State Complexity of Various Operations on Infix-free Regular Languages

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

Master of Engineering
In
Software Engineering

Submitted By
Krishan Kumar Agrawal
(800931011)

Under the supervision of:
Mr. Ajay Kumar Loura

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2011

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"State Complexity of Various operations on Infix-Free Regular Languages"**, 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 Loura*** and refers other researcher's work which are duly listed in the reference section. The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Krishan Kumar
(Krishan Kumar Agrawal)

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

A Kumar 16/6/11
(Mr. Ajay Kumar)
Assistant Professor
Computer Science & Engineering Department
Thapar University Patiala

Countersigned by:

(Dr. Maninder Singh)
Head 16/6/11
Computer Science and Engineering Department
Thapar University
Patiala

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

Acknowledgement

First of all I would like to thank the Almighty, who has always guided me to work on the right path of the life.

This work would not have been possible without the encouragement and able guidance of my supervisor **Mr. Ajay Kumar**. I thank my supervisors for their time, patience, discussions and valuable comments. Their enthusiasm and optimism made this experience both rewarding and enjoyable.

I am equally grateful to **Dr. Maninder Singh**, Associate Professor & Head, Computer Science & Engineering Department, a nice person, an excellent teacher and a well – credited researcher, who always encouraged me to keep going with work and always advised me with his invaluable suggestions.

I will be failing in my duty if I don't express my gratitude to **'Dr. Abhijit Mukherjee'**, Director of the University**,** for making provisions of infrastructure such as library facilities, computer labs equipped with net facilities, immensely useful for the learners to equip themselves with the latest in the field.

I am also thankful to the entire faculty and staff members of Computer Science and Engineering Department for their direct-indirect help, cooperation, love and affection, which made my stay at Thapar University memorable.

Last but not least, I would like to thank my family whom I dearly miss and without whose blessings none of this would have been possible. To my parents, I own thanks for their wonderful love and encouragement. I would also like to thank my brother, since he insisted that I should do so. I would also like to thank my close friends for their constant support.

Krishan Kumar Agrawal

Roll No. - 800931011

ABSTRACT

Regular languages are used in various 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 language can be described by regular expression or finite automata (deterministic or non-deterministic). Regular languages are of various types like infix free, prefix free, suffix free. These are used in various applications like pattern matching, computing forbidden words and text searching.

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 or non-deterministic finite automata that accepts the language obtained from the operation. For state complexity, we have to observe minimal state deterministic and non-deterministic models. For example, if the state complexity of the union of an m-state DFA language and an n-state DFA language is mn. It means that there exists two regular languages which are respectively accepted by an m-state DFA and an n-state DFA, such that their union is accepted by a mn-state DFA in the worst case.

A regular language is infix-free if, for all distinct strings x, y $\in \sum^*$, and x, y $\in$ L imply that x and y are not substrings of each other. State complexity for deterministic and non-deterministic model in case of infix-free language have been determined. A software is designed for checking whether the language is infix-free or not.

Table of Contents

List of Abbreviations

DFA	Deterministic Finite Automata
NFA	Non- Deterministic Finite Automata
FA	Finite Automata
RE	Regular Expression
DFS	Depth First Search

List of Figures

List of Tables

Chapter 1

Regular Language

This chapter gives an introduction to regular language, their operations and some basic definitions involved in regular language. This chapter also contains outline of thesis along with brief introduction to each chapter.

1.1 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). Regular languages are used in various fields of computer science like compilers [21], data compression [21], text processing [6], software engineering [21], and pattern matching [21] 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 [19], computing forbidden words [10] and text searching [7]. A language L is regular if and only if there is a regular expression E such that L = L(E).

1.2 Basic Notations and Definitions

1.2.1 Alphabet and Language

An alphabet is a finite, nonempty set of symbols, denoted by $\sum$. The notation $\sum^*$ means the set containing all the finite strings whose symbols are chosen from an alphabet $\sum$. Strings, which are finite sequences of letters, are also called words. Null language is denoted by Φ. Symbol ε denotes the null string. For a word x over an alphabet $\sum$, its length is the number of occurrences of letters in x. It is denoted by $|x|$. The a-length of the word x is the number of times that the letter 'a' appears in x [8]. It is denoted by $|x|_a$. A language over $\sum$ is a set of words which are chosen from $\sum^*$. The notation $|L|$ will be used to denote the cardinality of a language L, i.e., the number of words in L.

1.2.2 Regular Expression

A regular expression [13] is a pattern that describes some set of strings. A regular expression 'r' matches a string s if s is in the set of strings described by r.

They are usually used to give a concise description of a set, without having to list all elements.

Example 1: a|b* denotes {ε, a, b, bb, bbb, ...}

(a|b)* denotes the set of all strings with no symbols other than a and b, including the empty string: {ε, a, b, aa, ab, ba, bb, aaa, ...}

ab*(c|ε) denotes the set of strings starting with a, then zero or more b's and finally optionally c: {a, ac, ab, abc, abb, abbc, ...}

Set containing the three strings "Handel", "Händel", and "Haendel" can be described by the pattern `H(ä|ae?)ndel`. Alternatively, it is said that the pattern matches each of the three strings.

1.2.3 Deterministic Finite Automata

A deterministic finite-state automaton is a finite state machine accepting finite strings of symbols.

Definition 1: DFA [13] is defined by 5-tuples (Q, ∑, δ, s, F) where

Q is a finite set of states,

∑ is an input alphabet

δ: QX∑→Q is a transition function,

s is the starting state and belongs to Q.

F is subset of Q is a set of final states.

Example 2: Given a DFA M, with binary alphabet, which accept all string containing even number of 0s .Let M = (Q, Σ, δ, s, F) where Σ = {0, 1}, Q = {s_1, s_2}, F = {s_1}

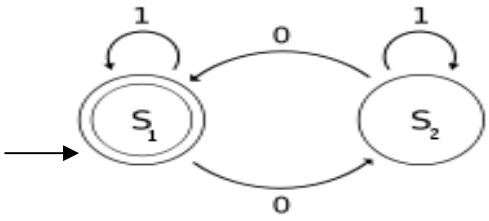


Figure 1.1: Deterministic Finite Automata M

The transition function δ can be defined by state transition table

States/Input	0	1
S_1	S_2	S_1
S_2	S_1	S_2

Table 1.1: State Transition table for given automata M

1.2.4 Non-Deterministic Finite Automata

Definition 2: A non-deterministic finite state automaton (NDFA) [13] is same as DFA except the transition function defined as $QX\sum \rightarrow 2^Q$.

Example 3: Given a NFA M (Q, Σ, δ, s, F) where Q = {1, 2, 3, 4, 5, 6}, Σ = {a, b}, s={1}, F = {6}

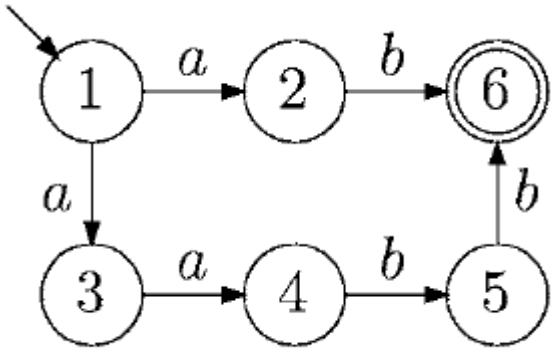


Figure 1.2: Non-deterministic finite automata M

The transition function δ can be defined by state transition table

States/Input	a	b
1	{2,3}	-
2	-	6
3	4	-
4	-	5
5	-	6
6	-	-

Table 1.2: State transition table for given automata M

|Q| denotes the number of states in Q. and |δ| denotes the number of transitions in δ. Then the size of automata A is denoted by |A| and equal to |Q|+|δ|. For a transition δ(p, a) = q in A, p has an out-transition and q has an in-transition. Furthermore, p is a source state of q and q is a target state of p.

Definition 3: A finite automaton A is non-returning if the start state of A does not have any in-transitions and A is non-exiting if all final states of A do not have any out-transitions.

A string x over ∑ is accepted by A if there is a labeled path from s to a state in F such that this path spells out the string x. Thus, the language L(A) of a finite-state

automaton A is the set of all strings that are spelled out by paths from s to a final state in F.

1.2.5 Operations on Regular Languages

There are various operations that can be performed on regular languages, which are described below:

a) Intersection of Regular Languages: Suppose L_1 and L_2 are two regular languages, then intersection of these regular languages is given by $L_1 \cap L_2$.

Example 4: suppose $L_1=\{a^nb\}$ and $L_2=\{ab^m\}$ then $L_1 \cap L_2=\{ab\}$

b) Union of Regular Languages: Suppose L_1 and L_2 are two regular languages, then union of these regular languages is given by $L_1 \cup L_2$, which represents language that accepts all string from L_1 and L_2. If E and F are regular expression, then E+F is a regular expression denoting the union of L(E) and L(F).

c) Concatenation of Regular Languages: If E and F are regular expressions, then E.F is a regular expression denoting the concatenation of L(E) and L(F). The Concatenation of languages L_1 and L_2 is represented by $L_1L_2= \{xy \mid x \in L_1, y \in L_2\}$. It denotes all words that are formed by concatenation of a word x from L_1 with a word y from L_2.

Example 5: If $L_1=\{a, ab\}$ and $L_2=\{aa, bb, abb\}$ then $L_1L_2=\{aaa, abb, aabb, abaa, abbb, ababb\}$

d) Closure or 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. It is denoted by L^*.

Example 6: If the regular language L contains alphabets $\sum=\{a, b\}$ then $L^*=\{\varepsilon,a, b, ab, ba, aa,\ldots\}$ i.e. any string of a and b.

e) Complement of Regular Languages: The complement of a regular language is defined with respect to $\sum^*$. The complement of a regular language L is given by $\overline{L} = \sum^* - L$

Example 7: Suppose $\sum =\{a\}$ and the language L consists of all non empty string of even length of a's. $L=\{aa, aaaa, aaaaaa, aaaaaaaa,\ldots\ldots\ldots\}$. Then the complement of the language L is given by $\overline{L}=\sum^* - L =\{\varepsilon, a, aaa, aaaaa,\ldots\ldots\}$i.e. $\overline{L}$ consists of all string of odd length of a's including null string.

f) Reversal of Regular Languages: Given an NFA A $(Q, \sum, \delta, s, F)$ for a regular language L, then a NFA A^R $(Q, \sum, \delta, f, s)$ for L^R can be constructed by flipping the

directions of all transitions and interchange the start state and the final state. Since L^R =L(A^R) and both A and A^R have the same set Q of states.

1.2.6 Sink States: A complete DFA is one that has transitions defined for each state in Q and each input symbol in ∑. A sink state is a state from which there exists no sequence of transitions to a final state [26].

1.2.7 Forms of Regular Languages

There are prefix free, infix free and suffix free forms of regular languages.

a) Prefix-Free: A language L is prefix free [19] if, for all distinct strings x, y ∈ ∑* and x, y ∈ L imply that x and y are not prefixes of each other. Given two strings x and y over ∑, x is a prefix of y if there exists z ∈ ∑* such that xz = y. 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.

b) Infix-Free: A language L is infix free [19] if, for all distinct strings x, y ∈ ∑*, and x, y ∈ 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. 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.

c) Suffix-Free: A language L is suffix free [19] if, for all distinct strings x, y ∈ ∑*, and x, y ∈ L imply that x and y are not suffixes of each other. Given two strings x and y over ∑, x is a suffix of y if there exists z ∈∑* such that zx = y. 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.

1.3 Outline of Thesis

The thesis is structured as follows:

Chapter 1 gives a brief introduction of thesis and background information relating to regular languages, various operations on regular languages and different forms of regular languages.

Chapter 2 gives a Literature survey of research results on various forms of regular languages, checking the various forms like infix-freeness, prefix-freeness, suffix-freeness and state complexities of various basic operations on various forms of regular languages.

Chapter 3 describes motivation behind thesis and discusses problem statement and state complexities of operations, including union, intersection, concatenation, reversal, and complement on infix free regular language.

Chapter 4 presents the implementation details of proposed algorithm for checking infix freeness of regular expression.

Chapter 5 concludes the thesis along with future work.

Chapter 2

Literature Review

This chapter describes the various forms of regular languages, and various approaches for checking the prefix freeness, infix freeness and suffix freeness as proposed by Han[19,22] and state complexity of various basic operations on prefix free and suffix free regular languages for both deterministic and non-deterministic finite automata.

2.1 Prefix Free Regular Languages

A language L is prefix free [19] if, for all distinct strings x, y $\in \sum^*$, x $\in$ L and y $\in$ L implies that x and y are not prefixes of each other. Given two strings x and y over $\sum$, x is a prefix of y, if there exists z $\in \sum^*$ such that xz = y. For example: given two strings x = abcbbab and y = abcbbababab. string x is prefix of string y. The language that consists of string x and y is not prefix free.

A regular expression E is prefix-free if the language L(E) is prefix-free and FA A is prefix-free if L(A) is prefix-free. Moreover, if L(A) is prefix-free, then A must be non-exiting. If a prefix-free FA A has several final states, then all final states are equivalent and, thus can be merged into a single final state[19]. Since A is non-exiting. therefore a minimal NFA for a prefix-free regular language must have a single final state[22].

2.1.1 Checking Prefix Freeness of Finite Languages

Han proposed an approach for checking prefix freeness of a finite language [22].Given a finite set of strings W = {w_1,w_2, . . . ,w_n}, where n is the number of strings in W. A trie is an ordered tree that is used to store a set of strings and each edge in the tree has a single character label [22]. A trie T is constructed for W.

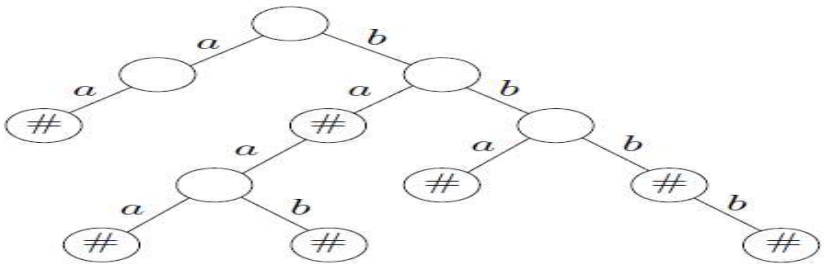


Figure 2.1: The trie for W = {aa, ba, baaa, baab, bba, bbb, bbbb}.

In figure 2.1 # denotes the end-marker for the corresponding string. If a node q in T has an end-marker, then it means that the corresponding string from the root to q is in W. Assume that string w_i is a prefix of string w_j , where $i \neq j$. It implies that $|w_i| < |w_j|$. Then w_i and w_j must have common path in T from the root to i^{th} node q. Therefore, if q is reachable while traversing the path for w_j in T, then w_i is a prefix of w_j.

Assume the case when construct a path for w_j first and then, construct a path for w_i in T. The path for w_i ends at the $|w_i|^{th}$ node q that already has a child node for the path for w_j. Therefore, w_i is a prefix of some other string. Trie for W can be constructed in $O(|w_1| + |w_2| + \cdots |w_n|)$ time, which is linear in the size W. Han [22] proposed that it can be determined whether a finite language is prefix free or not in linear time in the size of W by constructing a trie for W. A trie T for W is constructed in linear time and checked if any internal node has an end-marker while traversing T in linear time. If any internal node with an end-marker is identified, then W is not prefix-free. Otherwise, W is prefix-free.

2.1.2 Checking Prefix Freeness of Regular language

A regular language is represented by a FA or described by a regular expression. Y.Han [22] proposed an approach for determining whether a regular language is prefix free or not on basis of following conditions.

1. If the regular language is represented by a deterministic finite automata, then regular language L(A) is prefix-free, if and only if deterministic finite automata A is non-exiting. If A has out-transitions from a final state, then L(A) is not prefix-free[22]. If A is non-exiting and has several final states, then all final states are equivalent and, therefore, can be merged into a single final state.
2. If the regular language is represented by a non-deterministic finite automata, then if there are two strings s_1 and s_2 accepted by A and s_1 is a prefix of s_2. It means that there are two distinct paths in A that spell out s_1 and s_2 and these two paths spell out the same prefix s_1. For example, in figure 2.2, two paths for s_1 = abcbb and s_2 = abcbbab are different although they have the same sub-path for ab in common. If the path for s_1 is a sub path of the path for s_2, then it implies that there is another final state that has an out-transition. This contradicts that A is non-exiting. Han [22] proposed that if the regular language is represented by NFA in which there is an out transition from final

state then regular language is not prefix free. If the NFA is non exiting then first state pair graph is constructed to identify the unique path for common string.

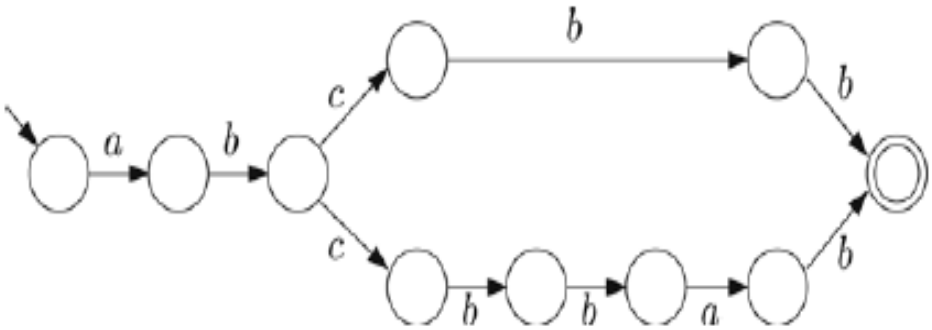


Figure 2.2: Two distinct paths for abcbb and abcbbab.

Given an NFA A = (Q, ∑, δ, s, f), then each state is uniquely numbered in order to construct state pair graph. By assigning a unique number for each state from 1 to m, where m is the number of states in Q. states q_i for $1 \leq i \leq m$ is used to denote the corresponding state in A. For example, q_1 denotes s and q_m denotes f .

Definition 2.1: Given an NFA A = (Q, ∑, δ, s, F), the state-pair graph G_A =(V, E), where V is a set of nodes and E is a set of edges, can be defined as follows[19].

$V = \{(i, j) \mid q_i \text{ and } q_j \in Q\}$ and

$E = \{((i,j),a,(x,y)) \mid (q_i,a,q_x) \text{ and } (q_j,a,q_y) \in \delta \text{ and } a \in \sum\}$

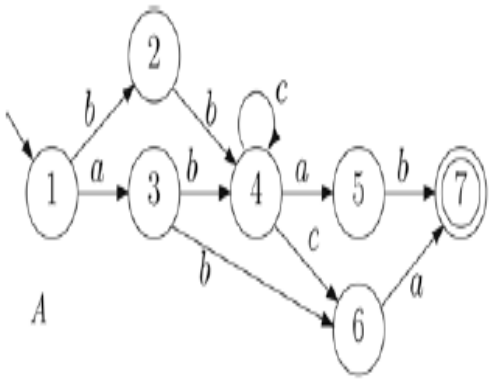


Figure 2.3(a): Given finite automata A

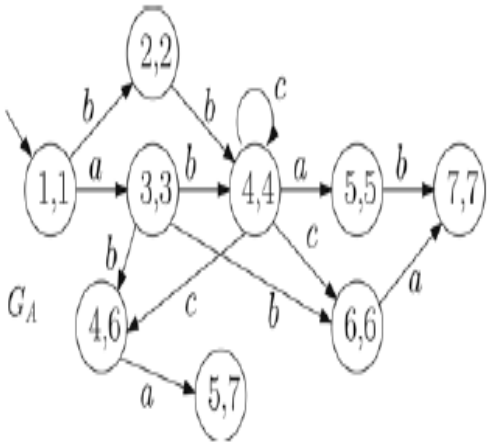


Figure 2.3(b): State pair graph for given finite automata A

Han [22] proposed that, L(A) is prefix-free if and only if there is no path from (1, 1) to (m, j), for any j ≠m, in G_A. All nodes that are unreachable from node (1,1) are

omitted in the state pair graph. Given L(A) is not prefix-free since A accepts both aba and abab. Prefix aba appears on the path (1, 1)⟶(3, 3)⟶ (4, 6)⟶(5, 7) in G_A.

2.2 Suffix Free Regular Languages

A language L is suffix free [19] if, for all distinct strings x, y ∈ ∑*, x∈ L and y ∈ L imply that x and y are not suffixes of each other. Given two strings x and y over ∑, x is a suffix of y if there exists z ∈∑* such that zx = y. Given two strings x = abb and y = aaabb. string x is suffix of string y. The language that consists of string x and y is not suffix free. A regular language L is said to be suffix-free if L is a suffix-free set [22]. A regular expression E is suffix-free if L (E) is suffix-free. Similarly, an FA A is suffix-free if L (A) is suffix-free. Moreover, if L(A) is suffix-free and non-empty, then A must be non-returning. The advantage of prefix-free codes is that a given encoded string can be decoded deterministically [22].

2.2.1 Checking Suffix Freeness of Finite language

Han [22] proposed an approach for checking suffix freeness of a finite language. Given a finite set of strings W = {w_1,w_2, . . . ,w_n}, where n is the number of strings in W. A trie is an ordered tree that is used to store a set of strings and each edge in the tree has a single character label [22]. A trie T is constructed for W^R. Suppose W={aa, ab, aaab, baab, abb, bbb, bbbb}

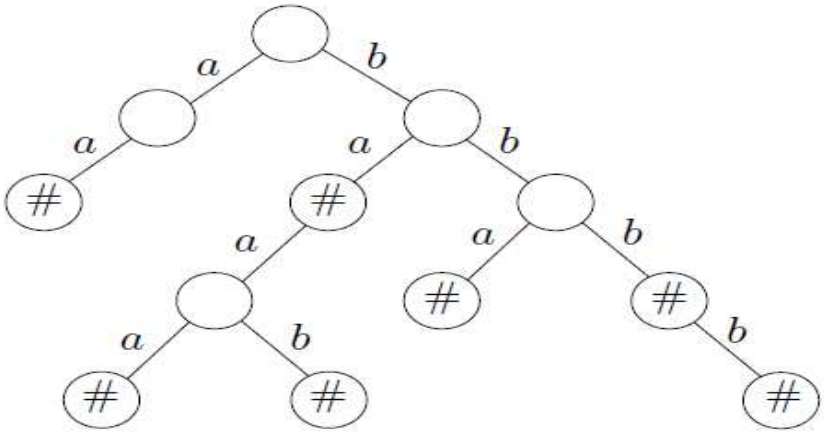


Figure 2.4: The trie for W^R = {aa, ba, baaa, baab, bba, bbb, bbbb}.

In figure 2.4 # denotes the end-marker of the corresponding string. A trie for W^R can be constructed in O($|w_1| + |w_2| + \cdots |w_n|$) time, which is linear in the size of W. Han [22] proposed that it can be determined whether a finite language is suffix free or not in linear time in the size of W by constructing a trie for W^R. A trie T for W^R is constructed in linear time and checked if any internal node has an end-marker while

traversing T in linear time. If any internal node with an end-marker is identified, then W^R is not prefix-free therefore W is not suffix free.

2.3 Infix Free Regular Language

A language L is infix free [19] if, for all distinct strings x, y ∈ ∑*, x ∈ L and y ∈ 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. Given two strings x = aba and y = aabab. string x is infix or substring of string y. The language that consists of string x and y is not infix free. Han [19] proposed that the family of infix-free regular languages is closed under concatenation and intersection but not under union, complement or star.

2.3.1 Checking Infix Freeness of Regular language

A regular language is represented by a finite-state automaton or described by a regular expression. it is assumed that ε does not belongs to L, where L is not equal to ε. otherwise, L is not infix-free since ε is an infix of all strings.

1) Consider the representation of a regular language L by an NFA A. If L (A) is not infix-free, then there are two distinct strings s_1 and s_2 accepted by A and s_1 is an infix of s_2. It implies that there are two distinct paths in A that spell out s_1 and s_2, respectively, and the path for s_2 has a sub-path that spells out s_1.
2) If a final state in A has an out-transition, then L(A) is not prefix-free and, therefore, not infix-free. Similarly, if start state of A has an in-transition, then L(A) is not suffix-free and, therefore, not infix-free. Thus, it can be assumed that A is non-returning and non-exiting. Furthermore, if A is non-exiting and has several final states, then all final states are equivalent and, therefore, are merged into a single final state.

For example, in figure 2.4, the given finite-state automaton accepts s_1 = abba and s_2 = aabbab and the sub path $q_2 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7 \rightarrow q_8$ of the path for s_2 also spells out s_1.

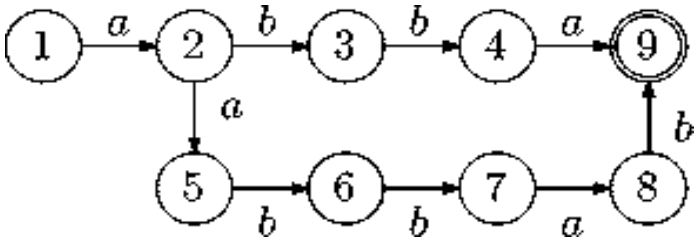


Figure 2.5: Strings abba and aabbab showing two distinct paths in a given finite-state automaton [19].

Han [19] proposed that if the regular language is represented by NFA in which there is no out transition from final state and no in-transition to start state. Then state pair graph is constructed as defined in definition 2.1 that is able to identify the case when two distinct paths in A spell out s_1 and s_2, and s_1 is an infix of s_2 as illustrated in figure 2.6(a). Given an NFA A = (Q, ∑, δ, s, f), then each state is uniquely numbered in order to construct state pair graph. By assigning a unique number for each state from 1 to m, where m is the number of states in Q. States q_i for $1 \leq i \leq m$ is used to denote the corresponding state in A.

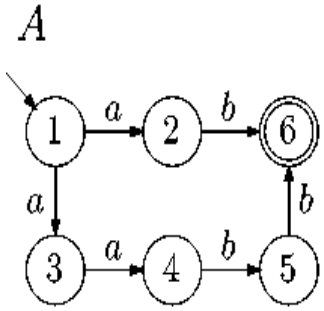


Figure 2.6(a): Given finite automata A

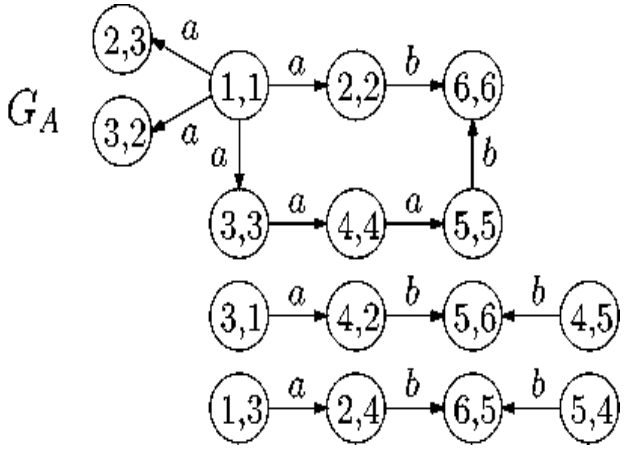


Figure 2.6(b): State pair graph for given finite automata A

For example, q_1 denotes s and q_m denotes f. Figure 2.6(b) illustrates the state-pair graph for a given finite-state automaton A. All nodes that have no out-transitions are omitted in state pair graph G_A. Han [19] proposed that if there is no path from (l,i) to (m,j) in G_A except from (1,1) to (m,m), for $1 \leq i$, $j \leq m$, then L(A) is infix-free. Language L(A) {ab, aabb} is not infix-free since ab is an infix of aabb. Note that the infix ab appears on the path from (1,3) to (6,5) in G_A .

Algorithm 2.1: Infix-Freeness (A = (Q, ∑, δ, s, F))

Step 1: if A is not non-exiting or not non-returning

then print " Not infix-free "

return

end if

Step 2: Construct state pair graph G_A = (V, E) from A

Step 3: Repeat step 4, 5 for each node (1, i) in V, where $2 \leq i \leq m$

Step 4: DFS ((1, i)) in G_A

Step 5: if we meet a node (m, j) for any j, $1 \leq j \leq m$

then output L(A) is not infix-free

end if

Step 6: DFS ((1, 1)) in G_A

Step 7: if a node (m, j) is found for any j, $2 \leq j < m$

then output L(A) is not infix-free

end if

Complexity of the state-pair graph G_A (V, E) for a given finite-state automaton A(Q, $\sum$, δ, s, f) is at most $|Q|^2$ nodes and $|\delta|^2$ edges. A sub-function DFS((i,j)) in infix-freeness (IF) algorithm is a depth-first search that starts at a node (i, j) in G_A. Although DFS((i, j)) is executed several times inside for loop in the algorithm, each node in G_A is visited at most twice. The construction of G_A = (V, E) from A takes $O(|Q|^2+|\delta|^2)$ time in the worst-case and DFS takes O(|V| + |E|) time. Therefore, the total running time for infix freeness is $O(|Q|^2 + |\delta|^2)$.

If a language is described by a regular expression, then construction of finite-state automata can be choosen that improves the worst-case running time. Since the complexity of the state-pair graph depends on the number of states and the number of transitions of a given automaton, so there is a need of finite-state automata construction that gives fewer states and transitions. Han [19] proposed an algorithm which uses Thompson construction to convert regular expression to NFA with null transitions. State pair graph have large number of states and transitions which can be improved by using an efficient approach for converting regular expression to finite automata which have less number of states.

2.4 State Complexity of Prefix free Regular Languages

The state complexity of prefix free languages for various kinds of operations can be deterministic or non-deterministic.

2.4.1 Non-Deterministic State Complexity of Prefix free Regular Languages

Han, Salomaa, and Wood [28] works on the non-deterministic state complexity of prefix free regular languages for various operations.

i) Intersection of Prefix-Free Regular Languages

Given two finite automata A and B, then a FA for the intersection of L(A) and L(B) can be constructed based on the cartesian product of states. Given finite automata $A(Q_1, \sum, \delta_1, s_1, F_1)$ and $B(Q_2, \sum, \delta_2, s_2, F_2)$. Let $M = (Q_1 \times Q_2, \sum, \delta, (s_1, s_2), F_1 \times F_2)$, where for all $p \in Q_1$ and $q \in Q_2$ and $a \in \sum$, and transition function δ is given by $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ and $L(M) = L(A) \cap L(B)$. This construction shows that mn states are sufficient in the worst case for the intersection of L(A) and L(B), where $|A| = m$ and $|B| = n$.

It has been observed that [28], in the case when L(A) and L(B) are prefix-free, M has some useless states and, thus M is not minimal. Approach used to reduce the number of states by identifying and removing these useless states from M based on structural properties of prefix-free NFAs. A unique number is assigned to each state from 1 to m in A and from 1 to n in B, where $|A| = m$ and $|B| = n$. Assume that the m^{th} state and the n^{th} state are the final states in A and B, respectively. Let $A \cap_c B$ denote the resulting intersection automaton that is computed based on the cartesian product of states. Then, (1, 1) is the start state and (m, n) is the unique final state. All states (m, i) and (j, n), for $1 \leq i \leq n-1$ and $1 \leq j \leq m-1$, do not appear in an accepting path in $A \cap_c B$ since L(A) and L(B) are prefix-free. If a state (m, i) for $i \neq n$ appears in an accepting path, then the final state m of A has an out transition and this contradicts that L(A) is prefix-free. All states (m, i) and (j, n), for $1 \leq i \leq n-1$ and $1 \leq j \leq m-1$ can be removed, since they are useless.

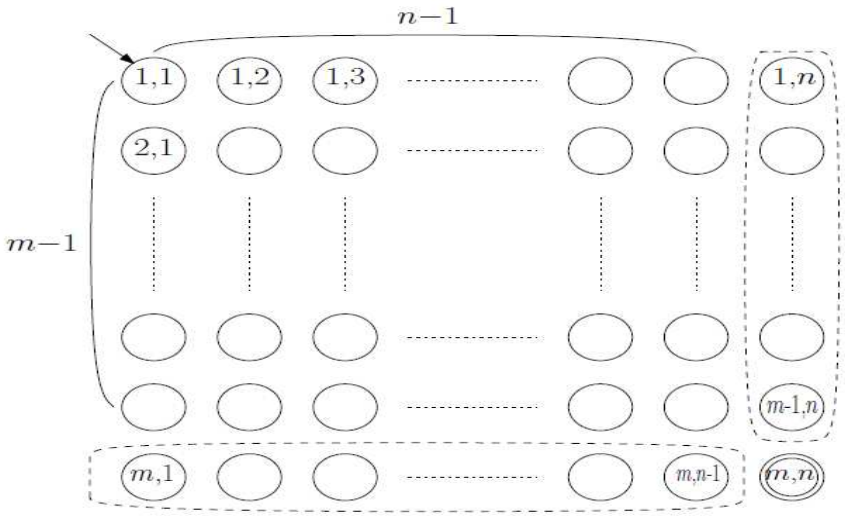


Figure 2.7: Intersection of two prefix-free minimal NFAs

All transitions are omitted, and all states inside the two dotted boxes are useless [28]. These useless states can be removed, the resulting automaton has $mn-(m-1)-(n-1) = mn-(m+n)+2$ states. Thus $mn-(m+n)+2$ states are sufficient for $L(A) \cap L(B)$, where $|A| = m$ and $|B| = n$.

Given two prefix-free regular languages L_1 and L_2, then the nondeterministic state complexity for $L_1 \cap L_2$ is $mn-(m+n)+2$, where m and n are nondeterministic state complexity of L_1 and L_2 respectively.

ii) Union of Prefix-Free Regular Languages

Given two prefix-free regular languages L_1 and L_2, then the nondeterministic state complexity for $L_1 \cup L_2$ is m + n, where m and n are nondeterministic state complexity of L_1 and L_2 respectively.

Let A $(Q_1, \sum, \delta_1, s_1, F_1)$, be a minimal NFA for L_1 and B $(Q_2, \sum, \delta_2, s_2, F_2)$, be a minimal NFA for L_2. NFA C for $L_1 \cup L_2$ is constructed by introducing a new start state for the alternation of L(A) and L(B). Namely, C = $(Q, \sum, \delta, s, \{f_1, f_2\})$, where $Q = Q_1 \cup Q_2 \cup \{s\}$ and for a state $q \in Q$ and a character $a \in \sum$.

$$\delta(p, a) = \begin{cases} \delta_1(s,a) \cup \delta_2(s,a) & \text{if } p=s \\ \delta_1(p,a) & \text{if } p \in Q_1 \\ \delta_2(p,a) & \text{if } p \in Q_2 \end{cases}$$

It can be observed that the two final states f_1 and f_2 are equivalent and, thus f_1 and f_2 can be merged. Thus $m + n + 1 - 1 = m + n$ states are sufficient for $L_1 \cup L_2$.

iii) Concatenation of Prefix-Free Regular Languages

Given two prefix-free regular languages L_1 and L_2, the nondeterministic state complexity for L_1L_2 is $m + n - 1$, where m and n represents nondeterministic state complexity of L_1 and L_2 respectively.

Let A = $(Q_1, \sum, \delta_1, s_1, f_1)$, be a minimal NFA for L_1 and B = $(Q_2, \sum, \delta_2, s_2, f_2)$, be a minimal NFA for L_2. It can be assumed that A and B have only one final state, that is non-exiting. An NFA C for L(A).L(B) can be constructed by merging the final state f_1 of A and the start state s_2 of B to give a single state that also eliminates all out-transitions of f_1 of A. C=$(Q, \sum, \delta, s_1, f_2)$ where $Q=Q_1 \cup Q_2 \setminus \{f_1\}$ and for each state $p \in Q$ and $a \in \sum$ and transition function δ is given by

$$\delta(p, a) = \begin{cases} s_2 & \text{if } p \in Q_1 \text{ and } \delta_1(p,a)=f_1 \\ \delta_1(p,a) & \text{if } p \in Q_1 \text{ and } \delta_1(p,a) \neq f_1 \\ \delta_2(p,a) & \text{otherwise} \end{cases}$$

It is easy to verify that L(C) = L(A)L(B) from the construction. Since Q = Q_1 UQ_2 \ {f_1} and therefore, the construction shows that m + n − 1 states are sufficient in the worst case for the concatenation of L (A) and L (B), where m and n are non-deterministic state complexity of A and B respectively.

iv) Reversal of Prefix-Free Regular Languages

Given a prefix-free regular language L, the non-deterministic state complexity for L^R is m where m is non-deterministic state complexity of L. Given a minimal NFA A =(Q, ∑, δ , s, f) that accepts L, where |Q| = m. A has a single final state and it is non-exiting. Based on the structural property, an NFA A^R for L^R can be obtained by flipping the directions of all transitions and interchange the start state and the final state. New obtained NFA A^R = (Q, ∑, δ, f, s) such that L^R =L(A^R). A and A^R both are having same set of states Q, Hence m states are sufficient for L^R.

v) Complementation of Prefix-Free Regular Languages

A n-state NFA can be transformed into DFA with at most 2^n states. The complementation of an n-state DFA does not change the state complexity since it simply interchanges final states and non-final states. Thus, based on the subset construction, 2^n states are sufficient for the complementation of an n-state NFA. Given an n-state prefix-free NFA A = (Q, ∑, δ, f, s). A can be converted into DFA A^1 using subset construction and it can have at most 2^n states (some of them may be useless states), and the final states are the subsets of Q that contain f. Since L(A) is prefix-free, L(A^1) is also prefix-free and, all 2^{n-1} final states can be merged into a single final state f^1 in A^1. Therefore, the number of states for a DFA for L ($\overline{A}$) is 2^{n-1}+1. Now interchange final states and non-final states, thus the resulting FA is an NFA for L(A). 2^{n-1}+1 states are sufficient for complement language L ($\overline{A}$) of n-state NFA.

vi) Kleene Star of Prefix-Free Regular Languages

Given a prefix-free minimal NFA A (Q, ∑, δ, s, f), a new NFA is constructed $A^{'}$ (Q, ∑, $\delta^{'}$, s, f) where transition function $\delta^{'}$ is given by

$$\delta'(p, a) = \begin{cases} \delta(p,a) & \text{if } p \neq f \\ \delta(s,a) & \text{if } p = f \end{cases} \text{----------(1)}$$

It shows L($A^{'}$)=L(A)$^+$ from above construction.

Theorem 2.1-: A is a minimal NFA if and only if $A^{'}$ is a minimal NFA [28].

Proof-: Assume that A is not minimal. This implies that there exists a minimal NFA C such that L(C) = L(A) and |C| < |A|. Since L(C) is prefix-free, it is non-exiting and

has a single final state. Then, same construction is used above and obtain a new NFA C' such that $L(C') = L(C)^+ = L(A)^+$.

Since $|C'| = |C| < |A| = |A'|$, there is a smaller NFA for $L(A')$. This theorem shows that m states are sufficient for $L(A)^+$. Since for every $m \geq 1$ there exists a prefix-free language L with non-deterministic state complexity m.

Given a prefix-free minimal NFA $A=(Q, \sum, \delta, s, f)$, a minimal NFA $A'=(Q, \sum, \delta', s, f)$ is constructed such that $L(A')=L(A)^+$. Now change the start state of A' to f and denote the resulting NFA as B $(Q, \sum, \delta', f, f)$. Since f is now a start state, $\varepsilon \in L(B)$. Let $w = w_1w_2 \cdots w_k$ be a string in $L(A')$. This implies that there is an accepting path for w in A'. Since $\delta'(s, w_1) = \delta'(f, w_1)$ by equation 1. Same set of states is reached after reading w_1 in both A' and B. Consequently, the same accepting path can be followed in B because of δ' is the same. Similarly it can be shown that if a string $w \neq \varepsilon \in L(B)$, then $w \in L(A')$. This gives an upper bound of the nondeterministic state complexity for L^*. Thus nondeterministic state complexity for L^* of a prefix-free regular language is m. Deterministic state complexity of kleene closure of regular languages [5] is $2^{n-1} + 2^{n-2}$ and non-deterministic state complexity of kleene closure of regular languages is m + 1 where m is the number of states in both cases.

2.4.2 Deterministic State Complexity of Prefix free Regular Languages

Han, Salomaa, and Wood [27] works on the deterministic state complexity of prefix free regular languages for various operations.

i) Intersection of Prefix-Free Regular Languages

Given two DFAs A and B, then a DFA for the intersection of L(A) and L(B) can be constructed based on the cartesian product of states Given two DFAs A $(Q_1, \sum, \delta_1, s_1, F_1)$, and B = $(Q_2, \sum, \delta_2, s_2, F_2)$. Let M = $(Q_1 \times Q_2, \sum, \delta, (s_1, s_2), F_1 \times F_2)$, where for all $p \in Q_1$ and $q \in Q_2$ and $a \in \sum$, and transition δ is given as $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ Then, $L(M) = L(A) \cap L(B)$.

Since the resulting automaton M is deterministic, the construction shows that mn states are sufficient for the intersection of L(A) and L(B), where m and n are deterministic state complexity of A and B respectively. State complexity of intersection of prefix-free regular languages can be investigated based on the structural properties of these minimal DFAs. Intersection of two prefix-free minimal DFAs can be determined based on the cartesian product of states as suggested by Hopcroft and Ullman [2].

Assign a unique number for each state from 1 to m in A and from 1 to n in B, where |A| = m and |B| = n. Assume that $(m-1)^{th}$ state and $(n-1)^{th}$ state are final states, and m^{th} state and n^{th} state are sink states in A and B, respectively. Let $A \cap_c B$ denote the resulting intersection automaton that is computed based on cartesian product of states. Then (1, 1) is the start state and (m−1, n−1) is a final state. Now identify all equivalent states in $A \cap_c B$ and merge them to compute the minimal DFA for L(A) ∩L(B).

Theorem 2.2: Given two prefix-free minimal DFAs A and B, mn−2(m+n)+6 states are necessary and sufficient in the worst-case for the minimal DFA of L(A) ∩L(B) [27].

Proof-: Let |A| is m and |B| is n. Let M be the minimal DFA for the intersection of L(A) and L(B). All equivalent states are merged in A ∩c B. Thus, M requires (m−2)(n−2) states. M also needs one final state and one sink state. It shows that M needs (m−2)(n−2)+ 2 states and, therefore, mn − 2(m+ n)+ 6 states are necessary

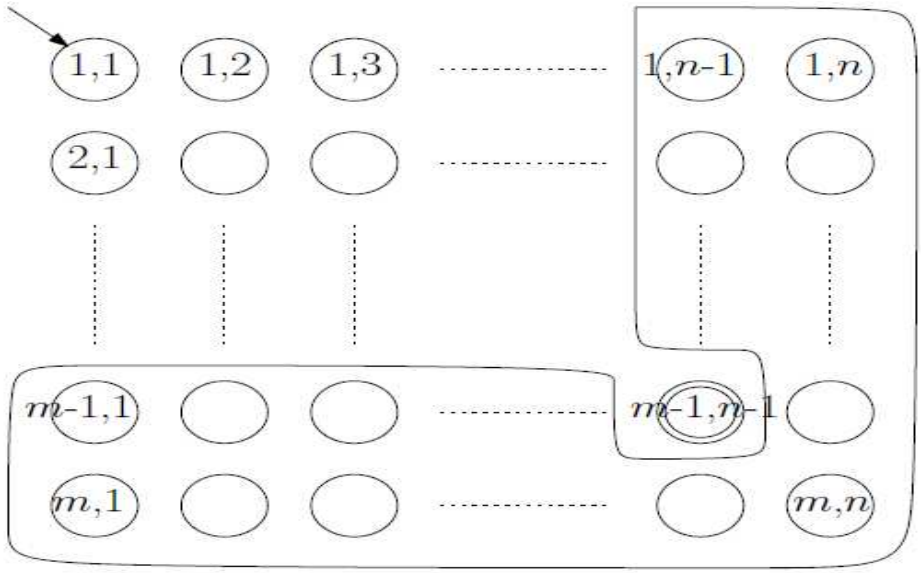


Figure 2.8: Intersection of two prefix-free minimal DFAs [27]

ii)Union of Prefix-Free Regular Languages

Given two DFAs A and B, then a DFA for the union of L(A) and L(B) can be constructed based on the cartesian product of states. Given two DFAs $A = (Q_1, \sum, \delta_1, s_1, F_1)$, and $B = (Q_2, \sum, \delta_2, s_2, F_2)$. Let $M (Q_1 \times Q_2, \sum, \delta, (s_1, s_2), F_1 \times F_2)$, denotes the union of A and B. For all $p \in Q_1$ and $q \in Q_2$ and $a \in \sum$, and δ is given by $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ and final state $F = \{(p, f_2) \mid p \in Q_1\} U \{(f_1, q) \mid q \in Q_2\}$.

L(M) = L(A) U L(B) and M is deterministic. Although the structure of A∩c B illustrated in Fig.2.8 and the structure of A Uc B are similar, following are two differences occur in A Uc and A∩c B [27]:

1. States in the second-last row and all states in the second-last column of A Uc B are final states.
2. Except for the state (n, m), all states in the last row and all states in the last column of A Uc B are not necessarily sink states.

Because of these two distinct properties, these states are not equivalent in A Uc B.

Theorem 2.3:-Given two prefix-free minimal DFAs A and B, mn−2 states are necessary and sufficient in the worst-case for the minimal DFA of L(A)UL(B) .[27]

Proof: In the resulting DFA M, L(m,n−1) = L_m U L_{n-1} ={ε} and L(m-1,n) = L_{m-1} U L_n ={ε} since A and B are non-exiting. Therefore, L(m,n−1) = L(m−1,n) = L(m−1,n−1). These three states (m, n−1), (m−1, n) and (m−1, n−1) are merged to reduce the number of states. It shows that mn − 2 states are sufficient for the union of the two prefix-free regular languages.

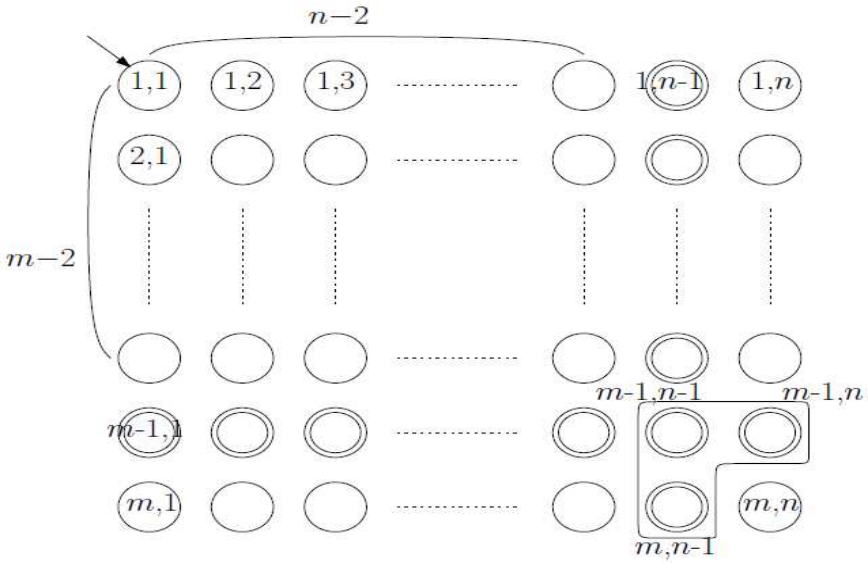


Figure 2.9: Union of two prefix-free minimal DFAs

iii) Concatenation of Prefix-Free Regular Languages

Let A and B be minimal DFAs for two regular languages L_1 and L_2 and |A| = m and |B| = n. Yu et al [12] obtained the size of the minimal DFA for L_1L_2 is $m2^n - 2^{n-1}$ in the worst-case. A regular language is Prefix free if and only if its minimal DFA is non-exiting [27].

Theorem 2.4: Given two prefix-free minimal DFAs A and B, m+n−2 states are necessary and sufficient in the worst-case for the minimal DFA of L(A).L(B), where m = |A| and n = |B|. [27]

Proof-: A and B are prefix free minimal DFAs. They are non-exiting and have only one final state. DFA for L(A)L(B) can be constructed by merging the final state of A and the start state of B to give a single state which also eliminates all out-transitions from the final state of A . Two sink states are also merged to give single sink state. The resulting automaton is deterministic and m + n − 2 states are sufficient in worst case for concatenation of two prefix free minimal DFAs.

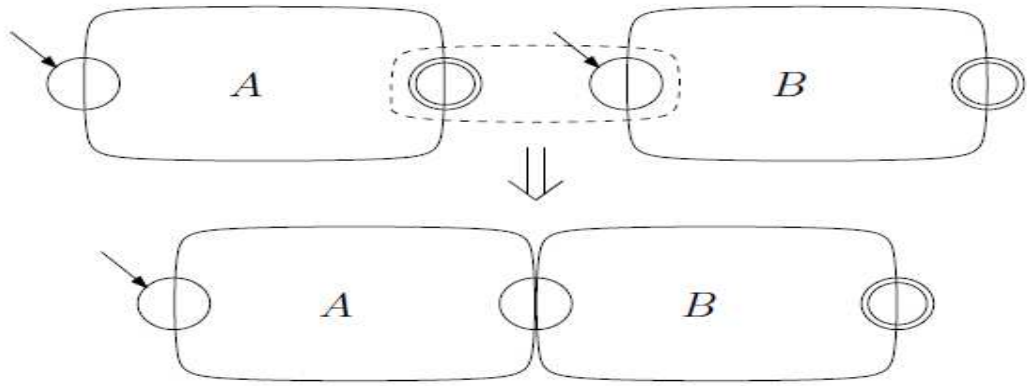


Figure 2.10: Concatenation of two prefix-free minimal DFAs.

iv) Kleene Closure of Prefix Free Regular Languages

Given a prefix-free minimal DFA A (Q, ∑, δ, s, F), a new NFA $A^{'}$ (Q, ∑, $\delta^{'}$, s, F) is constructed where transition function $\delta^{'}$ is given by $\delta^{'}$= δ U{(f, a, q) |f ∈ F and (s, a, q) ∈ δ for a ∈ ∑ }That is new out-transitions with the same label from all final states are added for each out-transition from the start state as in figure 2.11(b) such that L (A^{1})=L(A^{+})

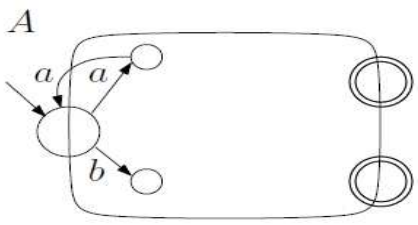


Figure 2.11(a): Given DFA A

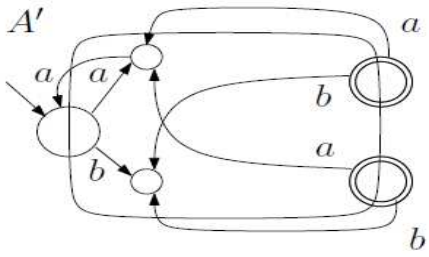


Figure 2.11(b): DFA for language $L(A^{+})$

Since A is a prefix-free minimal DFA, then A has only one final state and, therefore A′ also has one final state. For the minimal DFA for $L(A)^{*}$, ε-transition is also

considered. Add a new start state s′ and make a null-transition from s′ to s in A′ and make s′ as a final state.

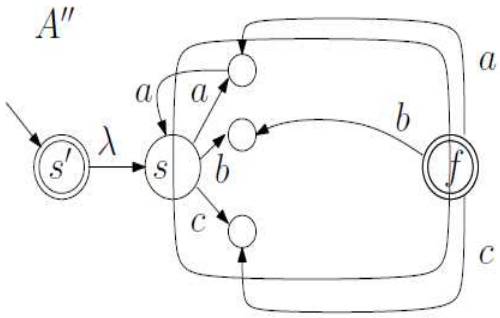


Figure 2.12(a): DFA for L(A)*

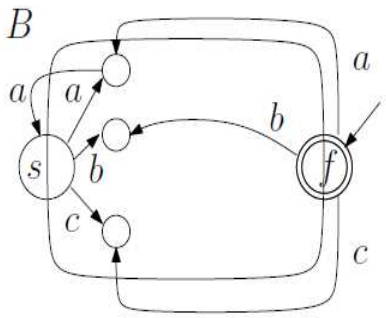


Figure 2.12(b): Minimal DFA for language L(A)*

Since s′ and f are equivalent states, these states can be merged and merging two equivalent states does not affect the language, L(A″) = L(B).

Theorem 2.5: Given a prefix-free minimal DFA A, n states are necessary and sufficient in the worst-case for the minimal DFA of L (A)*, where n = |A|. [27]

2.5 State Complexity of Suffix free Regular Languages

The state complexity of suffix free languages for various kinds of operations can be deterministic or non-deterministic.

2.5.1 Nondeterministic State Complexity of Suffix Free Regular Languages

Han, Salomaa, and Wood [29] works on the non-deterministic state complexity of suffix free regular languages for various operations like union, intersection, concatenation, reversal, kleene closure and complementation.

i) Intersection of Suffix-Free Regular languages

Given two FAs A and B, a FA for the intersection of L(A) and L(B) can be constructed based on the cartesian product of states. Given two FAs A (Q_1, $\sum$, δ_1, s_1, F_1), and B (Q_2, $\sum$, δ_2, s_2, F_2). Let M = ($Q_1 \times Q_2$, $\sum$, δ, (s_1, s_2), $F_1 \times F_2$) denotes the intersection of A and B. For all p $\in Q_1$ and q $\in Q_2$ and a $\in \sum$, and transition function δ is given by δ ((p, q), a) = (δ_1(p, a), δ_2(q, a)). Cartesian product shows that mn states are sufficient for the intersection of L(A) and L(B), where m and n are the numbers of states for A and B. Since A and B are suffix-free, this implies that both A and B are non-returning and, thus s_1 and s_2 do not have any in-transitions. All states (s_1, q) and (p, s_2), for p$\neq s_1 \in Q_1$ and q$\neq s_2 \in Q_2$, are unreachable from (s_1, s_2) in M. All

unreachable states are removed, so $mn-(m-1)-(n-1) = mn-(m+n)+2$ are sufficient for $L(A) \cap L(B)$.

ii) Union of Suffix-Free Regular languages

Han and Salomaa [29] showed that $mn-(m+n)+2$ is the state complexity of the union of an m-state suffix-free DFA and an n-state suffix-free DFA using the cartesian product of states. For the NFA state complexity, an NFA for the union of two suffix-free regular languages can be constructed without cartesian product. The construction relies on non-deterministic property and the fact that the suffix-free FA cannot have in transition to the start state.

Theorem 2.6: Given two suffix-free regular languages L_1 and L_2, the nondeterministic state complexity for $L_1 \cup L_2$ is $m+n-1$, where m and n are nondeterministic state complexity of L_1 and L_2 respectively.[29]

iii) Concatenation of Suffix-Free Regular languages

Theorem 2.7: Given two suffix-free regular languages L_1 and L_2, the nondeterministic state complexity for L_1L_2 is $m+n-1$, where m and n are nondeterministic state complexity of L_1 and L_2 respectively.[29]

iv) Kleene Star of Suffix-Free Regular languages

Han and Salomaa [26] investigated that the deterministic state complexity for kleene-closure of m-state minimal suffix free DFA is $2^{m-2}+1$. Han [29] suggest that if the language is suffix-free regular language L, then non-deterministic state complexity for L^* is m, where m is non-deterministic state complexity of regular language L.

v) Reversal of Suffix-Free Regular languages

Given a suffix-free regular language L, the non-deterministic state complexity for L^R is m+1, where m is non-deterministic state complexity of L. A minimal NFA $A = (Q, \sum, \delta, s, f)$ that accepts L, where $|Q| = m$. NFA A^R for L^R can be obtained by flipping the directions of all transitions and make the start state to be a final state and all final states to be start states of A. New obtained NFA A^R has multiple start states. Then a new start state is introduced and make a ε- transition from the new start state to the start states in L^R. Applying ε -transition removal technique [29], does not change the number of states. Thus, an m+1 state NFA for L^R is obtained. Han [29] proposed that if L is a suffix-free regular language recognized by an NFA with m states, then non-deterministic state complexity of reversal of given suffix free regular language L is m+1.

vi) Complementation of Suffix-Free Regular languages

A n-state NFA can be transformed into DFA with at most 2^n states [29]. The complementation of an n-state DFA does not change the state complexity since it simply interchanges final states and non-final states. Thus, based on the subset construction, 2^n states are sufficient for the complementation of an n-state NFA. Given an n-state suffix-free NFA A ($Q, \sum, \delta, f, s$). A can be converted into DFA A^1 using subset construction and it can have at most 2^n states (some of them may be useless states), and the final states are the subsets of Q that contain f. Since L(A) is suffix-free, all 2^{n-1} starting states can be merged into a single start state in A^1. Therefore, the number of states in DFA for L(A) is $2^{n-1}+1$. Now interchange final states and non-final states, thus the resulting FA is an NFA for L(A). $2^{n-1}+1$ states are sufficient for complement language L (A) of n-state suffix free NFA.

2.5.2 Deterministic State Complexity of Suffix free Regular Languages

Han, Salomaa, and Wood [26] works on the deterministic state complexity of suffix free regular languages for various operations.

i) Intersection of Suffix-Free Regular languages

Given two DFAs A and B, a DFA for the intersection of L(A) and L(B) can be constructed based on the cartesian product of states. Given two DFAs A =($Q_1, \sum, \delta_1, s_1, F_1$), and B = ($Q_2, \sum, \delta_2, s_2, F_2$). Let M = ($Q_1 \times Q_2, \sum, \delta, (s_1, s_2), F_1 \times F_2$) represents the intersection of DFA's A and B, where for all $p \in Q_1$ and $q \in Q_2$ and $a \in \sum$, and transition function δ is given by $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$.

Construction shows that mn states are sufficient for the intersection of L(A) and L(B), |A|= m and |B| = n. It has been observed that [26], in the case when L(A) and L(B) are suffix-free, M has some useless states and, thus, M is not minimal because L(A) and L(B) are suffix-free. Approach used here is to reduce the number of states by identifying and removing these useless states from M based on structural properties of suffix-free NFAs. Assign a unique number for each state from 1 to m in A and from 1 to n in B, the m^{th} state and the n^{th} state are the sink states in A and B, respectively. Let A $\cap$c B denote the resulting intersection automaton that is computed using the cartesian product of states then minimize A $\cap$c B by merging all equivalent states and removing unreachable states from the start state. All states (i, n) for $1 \leq i \leq m$ and all states (m, j) for $1 \leq j \leq n$ of A $\cap$c B are equivalent. All states (1, j) for $1< j \leq n$ of A $\cap$c B are not reachable from (1, 1) and therefore, these states are useless. Similarly all states (i, 1) for $1<i \leq m$ are useless.

Theorem 2.8: Given two suffix-free minimal DFAs A and B with m and n deterministic state complexity. Han [26] proposed that mn−2(m+n)+6 states are necessary and sufficient in the worst-case for the minimal DFA of L(A) ∩L(B).

ii) Union of Suffix-Free Regular languages

Given two DFAs A and B, DFA for the union of L(A) and L(B) can be constructed based on the cartesian product of states. Given two DFAs A =(Q_1, ∑, δ_1, s_1, F_1), and B = (Q_2, ∑, δ_2, s_2, F_2). Let M = ($Q_1 \times Q_2$, ∑, δ, (s_1, s_2), $F_1 \times F_2$) represents the union of A and B, where for all p ∈Q_1 and q ∈Q_2 and a ∈∑, and transition function δ is given by δ ((p, q), a) = (δ_1(p, a), δ_2(q, a)) and final states F = {(p, f_2) | p ∈Q_1}U {(f_1, q) | q ∈Q_2}. For a state (i, j) in A U_c B the language $L_{(i,j)}$ of (i, j) is the union of L_i in A and L_j in B. Since $L_{(m,j)}=L_m U L_n \neq \Phi$. It means all states (i,n) and (m,j) for $1 \leq i \leq m$ and $1 \leq j \leq n$ in AU_c B are not necessarily equivalent. Thus, these states cannot be merged. It is observed that all states (i, 1) and (1, j) for $1<i\leq m$ and $1<j\leq n$ are useless since L(A) and L(B) are suffix-free. Therefore, AU_c B is minimized by removing these m + n - 2 states.

Theorem 2.9: Given two suffix-free minimal DFAs A and B, mn−(m+n)+2 states are necessary and sufficient in the worst-case for the minimal DFA of L(A)UL(B), where m and n are the Deterministic state complexity of A and B respectively. [26]

iii) Reversal of Suffix-Free Regular languages

If a regular language L is accepted by an n-state minimal DFA, then its reversal L^R is accepted by an n-state NFA [26]. So the state complexity of L^R is at most 2^n since DFA can be obtained using subset construction of any n-state NFA. Reversal can be obtained by just flipping the direction of all transitions and interchange start and final state in any NFA. Then perform subset construction to find DFA which have at most 2^n states. Given DFA A (Q, ∑, δ, s, F) then its reversal can be represented by A^R (Q, ∑, δ, F, s). Since A is suffix-free DFA so it is non-returning, A^R will be non-exiting. But A^R has some useless and non-reachable states.

Theorem 2.10: Given m-state suffix-free minimal DFA A, then 2^{n-2} + 1 states are necessary and sufficient in the worst-case for the minimal DFA of $L(A)^R$. [26]

Chapter 3

Problem Statement and State complexity of Infix Free Language

3.1 Problem statement

Han proposed an approach for checking infix freeness of a non-deterministic finite automata based on state pair graph concept [19]. Han also checked the infix freeness of a regular expression by converting regular expression into NFA through thompson construction and then construct state pair graph and check the infix freeness. But this approach leads to large number of states and transitions. So checking infix freeness of a regular expression with less state complexity is required. Han 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. It is natural to investigate the deterministic and non-deterministic state complexity of various basic operations on infix free regular languages.

3.2 Objectives

1. Comparison among the state complexity for various basic operations on various forms of regular language.
2. To determine deterministic and non-deterministic state complexities of various operations on infix free regular languages.
3. To propose an approach for checking infix-freeness of a regular expression using state pair graph that has minimal number of states.
4. To design a software for checking infix-freeness of a regular expression.

3.3 Comparison among State Complexities of Various Forms of Regular Language

The state complexity of a regular language L is the number of states of the minimal FA that accepts L. Yu [12] proposed the state complexity of basic operations on regular languages. If a regular language L_1 is accepted by m-state DFA and a regular language L_2 is accepted by n-state DFA. Then the deterministic state complexity of basic operations on regular languages is represented in the table 3.1. S.Yu [11] also proposed the state complexity of basic operations on finite languages. If a language L_1 is accepted by an m-state DFA and a language L_2 is accepted by an n-state DFA and t

denote the number of final states in A_1. Then the deterministic state complexity of basic operations on finite languages is represented in the table 3.2.

Operation	$\|\sum\|=1$	$\|\sum\|>1$
$L_1 \cup L_2$	mn, for (m, n)=1	mn
$L_1 \cap L_2$	mn, for (m, n)=1	mn
$L_1 L_2$	mn, for (m, n)=1	$m2^n-2^{n-1}$
L_1^*	$(m-1)^2+1$	$2^{m-1}+2^{m-2}$
$\sum^* - L_1$	m	m
L_1^R	m	2^m

Table 3.1: Comparison among deterministic state complexity of regular languages [12]

Operation	$\|\sum\|=1$	$\|\sum\|>1$
$L_1 \cup L_2$	max (m, n) For (m, n)=1	O(mn)
$L_1 \cap L_2$	min (m, n) For (m, n)=1	O(mn)
$L_1 L_2$	m+n-1 for (m, n)=1	$O(mn^{t-1}+n^t)$
L_1^*	$m^2-7m+13$ for m>4	$2^{m-3}+2^{m-4}$ for $m \geq 4$
$\sum^* - L_1$	m	m
L_1^R	m	$2^{m/2}$ for $\|\sum\|=2$

Table 3.2: Comparison among deterministic state complexity of finite languages [11]

Y.Han proposed the non-deterministic and deterministic state complexity of various basic operations on prefix free regular languages and suffix free regular languages. The comparison of state complexity of prefix free [27,28] is represented in table 3.3 and comparison of state complexity of suffix free [26,29] is represented in table 3.4.

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

Table 3.3: Comparison of state complexity of prefix free regular languages [27,28]

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

Table 3.4: Comparison among state complexity of suffix free regular languages [26,29]

3.4 Non-Deterministic State Complexity of Operations on Infix-Free Regular Languages

3.4.1. Intersection of Infix-Free Regular Languages

Given two finite automaton A_1 and A_2, then a FA for the intersection of $L(A_1)$ and $L(A_2)$ can be constructed based on the cartesian product of states of both finite automata. Given finite automata A_1 $(Q_1, \sum, \delta_1, s_1, F_1)$ and A_2 $(Q_2, \sum, \delta_2, s_2, F_2)$. Let A $(Q_1 X\ Q_2, \sum, \delta, (s_1, s_2), F_1\ XF_2)$ denotes the intersection of A_1 and A_2 where for all $p \in Q_1$ and $q \in Q_2$ .and transition function δ is given by $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ for all $p \in Q_1$ and $q \in Q_2$. The construction shows that mn states are sufficient for the intersection of $L(A_1)$ and $L(A_2)$, where m and n are the non-deterministic state complexity of A_1 and A_2. But it is not minimal, since it has been observed that infix free regular languages are non returning and non exiting. So their intersection L(A) has some useless states, which can be removed to reduce the size of resulting automaton.

A unique number is assigned to each state from 1 to m in A_1 and from 1 to n in A_2, where $|A_1| = m$ and $|A_2| = n$. Assume that the m^{th} state and the n^{th} state are the final states in A_1 and A_2, respectively. Let $A_1 \cap A_2$ denote the resulting intersection automaton. Then, (1, 1) is the start state and (m, n) is the unique final state. Since $L(A_1)$ and $L(A_2)$ are infix free then all states (1, i) and (j, 1) for $2 \leq i \leq n$ and $2 \leq j \leq m$ are not reachable from (1,1), so these states are equivalent and can be merged with starting state(1,1). and all states (m, i) and (j, n) for $1 \leq i < n$ and $1 \leq j < m$ are useless states because they are non-accepting states and final state is unreachable from these states, so these states can be merged. After removing all useless states and merging all equivalent states, the resulting automaton has $mn-(m-1)-(n-1)-(m-2)-(n-2) = mn-2(m+n)+6$ states.

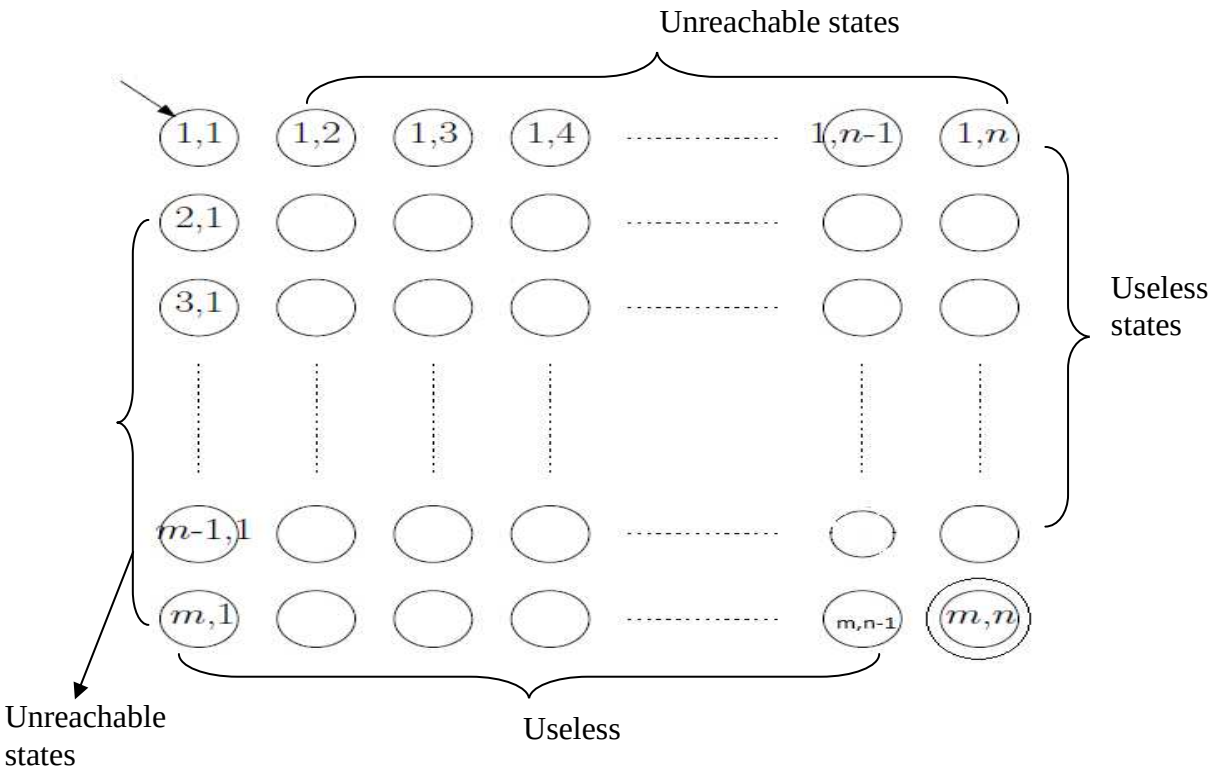


Figure 3.1: Intersection of two infix-free minimal NFAs

Thus mn-2(m+n)+6 states are sufficient in worst case for intersection of two infix free minimal NFA's.

3.4.2 Union of Infix-Free Regular Languages

Given two infix-free regular languages L_1 and L_2. Let A_1 (Q_1, $\sum$, δ_1, s_1, F_1) be a minimal NFA for L_1 and A_2 (Q_2, $\sum$, δ_2, s_2, F_2) be a minimal NFA for L_2. Then an NFA for L ($A_1 U A_2$) can be constructed by A (Q, $\sum$, δ, s, (F_1, F_2)) where $Q=Q_1 \cup Q_2 \cup \{s\}$ and for a state $q \in Q$, δ is defined as

$\delta(q,a) = \delta_1(s_1,a) \cup \delta_2(s_2,a)$ when $q=s$

$\delta(q,a) = \delta_1(q,a)$ when $q \in Q_1$

$\delta(q,a) = \delta_2(q,a)$ when $q \in Q_2$

Since A_1 and A_2 are infix free, so it can be observed that the two final states f_1 and f_2 are equivalent and, thus f_1 and f_2 can be merged, because final states do not have any out transitions. Starting state can't have any in transitions so they can be removed by replacing the transitions from s_1 and s_2 by same transitions from new start state s. After removing all useless states the resulting automaton has (m+n)+1-1-2= (m+n-2) states. Thus (m+n-2) states are sufficient in worst case for union of two infix free NFA's.

3.4.3 Concatenation of Infix-Free Regular Languages

Given two infix-free regular languages L_1 and L_2, Let $A_1 = (Q_1, \sum, \delta_1, s_1, f_1)$, be a minimal NFA for L_1 and $A_2 = (Q_2, \sum, \delta_2, s_2, f_2)$, be a minimal NFA for L_2. Since A_1

and A_2 are infix free. It can be assumed that A_1 and A_2 have only one final state, that is non-exiting. NFA A for $L(A_1).L(A_2)$ can be constructed by merging the final state f_1 from A_1 and the start state s_2 of A_2 to give a single state that also eliminates all out-transitions of f_1 of A_1. NFA $A=(Q, \sum, \delta, s_1, f_2)$, where $Q=Q_1 \cup Q_2 \setminus \{f_1\}$ and for each state $p \in Q$ and $a \in \sum$ and transition function δ is given by

$$\delta(p, a) = \begin{cases} s_2 & \text{if } p \in Q_1 \text{ and } \delta_1(p,a)=f_1 \\ \delta_1(p,a) & \text{if } p \in Q_1 \text{ and } \delta_1(p,a) \neq f_1 \\ \delta_2(p,a) & \text{otherwise} \end{cases}$$

It is easy to verify that $L(A) = L(A_1)L(A_2)$ from the construction. Since $Q = Q_1 \cup Q_2 \setminus \{f_1\}$ and therefore, the construction shows that $m + n - 1$ states are sufficient in the worst case for the concatenation of $L(A_1)$ and $L(A_2)$, where $|A_1| = m$ and $|A_2| = n$.

3.4.4 Complement of Infix-Free Regular Languages

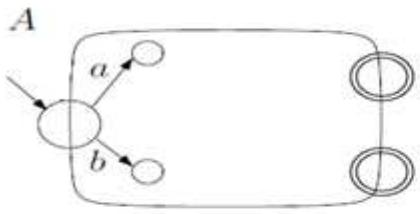
A n-state NFA can be transformed into a DFA with at most 2^n states. The complementation of an n-state DFA does not change the state complexity since it simply interchanges final states and non-final states. Thus based on the subset construction, we know that 2^n states are sufficient for the complementation of an n-state NFA. Given an n-state infix-free NFA $A = (Q, \sum, \delta, f, s)$. A can be converted into DFA A^1 using subset construction and it can have at most 2^n states (some of them may be useless states), Final states in A^1 are the subset of all states that includes f. Since in infix free language final states are non-exiting, so all final states are equivalent and can be merged. All 2^{n-1} final states can be merged into one final state. Also in infix free language all starting states do not have any in transitions, so these 2^{n-1} starting state can be merged into one starting state. But 2^{n-2} states are taken twice in subsets of start and final states. Therefore after reducing the useless states $2^n-(2^{n-1}-1)- (2^{n-1}-1)+2^{n-2} =2^{n-2} +2$ states are sufficient in worst case for complementation of an infix-free minimal NFA.

3.4.5 Reversal of Infix-Free Regular Languages

Given an infix-free regular language L. Suppose there is a minimal NFA A $(Q, \sum, \delta, s, f)$ that accepts language L where $|Q|=n$. A is non-exiting and non-returning. So A has single final state. NFA A^R $(Q, \sum, \delta^R, f, s)$ for L^R can be obtained by flipping the direction of transitions and interchange final and start state in A. A and A^R both are having the same number of states. Hence n states are sufficient for L^R. So worst case non-deterministic state complexity for reversal of infix free regular language is n.

3.4.6 Kleene Closure of Infix-Free Regular Languages

Given an n-state infix-free NFA A (Q, ∑, δ, s, F), a new NFA $A^{'}$ (Q, ∑, $\delta^{'}$, s, F) is constructed where transition function $\delta^{'}$ is given by $\delta^{'}= \delta \cup\{(f, a, q) \mid f \in F$ and $(s, a, q) \in \delta$ for $a\in \sum\}$. New out-transitions with the same label from all final states are added for each out-transition from the start state as in figure 3.2(b) such that $L(A^1) = L(A^+)$. Since A is an infix-free NFA, then A has only one final state and, therefore A′ also has one final state. In NFA for $L(A)^*$, ε-transitions are also considered. So add a new start state s′ and make a null-transition from s′ to s in A′ and make s′ as a final state.



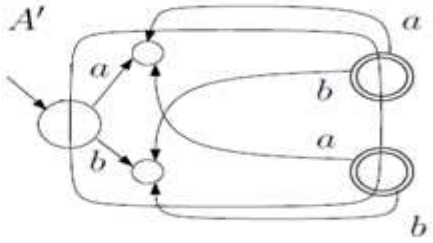
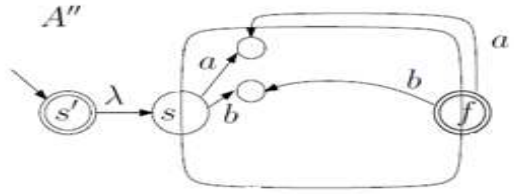


Figure 3.2(a): Given NFA A **Figure 3.2(b): NFA for language $L(A^+)$**

Since s′ and f are equivalent states, so these states can be merged as shown in figure 3.3(b) and merging two equivalent states does not affect the language, L(A″) = L(B).



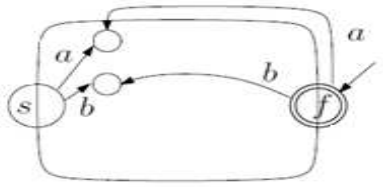


Figure 3.3(a): NFA for L(A)* **Figure 3.3(b): NFA after merging s′ and f**

Two equivalent states s′ and f in A″ are merged and state s is removed and obtain NFA B for language L(A)* as shown in figure 3.3(c), where L(B) = L(A)*. Thus n-1 states are necessary and sufficient in the worst-case for the minimal NFA for kleene closure of infix free NFA.

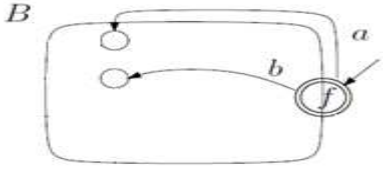


Figure 3.3(c): NFA for L(A)*

3.4.7 Comparison among Non-Deterministic State Complexity of Infix Free Regular Languages

Operation	Infix free NFAs
$L_1 \cap L_2$	mn-2(m+n)+6
L_1 U L_2	m+n-2
$L_1.L_2$	m+n-1
$\sum^* - L_1$	$2^{n-2}+2$
L_1^R	n
L_1^*	n-1

Table 3.5: Comparison among non-deterministic state complexity of infix free regular languages

3.5 Deterministic State complexity of Operations on Infix-free Regular Languages

3.5.1 Intersection of Infix-Free Regular Languages

Given two deterministic finite automata $A_1(Q_1, \sum, \delta_1, s_1, F_1)$ and $A_2(Q_2, \sum, \delta_2, s_2, F_2)$, then a deterministic finite automata A for the intersection of $L(A_1)$ and $L(A_2)$ can be constructed based on the cartesian product of states of both deterministic finite automata. Let A is given by $(Q_1 X\ Q_2, \sum, \delta, (s_1, s_2), F_1\ X\ F_2)$ and transition function δ is $\delta((p, q),a)= (\delta_1(p, a), \delta_2(q, a))$ for all $p \in Q_1$ and $q \in Q_2$. Then $L(A) = L(A_1) \cap L(A_2)$

It shows that mn states are sufficient for the intersection of $L(A_1)$ and $L(A_2)$, where $|A_1| = m$ and $|A_2|= n$. If L is a regular language, its minimal DFA does not necessarily have a sink state. However, if L is infix free, then its minimal DFA A must have a sink state since A is non-exiting. A unique number is assigned to each state from 1 to m in A_1 and from 1 to n in A_2, where $|A_1| = m$ and $|A_2| = n$. Assume that DFA A_1 and A_2 are complete. So each state has $|\sum|$ out- transitions and A may have sink state. Assume that $(m-1)^{th}$ state and $(n-1)^{th}$ state are the final states in A_1 and A_2, respectively. All out transitions from $(m-1)^{th}$ and $(n-1)^{th}$ states i.e. final states can only target to sink states, since A_1 and A_2 are non-exiting. Let m^{th} state and n^{th} state are the sink states in A_1 and A_2. Let $A_1 \cap A_2$ be the cartesian product of states for $L(A_1) \cap L(A_2)$.

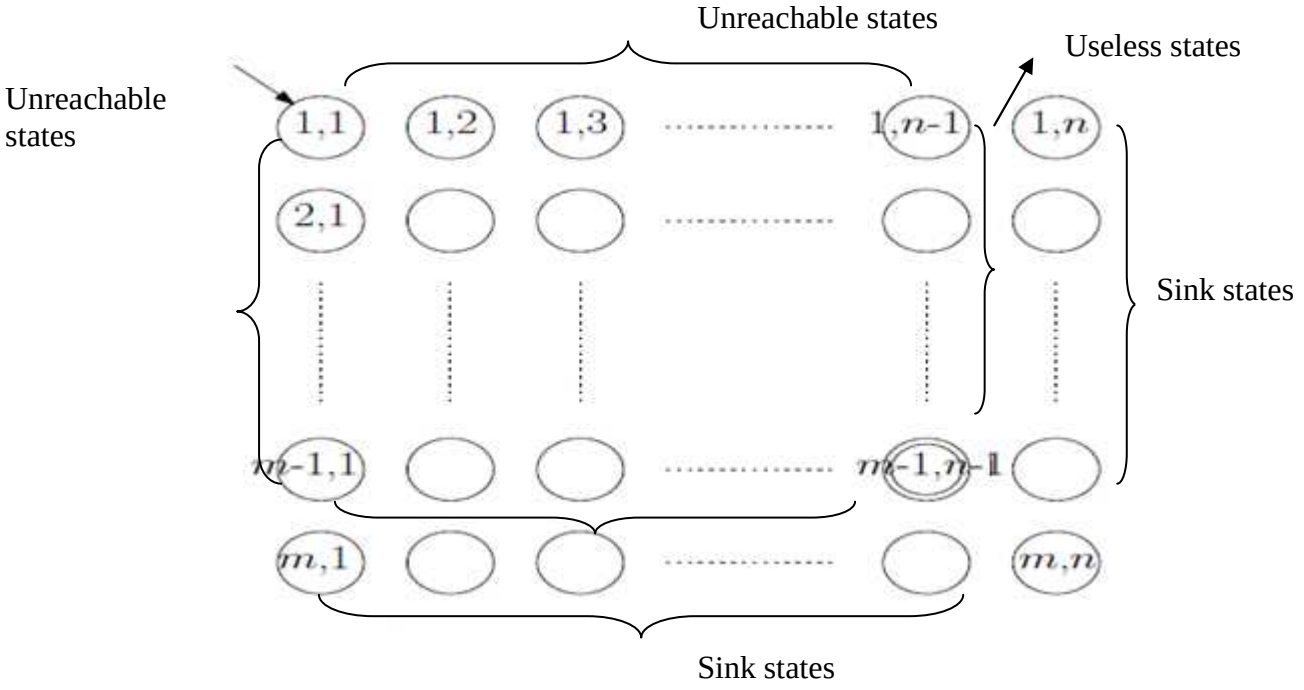


Figure 3.4: Intersection of two infix-free minimal DFAs

(1, 1) is the start state and (m-1, n-1) is the unique final state. Since L(A_1) and L(A_2) are infix free then all states (1,i) and (j,1) for 2≤i≤n and 2≤j≤m are not reachable from (1,1) so these states are equivalent and can be merged with starting state(1,1). All states (m, i) and (j, n) for 1≤i≤n and 1≤j≤m are sink states and in case of infix free regular languages all sink states are equivalent and can be merged. All states (m-1, i) and (j, n-1) are useless states because they are non-accepting states and final state is not reachable from these states. After removing all useless states, The resulting automaton has mn-((m-1)+(n-1))-((m-2)+(n-2))-((m-3)+(n-3))= mn-3(m+n)+12 states. Thus mn-3(m+n)+12 states are sufficient in worst case for intersection of two infix free minimal DFA.

3.5.2 Union of Infix-Free Regular Languages

Given two DFAs A_1 and A_2, then a DFA for the union of L(A_1) and L(A_2) can be constructed based on the cartesian product of states. Given two infix-free minimal DFAs A_1 (Q_1, ∑, δ_1, s_1, F_1), and A_2 (Q_2, ∑, δ_2, s_2, F_2). Let A ($Q_1 \times Q_2$, ∑, δ, (s_1, s_2), $F_1 \times F_2$) denotes the union of A_1 and A_2, where for all p ∈Q_1 and q ∈Q_2 and a ∈∑, and δ ((p, q), a) = (δ_1(p, a), δ_2(q, a)) and final state is given by F = {(p, f_2) | p ∈Q_1}U {(f_1, q) | q ∈Q_2}. Then, L(A) = L(A_1)UL(A_2)

All states in the last row and all states in the last column of A_1Uc A_2 are not necessarily sink states except for state (m, n).So these states are not equivalent in A_1Uc A_2. But L (m,n−1) = L_m U L_{n-1} = {ε} and L(m−1,n) = L_{m-1} U L_n = {ε}

Since A_1 and A_2 are non-exiting. Therefore, L(m, n−1) = L(m−1,n) = L(m−1,n−1). These three states (m, n−1), (m−1, n) and (m−1, n−1) can be merged to reduce the number of states. Since $L(A_1)$ and $L(A_2)$ are infix free so they are non-returning. All states (i, 1) and (1, j) for $1<i\leq m$ and $1<j\leq n$ are unreachable from (1, 1), so they are useless and can be removed to reduce the number of state in the resulting DFA. Therefore, $A_1 U_c A_2$ can be minimized by removing these m + n - 2 states. Remaining states in the resulting DFA are mn-(m+n-2)-2=mn-(m+n) states.

Thus mn-(m+n) states are sufficient in worst case for union of two infix free minimal DFA's.

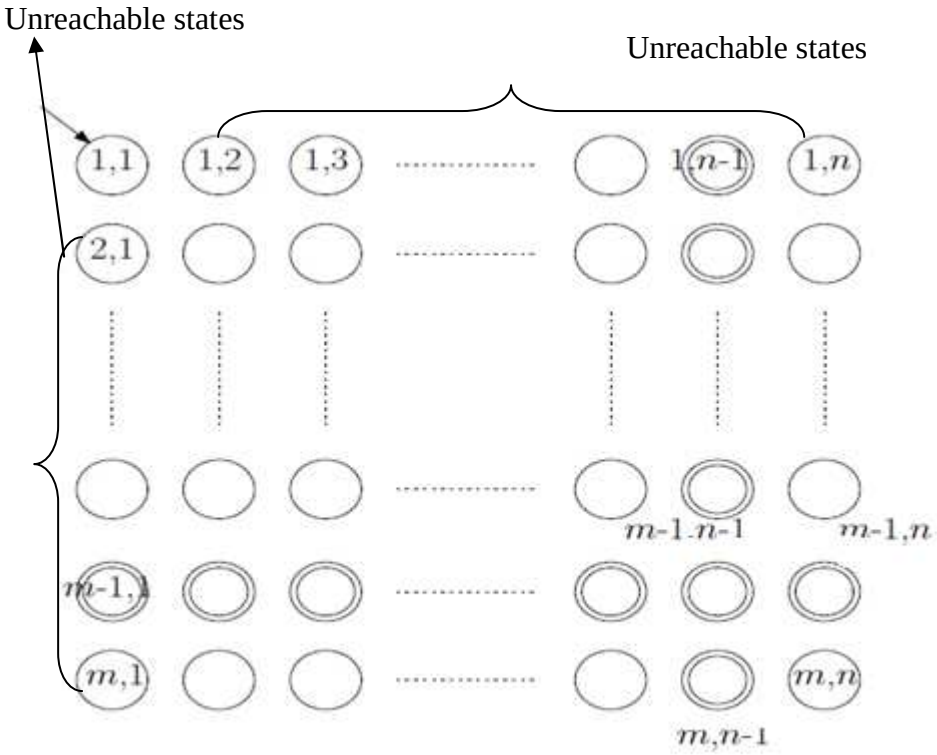


Figure 3.5: Union of two infix-free minimal DFAs

3.5.3 Concatenation of Infix-Free Regular Languages

Let A_1 and A_2 be minimal DFAs for two infix-free regular languages L_1 and L_2 where $|A_1|$ = m and |B| = n. If A regular language is infix free, then its minimal DFA is non-exiting and non-returning**.** Concatenation of two infix free DFA's can be obtained by merging the final state of first DFA with start state of second DFA and sink states in both DFA's can be merged. Given two infix-free minimal DFA's $A_1 (Q_1, \sum, \delta_1, s_1, F_1)$ and $A_2 (Q_2, \sum, \delta_2, s_2, F_2)$. Deterministic finite automata $A(Q, \sum, \delta, s_1, F_2)$ as $L(A_1. A_2)$ can be constructed by merging the final state of A_1 with start state of A_2 and merging

the sink states of A_1 and A_2. Hence m+n-2 states are sufficient in worst case for concatenation of two infix free DFA's.

3.5.4 Complement of Infix-Free Regular Languages

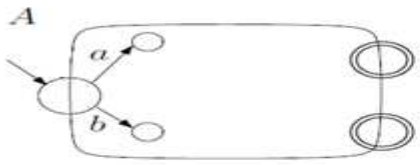
The complementation of an n-state DFA does not change the state complexity since it simply interchanges final states and non-final states. Hence n states are necessary and sufficient in worst case for determining complement of an infix-free DFA. Worst case deterministic state complexity of complementation of infix free regular language is n.

3.5.5 Reversal of Infix-Free Regular Languages

Given infix free minimal DFA A (Q, ∑, δ, s, F). If a regular language L is accepted by n-state minimal DFA, then its reversal L^R is accepted by n-state NFA. So reversal of A can be represented by A^R (Q, ∑, δ, F, s) and can be obtained by just flipping the direction of all transitions and interchange start and final state. A is an infix-free DFA, therefore it is non-exiting and non-returning. A^R will be non-returning and non-exiting. State complexity of L^R is at most 2^n since DFA can be obtained using subset construction from any n-state NFA. Since A is a minimal infix free DFA, so there will be a sink state in DFA. Let d is a sink state in A. then all state Q\{d} in A^R do not have any out transition to d in δ^R, so sink state d is useless in A^R and can be removed. Thus n-1 states are remaining in A^R. In subset construction 2^{n-1} subsets of states can be constructed. So 2^{n-2} final states and 2^{n-2} starting states can be merged. and 2^{n-3} states are common in both subsets of starting and final states. Hence total $2^{n-1}-(2^{n-2}-1)-(2^{n-2}-1)+2^{n-3}=2^{n-3}+2$ states are necessary and sufficient in worst case for reversal of n-state DFA.

3.5.6 Kleene Closure of Infix Free Regular Languages

Given an infix-free minimal n-state DFA A (Q, ∑, δ, s, F), a new DFA $A^{'}$ (Q, ∑, $\delta^{'}$, s, F) is constructed, where transition function $\delta^{'}$ is given by $\delta^{'}= \delta \cup \{(f, a, q) | f \in F$ and $(s, a, q) \in \delta$ for $a \in \sum \}$. New out-transitions with the same label from all final states are added for each out-transition from the start state as in figure 3.6 (b) such that $L(A^1)=L(A^+)$. Since A is a infix-free minimal DFA, then A has only one final state and, therefore, A′ also has one final state. For the minimal DFA for $L(A)^*$, ε-transition is also considered. So add a new start state s′ and make a null-transition from s′ to s in A′ and make s′ as a final state. s′ and f are equivalent states, so these states can be merged and merging two equivalent states does not affect the language, L(A″) = L(B).



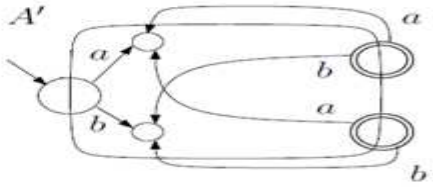
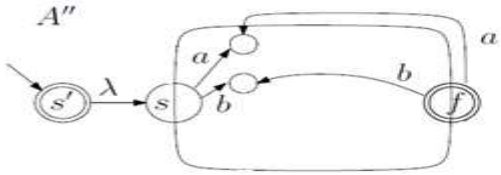


Figure 3.6(a): Given DFA A **Figure 3.6(b): DFA for language L(A^+)**

L(A) is infix-free, f has no out-transitions whose target state is not a sink state. Two equivalent states s′ and f in A″ are merged and s is removed and obtain minimal DFA B for language L(A)*, where L(B) = L(A)*. Thus n-1 states are necessary and sufficient in the worst-case for the minimal DFA for kleene closure of infix free DFA.



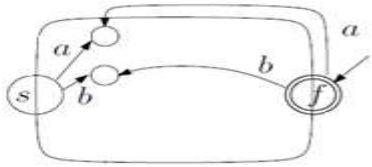


Figure 3.7(a): DFA for L(A)* **Figure 3.7 (b): DFA after merging s′ and f**

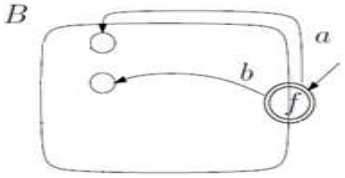


Figure 3.7(c): Minimal DFA for L(A)*

3.5.7 Comparison among Deterministic State Complexity of Infix Free Regular Languages

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

Table 3.6: Comparison among deterministic state complexity of infix free regular languages

Chapter 4

Implementation and Experimental results

4.1 Algorithm used for checking Infix Freeness of Regular Expression

We first convert a regular expression into DFA and then check whether DFA is infix-free or not.

4.1.1 Algorithm used for converting regular expression to DFA

We make use of the algorithm for direct conversion of RE to DFA []. After obtaining the DFA, we check for infix-freeness of the language.

Algorithm 1 (RE, DFA)

Input: This algorithm take a regular expression as input.

Output: DFA is generated from the regular expression.

Step 1: Syntax tree is created for augmented regular expression r#.

Step 2: Follow ( ) // call to procedure

Step 3: Consider firstpos (root) into the states of DFA as an unmarked state.

Step 4: while (there is an unmarked state S in the states of DFA) do

Step 5: mark S

Step 6: for each input symbol a do

Step 7: let $s_1,...,s_n$ are positions in S and symbols in those positions are a

Step 8: $S' \leftarrow followpos(s_1) \cup ... \cup followpos(s_n)$

Step 9: move(S,a)$\leftarrow$ S'

Step 10: if (S' is not empty and not in the states of DFA)

Step 11: Put S' into the states of DFA as an unmarked state.

end if

end for

end while

Step 12: Start state of DFA is first-pos(root) and accepting state of DFA are all states containing the position of #

Procedure follow ( )

Consider c_1 and c_2 are left and right child of a node while traversal. If a node is having single child consider it as c_1.

Step 1: Repeat step 2 to 5 for each node of the tree during post order Traversal

Step 2: if a node is a leaf node

First-pos of that node is node position;

Last-pos of that node is node position;

end if

Step 3: if a node is a concatenation node then

First-pos of concatenation node is first of c_1;

Last-pos of concatenation node is last position of c_2

Follow-pos of all lastpos of c_1 contains firstpos of c_2

end if

Step 4: if new node is union node then

First-pos of union node is union of firstpos of c_1 and c_2

Last-pos of union node is union of lastpos of c_1 and c_2

end if

Step 5: if new node is star or kleene closure then

First-pos of star node is firstpos of c_1

Last-pos of star node is lastpos of c_1

Add followpos of last node is firstpos of star

end if

end of procedure

4.1.2 Algorithm used for checking infix-freeness of DFA

Algorithm 2: Infix-freeness (DFA)

Step 1: if DFA A is not non-exiting or not non-returning

Then A is not infix free

Step 2: Construct State pair graph Construct $G_A = (V, E)$ from A

Step 3: Repeat step 4, 5 for each node (1, i) in V, where $2 \leq i \leq m$

Step 4: DFS ((1, i)) in G_A

Step 5: if we meet a node (m, j) for any j, $1 \leq j \leq m$

Then output L(A) is not infix-free

end if

Step 6: Otherwise L(A) is infix-free

4.2 Example for Checking Infix-Freeness of a Regular Expression

1. Converting Regular Expression into DFA Example

Given regular expression (a|b)*.a

The augmented regular expression is given by (a|b)*.a.#

Syntax tree with first-pos and last-pos for given regular expression

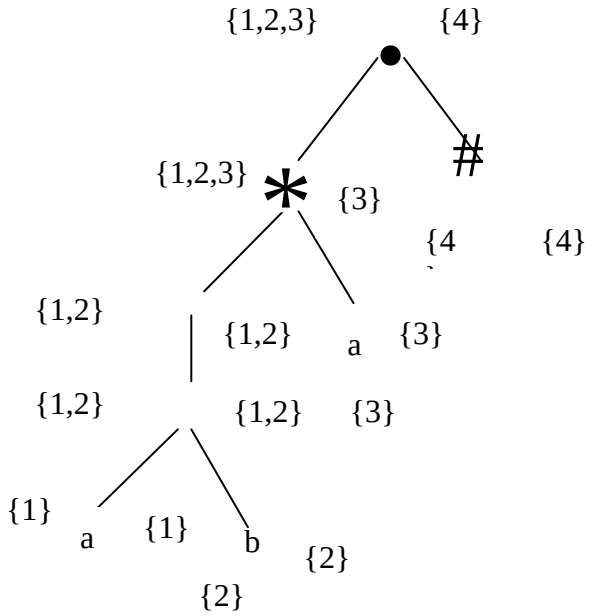


Figure 4.1: Syntax tree for regular expression (a|b)*.a.#

Follow-pos for given regular expression

Node n	followpos(n)
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{}

Table 4.1: Follow Position of Nodes

Starting state S_1=first-pos(root)={1,2,3}

⇓ mark S_1

a: followpos(1) ∪ followpos(3)={1,2,3,4}=S_2 move(S_1,a)=S_2

b: followpos(2)={1,2,3}=S_1 move(S_1,b)=S_1

⇓ mark S_2

a: followpos(1) ∪ followpos(3)={1,2,3,4}=S_2 move(S_2,a)=S_2

b: followpos(2)={1,2,3}=S_1 move(S_2,b)=S_1

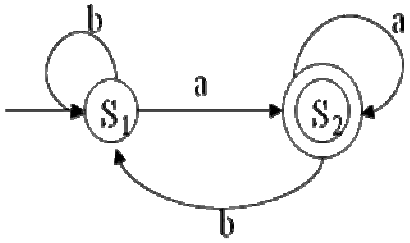


Figure 4.2: DFA for regular expression (a|b)*.a

2. Checking Infix-Freeness of DFA

Since there is in-transition on starting state, so DFA is not non-returning. So the given regular expression is not infix-free.

4.3 Experimental Results

A tool is designed for checking infix-freeness of a regular expression. We use Java platform for implementing the tool. This tool automates the procedure for checking infix-freeness of a regular expression. It takes regular expression as input and check regular expression for infix-freeness and gives the result.

Here some experimental results are presented in forms of snapshots.

Case 1: When DFA is non-returning and non-exiting

Input Regular Expression: (a.b)|(a|(b.b)).a*.b

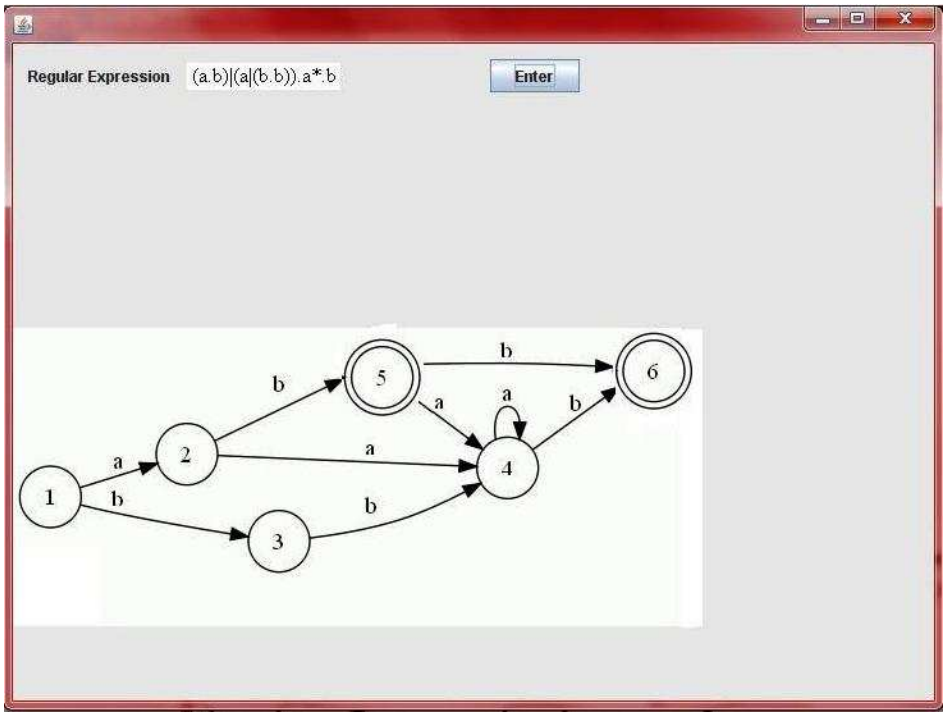


Figure 4.3: Non-returning and non-exiting DFA

State Pair Graph: In form of vertex and edges

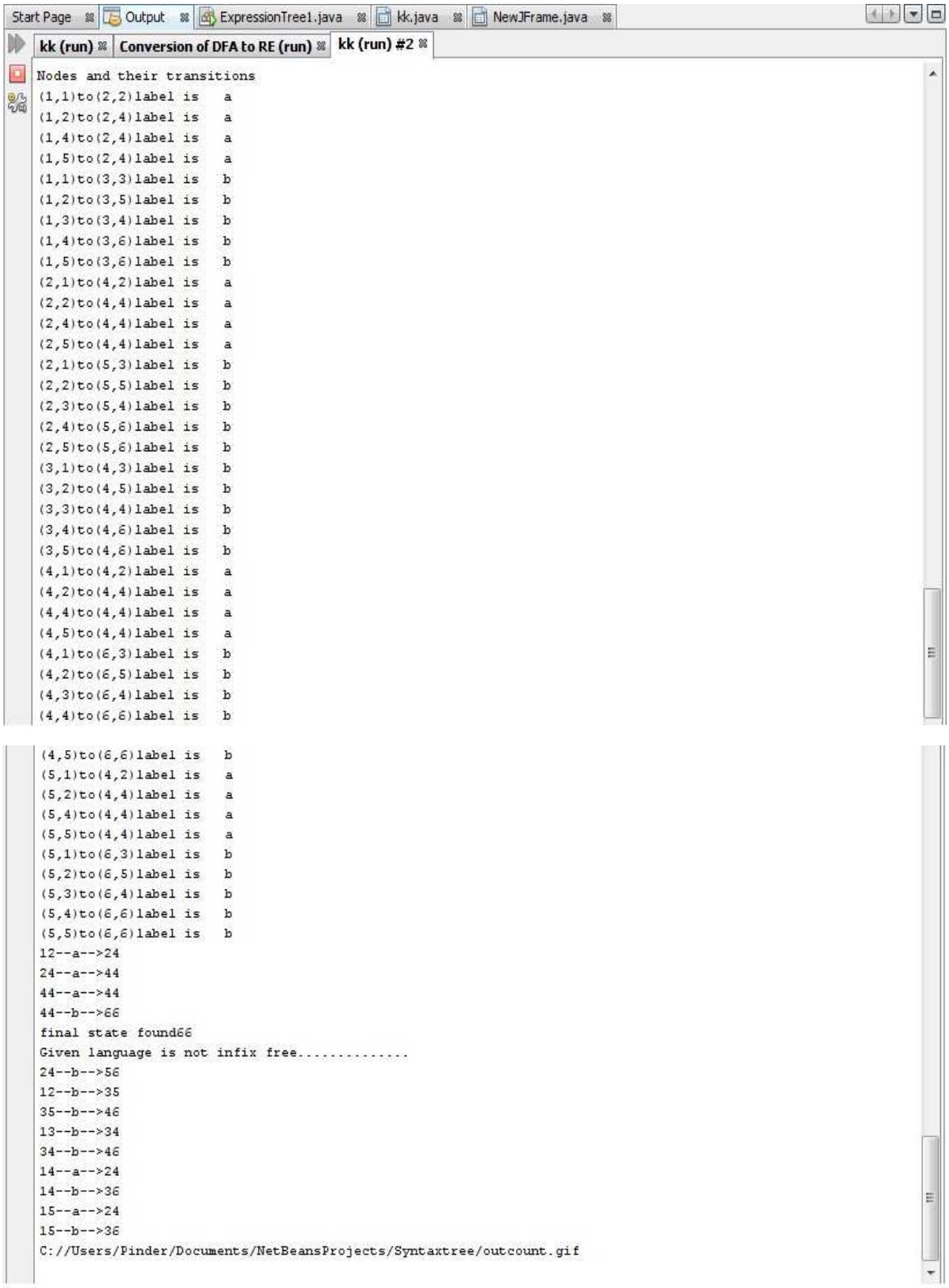


Figure 4.4: State pair graph for non-returning and non-exiting DFA

Output: There is a path from (1,2)→(2,4)→(4,4)→(6,6)

Given regular expression is not infix-free

Case 2: When DFA is not non-returning or not non-exiting

Input Regular Expression: ((a)*)*.a.(a|b)*

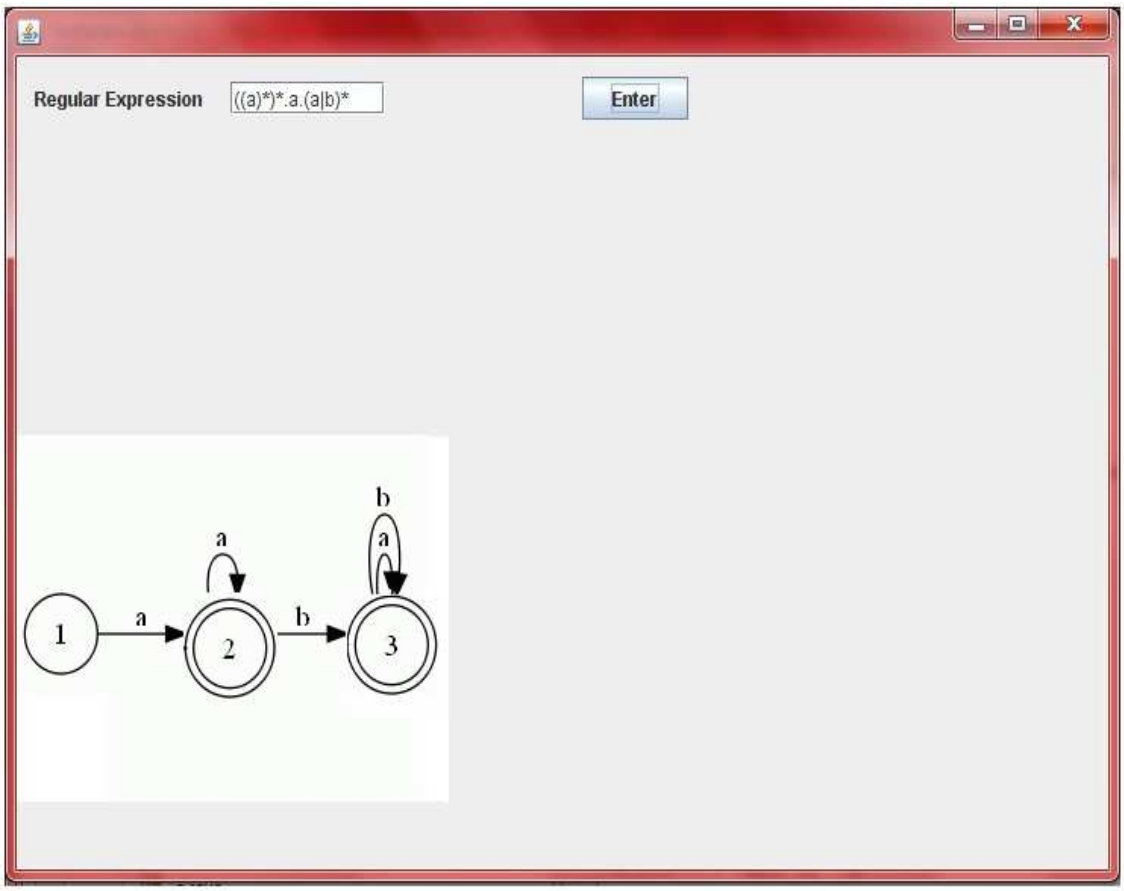


Figure 4.5: Not non-returning and not non-exiting DFA

State Pair Graph: In form of vertex and edges

Figure 4.6: State pair graph for not non-returning and not non-exiting DFA

Output: There is an out-transition from final state

Given regular expression is not infix-free

Case 3: Regular expression is infix-free

Input regular expression: a.b.b*.a

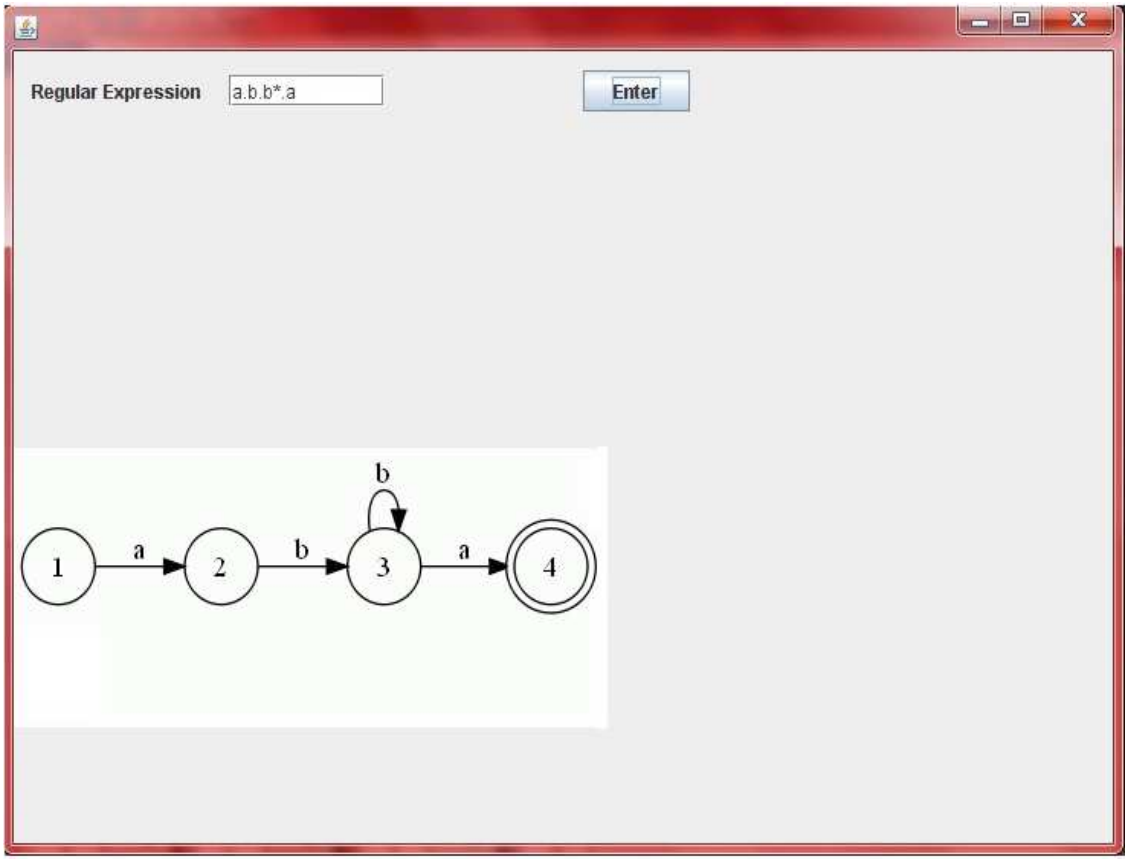


Figure 4.7: I**nfix-free DFA**

State Pair Graph: In form of vertex and edges

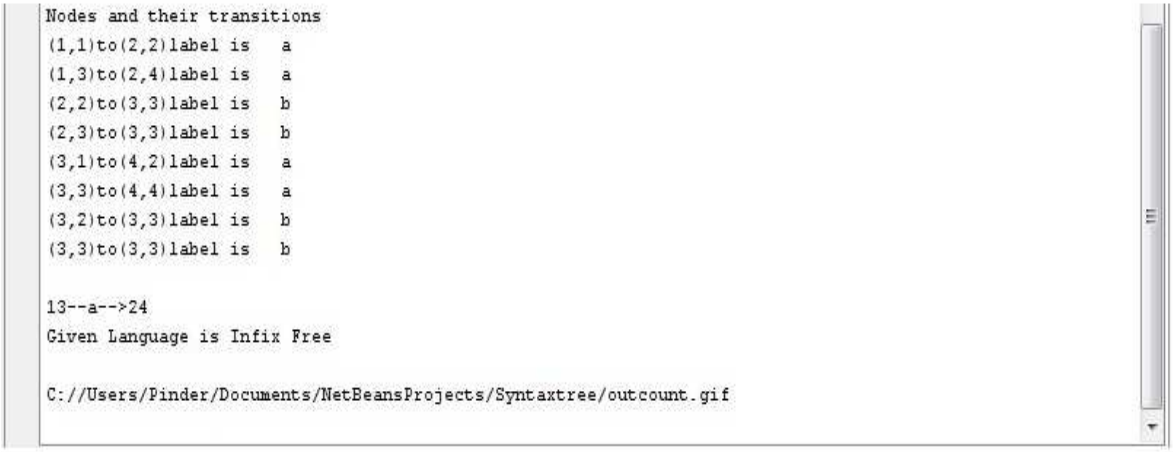


Figure 4.8: State pair graph for infix-free DFA

Output: Given regular expression is infix-free

Chapter 5

Conclusion and Future Scope

5.1 Conclusion

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. Further regular languages are categorized into infix free, prefix free, suffix free. These are used in various applications like pattern matching, computing forbidden words, text searching.

Comparative study of state complexity of various operations on suffix-free and prefix free are carried out. State complexities of various operations like union, intersection, concatenation, reversal, complement, and kleene closure on infix-free regular language are determined. A brief comparison between their state complexities is presented.

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

Table 5.1: Comparison among state complexity of infix free regular languages

A software is designed for checking infix-freeness of the language.

5.2 Future Scope:

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

1. To determine the precise state complexity of combination of multiple operations on various forms of regular languages.
2. To study the closure properties of various forms of regular languages.

References

1. R. Meyer and M. J. Fischer, “Economy of description by automata, grammars, and formal systems”, In Proceedings of the Twelfth Annual IEEE Symposium on Switching and Automata Theory, 188–191, 1971.
2. Hopcroft and J. Ullman “Introduction to Automata Theory, Languages, and Computation”, Addison-Wesley, Reading, MA, 2 edition, 1979.
3. D. Wood, “Theory of Computation”, John Wiley & Sons, Inc., New York, NY1987.
4. M. Ito, H. Jiirgensen, H.J. Shyr, and G. Thierrin, “Outfix and infix codes and related classes of languages”, Journal of Computer and System Sciences, 43:484-508, 1991.
5. S. Yu, Q. Zhuang, and K. Salomaa, “The state complexities of some basic operations on regular languages”, Theoretical Computer Science, 125(2):315-328, 1994.
6. M. Mohri, “On some applications of finite-state automata theory to natural language processing”, Natural Language Engineering, 61–80, vol. 2 (1996).
7. F. Pereira, M. Riley, “Speech recognition by composition of weighted finite automata “, Finite-State Language Processing, MIT Press, 431–453, 1996.
8. H.Jurgensen and S.Konstantinidis. Codes, “Word, Language, Grammar, volume 1 of Handbook of Formal Languages”, Springer-Verlag, 511–607, 1997
9. K. Salomaa and S. Yu., “NFA to DFA transformation for finite languages over arbitrary alphabets”, Journal of Automata, Languages and Combinatorics, 2(3):177–186, 1998.
10. M. Crochemore, F. Mignosi, and A. Restivo, “Automata and forbidden words” ,Information Processing Letters, 67(3):111—117, 1998.
11. 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, 60-70, 2001.
12. S. Yu, “State complexity of regular languages”, Journal of Automata, Languages and Combinatorics, 6(2):221-234, 2001.
13. J. Hopcroft, R. Motwani and J. Ullman, “Introduction to Automata theory, Languages, and Computation”, 2nd ed., Addison-Wesley, 2001

14. C. Cˆampeanu, K. Salomaa, and S. Yu., “Tight lower bound for the state complexity of shuffle of regular languages”, Journal of Automata, Languages and Combinatorics,7(3):303–310, 2002.
15. M. Holzer and M. Kutrib, “Nondeterministic descriptional complexity of regular languages”, International Journal of Foundations of Computer Science, 14(6):1087–1102, 2003.
16. Salomaa, D. Wood, and S. Yu, “On the state complexity of reversals of regular languages”, Theoretical Computer Science, 320(2-3):315–329, 2004.
17. G. Jir´askov´a, “State complexity of some operations on binary regular languages”, Theoretical Computer Science, 330(2):287–298, 2005.
18. Steven Bird Ewan Klein, “Regular Expressions for Natural Language Processing”, Creative Commons Attribution-ShareAlike License 01-29, 2006.
19. Y.S.Han, Yajun. Wang, and D. Wood, “Infix free regular expressions and Languages”, International journal of foundations of computer science, 379-393 vol.17, No. 2(2006).
20. S. Yu , “On the state complexity of combined operations”, In Proceeding of CIAA’06, Lecture Notes in Computer Science 4094, 11–22, 2006.
21. Brian A. Carter, Leah A. Hubert, and Arthur C. Walton, “Applications of regular expressions”, 2007.
22. Y.-S. Han, D. Wood, “ Outfix-Free Regular Languages and Prime Outfix-Free Decomposition” , Fundamenta Informaticae, 81(1-2):441–457, 2007.
23. Salomaa, K. Salomaa, and S. Yu, “State complexity of combined operations” ,Theoretical Computer Science, 383(2-3):140–152, 2007.
24. Y.S.Han and K. Salomaa, “State complexity of union and intersection of finite languages”, International Journal of Foundations of Computer Science, 19(3):581–595, 2008.
25. Y. Gao, K. Salomaa, and S. Yu, “The state complexity of two combined operations: Star of catenation and star of reversal”, Fundamenta Informaticae, 83(1-2):75–89, 2008.
26. H-S. Han and K. Salomaa, “State complexity of basic operations on suffix-free regular languages”, Theoretical Computer Science, 410(27-29):2537–2548, 2009.

27. Y.-S. Han, K. Salomaa, and D. Wood, “Operational state complexity of prefix-free regular languages”, In Automata, Formal Languages, and Related Topics ,99–115, 2009.

28. Y.-S. Han, K. Salomaa, and D. Wood, “Nondeterministic state complexity of basic operations for prefix-free regular languages”, Fundamenta Informaticae, 90(1-2):93–106, 2009.

29. Y.-S. Han, K. Salomaa, and D. Wood, “ Nondeterministic state complexity of basic operations for suffix-free regular languages” , Fundamenta Informaticae, 90(1-2):63–109, 2010.

Publications

1. Krishan Kumar Agrawal, Ajay Kumar, “Non-deterministic state complexity of various operations on infix-free regular languages” , Paper is communicated to International Conference on issues & challenges in Networking, Intelligence & computing technology, Department of Computer Science and engineering , KIET Ghaziabad on Sept 2-3, 2011.
2. Krishan Kumar Agrawal, Ajay Kumar, “State complexity of various operations on infix-free regular languages” , Paper is communicated to International Journal of Advanced Research in Computer Science.