar.

1.4 Regular Expressions

The origins of regular expressions lie in automata theory and formal language theory, both of which are part of theoretical computer science. These fields study models of computation (automata) and ways to describe and classify formal languages. In the 1950s, J. Hopcroft [13] described these models using his mathematical notation called regular sets. A regular expression is a set of characters that specify a pattern. The term regular has nothing to do with a high-fiber diet. It comes from a term used to describe grammars and formal languages.

Regular expressions are used when you want to search for specify lines of text containing a particular pattern. Most of the UNIX utilities operate on ASCII files a line at a time. Regular expressions search for patterns on a single line, and not for patterns that start on one line and end on another. It is simple to search for a specific word or string of characters. Almost every editor on every computer system can do this. Regular expressions are more powerful and flexible. Regular expressions are more powerful and flexible [9]. You can search for words of a certain size. You can search for a word with

four or more vowels that end with an "s". Literal and Meta characters are two types of characters in Regular Expressions. In Literal, every normal text character is a regular expressions and in Meta-characters, special characters that allow you to combine REs in various ways. Regular expressions are a powerful text processing component of programming languages such as PERL and Java. A regular expression is a mathematical model for describing a particular type of language.

The regular expression is defined recursively.

1) ε represent regular expression for the language $\{\varepsilon\}$.
2) $\emptyset$ represent regular expression for the empty language.
3) r_1+r_2, where r_1 and r_2 are regular expressions and + signifies union.
4) r_1r_2, where r_1 and r_2 are regular expressions and this signifies concatenation.
5) r^* where r is a regular expression and signifies closure.

Example 5: The following are some regular expressions over alphabet $\Sigma = \{a, b\}$ and the corresponding regular languages

Regular Expression	Regular Language
$a + b$	$\{a, b\}$
$(a + b)^*$	$\{ \varepsilon,a,b,aa,bb,ab...\}$
a^+	$\{a,aa,aaa,.....\}$
a^*	$\{ \varepsilon,a,aa,aaa,...\}$

1.4.1 Applications for Regular Expressions

Following are the applications of regular expressions

1) Lexical analysis
2) Regular Expressions in Web Search Engines
3) Regular Expressions in Software Engineering
4) Pattern matching

1.5 Regular Languages

There are four levels of languages according to the Chomsky hierarchy of formal languages, which are regular languages, context-free languages, context-sensitive languages and recursively enumerable languages. Regular language can be described by regular expression or finite automata (deterministic or non-deterministic).

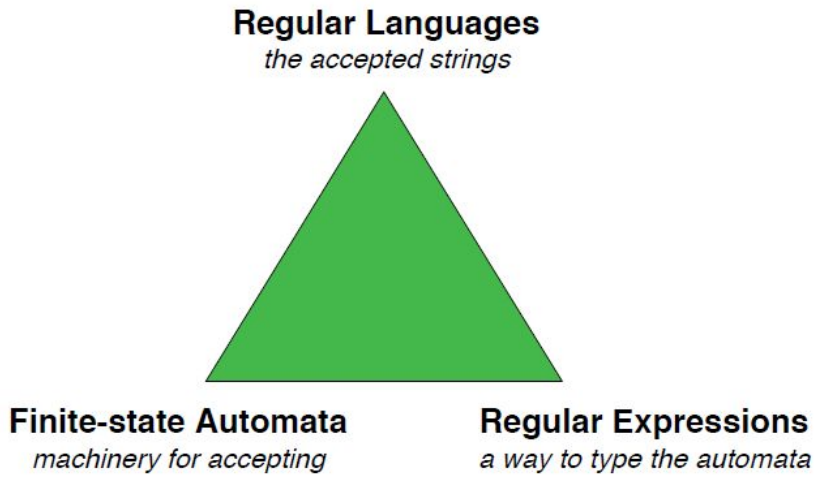

Figure 1.10: Relation between FA, RE AND RL [2]

Regular languages are used in various fields of computer science like compilers [17], data compression, text processing, software engineering, and pattern matching and many more areas. Further regular languages can be classified into infix free, prefix free and suffix free. These are used in various applications like pattern matching [2], computing forbidden words [17] and text searching. The languages represented by regular expressions are called Regular Languages. A language L is regular if and only if there is a regular expression E such that L = L(E).

1.5.1 Forms of Regular Languages

Regular language can be of prefix-free, suffix free or infix free.

1) Prefix-Free: A language L is prefix-free [30] if, for all distinct strings $x, y \in \Sigma^*$ and $x, y \in L$ imply that x and y are not prefixes of each other. A regular language is prefix-free if and only if its minimal DFA M has only one final state and the final state has no out transitions whose target state is not a sink state [32]. A sink state is a state from which there exists no sequence of transitions to a final state [33].

Example 6: Given two strings $x = abcbba$ and $y = abcbbab$. String x is in prefix of string y. The language that consists of string x and y is not prefix free.

2) Suffix-Free: A language L is suffix free [29] if, for all distinct strings $x, y \in \Sigma^*$, and $x, y \in L$ imply that x and y are not suffixes of each other. Given two strings x and y over Σ, x is a suffix of y if there exists $z \in \Sigma^*$ such that $zx = y$. A regular language is suffix-free if

and only if its minimal DFA *M* has a unique sink state and its starting state does not have any in-transitions [33].

Example 7: Given two strings $x = abb$ and $y = aaabb$. String x is suffix of string *y*. It means that the language consists of string *x* and *y* is not suffix free.

3) Infix-Free: A language *L* is infix free [28] if, for all distinct strings $x, y \in \Sigma^*$, and $x, y \in L$ imply that *x* and *y* are not substrings of each other. *x* is said to be substring or an infix of *y* if there are two strings u and v such that $uxv = y$. A regular language is infix-free if a DFA *M* is minimal and is non-returning and non-exiting.

Example 8: Given two strings $x = aba$ and $y = aabab$. String *x* is infix or substring of string *y*. It means that the language consists of string *x* and *y* is not infix free.

1.6 Operations on Regular Languages

Following operations are defined as regular languages

1) Intersection of Regular Languages

Suppose L_1 and L_2 are two regular languages [5,18], then intersection of these regular languages is given by $L_1 \cap L_2$.

Example 9: If $L_1 = \{a^n b\}$ and $L_2 = \{ab^m\}$ then $L_1 \cap L_2 = \{ab\}$.

2) Union of Regular Languages

Suppose L_1 and L_2 are two regular languages [5,18], then union of these regular languages is given by $L_1 U L_2$, which represents language that accepts all string from L_1 and L_2.

Example 10: Suppose $L_1 = \{a^n b\}$ and $L_2 = \{ab^m\}$ then $L_1 U L_2 = \{a^n b^m\}$.

3) Concatenation of Regular Languages

The Concatenation of languages L_1 and L_2 is represented by $L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$ [5]. It denotes all words that are formed by concatenation of a word *x* from L_1 with a word y from L_2.

Example 11: If $L_1 = \{a, ab\}$ and $L_2 = \{aa, bb, abb\}$ then $L_1 L_2 = \{aaa, abb, aabb, abaa, abbb, ababb\}$.

4) Kleene Closure of Regular Languages

The closure or kleene closure of a given regular language *L* is the collection of all possible finite-length strings generated from the symbols in *L* including null string [5]. It is denoted by L^*.

Example 12: If the regular language L contains alphabets Σ= *{a, b}* then L^*= *{ε, a, b, ab, ba, aa,...}* i.e. any string of *a* and *b*.

5) Complement of Regular Languages

The complement of a regular language is defined with respect to Σ^*. The complement of a regular language [5] *L* is given by $L = \Sigma^* - L$.

Example 13: Suppose Σ = *{a}* and the language *L* consists of all non empty string of even length of *a's*. *L= {aa, aaaa, aaaaaa,aaaaaaaa,……....}*. Complement of the language *L* is given by $L=\Sigma^* - L$ = *{ε, a, aaa, aaaaa ...}* i.e. *L* consists of all string of odd length of *a's* including null string.

6) Reverse of Regular Languages

Reverse [1] of a string contains all letters of a string in reverse order. Reversal of *x* is denoted by x^R. For null string, reverse of the string is also *ε*. For a language L over an alphabet ∑ [5], the reversal of *L* is denoted by L^R, and $L^R = \{x^R \mid x \in L\}$.

1.7 Thesis Outline

This thesis is organized into five chapters. Chapter 1 gives the description about the basic concepts of automata, regular expressions and regular languages. Chapter 2 describes information related to regular languages and state complexities. Chapter 3 describes the motivation behind the thesis, discusses the problem statement and its objectives. Chapter 4 determines whether closure properties of regular languages are prefix-free or suffix-free. Chapter 5 summarizes the conclusions drawn from the thesis along with the directions regarding the future work.

Chapter 2

LITERATURE SURVEY

This chapter describes the various works that have been done in field of finite state automata and regular languages.

2.1 Introduction

Regular languages are used in various fields of computer science like data compression, network intrusion detection, compilers, text processing, software engineering, pattern matching and various others fields [14]. Modern network devices need to perform deep packet inspection at high speed for security and application-specific services. Finite Automata (FAs) are used to implement regular expressions matching, but they require a large amount of memory. Nowadays, state-of-the-art systems replace string sets with regular expressions, due to their superior expressive power and flexibility, as first shown in [6]. It has been proved that DFAs corresponding to a large set of regular expressions can blow up in space, and many recent works have been presented with the aim of reducing their memory footprint. F. Yu et al. [9] develop a grouping scheme that can strategically compile a set of regular expressions into several DFAs evaluated by different engines, resulting in a space decrease, while the required memory bandwidth linearly increases with the number of active engines.

Domenico Ficara et al. [6] give a new representation for deterministic finite automata, called Delta Finite Automata (δFA). In Delta transition function, we have reduced states and transitions. It requires a transition per character only, thus allowing fast matching. A new state encoding scheme is proposed and the comprehensive algorithm is tested for use in the packet classification area.

Several works in the recent years have focused on memory reduction of DFAs to improve the speed of regular expression engines. A technique D^2FA is where an input character can require a number of additional steps through the automaton before reaching the right state.

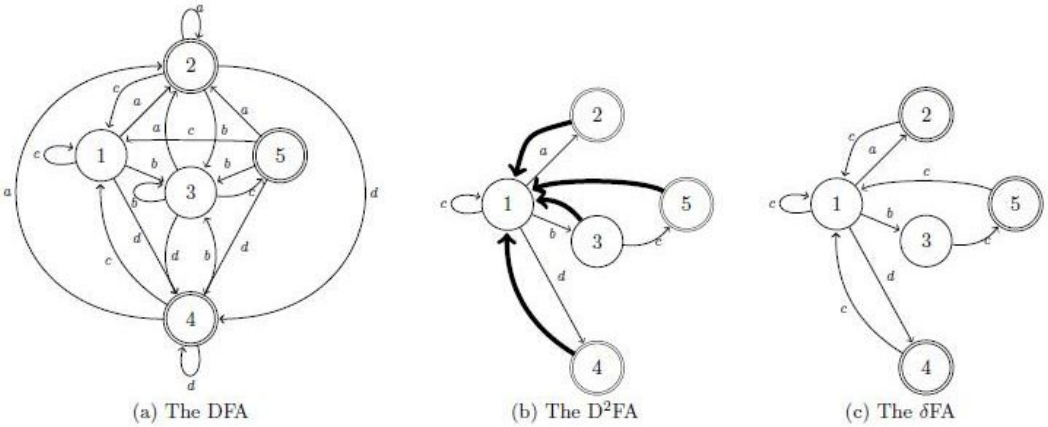

Figure 2.1: Automata recognizing (a^+),(b^+c) and ($c\Box d^+$)[6].

Junghoo Cho and Sridhar Rajagopalan [11] describe the design and architecture of a fast regular expression indexing engine FREE. FREE uses a prebuilt index to identify the text data units which may contain a matching string and only examines these further. In this way, FREE shows orders of magnitude performance improvement in certain cases over standard regular expression matching systems, such as lex, awk and grep [11].

2.2 State Complexity

State complexity is a complexity measure for regular languages[25]. The state complexity of a regular language *L* is the number of states of the minimal DFA that accepts *L*. State complexity has a fundamental research area in automata and formal language theory. The implementations as well as applications of regular language motivate the study of state complexity of regular languages. There are different types of complexity shown in figure 2.2. Individual operations includes union, intersection, catenation, Kleene star, complement and reversal. State complexities of some combined operations like star of union and intersection, star of catenation and reversal, reversal of union and intersection, reversal of catenation and star have been studied [2,3].

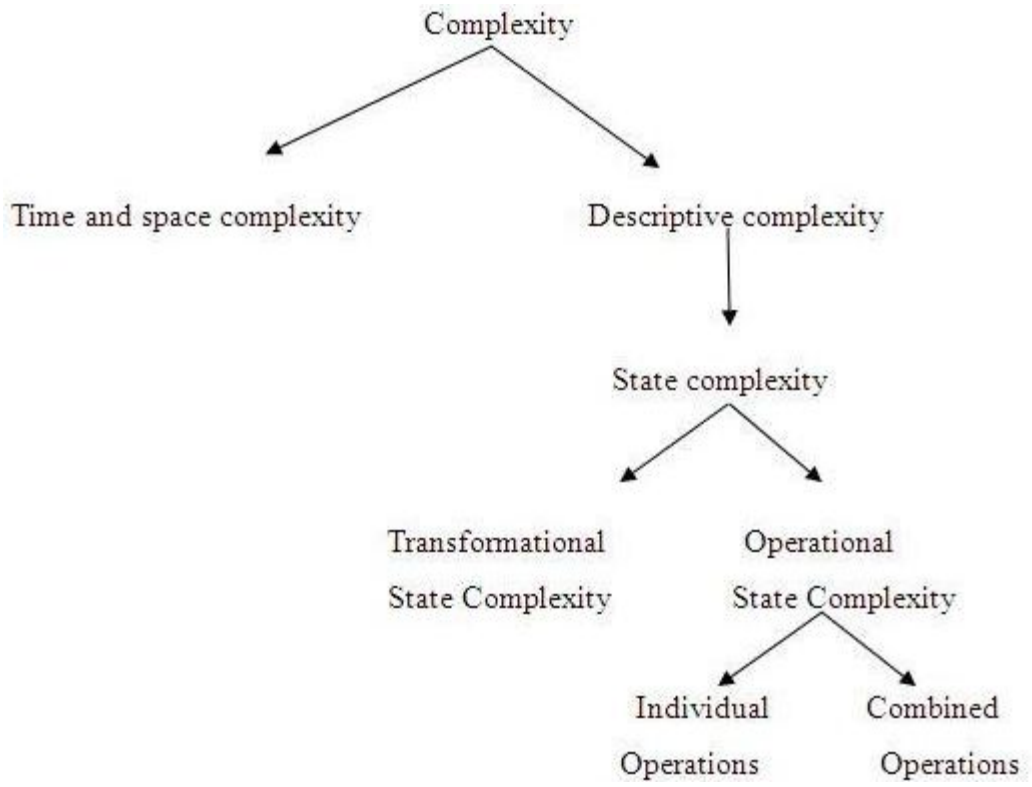

Figure 2.2: Types of Complexity[26]

Xiaoxue Piao and Kai Salomaa [26] studied the operational state complexity of two variants of deterministic unranked tree automata. For union and intersection, tight upper bounds on the number of vertical states were established for both strongly and weakly deterministic automata. An almost tight upper bound on the number of horizontal states was obtained in the case of strongly deterministic unranked tree automata. For weakly deterministic automata, lower bounds on the numbers of horizontal states are hard to establish because there can be trade-offs between the numbers of vertical and horizontal states. This is indicated by the fact that minimization of weakly deterministic unranked tree automata is intractable and the minimal automaton need not be unique [26]. For deterministic unranked tree automata, the state complexity of concatenation of an m state and an n state automaton is at most $(n+1)((m+1)2^n - 2^{n-1})-1$ and that this bound can be reached in the worst case. The bound is of a different order than the known state complexity $m2^n-2^{n-1}$ of concatenation of regular languages.

A language is prefix-free, if it does not contain any pair of words such that one is a proper prefix of the other. A language is suffix-free, if it does not contain any pair of

words such that one is a proper suffix of the other. It is bifix-free if it is both prefix- free and suffix-free.

With regard to free languages, Han, Salomaa and Wood examined prefix-free languages and suffix-free languages. Janusz Brzozowski et al.[11] studied the syntactic complexity of automata with unary and binary alphabets.Table 2.1 shows the syntactic complexity bounds of prefix-free, suffix-free, and bifix-free languages, in that order, with a particular alphabet size. The asterisk ⍰ indicates that the bound is already tight for a smaller alphabet. The last three rows include the tight upper bound n^{n-2} for prefix-free languages, conjectured upper bounds *p(n)* for suffix-free and *r(n)* for bifix-free languages, and weaker upper bounds *g(n)* for suffix-free and *h(n)* for bifix-free languages.

Table 2.1: Syntactic complexities of prefix-free, suffix-free and bifix-free languages [11]

	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$\|\Sigma\|=1$	**1**	**2**	**3**	**4**	**5**
$\|\Sigma\|=2$	$*$	**3/3/**$*$	**11/11/7**	**49/49/20**	?
$\|\Sigma\|=3$	$*$	$*$	**14/13/**$*$	**95/61/31**	?
$\|\Sigma\|=4$	$*$	$*$	**16/** $*$ **/**$*$	**110/67/32**	?
$\|\Sigma\|=5$	$*$	$*$	$*$	**119/73/33**	?
$\|\Sigma\|=6$	$*$	$*$	$*$	**125/** ? **/34**	? /501/ ?
$\cdots$					
$n^{n-2}/p(n)/r(n)$	**1/1/1**	**3/3/2**	**16/13/7**	**125/73/34**	**1296**/501/209
Suffix-free : $g(n)$	1	3	17	145	1,649
Bifix-free : $h(n)$	1	2	7	43	381

State complexity is defined as the number of distinct quotients of the language and it is also known as “quotient complexity”. Consider the problem of finding the quotient complexity of a language f(K,L) for union or concatenation, where K and L are regular languages and f is a regular operation. Since quotients can be represented by derivatives, we can find a formula for the typical quotient of f(K,L) in terms of the quotients for K and L. To obtain an upper bound[14.31] on the number of quotients of f(K,L) all one has to do is count how many such quotients are possible, and this makes automaton constructions unnecessary. Moreover, new general observations are presented to help in the estimation of the upper bounds on quotient complexity of regular operations. Quotients supply a uniform approach for finding upper bounds for the complexity of operations on regular languages and verifying that particular languages meet these

bounds. It is hoped that this is a step towards a theory of complexity of languages and automata [14].

Y.-S. Han et al. investigate the non-deterministic state complexity of basic operations for prefix-free regular languages. If a minimal NFA *M* is prefix-free, then *M* has only one final state and non-exiting [24, 25]. The non-deterministic state complexity of an operation is the number of states that are necessary and sufficient in the worst-case for a minimal nondeterministic finite-state automaton that accepts the language obtained from the operation. The precise state complexity of catenation, union, intersection, Kleene star, reversal and complementation for prefix-free regular languages are shown in table 2.2.

Table 2.2: State Complexity of Basic Operations of Prefix-Free DFAs and NFAs [30]

Operations	Prefix-free DFAs	Prefix-free NFAs
$L_1 . L_2$	$m + n - 2$	$m + n - 1$
$L_1 \cup L_2$	$mn - 2$	$m + n$
$L_1 \cap L_2$	$mn - 2(m + n) + 6$	$mn - (m + n) + 2$
L_1^*	m	m
L_1^R	$2^{m-2} + 1$	m
L_1^c	m	2^{m-1} or $2^{m-1} + 1$

Y.-S. Han and K. Salomaa [31] investigate the state complexity of basic operations for suffix-free regular languages. The state complexity of an operation for regular languages is the number of states that are necessary and sufficient in the worst-case for the minimal deterministic finite-state automaton that accepts the language obtained from the operation [18, 29].

Han et al. [30] examined the state complexity of prefix-free regular languages. They tackled the problem based on the structural property of prefix-free DFAs. A prefix-free DFA must be non-exiting assuming all states are useful. State complexity for the prefix-free case is strictly less than the corresponding state complexity for regular languages over some basic operations. If a language L is prefix-free, then its reversal L^R is suffix-free. Moreover, if L is regular and non-empty, then the starting state of L^R should not have any in-transitions.

Table 2.3: State Complexity of Basic Operations of Suffix-Free DFAs and NFAs [29]

Operations	Suffix-free DFAs	Suffix-free NFAs
$L_1 . L_2$	$(m-1)2^{n-2}+1$	$m+n-1$
$L_1 \cup L_2$	$mn-(m+n)+2$	$m+n-1$
$L_1 \cap L_2$	$mn-2(m+n)+6$	$mn-2(m+n)+2$
L_1^*	$2^{m-2}+1$	m
L_1^R	$2^{m-2}+1$	$m+1$
L_1^c	m	$2^{m-1} \pm 1$

Han [28] proposed an approach for checking infix-freeness of non-deterministic finite automata using following steps:

1) Regular expression into NFA

2) Construct state pair graph

3) Using state pair graphs to check whether language is infix-free or not. Han et al. [28] determined the state complexity of various basic operations (union, intersection, concatenation, reversal, complement, kleene closure) on prefix free regular languages and suffix free regular languages.

Table 2.4: State Complexity of Basic Operations of Infix-Free DFAs [28] and NFAs

Operations	Infix-free DFAs	Infix-free NFAs
$L_1 . L_2$	$m+n-2$	$m+n-1$
$L_1 \cup L_2$	$mn-(m+n)$	$m+n-2$
$L_1 \cap L_2$	$mn-3(m+n)+12$	$mn-2(m+n)+6$
L_1^*	$n-1$	$n-1$
L_1^R	$2^{n-3}+2$	n
L_1^c	n	$2^{n-2}+2$

Consider we have two minimal DFA with state *m* and *n* recoginizing finite languages.Yo-Sub Han and Kai Salomaa [34] have proved that for union and intersection *mn-(m+n)* and *mn-3(m+n)+12* states are necessary and sufficient.

Bo Cui [3] has studied the state complexities of reversal combined with concatenation and star combined with concatenation. Due to the structural properties of DFAs obtained from reversal and star, the state complexities of these two combined operations are not equal [12, 20].

2.3 Closure Properties of Regular Languages

A regular language is way to represent the string or words of a given language. If r is a regular expression then, L(r) is a languages generated by the regular expression. Regular language is closed under union, kleene closure and concatenation by the definition of regular expression.

Preposition 2.1: Regular languages are closed under complementation [8].

Proof: A DFA for L^C is obtained from M by using following rules:

1. Make the accepting state to non accepting state.
2. Make the non accepting state to accepting state.

Preposition 2.2: Regular languages are closed under intersection [8].

Proof: Let L_1 and L_2 are two regular languages represented by M and N DFA respectively

$L_1 \cap L_2 = (L_1^C + L_2^C)^C$

By using preposition 2.1, L_1^C and L_2^C are regular languages. Therefore, $(L_1^C + L_2^C)$ is also a regular languages. Hence, $(L_1^C + L_2^C)^C$ i.e. $L_1 \cap L_2$ is a regular languages.

Chapter 3

PROBLEM STATEMENT

This chapter includes the problem statement and objectives followed by the motivation behind the thesis.

3.1 Problem Statement

The given problem relates to the closure properties of prefix-free and suffix-free regular languages. Yo-Sub Han [32] investigated the deterministic and non-deterministic state complexity of basic operations for prefix-free regular language. If a minimal DFA/NFA *M* is prefix-free, then *M* has only one final state and *M* is non-exiting. Yo-Sub Han and Kai Salomaa [29, 33] investigated the state complexity of basic operations for suffix-free regular language. The State complexity of basic operation for regular languages is the number of states that are necessary and sufficient in the worst-case for the minimal deterministic finite state automaton that accepts the language obtained from the operation. State complexity of Infix-free, prefix-free and suffix-free regular expressions have been developed. It is natural to investigate the closure properties of various forms of regular languages.

3.2 Thesis Objectives

Regular expression used in various fields such as linguistics, computational biology, pattern recognition, text retrieval etc. An elegant theory gives the support to easily and efficiently solve many complex problems by mapping them to regular expressions, then obtaining a non-deterministic finite automaton (NFA) that recognizes it, and finally converting it into a deterministic (a DFA).

Following points represent the main objectives for the work done in this thesis report:

1. Study the various forms of regular languages.
2. Closure properties of prefix-free regular languages.
3. Closure properties of suffix-free regular languages.

3.3 Thesis Motivation

There exist different approaches to find state complexities of catenation, kleene star, reversal and the boolean operations for suffix-free and prefix-free regular languages. The state complexity of an operation for regular languages is the number of states that are necessary and sufficient in the worst-case for the minimal deterministic finite-state automaton that accepts the language obtained from the operation. Motivated by the successful application of the theory of regular languages to formal verification of finite-state systems, there is a renewed interest in closure properties of prefix-free and suffix-free regular languages.

Chapter 4

Closure Properties of Prefix-Free and Suffix-Free Regular Languages

This chapter contains the discussion of various properties of prefix-free and suffix-free regular languages.

4.1 Closure Properties of Prefix-Free Regular Languages

A regular language is prefix-free if and only if its minimal DFA *M* has no more than one final state and the final state has no out-going transitions whose target state is not a sink state. A sink state is a state from which there exists no progression of transitions to a final state.

4.1.1 Union Operation on Prefix-Free Regular Languages

Theorem 1: Prefix-free regular languages are closed under union operation.

Proof: Suppose L_1 and L_2 are prefix-free regular languages represented by DFA $M_1 (Q_1, \sum_1, \delta_1, q_{01}, F_{01})$ and $M_2 (Q_2, \sum_2, \delta_2, q_{02}, F_{02})$.

Since both M_1 and M_2 contains a single final state and no out-going transition from their final states. $L_1 \cup L_2$ is found by adding a new starting state q_0 to $q_{01}$and q_{02}. Final states of M_1 and M_2 are combined and there is no transition from the final state.

After removing null transitions, we obtain a DFA representing a single final state and no out-going transition from the final state. Hence, the resulting language L is prefix-free regular language. We can say that prefix-free regular languages are closed under union.

Example 4.1: Given $L_1 = \{ab\}$ and $L_2 = \{aabb\}$ are prefix-free regular languages, then $L = L_1 \cup L_2$ is also prefix-free regular languages. L_1 and L_2 can be represented by figure 4.1 and 4.2 respectively.

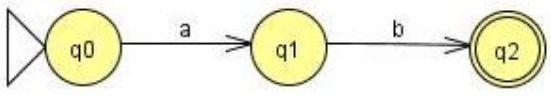

Figure 4.1: DFA for $L_1 = \{ab\}$

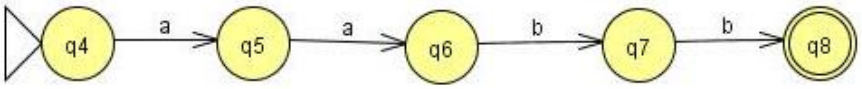

Figure 4.2: DFA for $L_2 = \{aabb\}$

Union of L_1 and L_2 is represented by a DFA as shown in figure 4.3.

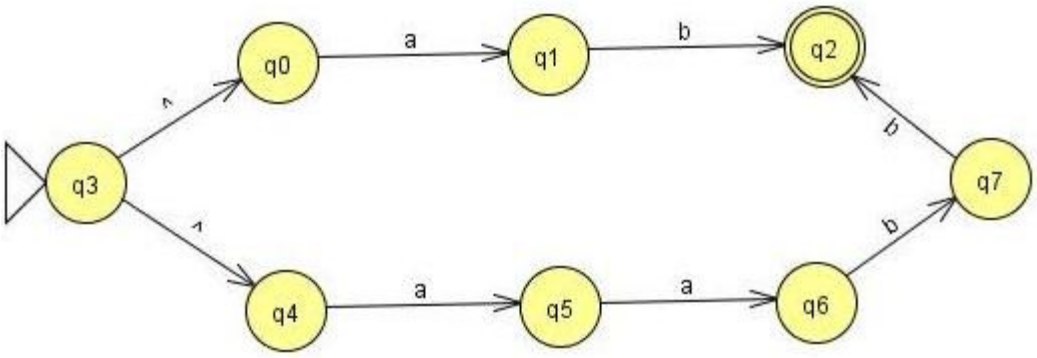

Figure 4.3: Union of L_1 and L_2

After removal of ε-transition and minimization of DFA, we obtain the DFA as shown in figure 4.4.

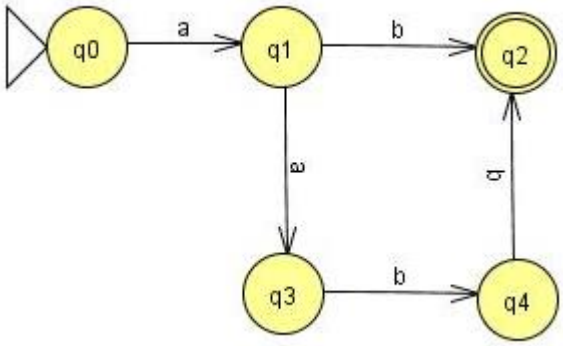

Figure 4.4: Union after ε-transition

Clearly, $L = L_1 \ U \ L_2$ is a prefix-free regular language as there is no out-going transition from the final state.

4.1.2 Concatenation Operation on Prefix-Free Regular Languages

Theorem 2 Prefix-free regular languages are closed under concatenation operation.

Proof: Suppose L_1 and L_2 are prefix-free regular languages represented by DFA $M_1 (Q_1, \sum_1, \delta_1, q_{01}, F_{01})$ and $M_2 (Q_2, \sum_2, \delta_2, q_{02}, F_{02})$.

Since both M_1 and M_2 contains a single final state and no out-going transition from their final states. L_1 and L_2 are obtained by merging the final state of M_1 and starting state of M_2. It will not cause any problem as final state of M_1 is not having any out-going transition. We can say that prefix-free regular languages are closed under concatenation.

Example 4.2: Given $L_1 = \{ab\}$ and $L_2 = \{ba\}$ are prefix-free regular languages, then $L = L_1 L_2$ is also prefix-free regular languages. L_1 and L_2 can be represented by figure 4.5 and 4.6 respectively.

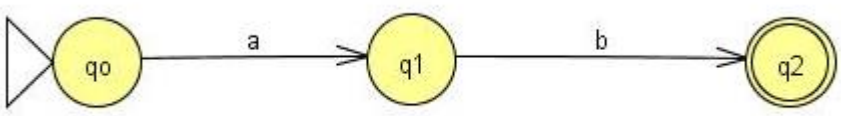

Figure 4.5: Language $L_1 = \{ab\}$

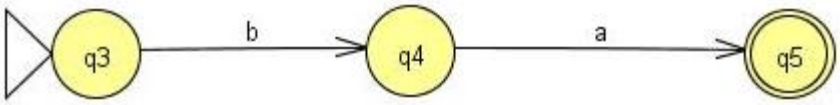

Figure 4.6: Language $L_2 = \{ba\}$

Concatenation of L_1 and L_2 is represented by a DFA as shown in figure 4.7.

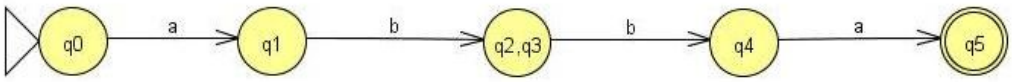

Figure 4.7: $L_1. L_2 = \{abba\}$

We just have to merge the final state of the language L_1 and the starting state of the language L_2. For simplification, Let $q6 = \{q2, q3\}$.

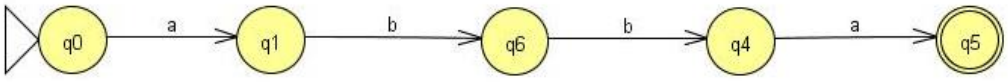

Figure 4.8: Final diagram of $L_1. L_2 = \{abba\}$

Clearly, $L = L_1.L_2$ is a prefix-free regular language as there is no out-going transition from the final state.

4.1.3 Complementation Operation on Prefix-Free Regular Languages

Theorem 3 Prefix-free regular language is not closed under complementation operation.

Proof: We prove it by giving an example, in which complement of a prefix-free regular language is not a prefix-free.

Example 4.3: Let *L* be a prefix-free regular language represented as shown in figure.

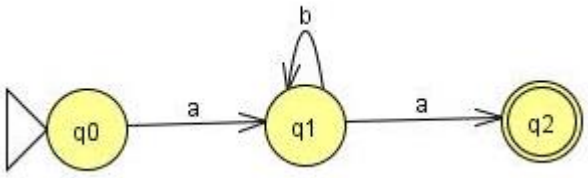

Figure 4.9: L = {ab*a}

In complement operation, all the strings which are accepting in original language become non-accepting and all non-accepting strings become accepting strings.

For complement, we perform following tasks:

1. Add an additional state called dead state (q_d).
2. Repeat for each state (q) of the DFA except q_d .
   2.1 Find the alphabet which is not having out-transition from q.
   2.2 Add an edge with out- transition from q to dead state with labeled the input found from step 2.1.
3. Convert accepting state into non- accepting state.
4. Convert non- accepting state into accepting state.

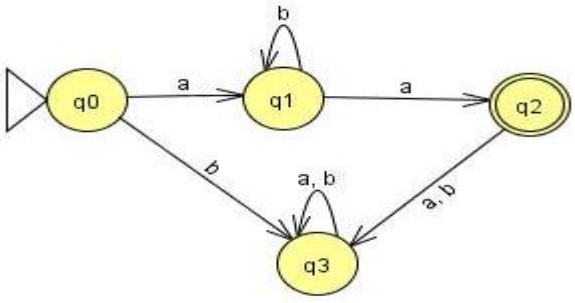

Figure 4.10: *L* with a dead state

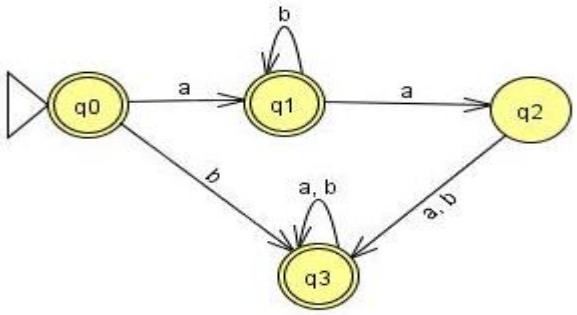

Figure 4.11: L^c with a dead state

Clearly, DFA representing L^c is not a prefix-free regular language.

4.1.4 Intersection Operation on Prefix-Free Regular Languages

Theorem 4 Prefix-free regular languages are not closed under intersection operation.

Proof: We proof it by contradiction. Let us assume intersections of L_1 and L_2 prefix-free regular language are closed. Then, $L_1 \cap L_2 = (L_1^C + L_2^C)^C$ where L_1^C represent complement of L_1.

By theorem 4.3, we have proved that a language is not closed under complement. It implies that $L_1 \cap L_2$ may or may not be a prefix-free regular language. Hence for intersection operation, regular languages are not closed.

4.1.5 Kleene closure Operation on Prefix-Free Regular Languages

Theorem 5 Prefix-free regular languages are not closed under kleene closure operation.

Proof: Let L be a prefix-free regular language represented by a DFA $M\ (Q, \sum, \delta, q_0, F)$.

We proof it by giving an example in which kleene closure of a prefix-free regular language is not a prefix-free regular language.

Example 4.4: Let $L_1 = \{ab\}$ represented by DFA M as shown in figure 4.12.

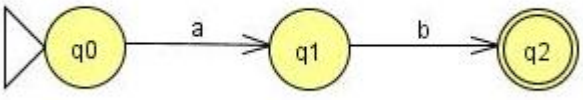

Figure 4.12: kleene closure $L_1 = \{ab\}$

Kleene closure of M can be obtained as shown in figure 4.13.

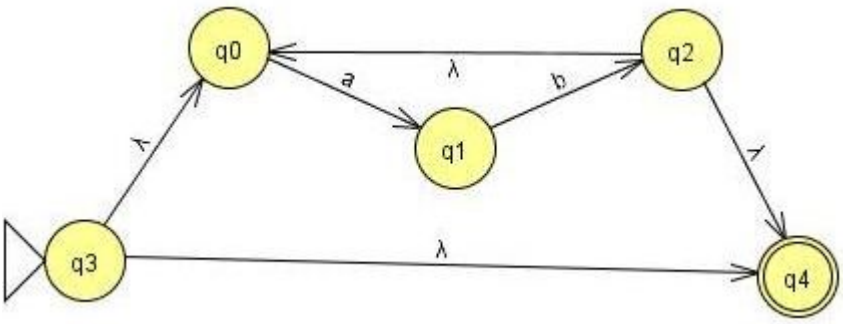

Figure 4.13: DFA for L^*

ε- Closure (q_3) = {q_3, q_0, q_4} = *A*

In reading *a* on state q_0 from *A*, we reach to q_1. ε- Closure (q_1) = { q_1} = *B*

We can able to read *b* on *A* as there is no out transition from q_0, q_3 and q_4 labeled *b*.

Reading of *a* on state *B* is not possible as there is no out transition from q_1 labeled *a*.

In reading *b* on state *B*, we reach to q_2. ε- Closure (q_2) = {q_0, q_2, q_4} = *C*

Similarly, on reading *a* on state *C* we reach to state *B*.

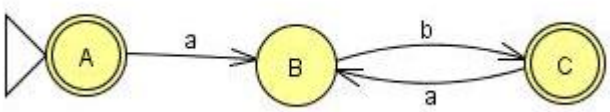

Figure 4.14: DFA with ε- Closure

Clearly, under kleene closure prefix-free regular languages are not closed.

4.1.6 Reversal Operation on Prefix-Free Regular Languages

Theorem 6 Prefix-free regular languages are not closed under reversal operation.

Proof: Let *L* be a prefix-free regular language represented by a DFA *M (Q, ∑, δ, q_0, F)*.

We proof it by giving an example in which reversal of a prefix-free regular language is not a prefix-free regular language.

Example 4.5: Let L_1 = *{ abc }* represented by DFA *M* as shown in figure 4.15.

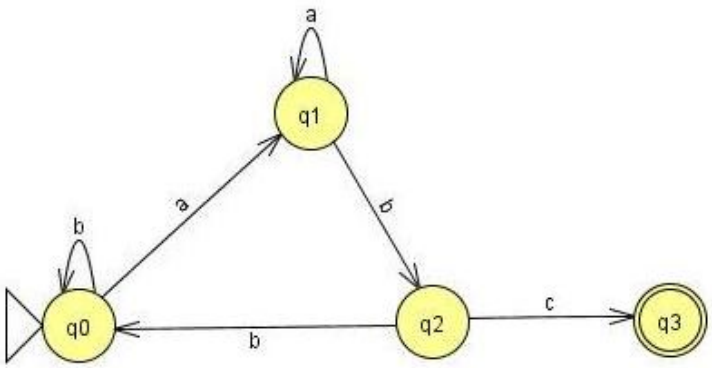

Figure 4.15: $L_1 = (abc)$

Reversal of *M* can be obtained as shown in figure 4.16.

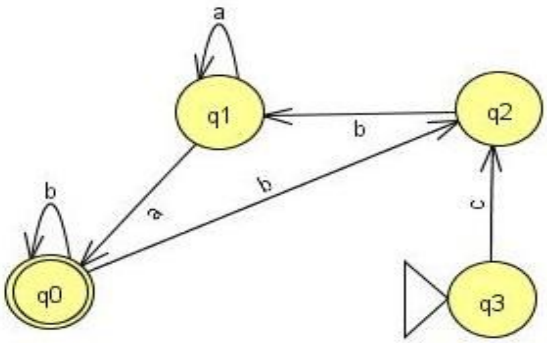

Figure 4.16: Reversal of $(L_1)^R = \{cba\}$

Hence for reversal operation, regular languages are not closed.

Table 4.1: Closure Properties of Prefix-Free Regular Languages

Closure properties of regular languages	Results with Prefix-free
Union	Yes
Concatenation	Yes
Complement	No
Intersection	No
Kleene closure	No
Reversal	No

Prefix-free regular languages are closed under union and concatenation but not closed under complementation, intersection, kleene closure and reversal.

4.2 Closure Properties of Suffix-Free Regular Languages

A regular language is suffix-free if and only if its minimal DFA have an exclusive sink state and its initial state does not have any in-coming transitions.

4.2.1 Union Operation on Suffix-Free Regular Languages

Theorem 7: Suffix-free regular languages are closed under union operation.

Proof: Suppose L_1 and L_2 are suffix-free regular languages represented by DFA $M_1 (Q_1, \sum_1, \delta_1, q_{01}, F_{01})$ and $M_2 (Q_2, \sum_2, \delta_2, q_{02}, F_{02})$.

Since both M_1 and M_2 contains a single final state and no in-coming transition from their initial states. $L_1 \cup L_2$ is found by adding a new starting state q_0 to $q_{01}$and q_{02}. Final states of M_1 and M_2 are combined and there is no in-coming transition from the initial state.

We can say that suffix-free regular languages are closed under union.

Example 4.6: Given $L_1 = \{abc^*\}$ and $L_2 = \{j^+k\}$ are suffix-free regular languages, then $L = L_1 \cup L_2$ is also suffix-free regular languages. L_1 and L_2 can be represented by figure 4.17 and 4.18 respectively.

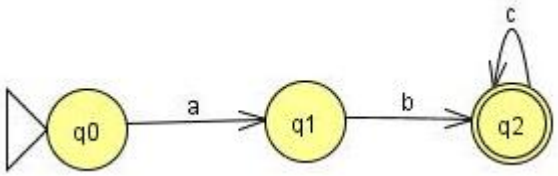

Figure 4.17: $L_1 = \{abc^*\}$

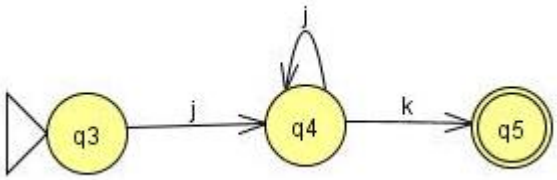

Figure 4.18: $L_2 = \{j^+k\}$

Union of L_1 and L_2 is represented by a DFA as shown in figure 4.19.

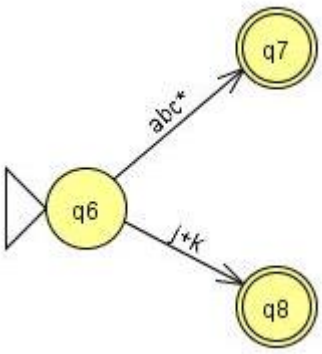

Figure 4.19: Union of L_1 and L_2

After breaking both L_1 and L_2, we get

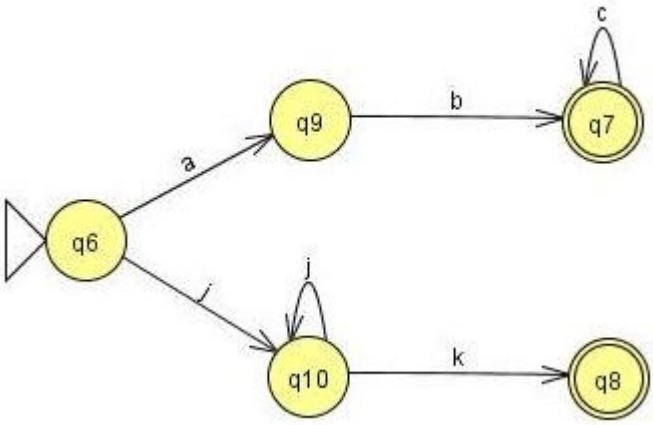

Figure 4.20: $L_1UL_2 = \{abc^* U j^+k\}$

Clearly, $L = L_1 U L_2$ is a suffix-free regular language as there is no in-coming transition from the initial state.

4.2.2 Concatenation Operation on Suffix-Free Regular Languages

Theorem 8 Suffix-free regular languages are closed under concatenation operation.

Proof: Suppose L_1 and L_2 are Suffix-free regular languages represented by DFA $M_1 (Q_1, \sum_1, \delta_1, q_{01}, F_{01})$ and $M_2 (Q_2, \sum_2, \delta_2, q_{02}, F_{02})$.

Since both M_1 and M_2 contains a single final state and no in-coming transition from their initial states. L_1 and L_2 are obtained by merging the final state of M_1 and starting state of M_2. It will not cause any problem as initial state of M_1 is not having any in-coming transition. We can say that suffix-free regular languages are closed under concatenation.

Example 4.7: Given $L_1 = \{x \mid y\}$ and $L_2 = \{z\}$ are suffix-free regular languages, then $L = L_1 L_2$ is also suffix-free regular languages. L_1 and L_2 can be represented by figure 4.21 and 4.22 respectively.

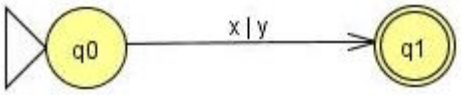

Figure 4.21: $L_1 = \{x \mid y\}$

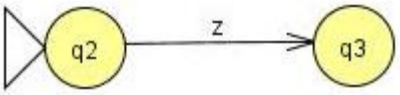

Figure 4.22: $L_2 = \{z\}$

Concatenation of L_1 and L_2 is represented by a DFA as shown in figure 4.23.

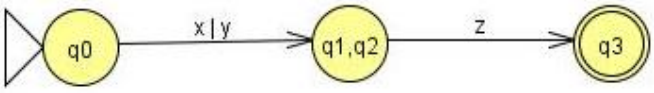

Figure 4.23: $L_1 . L_2 = (x \mid y)z$

Let q4 = {q1, q2}

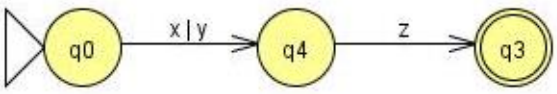

Figure 4.24: After simplification of (x | y)z

Clearly, $L = L_1.L_2$ is a suffix-free regular language as there is no in-coming transition from the initial state.

4.2.3 Complementation Operation on Suffix-Free Regular Languages

Theorem 9 Suffix-free regular languages are closed under complementation operation.

Proof: We prove it by giving an example, in which complement of a suffix-free regular language is a suffix -free. In complement operation, all the strings which are accepting in original language become non-accepting and all non-accepting strings become accepting strings. For complement, we perform following tasks:

1. Add an additional state called dead state (q_d).
2. Repeat for each state (q) of the DFA except q_d .
3. Convert accepting state into non- accepting state.
4. Convert non- accepting state into accepting state.

Example 4.8: Given *L* are suffix-free regular languages, then L^c is also suffix-free regular languages. *L* and L^c can be represented by figure 4.25and 4.26 respectively. Figure 4.25 shows a DFA that only accepts the string *aba* and *abb.*

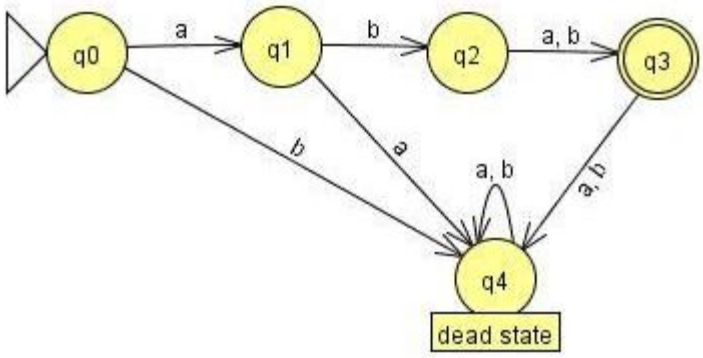

Figure 4.25: String that accepts only *aba* and *abb*

Figure 4.26 shows a DFA that accepts all the strings other than *aba* and *abb*.

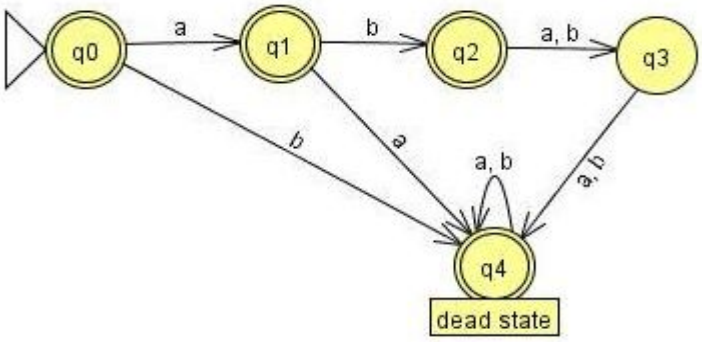

Figure 4.26: String that does not accepts only *aba* and *abb*

Clearly, DFA representing L^c is a suffix-free regular language.

4.2.4 Intersection Operation on Suffix-Free Regular Languages

Theorem 10 Suffix-free regular languages are closed under intersection operation.

Proof: We proof it by contradiction. Let us assume intersections of L_1 and L_2 suffix-free regular language are not closed. Then, $L_1 \cap L_2 = (L_1^C + L_2^C)^C$ where L_1^C represent complement of L_1.

By theorem 9, we have proved that a language is closed under complement. It implies that $L_1 \cap L_2$ are suffix-free regular language. Hence for intersection operation, regular languages are closed.

Example 4.9: Given $L_1 = \{a^n\ b\}$, n>=0 and $L_2 = \{a\ b^m\}$, m>=0 are suffix-free regular languages, then $L = L_1 \cap L_2$ is also suffix-free regular languages. L_1 and L_2 can be represented by figure 4.27 and 4.28 respectively.

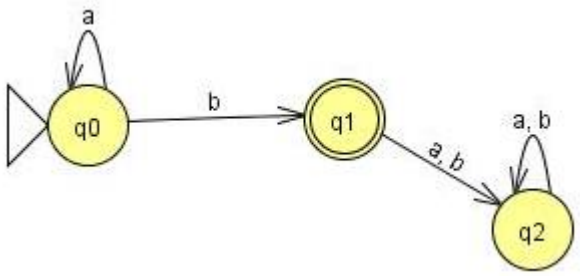

Figure 4.27: $L_1 = \{a^n\ b\}$

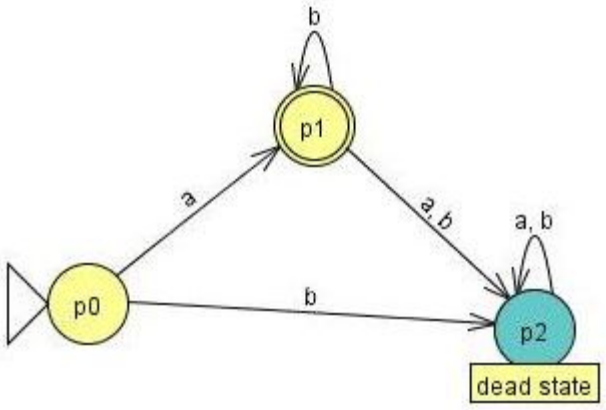

Figure 4.28: $L_1 = \{a\ b^m\}$

Intersection of L_1 and L_2 is represented by a DFA as shown in figure 4.29.

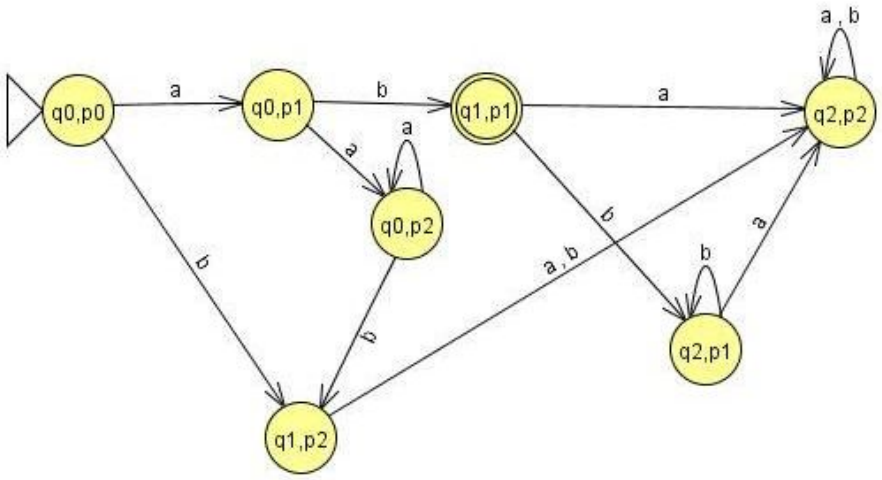

Figure 4.29: Suffix-free of $L_1 \cap L_2 = \{ab\}$

Clearly, DFA representing $L_1 \cap L_2$ is a suffix-free regular language.

4.2.5 Kleene closure Operation on Suffix-Free Regular Languages

Theorem 10 Suffix-free regular languages are closed under kleene closure operation.

Proof: Let L be a suffix-free regular language represented by a DFA $M\ (Q, \sum, \delta, q_0, F)$. We proof it by giving an example in which kleene closure of a suffix-free regular language is a suffix-free regular language.

Example 4.10: Given a languages $L_1 = \{a\}$ and $L_2 = (bc)$ which is suffix-free. L_1 and L_2 can be represented by figure 4.30and 4.31 respectively.

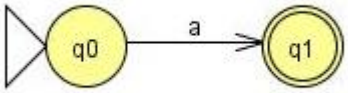

Figure 4.30: $L_1 = \{ a\}$

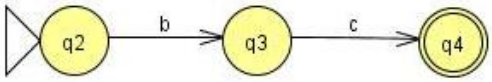

Figure 4.31: $L_2 = (bc)$

Union of L_1 and L_2 is represented by a DFA as shown in figure 4.32.

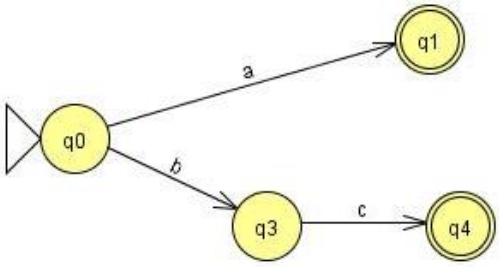

Figure 4.32: Union of (*a U bc)*

Kleene closure of *M* can be obtained as shown in figure 4.33.

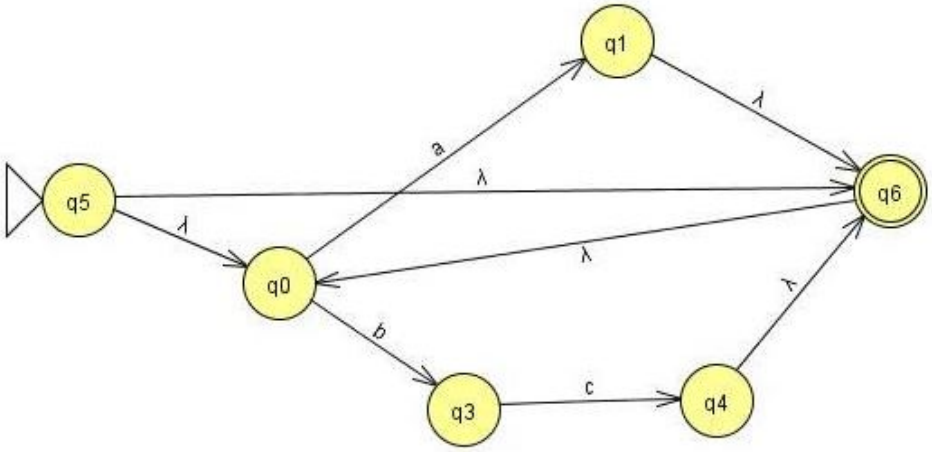

Figure 4.33: Kleene-star of *(a U bc)**

We can say that suffix-free regular languages are closed under kleene closure operation. Because there is no incoming transition possible through the initial state of the language.

4.2.6 Reversal Operation on Suffix-Free Regular Languages

Theorem 11 Suffix-free regular languages are closed under reversal operation.

Proof: Let *L* be a suffix-free regular language represented by a DFA $M\ (Q, \sum, \delta, q_0, F)$. We proof it by giving an example in which reversal of a suffix-free regular language is a suffix-free regular language.

Example 4.11: Given *L* are suffix-free regular languages, then L^R is also suffix-free regular languages. *L* and L^R can be represented by figure 4.34 and 4.35 respectively.

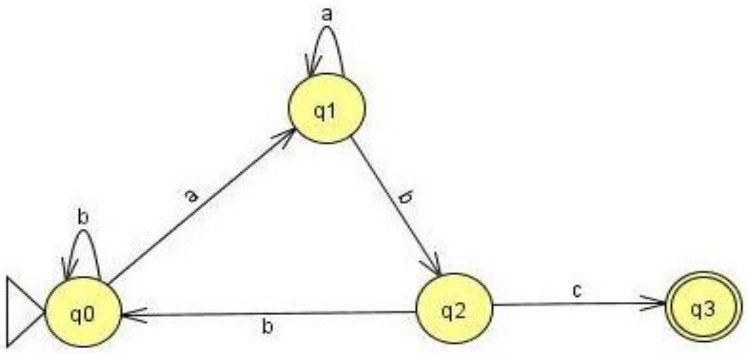

Figure 4.34: *L* is the suffix-free language

Reversal of *M* can be obtained as shown in figure 4.35.

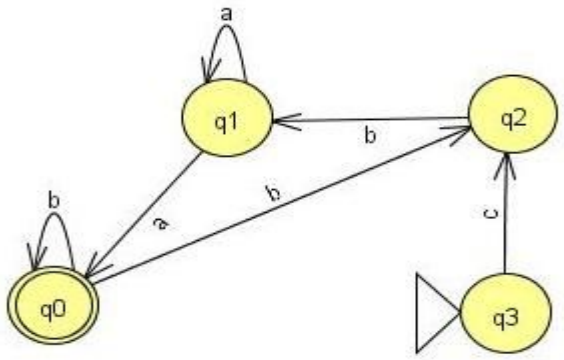

Figure 4.35: $(L_1)^R$ is the suffix-free language

We can say that suffix-free regular languages are closed under reversal operation.

Table 4.2: Closure Properties of Suffix-Free Regular Languages

Closure properties of regular languages	Results with suffix-free
Union	Yes
Concatenation	Yes
Complement	Yes
Intersection	Yes
Kleene closure	Yes
Reversal	Yes

Suffix-free regular languages are closed under all operations i.e. union, concatenation, complementation, intersection, kleene closure and reversal.

Chapter 5

Conclusion and Future Scope

Regular languages are used in a various fields of computer science such as data compression, network intrusion detection, compilers, text processing, software engineering, pattern matching etc. Regular languages are categorized into infix free, prefix free and suffix free. In this thesis work, various properties of prefix-free and suffix-free regular languages are studied. In prefix-free, regular languages are closed under union and concatenation but regular languages are not closed under complementation, intersection, kleene closure and reversal operations. Similarly, suffix-free regular language are closed under union, concatenation complementation, intersection, kleene closure and reversal operations.

Following are the future directions in which work can be carried out:

1) Closure properties of infix-free regular languages.

2) To determine state complexity of various combined operations of different form of regular languages.

References

[1] A. Salomaa, D. Wood and S. Yu: "*On the state complexity of reversals of regular languages*", Theoretical Computer Science, pp.315–329, 2004.

[2] A. Salomaa, K. Salomaa and S. Yu: "*State complexity of combined operations*", Theoretical Computer Science, vol. 383, pp. 140–152, 2007.

[3] Bo Cui, Yuan Gao, Lila Kari and Sheng: "*State Complexity of Two Combined Operations: Reversal-Catenation and Star-Catenation*", 2010.

[4] B. Krawetz, J. Lawrence and Shallit: "*State complexity and the monoid of transformations of a finite set*", 2003.

[5] C. Campeanu, K. Culik II, K. Salomaa and S. Yu: "*State complexity of basic operations on finite languages*", In Proceedings of WIA '99, Lecture Notes in Computer Science 2214, pp. 60-70, 2001.

[6] Domenico Ficara, Stefano Giordano, Gregorio Procissi, Fabio Vitucci, Gianni Antichi and Andrea Di Pietro: "*An Improved DFA for Fast Regular Expression Matching*", 2008.

[7] D. Wood: "*Theory of Computation*", New York, 1987.

[8] D. Goswami and K. V. Krishna: "*Properties of Regular Languages*", 2012.

[9] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman and R. H. Katz: "*Fast and memory-efficient regular expression matching for deep packet inspection*", In Proceedings of ANCS, pp. 93-102, 2006.

[10] H. Gruber and S. Gulan: "Simplifying *regular expressions: A quantitative perspective*", 2009.

[11] Junghoo Cho and Sridhar Rajagopalan: "*A Fast Regular Expression Indexing Engine*", Proceedings of the 18th International Conference on Data Engineering pp.1063-6382, 2002.

[12] J. Jirasek, G. Jiraskova and A. Szabari: "*State complexity of concatenation and complementation of regular languages*", Springer, Heidelberg, pp. 178–189, 2005.

[13] J. Hopcroft, R. Motwani and J. Ullman: *"Introduction to Automata theory, Languages, and Computation*", Publication Addison-Wesley, Second edition, 2001.

[14] Janusz Brzozowski: "*Quotient Complexity of Regular Languages*", 2009.

[15] K.L.P. Mishra & N. Chandrasekaran: "*Theory of Computer Science (Automata Language and. Computation)*", Second edition, (1998).

[16] K. Salomaa and S. Yu: "*On the state complexity of combined operations and their estimation*", International Journal of Foundations of Computer Science, vol. 18, pp. 683–69, 2007.

[17] M. Holzer and Konig: "*On deterministic finite automata and syntactic monoid size*", Theory of Computer Science, pp. 319 – 347, 2004.

[18] M. Hricko, G. Jiraskova and A. Szabari: "*Union and intersection of regular languages and descriptional complexity*", Proceedings of DCFS, pp. 170–181, 2005.

[19] M. Holzer and M. Kutrib: "*Nondeterministic finite automata*", Proceedings of CIAA*, pp*. 1–16, 2008.

[20] M. Holzer and M. Kutrib: "*Descriptional and computational complexity of finite automata*", Proceeding of LATA, Springer, pp.23–42, 2009.

[21] M. Parthasarathy and M. Fleck: "*DFA Minimization",* University of Illinois at Urbana-Champaign, 2007.

[22] Noam Chomsky: "*On Certain Formal Properties of Grammars*", Information and Control, Vol. 2, pp.137-167, 1959.

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

[24] R. Meyer and M. J. Fischer: "*Economy of description by automata, grammars and formal systems*", In Proceedings of the 12th Annual IEEE Symposium on Switching and Automata Theory, pp. 188–191, 1971.

[25] S. Yu, Q. Zhuang and K. Salomaa: "*The state complexities of some basic operations on regular languages*", Theoretical Computer Science, pp. 315-328, 1994.

[26] Xiaoxue Piao and Kai Salomaa: "*Operational State Complexity of Deterministic Unranked Tree Automata*", In Proceedings of the 12th International Workshop on Descriptional Complexity of Formal Systems, pp. 149–158, 2010.

[27] Y. Gao and S. Yu: "*State complexity approximation*", Proceedings of Descriptional Complexity of Formal Systems 11th Workshop (DCFS 2009), Magdeburg, Germany, pp. 163-174, 2009.

[28] Yo-Sub Han and Kai Salomaa: "*Deterministic State Complexity for Infix-Free Regular Languages*", 2010.

[29] Yo-Sub Han and Kai Salomaa: "*Nondeterministic State Complexity for Suffix-Free Regular Languages*", 2010.

[30] Y.-S. Han, K. Salomaa and D. Wood: "*Nondeterministic state complexity of basic operations for prefix-free regular languages*", Fundamental Information, pp.93–106, 2009.

[31] Yo-Sub Han and Derick Wood: "*Obtaining shorter regular expressions from finite-state automata*", Theoretical Computer Science pp. 110-120, 2007.

[32] Y.-S. Han, K. Salomaa and D. Wood: "*Operational state complexity of prefix-free regular languages*", In Automata, Formal Languages, and Related Topics, pp. 99–115, 2009.

[33] Y.-S. Han and K. Salomaa: "*State complexity of basic operations on suffix-free regular languages*", Theoretical Computer Science, pp. 27-29, 2009.

[34] Yo-Sub Han and Kai Salomaa: "*State Complexity of Union and Intersection of Finite languages*", 2008.

[35] Y. S. Han, Yajun. Wang and D. Wood: "*Infix free regular expressions and Languages*", International journal of foundations of computer science, vol.17 No. 2, pp. 379-393, (2006).

List of Publications

Published

M. Lochan, S. Garhwal: "*State Complexities of Different Types of Regular Languages: A Systematic Review*", paper is presented in International Conference on Recent Trends in Computing, Mechatronics and Communication, at OITM Hisar, pp.375-378, 2012.

Communicated

M. Lochan, S. Garhwal: "*Closure Properties of Prefix-free Regular Languages*", International Journal of Computer Applications.