Normalizing Regular Expression by using Starred Trie Representation

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

Master of Engineering

in

Software Engineering

Submitted By

Jitendra Singh Khedar

(Roll No. 801131013)

Under the supervision of

Mr. Ajay Kumar

Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

June 2013

Certificate

I hereby certify that the work which is being presented in the thesis entitled, *"Normalizing of Regular Expression by using Starred Trie Representation"*, in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mr. Ajay Kumar* 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 the award of any other degree of this or any other University.

Jitendra

Signature:

(Jitendra Singh Khedar)

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

A Kumar 15/7/13

(Mr. Ajay Kumar)

Assistant Professor

Computer Science and Engineering Department

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

First of all, I am thankful to God for his blessings and showing me the right direction. With His mercy, it has been made possible for me to reach so far. It gives me great pleasure to express my gratitude towards the guidance and help I have received from Mr. Ajay Kumar. I am thankful for her continual support, encouragement, and invaluable suggestion. He not only provided me help whenever needed, but also the resources required to complete this thesis report on time. I am also thankful to Dr. Maninder Singh, Head, Computer Science and Engineering Department for his kind help and cooperation. I express my gratitude to all the staff members of Computer Science and Engineering Department for providing me all the facilities required for the completion of my thesis work. I would like to say thanks to all my friends especially Saurabh Jain, Desh deepak, and Abhishek for their support. I want to express my appreciation to every person who contributed with either inspirational or actual work to this thesis. Last but not the least I am highly grateful to all my family members for their inspiration and ever encouraging moral support, which enables me to pursue my studies.

Abstract

A regular expression is a set of characters that specify a pattern. A regular expression can be defined for every finite automaton, and is used for pattern matching. Here we are considering about normalized and un-normalized regular expression. Methodology has been developed to find out the normalized regular expression of a given un-normalized regular expression. Both normalized and un-normalized regular expression should be equivalent, means they should generate same strings over the given alphabets. Normalized regular expression is less time consuming in pattern matching as compare to un-normalized regular expression.

Implementation of methodology is rule based and using starred trie representation. Implementation techniques for both methodologies are described in chapter 4 "Normalization of regular expression". Both methodologies are developed in Java programming language. This thesis contains introduction of automaton, regular expression, literature review, problem statement, normalization techniqes, implementation environment followed by conclusion and future scope.

Table of Content

List of Figures and Table

Table

Chapter 1

Introduction

This chapter gives a basic introduction of automaton, language, regular expression, normalization, and applications of regular expression.

1.1 Automaton

Automaton theory is the study of the abstract computing device or machines. Automaton has a set of states and its control moves from state to state in response to external inputs. These inputs are set of strings. The most general model of the automaton is called finite automaton. "Finite" because the number of possible states and the number of letters in a string are finite, and "automaton" because the change of state is totally governed by the input string [1,2].

A finite automaton is a set of three things [2]:

1. A finite set of states, one of which is stated the initial state, called the start state and some of which are designated as the final state.
2. An alphabet $\sum$ set of possible input letters, from which are formed string that are to be read one letter at a time.
3. A finite set of transitions that tell for each state and for each letter of the input alphabet which state to go next.

1.1.1 Working of Automaton

Automaton works by given input strings made by letters, that automaton reads letter by letter starting from the leftmost letter from a string. Beginning at the start state the string determines a sequence of states. The sequence ends when the last input letter of a string has been read by the automaton [2]. The input string is accepted or rejected at final state.

1.1.2 Example of Automaton

To describe the mathematical theory of finite automaton, the Fig. 1.1 depicts a finite automaton called M_1 [3]. Fig 1.1 is called the state diagram of the finite automaton M_1. It has two states, labeled with q_1 and q_2. The start state q_1 is indicated by the arrow pointing at it from nowhere. A state with double circled is called final state, q_2 is the final state in given automaton. By giving a letter machine go from one state to another that is called transition.

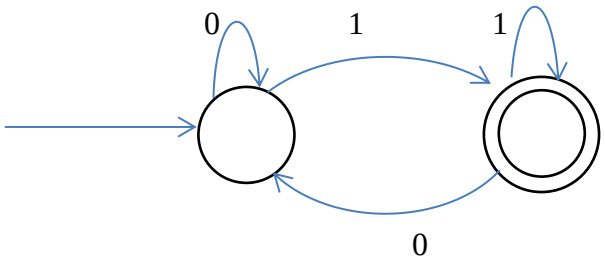


Fig.1.1: A finite automaton M_1, that has two states.

When an input string such as 1101 given to this automaton, it processes the given string and produces an output. The output is either accepted or rejected. The processing begins in M_1's starting state. The automaton receives the symbols from the input string one by one from left to right. After reading each symbol, M_1 moves from one state to another along the transition according with its label. When it reads the last symbol, M_1 produces its output. The output is accepted by given string because M_1 is finally in accepting state.

1.1.3 Application of Automaton

The finite automaton is used in lexical analysis in programming language [4]. Automaton theory serves as the basis for pattern recognition in natural language [5] and speech processing [4]. Najim and Poznyak introduced a multimodal searching technique which is based on learning automata with a changing number of actions [6]. Oommen and De St. Croix [7] proposed an application of learning automata in a pattern recognition problem that involves comparing a noisy string with every element of a dictionary.

1.2 Basic Definitions and Notation

Some basic definitions and notations related with automaton are discussed here.

Definition 1: An alphabet is a finite and nonempty set of symbols, denoted by $\sum$ [8], and $\sum^*$ be the set of all words over the alphabet $\sum$ including the empty word ε.

Definition 2: A string is a finite sequence of symbols over some alphabets. For example, 0111010 is a string from the binary alphabet $\sum = \{0,1\}$.

An empty word is denoted by ε, and it does not contain any instance of alphabets. The length of a word $'z'$ over an alphabet $\sum$ is the number of occurrences of letters in $'z'$. It is denoted by $|z|$.

Definition 3: If A is the set of all strings that machine M_1 accepts, we say that A is the language of machine M_1 and write $L(M_1) = A$. We say that M_1 recognizes A or that M_1 accepts A.

Definition 4: The representation of certain sets of strings in an algebraic notation is called a regular expression. Regular expressions describe the languages accepted by finite state automaton [1].

R is a regular expression if the following conditions hold [3]:

- $\in$
- ϕ
- Each symbol $a \in \sum$ is a regular expression
- If R_1 and R_2 are regular expressions, then $(R_1 \cup R_2)$, $R_1 \cdot R_2$, $(R_1)^*$ are also regular expressions.

Definition 5: The term deterministic refers to the fact that on each input there is one and only one state to which an automaton can transition from its current state [1]. A deterministic finite automata (DFA) M is defined in 5-tuple notation $M = (Q, \sum, \delta, q_0, F)$, where

- Q is a finite set of states,
- $\sum$ is a finite set called alphabet,
- $\delta : Q \times \sum \to Q$ is a transition function,
- q_0 is a starting state,
- $F \subseteq Q$ is a set of final state.

A language L is accepted by a DFA M, if and only of $L = \{w \mid \delta^*(q_0, w) \in F\}$ [1].

Example 1.1: Let $M = (\{s_1, s_2\}, \{a, b\}, \delta, \{s_1, s_2\})$ be a deterministic finite automaton, its state transition diagram is shown in Figure 1.2

DFA M is represented by transition table, a transition table is a tabular representation of a function δ that takes two arguments an input symbol and a state and then returns a value [1]. The rows of the table correspond to the states and the columns correspond to the inputs.

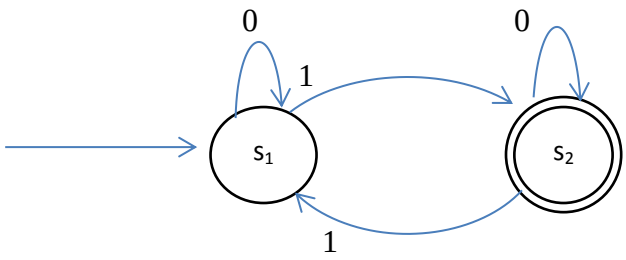


Fig 1.2: State transition diagram for DFA M

Table 1.1: Transition table for DFA M

Symbols / States	0	1
s_1	s_1	s_2
s_2	s_2	s_1

The language accepted by DFA is set of all strings accepted by the DFA.

Definition 6: As DFA, finite set of states are also in non-deterministic finite automaton(NFA) [1], a finite set of input symbols, one start state, a transition function and a set of accepting states. The difference between DFA and NFA is in the type of transition function (δ). For the NFA, δ is a function that takes a state and input symbol as arguments, and return a set of zero, one or more states. An NFA is defined in 5-tuple notation $M = (Q, \sum, \delta, q_0, F)$ where,

- Q is a finite set of states,
- $\sum$ is a finite set called alphabet,
- $\delta : Q \times (\sum \cup \lambda) \rightarrow 2^Q$ is a transition function,
- q_0 is a starting state,
- $F \subseteq Q$ is a set of final state.

Example 1.2: Let $N = (\{s_1, s_2\}, \{a, b\}, \delta, \{s_1\}, \{s_2\})$ be an non-deterministic finite automaton, its state transition diagram shown in Figure 1.3

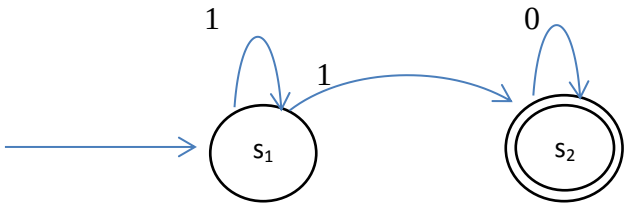


Fig. 1.3: State transition diagram for NFA N

NFA N can be represented by transition table, shown in table 1.2 . It is to be noted that $\delta(s_1, b)$ has more than one state and $\delta(s_2, a)$ has no state.

Table 1.2: Transition table for NFA N

Symbols / States	0	1
s_1	-	$\{s_1, s_2\}$
s_2	s_2	-

Definition 7: A language is regular if it is accepted by a DFA or an NFA, and it is expressed using a regular expression [2].

1.2.1 Length of Regular Expression

The length of regular expression can be defined in several different ways [9,10]:

1. Ordinary length: The ordinary length of regular expression is the count of total number of symbols, parentheses, and null character (ε), etc.
    - $(0+01)^*(1+\varepsilon)$ has ordinary length 12.
2. Reverse polish length: Reverse polish length includes a symbol $(\cdot)$, and its is parentheses free.
    - $(0+01)^*(1+\varepsilon)$ in reverse polish would be $010\cdot+*1\varepsilon+\cdot$

    Reverse polish length of this notation is 10.

3. Alphabetic length: Alphabetic length of a regular expression is total number of symbols from Σ, it does not include ε, parentheses and operators.
    - $(0+01)^*(1+\varepsilon)$ has alphabetic length 4.

Each length measure has its own advantage and disadvantage. The ordinary length is the most direct way to measure length of regular expression. Concatenation operator $(\cdot)$ can be omitted in ordinary length. For example, the regular expression $(a \cdot b)+(c \cdot d)$ has ordinary length 11. It can be written more briefly as $ab+cd$, which has ordinary length 5. Reverse polish length is a measure for the amount of memory required to store the parse tree of a regular expression. Alphabetic length is often used in proofs of upper and lower bounds. The drawback of alphabetic length is that it may be far actual size of regular expression. For example, $((\varepsilon+\phi)^*+\varepsilon)^*$ has alphabetic length is zero.

1.2.2 Application of Regular Expression

Regular expressions is used in various field like pattern matching, For example: (a + b)* is a valid regular expression that would match the string 'aabb', 'babaaa', abbbbbbb' etc, regular expression is also used in web search Engine and in lexical Analysis. Regular expressions are powerful and flexible in terms of searching, which is used to search for a word or string of characters in the pattern.

- **Regular expression in pattern matching**: Regular expression is widely used in Pattern Matching. As Normalized Regular Expression is used for fast pattern matching.
- **Regular expression in web search engine:** Once the World Wide Web started to take form, the first search engines used regular expressions to search through their indexes. It is a fairly trivial task to convert search strings into regular expressions that accept only strings that have some relevance to the query.
- **Regular expressions in lexical analysis:** Lexical analysis is the process of tokenizing a sequence of symbols to eventually parse a string. To perform lexical analysis, two components are required: a scanner and a tokenizer. A token is a block of symbols, known as lexeme. The purpose of tokenization is to categorize the lexemes found in a string to sort them by meaning. For example, the C programming language contain tokens such as identifiers (variable names), string constants, numbers, characters, operators, and keywords.

1.3 Relation between Regular Expression and Regular Language

A very close relation exists between regular language and regular expression. For every regular language there is a regular expression [16], and for every regular expression there is a regular language.

1.3.1 Regular expression denote regular language

If r is a regular expression then L(r) is a regular language. A language is regular if it is accepted by DFA or NFA. In this section, we will discuss an automaton can be constructed using a regular expression r that accepts language L(r).

Theorem 1: Let r is a regular expression then there exists a NFA that accepts L(r).

Proof: We begin with automaton that accept the languages for the simple regular expressions ϕ , λ , and $a \in \Sigma$. These are shown in Figure 1.4,1.5 and 1.6

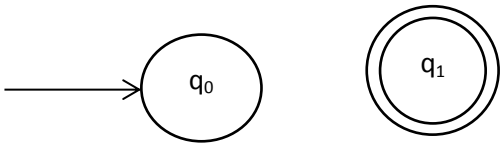


Fig. 1.4 NFA accepts ϕ

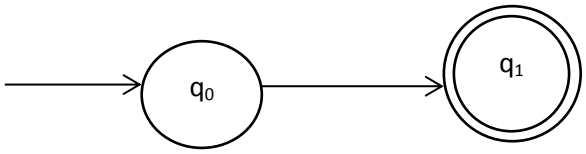


Fig. 1.5 NFA accepts λ

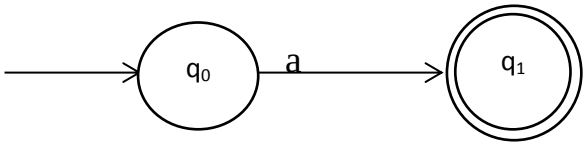


Fig. 1.6 NFA accepts *a*

Example 1.3: DFA which accepts L(r),where r is given below

$$r = (a + ba^*b)^* ba^*$$

Automaton for this regular expression shown in Fig. 1.7, Kleene closure (*) constructed using a loop on a state.

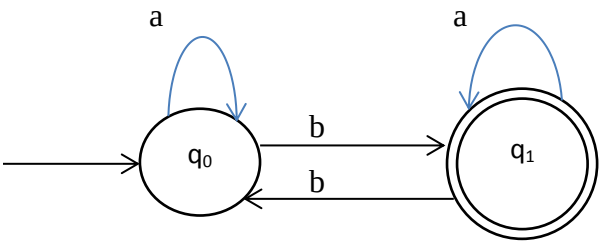


Fig. 1.7 DFA accepts $L((a+ba^*b)^*ba^*)$

1.3.2 Regular expressions for regular languages

Every regular language has an associated finite automaton hence a transition graph. In this section we find regular expression capable of generating the labels of all the walks from q_0 to any final state in an automaton [16]. The label of any walk from the initial state to final state is the concatenation of several regular expressions. An edge of an automaton is labeled with a single symbol *a* is interpreted with the expression *a*, while an edge labeled with multiple symbols a,b,.. is interpreted as an edge labeled with the expression a+b+.. . The strings denoted by such regular expressions are a subset of the language accepted by the automaton, full language being the union of all such generated subsets.

Example 1.4: Fig. 1.8 represents an automaton.

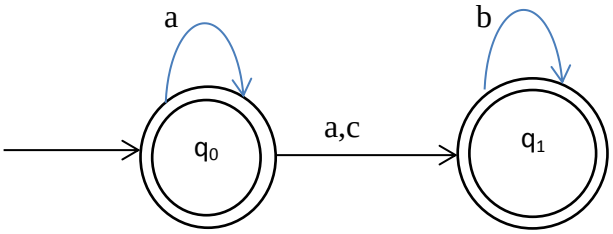


Fig. 1.8 Example of Automaton to show Regular Expression

The language accepted by this automaton is $L(a^*+a^*(a+c)b^*)$. The edge (q_0,q_0) labels is a loop that can generate any number of a's, it represents $L(a^*)$.

1.4 Feature of Java Programming Language

Java was conceived by James Gosling [15], Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991. Core concepts behind to develop Java programming

language is object oriented programming (OOP). Java programming language was choosen to implement this thesis work because Java has some unique features.

1.4.1 Significant features of programming language Java

Programming language Java has so many features to develop a project or a methodology. These features are based on operating system, technologies, functions etc.

- **Platform Independence** - Java compilers do not produce native object code for a particular platform but rather 'byte code' instructions for the Java Virtual Machine (JVM). Making Java code work on a particular platform is then simply a matter of writing a byte code interpreter to simulate a JVM. What this all means is that the same compiled byte code will run unmodified on any platform that supports Java.
- **Object Orientation** - Java is a pure object-oriented language. This means that everything in a Java program is an object and everything is descended from a root object class.
- **Rich Standard Library** - One of Java's most attractive features is its standard library. The Java environment includes hundreds of classes and methods in six major functional areas.
    - Language Support classes for advanced language features such as strings, arrays, threads, and exception handling.
    - Utility classes like a random number generator, date and time functions, and container classes.
    - Input/output classes to read and write data of many types to and from a variety of sources.
    - Networking classes to allow inter-computer communications over a local network or the Internet.
    - Abstract window toolkit for creating platform-independent GUI applications.
    - Applet is a class that lets you create Java programs that can be downloaded and run on a client browser.
- **Applet Interface** - In addition to being able to create stand-alone applications, Java developers can create programs that can downloaded from a web page and run on a client browser.
- **Familiar C++-like Syntax** - One of the factors enabling the rapid adoption of Java is the similarity of the Java syntax to that of the popular C++ programming language.

- **Garbage Collection** - Java does not require programmers to explicitly free dynamically allocated memory. This makes Java programs easier to write and less prone to memory errors.

1.4.2 Area of application of Java

- World wide web applets
- Cross-platform application development
- Network applications

1.5 Thesis Outline

The chapters in thesis are organized as follows:

Chapter 1 describes automaton, regular language, regular expression and programming language Java. Chapter 2 covers literature survey, equivalence of regular expression, normalization of regular expression, star normal form, starred trie representation and Java regular expression. Chapter 3 discusses problem statement, objective and methodology of the thesis. Chapter 4 describes the normalization of regular expression using starred trie representation and rule based normalization system. Chapter 5 includes implementation environment, analyzing a regular expression and snapshots of the implementation.

Chapter 2

Literature survey

This chapter describes in detail the literature survey, equivalence of two regular expressions, normalization of regular expression and star normal form.

2.1 Equivalence of Regular Expressions

Two regular expressions are equivalent if both generate same set of strings and one regular expression can be transformed to another regular expression. Equivalent regular expression represents the same regular language.

The classical approach [11] for comparing two regular expressions r_1 and r_2, i.e. deciding if $L(r_1) = L(r_2)$ consists of following steps:

1. Obtain non-deterministic finite automaton N_1 and N_2, which accepts the same language as r_1 and r_2 respectively.
2. Convert the non-deterministic finite automaton N_1 and N_2 into deterministic finite automaton D_1 and D_2, such that $L(N_1) = L(D_1)$ and $L(N_2) = L(D_2)$.
3. Minimize both D_1 and D_2.
4. Check equivalence of D_1 and D_2.

2.2 Equivalence of Regular Expressions and Finite Automata

Regular expressions and finite automata have equivalent expressive power:

- For every regular expression R, there is a corresponding finite automaton that accepts the set of strings generated by R.
- For every finite automaton A, there is a corresponding regular expression that generates the set of strings accepted by A.

2.3 Normalization of regular expression

Normalized regular expression is equivalent to un-normalized regular expression having less ordinary length. Finite automaton constructed by normalized regular expression having less number of states and transitions as compared to finite automaton constructed by un-normalized

regular expression. Normalized regular expression should generate same strings as un-normalized regular expression and denotes the same language.

Un-normalized regular expression can be converted into a normalized regular expression, if both regular expressions generate same strings. A regular expression E is said to be normalized [12] if it satisfies the following conditions:

1. The regular expression E is reduced if it has some extra un-useful operators and symbols.
    a. E+0=E
    b. 0+E=E
    c. E.1=E
    d. 1.E=E
    e. E.0=0
    f. 0.E=0
2. A regular expression E is normalized if it holds the conditions star normal form (SNF). A regular expression E is said to be in star normal form [13] if for each H such that H^* is a sub-expression of E, we have

    $$\forall x \in Last(H), Follow(H, x) \cap First(H) = \phi$$

3. Un-useful Kleene closure is removed from un-normalized regular expression to reduce the ordinary length of regular expression. Automaton constructed by a regular expression having less Kleene closure contains less transition function. Some basic examples are shown in the following example:

    Example 2.1:

    $$(a^*)^* = a^*$$

    $$((a+b)^*)^* = (a+b)^*$$

    $$(a^* + b)^* = (a+b)^*$$

2.4 Glushkov Automaton

In order to construct a non-deterministic finite automaton, recognizing $L(E)$, Glushkov [14] has introduced four functions:

- $Null(E)$ is equal to $\{\varepsilon\}$, if $\varepsilon \in L(E)$.
- $First(E)$ is the set of initial positions of the string for language $L(E)$.

- $Last(E)$ is the set of final positions of the string for language $L(E)$.
- $Follow(E,x)$ is the set of positions which follow immediately the position x in the expression E

2.5 Star normal form

A regular expression E is said to be in star normal form [13] if for each H such that H^* is a sub-expression of E, we have

$$\forall x \in Last(H), Follow(H,x) \cap First(H) = \phi$$

Example 2.2:

For the regular expression $E = (a+b)^* ab$, we have

- $E' = (a_1 + b_2)^* a_3 b_4$
- $Null(E) = \phi$
- $First(E) = \{1,2,3\}$
- $Last(E) = \{4\}$
- $Follow(E,1) = \{1,2,3\}$
- $Follow(E,2) = \{1,2,3\}$
- $Follow(E,3) = \{4\}$
- $Follow(E,4) = \phi$

Now for a given regular expression following conditions must hold in the star normal form:

$$b_4 \in Last(H), Follow(H,b_4) \cap First(H) = \phi$$

i.e. $\phi \cap \{1,2,3\} = \phi$

It holds the condition of star normal form. Hence the given regular expression is a normalized regular expression.

Theorem 1: For every regular expression E, there is a regular expression E' called the star normal form [13] of E such that

- E' satisfies the condition of star normal form.
- Automaton constructed by both regular expression is equivalent, $M_E = M_E'$

Then we can say E is normalized regular expression.

2.6 Starred Trie Representation

Starred trie representation is used to represent the regular expression with empty, non-empty finite language. Non-empty finite languages not containing ε admit a standard representation via a trie structure [9].

Starred trie representation is used to represent the finite or infinite language. Representation of such languages is shown in diagram with some example of regular expression.

1. **Representing the finite language:** Starred trie representation of finite language is shown in Fig.2.1 using an example of regular expression.

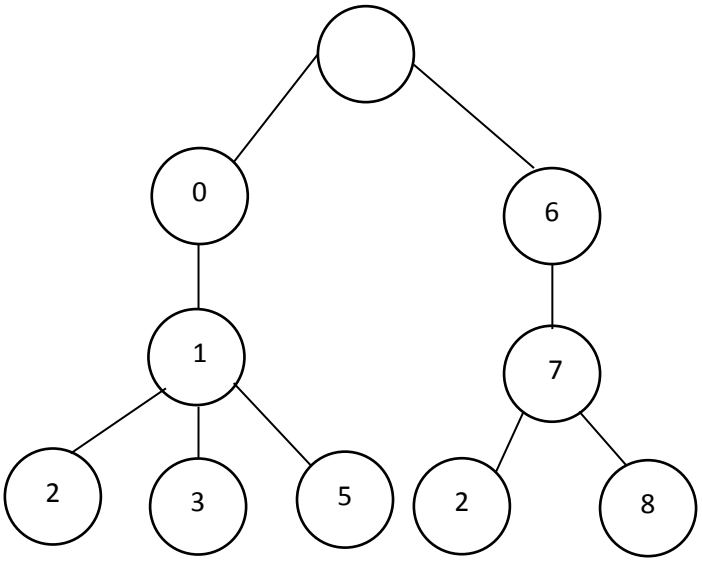


Fig. 2.1: Starred trie representation of language $01(2+3+5)+67(2+8)$

2. **Representation of infinite language:** Starred trie representation of infinite language can be shown by two ways, containing null and non-empty.

- Representation of non-empty infinite language: Starred trie representation of non-empty infinite language is shown in Fig. 2.2
- Representation of infinite language containing null: Starred trie representation of infinite language containing null is shown in Fig. 2.3

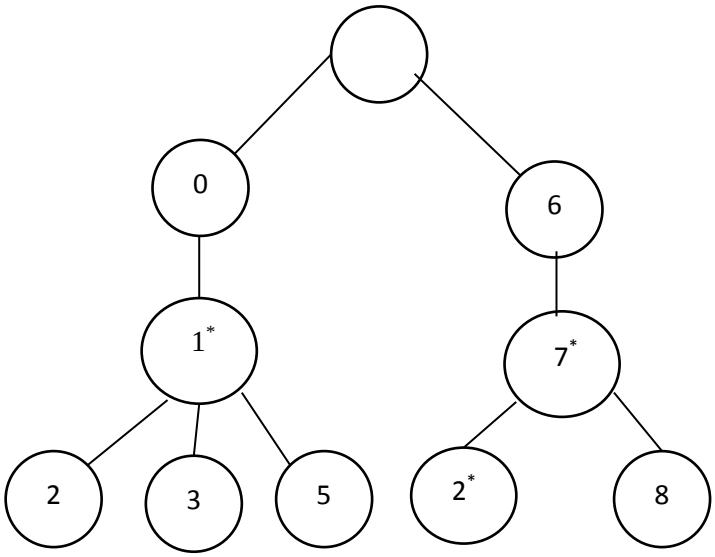


Fig. 2.2: Starred trie representation of language $01^*(2+3+5)+67^*(2^*+8)$

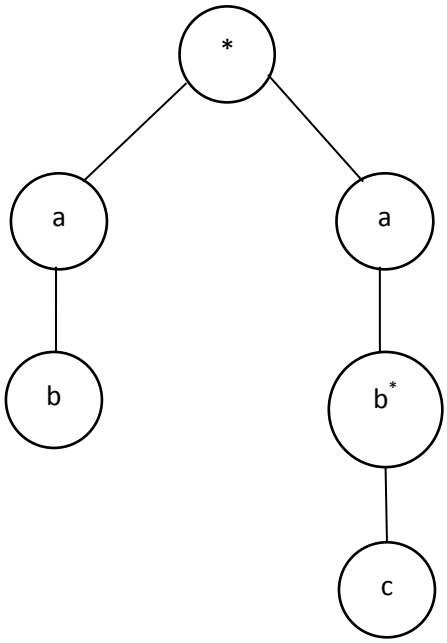


Fig. 2.3: Starred trie representation of language $(ab+ab^*c)^*$

2.7 Java Regular Expression

A Package *java.util.regex* is available in Java programming language for pattern matching with regular expression [17, 18]. A Java program which use this package take inputs a regular expression and a string, and gives the output whether given string is accepted or not accepted by particular regular expression. The *java.util.regex* package primarily consists of three classes: Pattern, Matcher, and PatternSyntaxException.

1. **Pattern:** A Pattern object is a compiled representation of a regular expression. The Pattern class provides no public constructors. To create a pattern, you must first invoke one of its public static compile methods, which will then return a Pattern object. These methods accept a regular expression as the first argument; the first few lessons of this trail will teach you the required syntax.
2. **Matcher:** A Matcher object is the engine that interprets the pattern and performs match operations against an input string. Like the Pattern class, Matcher defines no public constructors. You obtain a Matcher object by invoking the matcher method on a Pattern object.
3. **PatternSyntaxException:** A PatternSyntaxException object is an unchecked exception that indicates a syntax error in a regular expression pattern.

Test Harness: A Java program is available on website by this name [20]. It is for exploring the regular expression supported by application program interface. The application loops repeatedly, prompting the user for a regular expression and input string.

2.8 Related work

A method to construct λ - free automaton from a regular expression, whose size is bounded by the size of both partial derivatives and follow automaton [21]. J.M. Champarnaud [12] solved the theoretical question about regular expressions, by defining a set of regular expression and normalized regular expression and showed that every regular expression can be normalized in linear time. Lower bounds on the minimum required size for the conversion of deterministic finite automata into regular expressions and on the required size of regular expressions resulting from applying some basic language operations on them, given by Gelade and Neven [22]. Antimirov and Mosses [23] presented a rewrite system for deciding the equivalence of two extended regular expressions (i.e. with intersection) based on a new complete axiomatization of

the extended algebra of regular sets. Djelloul Ziadi, and J.M. Champarnaud [13] describe a CREW-PRAM optimal algorithm which converts a regular expression of size s into its Glushkov automaton in $O(\log s)$ time using $O(s^2/\log s)$ processors, This algorithm makes use of the star-normal form of an expression and is based on the sequential algorithm. Marco Almeida, Nelma Moreira, and Rogerio Reis [11] presented a functional approach of a variant of Antimirov and Mosses method, prove the correctness and give some experimental comparative result. Bruggemann-Klein [14] shown that every regular expression can be turned into star normal form in linear time, and several algorithms depend on star normal form. Haiming Chen [24] derives derivatives, partial derivatives and automaton in case of regular expressions in star normal form. Hermann Gruber and Stefan Gulan [25] suggest a quantitative comparison of heuristics for simplifying regular expressions and proposed a normal form for regular expression, they applied this normal form to determine an exact bound for the relation between the two most common size measures for regular expressions. Thompson automaton [19], which is an inductive tool that defines the automaton for the basic regular expressions together with rules to construct the automata for the different operations involved in a regular expression α. Derivatives, partial derivatives and automata in the case of regular expressions in star normal form, defined by Bruggemann-Klein [14], and of one-unambiguous expressions, defined by Bruggemann-Klein and Wood [26]. In syntactic specifications of programming languages regular expression describe lexical tokens, and in text manipulation systems they describe textual patterns that trigger processing actions [27]. Ilie and Yu [16] proposed the problem of determining the optimal quotient if we replace expression size with alphabetic width [28].

Chapter 3

Problem Statement

This chapter includes the problem statement, rule based normalization system of regular expression followed by the motivation behind the thesis.

3.1 Problem Statement

Normalization of regular expressions and equivalence of regular expressions have been studied by various researchers from years. In this thesis normalization of regular expression, equivalence of regular expression, star normal form of regular expression and starred trie representation of regular expression are discussed.

Two regular expressions are equivalent if both generate the same set of strings. One regular expression is normalized if its ordinary length is less than ordinary length of another regular expression and both are equivalent. Rule based normalization system is designed to convert un-normalized regular expression into normalized regular expression, when r_1 and r_2 consist of one or two symbols are only *a* and *b*. This system suffers the problem whenever length of r1 and r2 are greater than two.

A generalized approach is proposed to normalize a regular expression using the starred trie representation. This methodology is generalized and work for any length of r1 and r2. This methodology faces the problem when regular expression has more than one round brackets or regular expression is without round brackets.

3.2 Proposed Work

The work in this thesis is to normalize a regular expression from un-normalized regular expression. The objective of the proposed work in this thesis is given below:

1. Analysis of various techniques and methodologies in the field of regular expression, normalization of regular expression, starred trie representation and star normal form.
2. Proposed an approach for the conversion of un-normalized regular expression to normalized regular expression. Two approaches are proposed, first is rule based

normalization of regular expression and second is normalization using starred trie representation.

3. Implementation of the proposed approach in the Java language.

Chapter 4

Normalization of Regular Expression

This chapter includes the rule based normalization system of regular expression and starred trie representation of regular expression, those are in the form of $(r_1)^*$ and $(r_1 + r_2)^*$. Rule based normalization system is described in five major rules in tables and starred trie representation is discussed with the representation of regular expression and normalization techniques to convert an un-normalized regular expression into normalized regular expression.

4.1 Rule based Normalization System

Rule based normalization system was developed to convert an un-normalized regular expression to a normalized regular expression. This system is based on five major rules. Rules are constructed for performing the normalization of regular expressions.

Regular expression $(a+b)^*$ is superset of all strings over *a* and *b*. This regular expression is divided in two parts r_1 and r_2. Consider this regular expression in the form $(r_1 + r_2)^*$.

$|r_1|$ and $|r_2|$: Rules in rule based normalization system are based on $|r_1|$ and $|r_2|$. Length of r_1 and r_2 should be one or two, and over $\sum = \{a,b\}$. Five major rules have been developed for this methodology.

Rule 1: If $|r_1| = |r_2| = 1$, Six cases can be reduced to a normalized regular expression $(a+b)^*$, reducible expressions are shown in table 1 and the non-reducible case is:

- When, both r_1 and r_2 are same,

  Example: $(a+a)^*$.

Rule 2: If $|r_1| = 1$ and $|r_2| = 2$, Sixteen regular expressions can be reduced to $(a+b)^*$, reducible expressions are shown in table 2 and the non-reducible cases are:

1. If r_2 containing either $'ab'$ or $'ba'$.

   Example: $(a+ab)^*, (a+ba)^*$, etc.

2. If $r_1 = a$ and r_2 containing $'a'$ is without Kleene closure.

   Example: $(a+ab^*)^*$ etc.

Rule 3: If $|r_1|=2$ and $|r_2|=1$, Sixteen regular expressions can be reduced to $(a+b)^*$, reducible expressions are shown in table 3 and the non-reducible cases are:

1. If r_1 containing either $'ab'$ or $'ba'$.

   Example: $(ab+a)^*, (ba+a)^*$ etc.

2. If $r_2 = a$ and r_1 containing $'a'$ is without Kleene closure.

   Example: $(ab^*+a)^*$ etc.

Rule 4: If $|r_1| = |r_2| = 2$, Twenty eight cases can be normalized to $(a+b)^*$, reducible expressions are shown in table 4 and the non-reducible cases are:

1. If either of r_1 or r_2 is $'aa','ab','ba','bb'$.
2. Symbol $'a'$ in both r_1 and r_2 having Kleene closure and another symbol $'b'$ does not having Kleene closure.

   Example: $(a^*b+a^*b)^*$, and vice versa.

Rule 5: When either r_1 or r_2 having two $'a'$ or two $'b'$, and $|r_1| = |r_2| = 2$, reducible expressions are shown in table 5, and non-reducible cases for Rule 5 are:

1. If $r_1 = 'aa'$ or $r_2 = 'bb'$, or vice versa.
2. If $r_1 = 'a^*a^*'$ or $r_2 = 'b^*b^*'$, or vice versa.

Table 4.1: Normalization of regular expression into $(a+b)^*$ for $|r_1| = |r_2| = 1$

$r_1 = a$ and $r_2 = b$	$r_1 = b$ and $r_2 = a$
$(a+b^*)^*$	$(b+a^*)^*$
$(a^*+b)^*$	$(b^*+a)^*$
$(a^*+b^*)^*$	$(b^*+a^*)^*$

Table 4.2: Normalization of regular expression into $(a+b)^*$ for $|r_1|=1$ and $|r_2|=2$

$r_1 = a$	$r_1 = a^*$	$r_1 = b$	$r_1 = b^*$
$(a+a^*b)^*$	$(a^*+a^*b)^*$	$(b+ab^*)^*$	$(b^*+ab^*)^*$
$(a+ba^*)^*$	$(a^*+ba^*)^*$	$(b+a^*b^*)^*$	$(b^*+a^*b^*)^*$
$(a+a^*b^*)^*$	$(a^*+a^*b^*)^*$	$(b+b^*a)^*$	$(b^*+b^*a)^*$
$(a+b^*a^*)^*$	$(a^*+b^*a^*)^*$	$(b+b^*a^*)^*$	$(b^*+b^*a^*)^*$

Table 4.3: Normalization of regular expression into $(a+b)^*$ for $|r_1|=2$ and $|r_2|=1$

$r_2 = a$	$r_2 = a^*$	$r_2 = b$	$r_2 = b^*$
$(a^*b+a)^*$	$(a^*b+a^*)^*$	$(ab^*+b)^*$	$(ab^*+b^*)^*$
$(ba^*+a)^*$	$(ba^*+a^*)^*$	$(b^*a+b)^*$	$(b^*a+b^*)^*$
$(a^*b^*+a)^*$	$(a^*b^*+a^*)^*$	$(a^*b^*+b)^*$	$(a^*b^*+b^*)^*$
$(b^*a^*+a)^*$	$(b^*a^*+a^*)^*$	$(b^*a^*+b)^*$	$(b^*a^*+b^*)^*$

Table 4.4: Normalization of regular expression into $(a+b)^*$ for $|r_1|=|r_2|=2$

$First(r_1)=First(r_2)=a$	$First(r_1)=a$ and $First(r_2)=b$	$First(r_1)=b$ and $First(r_2)=a$	$First(r_1)=First(r_2)=b$
$(a^*b+ab^*)^*$	$(a^*b+b^*a)^*$	$(ba^*+ab^*)^*$	$(ba^*+b^*a)^*$
$(a^*b^*+ab^*)^*$	$(a^*b^*+b^*a)^*$	$(b^*a^*+ab^*)^*$	$(ba^*+b^*a^*)^*$
$(a^*b+a^*b^*)^*$	$(a^*b+b^*a^*)^*$	$(ba^*+a^*b^*)^*$	$(b^*a^*+b^*a)^*$
$(ab^*+a^*b)^*$	$(ab^*+ba^*)^*$	$(b^*a+a^*b)^*$	$(b^*a+ba^*)^*$
$(ab^*+a^*b^*)^*$	$(ab^*+b^*a^*)^*$	$(b^*a+a^*b^*)^*$	$(b^*a+b^*a^*)^*$
$(a^*b^*+a^*b)^*$	$(a^*b^*+ba^*)^*$	$(b^*a^*+a^*b)^*$	$(b^*a^*+ba^*)^*$
$(a^*b^*+a^*b^*)^*$	$(a^*b^*+b^*a^*)^*$	$(b^*a^*+a^*b^*)^*$	$(b^*a^*+b^*a^*)^*$

Table 4.5: Normalization of regular expression into $(a+b)^*$ for $|r_1|=|r_2|=2$

r_1 having two 'a' and r_2 having two 'b'	r_1 having two 'b' and r_2 having two 'a'
$(aa^*+bb^*)^*$	$(bb^*+aa^*)^*$
$(a^*a+bb^*)^*$	$(b^*b+aa^*)^*$
$(aa^*+b^*b)^*$	$(bb^*+a^*a)^*$
$(a^*a+b^*b)^*$	$(b^*b+a^*a)^*$

4.2 Normalization of union free regular expression using starred tire representation

When a regular expression is given in the form of $(r_1)^*$, with any number of symbols of any length. First of all the regular expression is represented in starred trie representation, then the proposed techniques will be applied to convert un-normalized regular expression into normalized regular expression. If a regular expression is already in normalized form then we need to check whether it is in normalized form. Let assume that all the regular expressions are whole starred.

1. A regular expression with a single symbol without Kleene closure $(a)^*$: Regular expression is converted into starred trie representation in Fig. 4.1

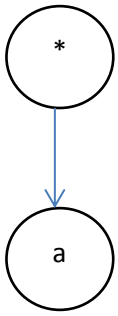


Fig. 4.1: Starred trie representation of $(a)^*$

- Technique applied: One child with single node having a symbol without Kleene closure is a normalized regular expression. Such type of regular expression cannot reduce further.

2. A regular expression with a single symbol with starred $(a^*)^*$: Regular expression is converted into starred trie representation in Fig. 4.2

- Technique applied: One child with single node having a symbol with Kleene closure is an un-normalized regular expression. To make this regular expression in normalized form, remove the Kleene closure from the child node. Now read the regular expression as its definition.

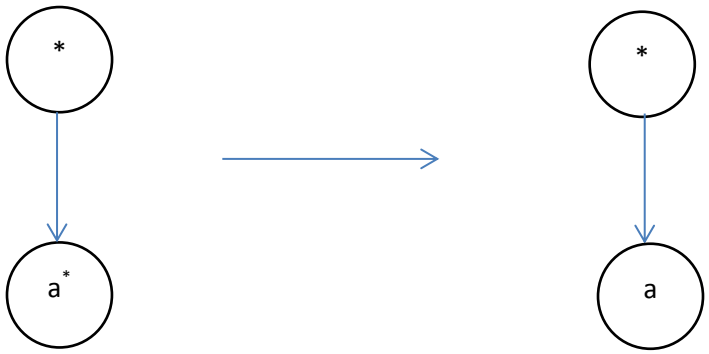
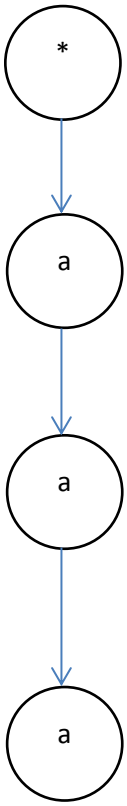


Fig. 4.2: Starred trie representation of $(a^*)^*$ and $(a)^*$

3. A regular expression with existence of a symbol more than one time and all symbols are without Kleene closure $(aaa)^*$: Regular expression is converted into starred trie representation in Fig. 4.3
- Technique applied: If same symbol exists more than one time, without having Kleene closure is a normalized regular expression.
4. A regular expression with existence of different symbols more than one time without Kleene closure $(abcd)^*$: Starred trie representation of this regular expression is shown in Fig. 4.4
- Technique applied: If different symbols exist more than one time, without Kleene closure is a normalized regular expression.
5. A regular expression with two same symbols and one of them having Kleene closure $(aa^*)^*$: Starred trie representation of this regular expression is shown in Fig. 4.5
- Technique applied: When two same symbols are in regular expression and one of them having Kleene closure then remove the node whose symbol with Kleene closure. Starred

trie representation of normalized regular expression is shown in Fig. 4.6. Normalized regular expression would be $(a)^*$.



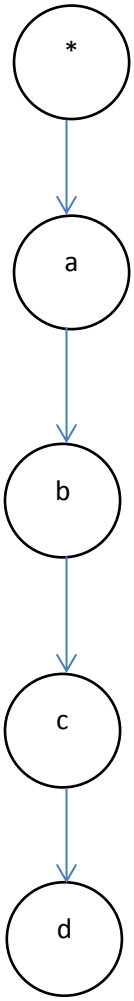


Fig. 4.3: Starred trie representation of $(aaa)^*$ **Fig. 4.4:** Representation of $(abcd)^*$

6. A regular expression with two or more different with one or more Kleene closure symbol but not all Kleene closure $(ab^*c)^*$: Starred trie representation of this regular expression is shown in Fig. 4.7

- Technique applied: When a regular expression having two or more different symbols with one or more Kleene closure symbols but not all Kleene closure is a normalized regular expression. It cannot reduce further.

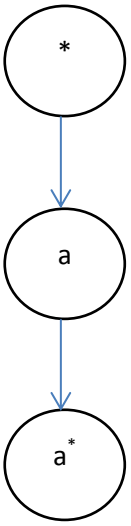


Fig. 4.5: Starred trie representation of $(aa^*)^*$

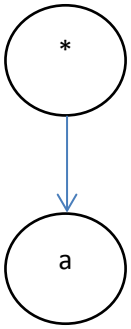


Fig. 4.6: Representation of $(a)^*$

7. A regular expression with existence of a symbol more than one time and all are with Kleene closure $(a^*a^*a^*)^*$: Starred trie representation of this regular expression is shown in Fig. 4.8

- Technique applied: If a regular expression with same symbol more than one time and all are Kleene closure, then first remove all nodes except one node then remove the Kleene closure of last node. Starred trie representation of normalized regular expression is shown in Fig. 4.9

8. A regular expression with more than two different symbols and all symbols are with Kleene closure $(a^*b^*c^*)^*$: Starred trie representation of this regular expression is shown in Fig. 4.10

- Technique applied: A regular expression with more than two different symbols and all symbols are with Kleene closure is equivalent to normalized regular expression as shown below:

$$(a^*b^*c^*)^* = (a+b+c)^*$$

Here, ordinary length of un-normalized regular expression is reduced by one in normalized regular expression. So, first we need to store all symbols and then make them different children of starred trie representation. Now read regular expression as definition

of starred trie representation. Starred trie representation of normalized regular expression is shown in Fig. 4.11

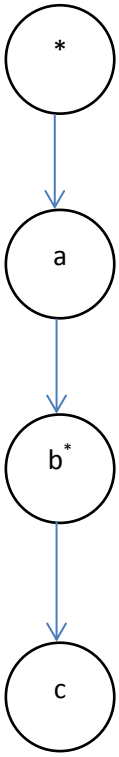


Fig. 4.7: Starred trie representation of $(ab^*c)^*$

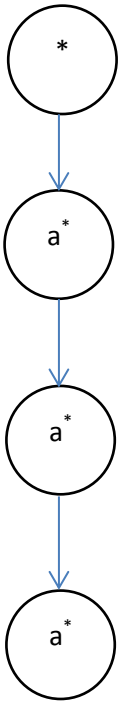


Fig. 4.8: Representation of $(a^*a^*a^*)^*$

9. A regular expression with a same symbol twice and one of them with Kleene closure along with a another symbol $(aa^*b)^*$: Starred trie representation of this regular expression is shown in Fig. 4.12(a)

- Technique applied: As mentioned in technique 5 same symbol exists twice and one with Kleene closure $(aa^*)^*$ is reduced to $(a)^*$. But if one more different symbol present with $(aa^*)^*$ is made it $(aa^*b)^*$, and this is normalized regular expression, it cannot be reduce further.

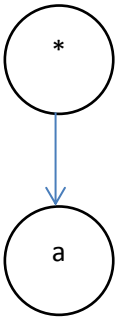


Fig. 4.9: Starred trie representation of $(a)^*$

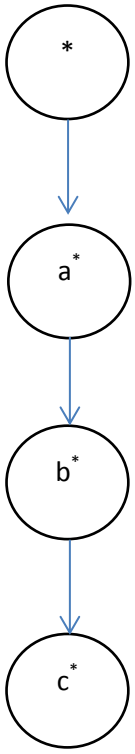


Fig. 4.10: Starred trie representation of $(a^*b^*c^*)^*$

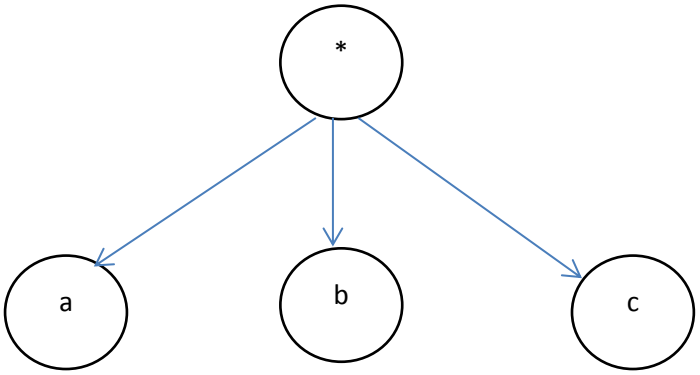


Fig. 4.11: Starred trie representation of $(a+b+c)^*$

10. A regular expression with different symbols having at least one Kleene closure but not all Kleene closure $(ab^*c)^*$: Starred trie representation of this regular expression is shown in Fig. 4.12(b)

- Technique applied: If a regular expression given with different symbols having at least one Kleene closure but not all Kleene closure is a normalized regular expression. The subset of a regular expression aa^* is not included in this case.

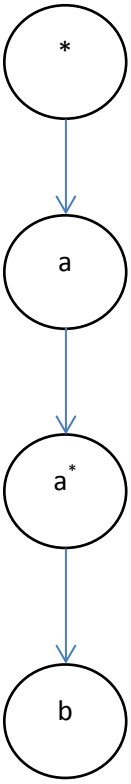


Fig. 4.12(a)

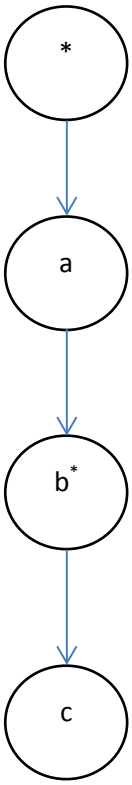


Fig. 4.12(b)

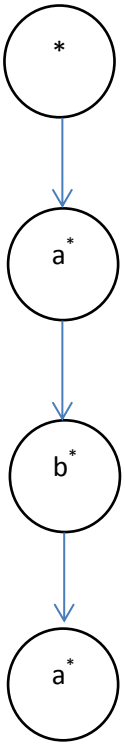


Fig. 4.12(c)

Fig. 4.12: Starred trie representation of $(aa^*b)^*$, $(ab^*c)^*$ and $(a^*b^*a^*)^*$

11. A regular expression with different symbols and all of them having Kleene closure, and one symbol exists more than one time $(a^*b^*a^*)^*$: Starred trie representation of this regular expression is shown in Fig. 4.12(c)

- Technique applied: If a regular expression is given with all symbols having Kleene closure and one symbols exists more than one time. First apply technique 8 to make

different child for each symbol. Then keep one node for same symbols, remaining nodes are removed. These states are shown in Fig. 4.13 and Fig. 4.14

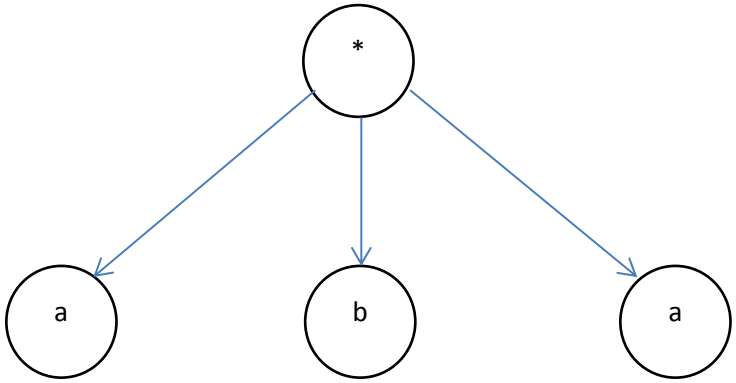


Fig. 4.13: Starred trie representation of $(a+b+a)^*$

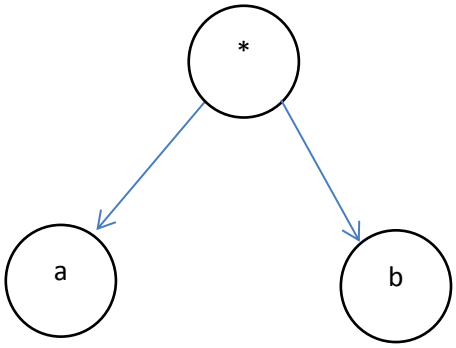


Fig. 4.14: Starred trie representation of $(a+b)^*$

12. A regular expression with more than one symbol and same symbols exits more than one time simultaneously with Kleene closure $(a^*a^*b)^*$: Starred trie representation of this regular expression is shown in Fig. 4.15(a)

- Technique applied: To normalize such regular expression, delete one of node that contains same symbol with Kleene closure. The resulting starred trie representation is shown in Fig. 4.15(b) and normalized regular expression would be $(a^*b)^*$.

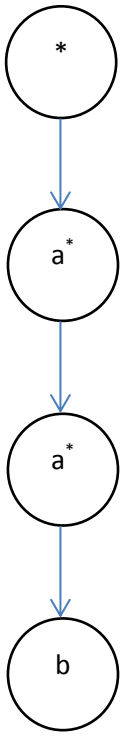


Fig.4.15 (a)

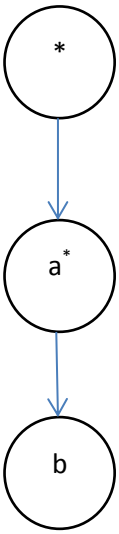


Fig. 4.15(b)

Fig. 4.15: Starred trie representation of $(a^*a^*b)^*$ and $(a^*b)^*$

4.3 Normalization of Regular Expression with union using starred trie representation

A regular expression in the form $(r_1)^*$ has been normalized in section 4.1. This section describes techniques to convert un-normalized regular expression into normalize a regular expression which is in the form of $(r_1 + r_2)^*$. First, all the techniques applied on $(r_1)^*$ will be applied on separate r_1 and r_2. Then some techniques will be applied on $(r_1 + r_2)^*$. Starred trie representation of $(r_1 + r_2)^*$ shown with two children.

1. A regular expression with both r_1 and r_2 are same $(a^*b+a^*b)^*$: Starred trie representation of this regular expression is shown in Fig. 4.16

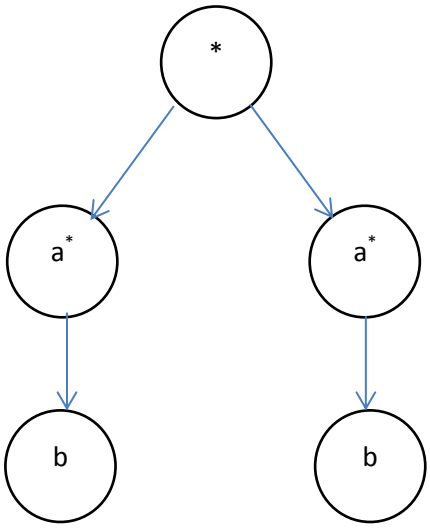


Fig. 4.16: Starred trie representation of $(a^*b+a^*b)^*$

- Technique applied: If two children are exactly same, remove all nodes of one child and keep only one child. Its normalized regular expression would be $(a^*b)^*$ and starred trie representation of normalized regular expression is shown in Fig. 4.17

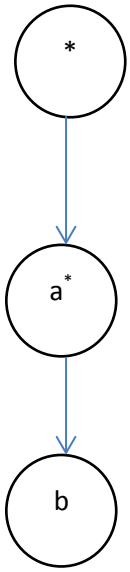


Fig. 4.17: Starred trie representation of $(a^*b)^*$

2. A regular expression where either r_1 or r_2 is single symbol with Kleene closure $(a^* + b^*)^*$: Starred trie representation of this regular expression is shown in Fig. 4.18

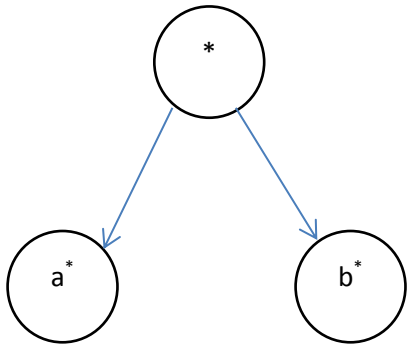


Fig. 4.18: Starred trie representation of $(a^* + b^*)^*$

- Technique applied: If r_1 or r_2 are single symbols with Kleene closure then remove the Kleene closure of all symbols. Its normalized regular expression would be $(a+b)^*$. Starred trie representation of normalized regular expression is shown in Fig. 4.19

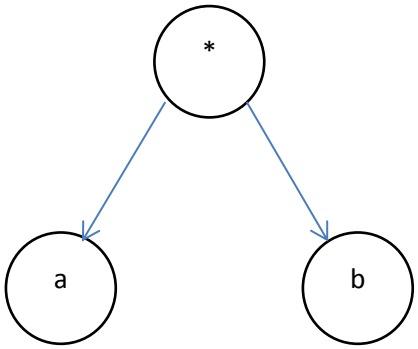


Fig. 4.19: Starred trie representation of $(a+b)^*$

3. A regular expression where both r_1 and r_2 have one or more than one symbol and all symbols are without Kleene closure $(ab+cd)^*$: Starred trie representation of this regular expression is shown in Fig. 4.20

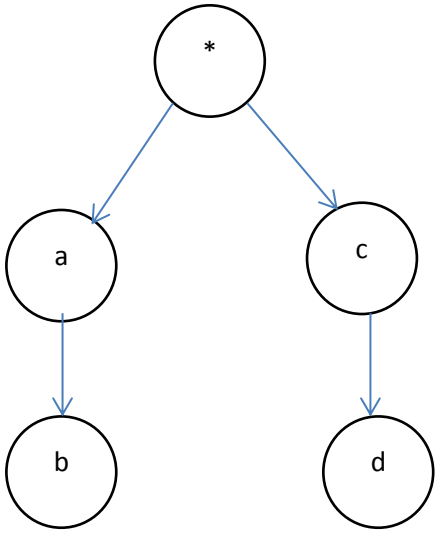


Fig. 4.20: Starred trie representation of $(ab+cd)^*$

- Technique applied: A regular expression where both r_1 and r_2 have length greater than one, with different symbols and without Kleene closure is a normalized regular expression. It cannot be further reduced.

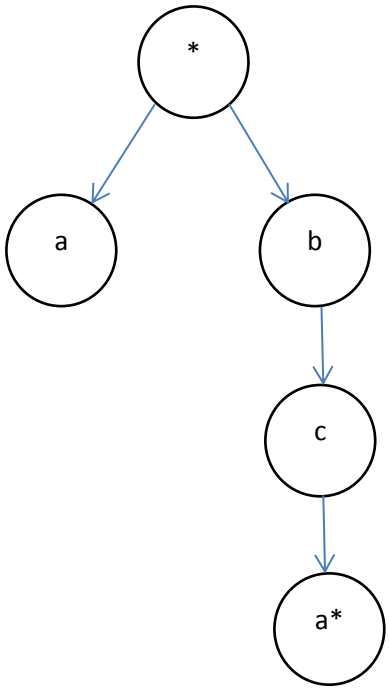


Fig. 4.21: Starred trie representation of $(a+bca^*)^*$

4. A regular expression where r_1 has length one and same symbol of r_1 is at corner in r_2 with Kleene closure and vice versa $(a+bca^*)^*$: Starred trie representation of this regular expression is shown in Fig. 4.21

- Technique Applied: Remove the node of r_2 at corner with Kleene closure which is same as r_1. The regular expression after normalization would be $(a+bc)^*$. Starred trie representation of normalized regular expression is shown in Fig. 4.22

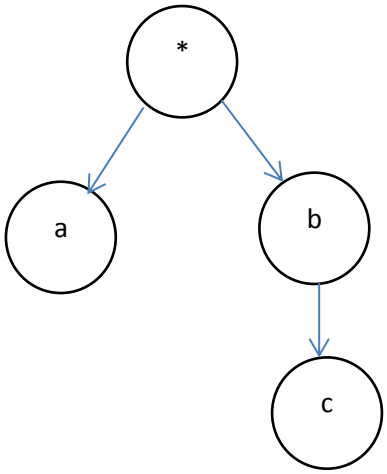


Fig. 4.22: Starred trie representation of $(a+bc)^*$

5. A regular expression where r_1 has length one and same symbol of r_1 is at corner in r_2 without Kleene closure and vice versa $(a+ba)^*$: Starred trie representation of this regular expression is shown in Fig. 4.23

- Technique Applied: A regular expression where r_1 has length one and same symbol of r_1 is at corner in r_2 without Kleene closure and vice versa is normalized regular expression. It cannot be reduced further.

6. A regular expression where r_1 has length one and same symbol of r_1 is present in r_2 but not in corner with Kleene closure and vice versa $(a+ba^*c)^*$: Starred trie representation of this regular expression is shown in Fig. 4.24

- Technique Applied: A regular expression where r_1 has length one and same symbol of r_1 is present in r_2 but not in corner with Kleene closure and vice versa is normalized regular expression. It cannot be reduced further.

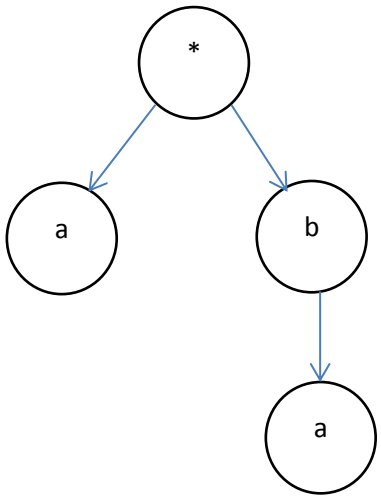


Fig. 4.23: Starred trie representation of $(a+ba)^*$

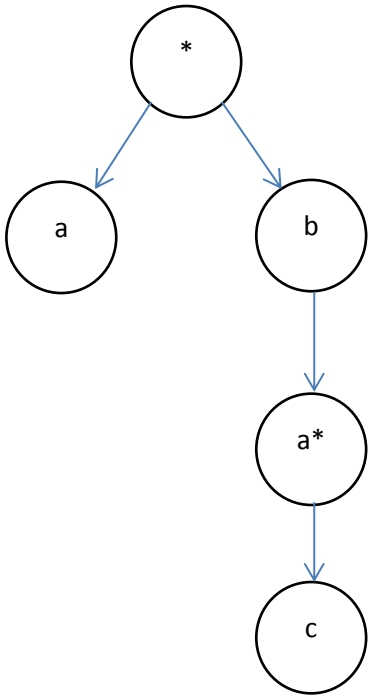


Fig. 4.24: Starred trie representation of $(a+ba^*c)^*$

7. A regular expression with length of both r_1 and r_2 is greater than one and r_1 is subset of r_2, remaining part of r_2 having Kleene closure $(ab+abc^*)^*$: Starred trie representation of this regular expression is shown in Fig. 4.25

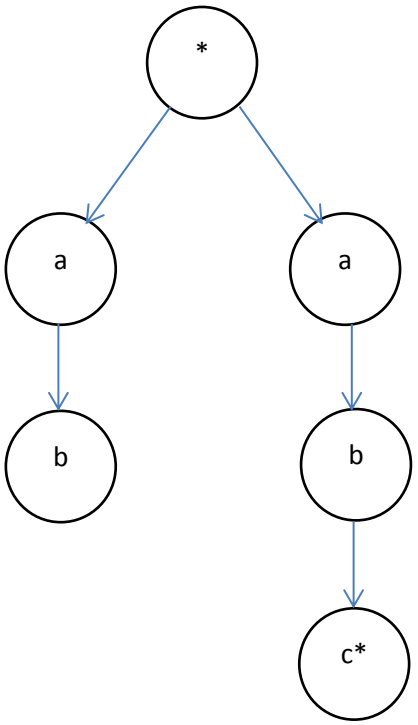


Fig. 4.25: Starred trie representation of $(ab+abc^*)^*$

- Technique applied: Remove the all nodes of a child which is subset of another child. After normalization it will contain only one child. After normalization regular expression would be $(abc^*)^*$. Starred trie representation of normalized regular expression is shown in Fig. 4.26

8. A regular expression with length of both r_1 and r_2 is greater than one and r_1 is subset of r_2, remaining part of r_2 is without Kleene closure $(ab+abc)^*$: Starred trie representation of this regular expression is shown in Fig. 4.27

- Technique applied: A regular expression with length of both r_1 and r_2 is greater than one and r_1 is subset of r_2, remaining part of r_2 is without Kleene closure is a normalized regular expression. It cannot be reduced further.

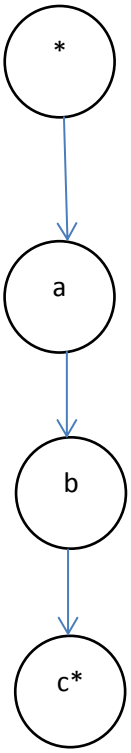


Fig. 4.26: Starred trie representation of $(abc^*)^*$

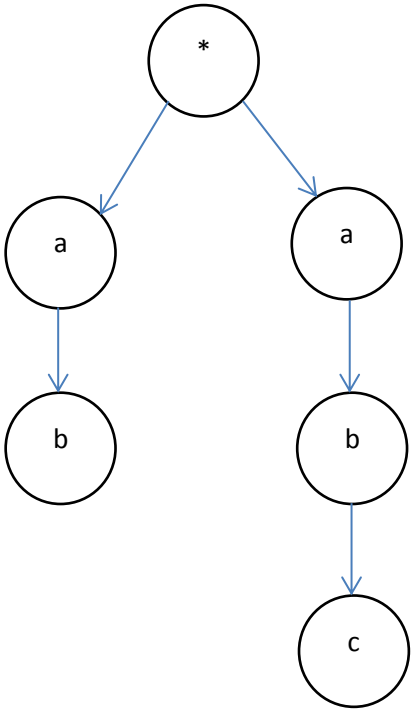


Fig. 4.27: Starred trie representation of $(ab + abc)^*$

Chapter 5

Implementation Environment

This chapter gives the detail of implementation of the methodology. The proposed methodologies have been implemented in Java programming language.

5.1 Implementation of methodology

The proposed methodologies in this thesis have been implemented in Java programming language. In starred trie representation, first the un-normalized regular expression is represented in starred trie representation. This representation is similar a tree representation. Initially this methodology stored a given regular expression in tree form and then rules are applied to normalize a regular expression from an un-normalized regular expression.

5.2 Analyzing a Regular Expression

The proposed methodology makes the Kleene closure to root node. Then every string in a given regular expression is become a child and every symbol become a node. A root in the tree can have minimum one child and maximum two children. When only r_1 is present in given regular expression then comparison is needed in child itself only. When both r_1 and r_2 are given then comparison is needed in one child itself and between both children also. After normalization the regular is read according to starred trie representation format.

5.3 GUI for Rule based normalization system

The rule based normalization system was developed in programming language Java. All the cases that are reduced in $(a+b)^*$, give the result "Normalized regular expression: $(a+b)^*$". All other cases give the result "cannot be normalized into $(a+b)^*$". Result of all rules is shown in Fig. 5.1- Fig. 5.5 respectively.

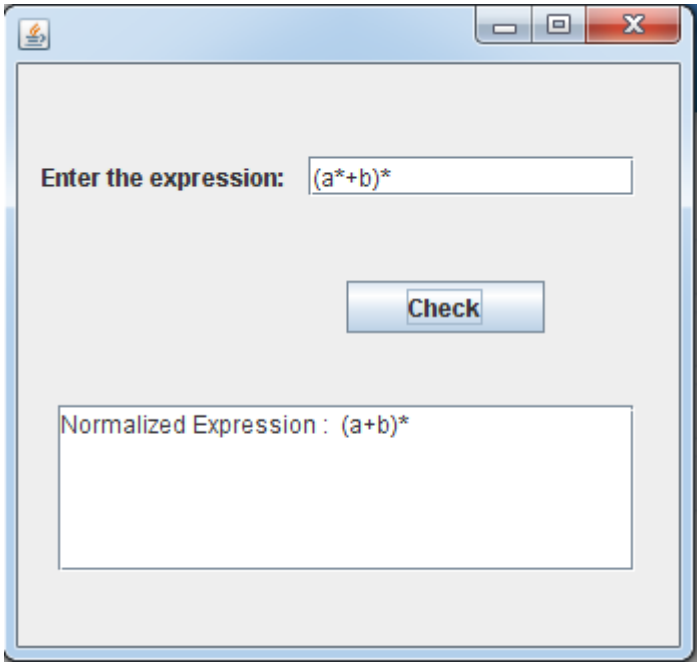


Fig. 5.1: GUI for normalizing a regular expression $(a^* + b)^*$

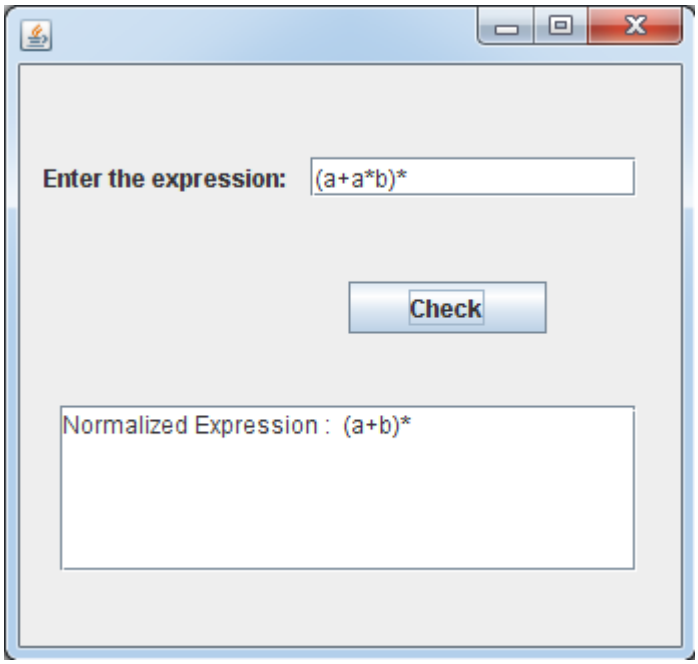


Fig. 5.2: GUI for normalizing a regular expression $(a + a^*b)^*$

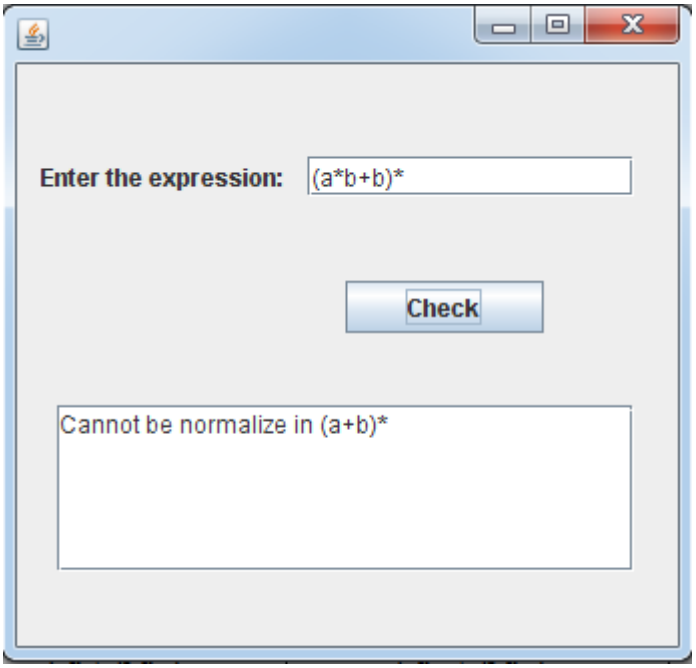


Fig. 5.3: GUI for normalizing a regular expression $(a^*b+b)^*$

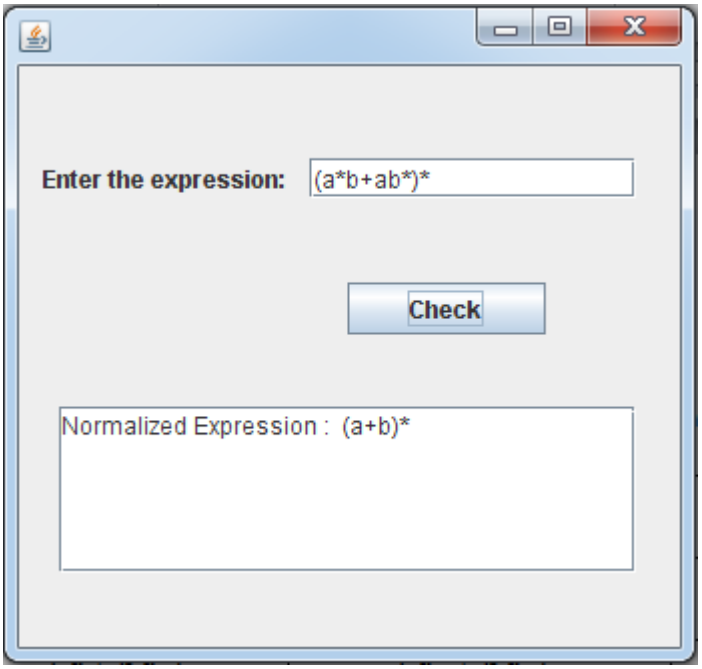


Fig. 5.4: GUI for normalizing a regular expression $(a^*b+ab^*)^*$

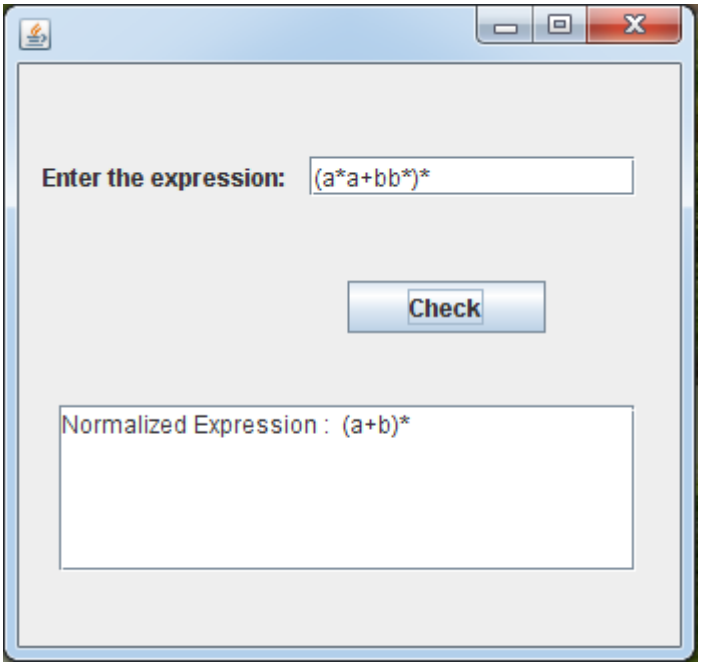


Fig. 5.5: GUI for normalizing a regular expression $(a^*a+bb^*)^*$

5.4 GUI for normalizing a union free regular expression using starred trie representation

This section describes the methodology to normalize the regular expression when only r_1 is given in regular expression. The rules and representation of regular expression is shown in chapter 4. The snapshots of regular expression which containing r_1 is shown in following snaps:

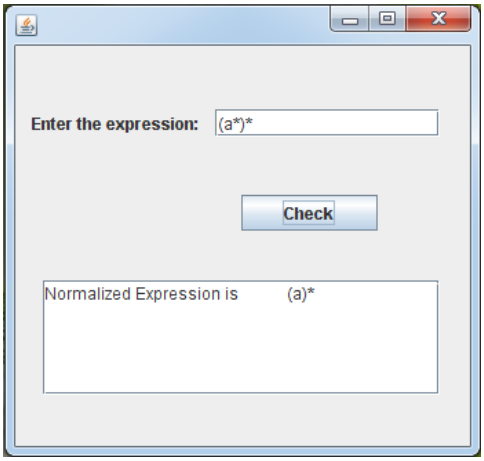


Fig. 5.6: GUI to normalize a regular expression $(a^*)^*$

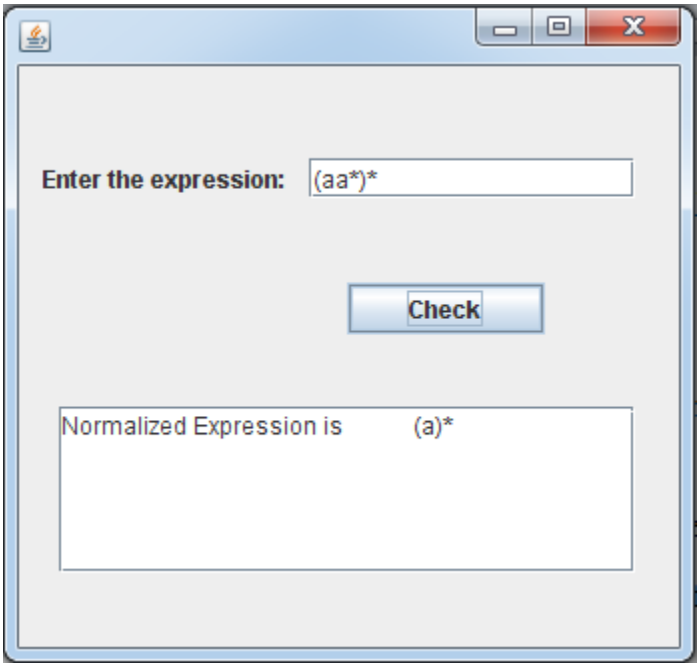


Fig. 5.7: GUI to normalize a regular expression $(aa^*)^*$

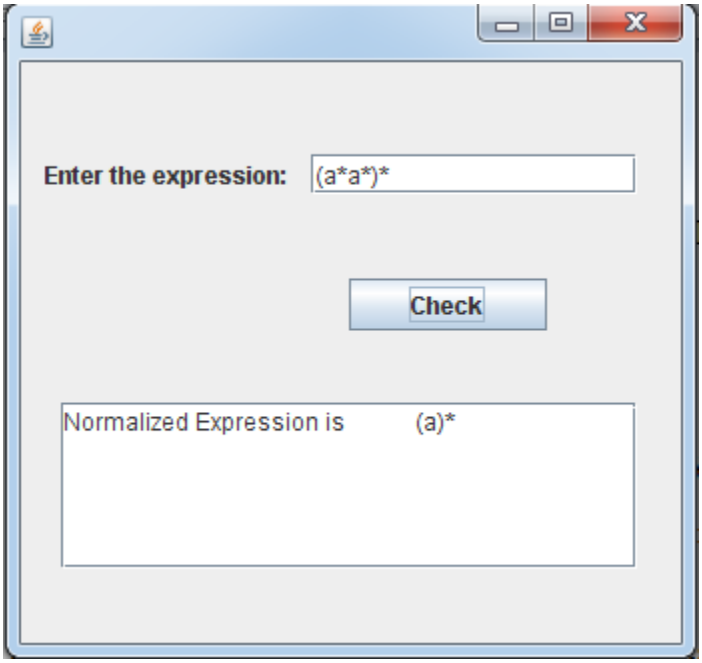


Fig. 5.8: GUI to normalize a regular expression $(a^*a+bb^*)^*$

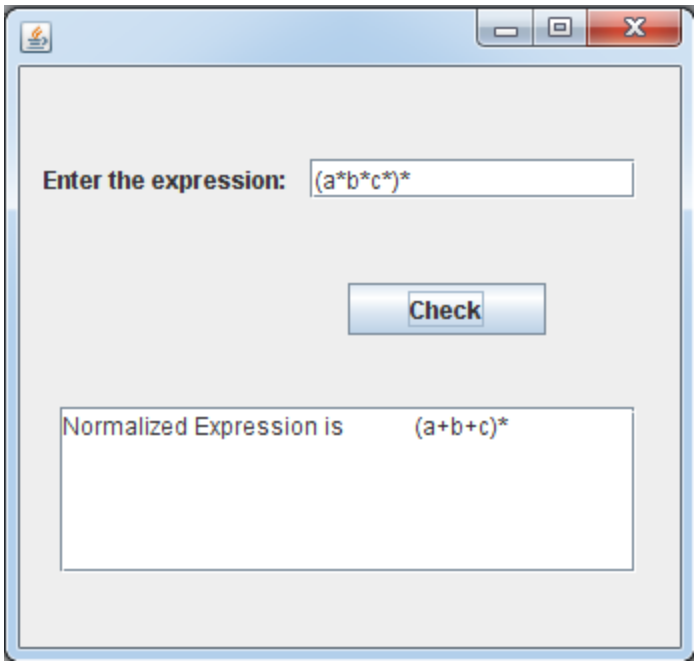


Fig. 5.9: GUI for normalizing a regular expression $(a^*b^*c^*)^*$

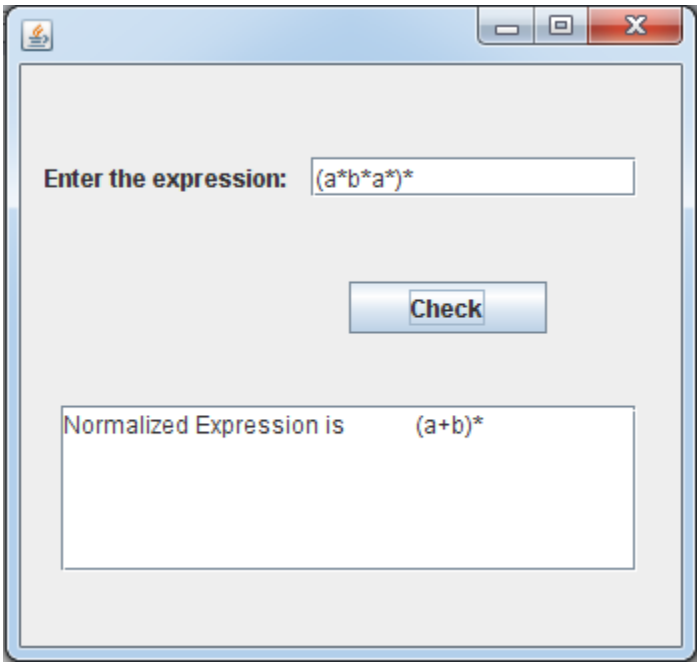


Fig. 5.10: GUI for normalizing a regular expression $(a^*b^*a^*)^*$

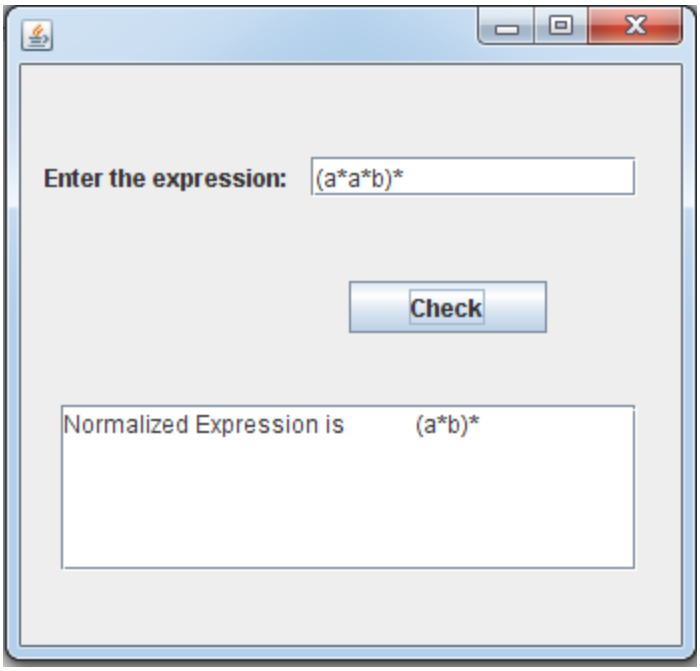


Fig. 5.11: GUI for normalizing a regular expression $(a^*a^*b)^*$

5.5 GUI for normalizing a regular expression in form of $(r_1+r_2)^*$

This section describes the methodology to normalize the regular expression when both r_1 and r_2 are given in regular expression. The rules and representation of regular expression is shown in chapter 4.

The snapshots of regular expression which containing both r_1 and r_2 is shown in following snaps:

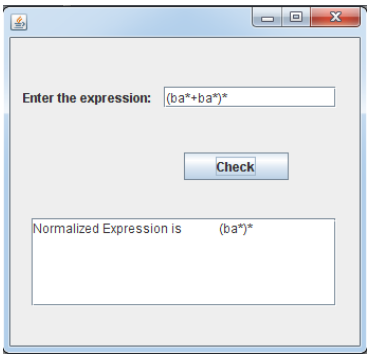


Fig. 5.12: GUI for normalizing a regular expression $(ba^*+ba^*)^*$

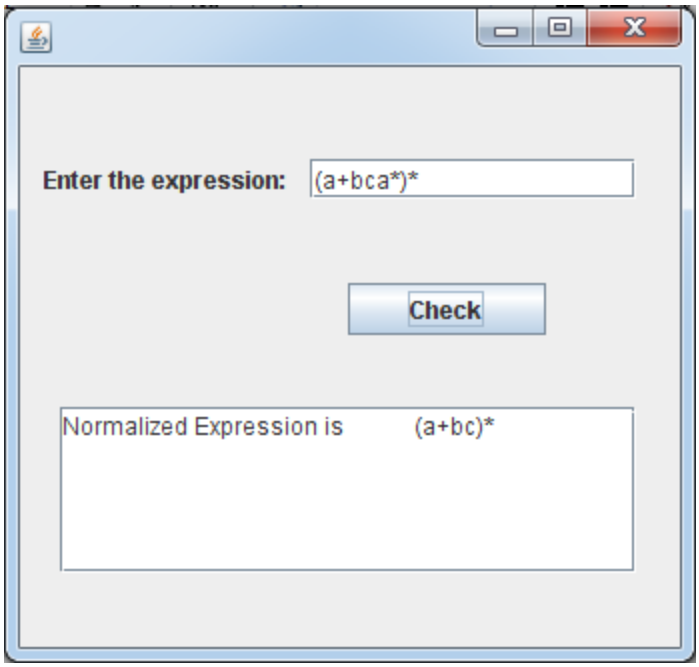


Fig. 5.13: GUI for normalizing a regular expression $(a+bca^*)^*$

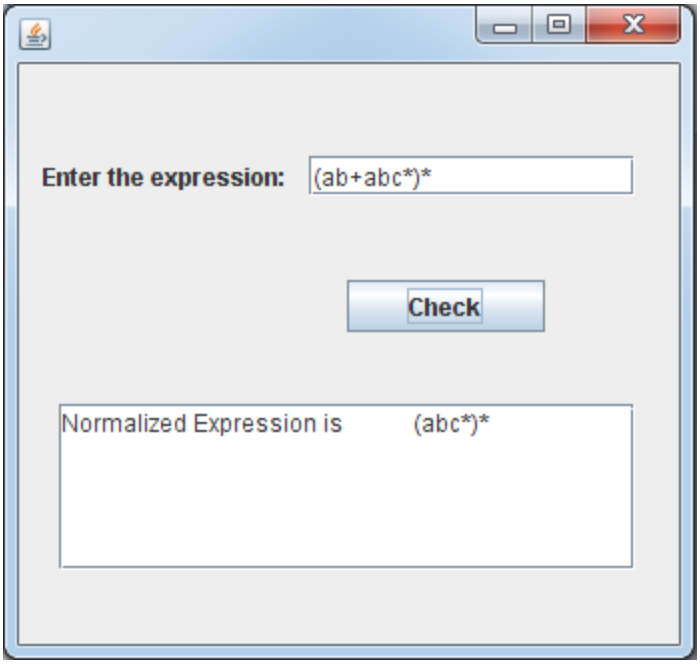


Fig. 5.14: GUI for normalizing a regular expression $(ab+abc^*)^*$

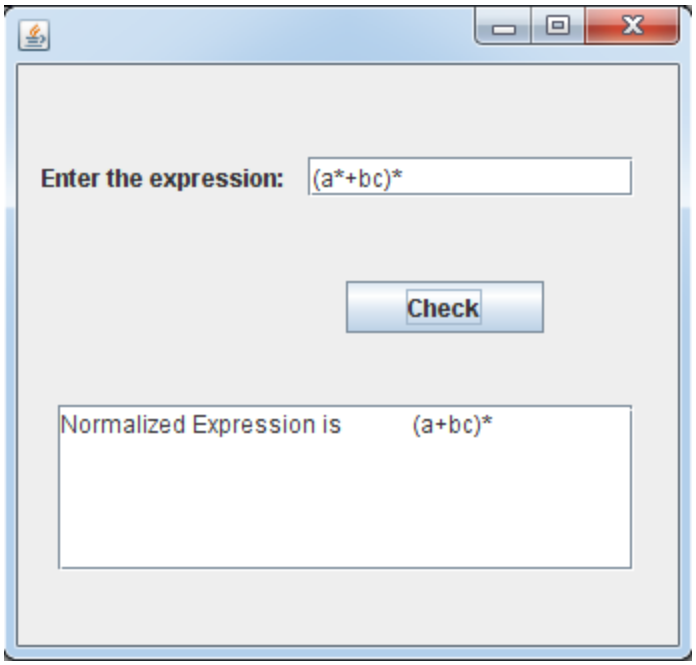


Fig. 5.15: GUI for normalizing a regular expression $(a^* + bc)^*$

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

Normalization of a regular expression will cause the reduction in length of a regular expression which will enhance the pattern matching. A DFA can be constructed with lesser number of states corresponding to normalized regular expression as compared with un-normalized regular expressions. Normalized regular expression is an equivalent regular expression with lesser ordinary length. Two regular expressions are equivalent if both generate same set of strings.

6.2 Future Scope

Two methodologies are proposed in this paper to normalize a regular expression from un-normalized regular expression. First methodology is rule based and second is based on starred trie representation of regular expression. Rule based normalization system covers regular expression with whole Kleene closure which is in the form $(r_1 + r_2)^*$, where length of r_1 and r_2 is one or two and over *a* and *b*. Methodology based on starred trie representation make overcome this problem and cover all the regular expression which is in the form $(r_1 + r_2)^*$.

Both methodologies normalize the regular expression with one round bracket. Future work can be done with the un-normalized regular expression of more than one round bracket. Methodology based on starred trie representation normalize a regular expression which in the form $(r_1)^*$ and $(r_1 + r_2)^*$. Future work can be done to normalize a regular expression with more than one union in regular expression. Methodology is to run on different machines and to develop for live examples.

References

[1] John E. Hopcraft, R. Motwani and J. D. Ullman, Introduction to automata theory, Languages and computation, Pearson Education, 2001.

[2] D. Cohen, Introduction to Computer Theory, USA: John wiley & sons, 2003.

[3] M. Sipser, Introduction to the Theory of Computation, Boston, Massachusetts: Thomson Course Technology, 2006.

[4] Fernando C. N. Pereira and Michael D. Riley, "Speech Recognition by Composition of Weighted Finite Automata", *AT&T Research*, 1996.

[5] M. Mohri, "On some application of finite state automata theory to natural language processing", *Journal of Natural Language Engineering*, Vol. 2, Issue 1, Pages 61-80, 1996.

[6] K. Najim and A. S. Poznyak, "Multimodal Searching Technique Based on Learning Automata with Continuous Input and Changing Number of Actions", *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B: CYBERNETICS*, Vol. 26, No. 4, AUGUST 1996.

[7] B. John Oommen and Edward V. de St. Croix, "String Taxonomy Using Learning Automata", *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS*, VOL. 27, NO. 2, APRIL 1997

[8] H. Gruber and M. Holzer, "Finite Automata, Digraph Connectivity, and Regular Expression Size", "*Springer-Verlag Berlin Heidelberg*", Part 2, Page 39-50, 2008.

[9] H. Gruber, J. Lee and J. Shallit, "Enumerating Regular Expressions and their Language", *Mathematics subject classification*, cite as "arXiv:1204.4982", 2010.

[10] K. Ellul, B. Krawetz, J. Shallit, and Ming-wei Wang, "Regular Expressions: New Results and Open Problems", *Journal of Automata, Languages and Combinatorics,* Vol.9, Issue2-3, Pages 233-256, 2004.

[11] M. Almeida, N. Moreira and R. Reis, "Testing the equivalence of Regular Expressions," *Universidade do Porto,Technical report series: DCC-2007-07,* vol. 1.1, 2008.

[12] J. Champarnaud, F. Ouardi and D. Ziadi, "Normalized Expressions and Finite Automata," *International Journal of Algebra and Computation*, vol. 17, page 141-154, 2007.

[13] D. Ziadi and Jean-Marc Champarnaud, "An optimal parallel algorithm to convert a regular expression into its Glushkov automaton", *ELSEVIER, Theoretical Computer Science,* Vol. 215, Issues 1–2, Pages 69–87, 1999.

[14] A. Brüggemann-Klein, "Regular expressions into finite automata", *ELSEVIER,* Vol. 120, Issue 2, Pages 197–213, 22 November 1999.

[15] H. Schildt, "Java : The complete Reference", *McGraw Hill Publication,* Seventh Edition, 2007.

[16] P. Linz, An Introduction to formal Languages and Automata, New Delhi: Narosa Publication House, 2003.

[17] Oracle,"The Java Tutorials," Available: http://docs.oracle.com/javase/tutorial/essential/regex/intro.html

[18] Oracle,"The Java Tutorials," Available: http://docs.oracle.com/javase/tutorial/essential/regex/index.html

[19] K. Thompson, "Regular expression search algorithm", *Communications of the ACM 11 (6),* Page 419–422, 1968.

[20] Oracle,"The Java Tutorials," Available: http://docs.oracle.com/javase/tutorial/essential/regex/test_harness.html

[21] P. Garcia, D. Lopez, J. Ruiz and G. I. Alvarez, "From regular expressions to smaller NFAs", *ELSEVIER,* Vol. 412, Issue 41, P. 5802–5807, 23 September 2011.

[22] W. Gelade and F. Neven, "Succinctness of the Complement and Intersection of Regular Expressions ", *ACM Transactions on Computational Logic,* P. 1-21, 2008.

[23] V. M.Antimirov and Peter D. Mosses, "Rewriting extended regular expressions", *Theoretical Computer Science 143,* Pages 51-72, 1995.

[24] H. Chen, "Finite Automata of Expressions in the Case of Star Normal Form and One Unambiguity ", *State Key Laboratory of Computer Science, Institute of Software,* ISCAS-LCS-10-11, 2010.

[25] H. Gruber and S. Gulan, "Simplifying Regular Expressions: A Quantitative Perspective", *IFIG Research Report 0904,* August 2009.

[26] A. Bruggemann-Klein and D. Wood, "One-unambiguous regular languages, Information and Computation", *ELSEVIER, Information and computation,* Vol. 140, Issue 2, Pages 229–253, 1 February 1998.

[27] D. Dougherty AND Tim O'Rrilly, Unix Text Processing, Hayden Books: A Division of Howard W. Sams & Company , 2004

[28] Lucian Ilie and Sheng Yu, "Follow automata", *ELSEVIER, Information and computation*, Volume 186, Issue 1, Pages 140–162, 10 October 2003.

List of Publication

Published

[1] J.S. Khedar and Mr. A. Kumar, "Rule based normalization of Regular Expression", International Journal of Advanced Research in computer science and Software Engineering, ISSN: 2277 128X, Vol. 3, Issue 5, 2013.