State Complexity of Combined Operations for Suffix-free Regular Languages

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

Master of Engineering
in
Software Engineering

Submitted By
Darvena Rohilla
(801031006)

Under the supervision of:
Mr. Ajay Kumar
(Assistant Professor)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

July 2012

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*State Complexity of Combined Operations for Suffix-free Regular Languages*", in partial fulfilment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Mr. Ajay Kumar and refers other researcher's work which are duly listed in the reference section.

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

(Darvena Rohilla)

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

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

Countersigned by

(Dr. Maninder Singh) Head Computer Science and Engineering Department Thapar University Patiala	**(Dr. S. K. Mohapatra)** Dean (Academic Affairs) Thapar University Patiala

Acknowledgement

I would like to express my sincere gratitude to all those who have made the fulfilment of this work possible.

Firstly, I would like to thank my guide, Mr. Ajay Kumar, Assistant Professor, CSED, Thapar University, Patiala for the time, patience, guidance and invaluable advises that he has given me not only while my thesis work but throughout the course. It was a great opportunity to work under his supervision.

Then I would like to thank Dr. Maninder Singh, Head of the Department, CSED, Thapar University, Patiala for providing all the facilities and environment. I would also like to thank all my teachers for their support and invaluable suggestions during the period of my work.

I would also like to thank my parents for always supporting me in the tough and happy moments, for their never ending support and inspiration.

Finally, I wish to thank to thank my friend, Lft. Vikas Rohilla for being with me through the good and the bad.

Darvena Rohilla

(801031006)

Abstract

A regular expression is a pattern that describes a set of strings of particular type described by finite automata. Regular languages are classified into prefix-free, suffix-free, infix-free languages.

Study of state complexity is strongly motivated by applications of finite automata in software engineering, programming languages, natural language, speech processing and other practical areas. Since many of these applications use automata of large sizes, it is important to know the size of automata which is defined by the number of states in the automata. Number of states in the DFA accepting the language defines the state complexity of languages. Recently researchers have studied the state complexities of prefix free regular languages for individual and combined operations and suffix-free regular languages for individual operations.

This thesis focuses on estimating the state complexities of combined operations for suffix-free regular languages.

Abbreviations

NFA	Nondeterministic Finite Automata
DFA	Deterministic Finite Automata
FA	Finite Automata
NSC	Nondeterministic State Complexity
DSC	Deterministic State Complexity
SC	State Complexity

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This chapter gives basic introduction to automata, languages, operations on languages and its applications.

1.1 Automata

In theoretical computer science, automata theory is the study of mathematical objects called abstract machines [1]. An automaton can be defined as a system where energy, material and information are transformed, transmitted and used for performing some functions without direct participation of human being. Automata theory is one of the oldest research areas in computer science and is closely related to formal language theory. Automata play a major role in theory of computation, compiler design, parsing and formal verification [1].The use of finite automata has been used in lexical analysis in programming languages [30]. In parallel programming, automata theory has been associated with optimization problems [26]. Automata theory also serves as the basis for pattern recognition in natural language [29] and speech processing [30]. These applications motivate the study of state complexity, a fundamental subarea in automata theory. An automaton consists of a finite set of states, a single circle represents an intermediate state, a double circle represents a final state or accepting state and a circle with an incoming arrow represents a start state. At some instant, the automaton is in one of its states, and at any step when it reads a symbol, it jumps or transits to a next state that is decided by the transition function which works by reading the input symbols from a word until it is completely read. When the word is completely read, the automata stop at final state. The automaton accepts an input word if the final state is the accepting state, otherwise the word is rejected. The set of all the words accepted by an automaton is called the language recognized by the automaton.

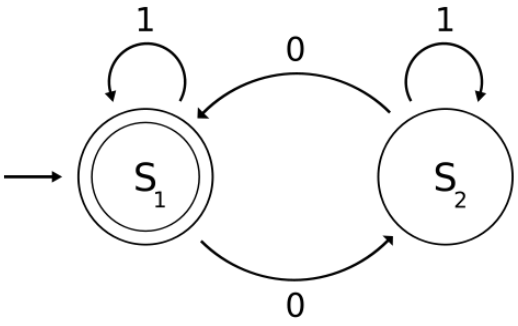

Fig 1.1: Example of automata [1]

The automata represented in the Fig 1.1 accepts the strings that reads consecutive 1's or strings with consecutive 1's starting and ending with a 0.

1.2 Basic Definitions and Notation

This section contains some basic definitions and notations. An automaton runs on some given sequence of inputs in discrete time steps. At each step, it gets one input that is picked up from a set of symbols or letters.

Def. 1: An alphabet [34] is a finite, nonempty set of symbols, denoted by $\sum$. The notation $\sum^*$ denotes the set containing all finite strings whose symbols are chosen from $\sum$.

Def. 2: At any time, the symbols so far fed to the automaton as input form a finite sequence of symbols called strings [25]. Strings are also called words.

An empty word is denoted by ε, and it does not contain any instance of alphabets. The length of a word "z" over an alphabet $\sum$ is the number of occurrences of letters in z. It is denoted by |z|. The a-length [25] of the word z is the number of times symbol "a" appears in z. It is denoted by $|z|_a$.

Def. 3: A language [25] over $\sum$ is a set of words which are chosen from $\sum^*$. The languages { } and $\emptyset$ are over any alphabet.

Def. 4: The representation of certain sets of strings in an algebraic notation is called regular expression. Regular expressions [25] describe the languages accepted by finite state automata. Given the regular expression $r = (0 + 1)^* 1$ represents a language consisting of all the words over {0, 1} that end with 1.

Finite automata can be deterministic or non-deterministic. Non determinism [25] means there are choices of moves for automata, rather than moving in a particular unique direction it allows a set of possible moves.

Def. 5: A deterministic finite automata [4] is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

1. Q is a finite set of states,
2. Σ is a finite set called the alphabet,
3. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$,
4. $q_0 \in Q$ is known as starting state,
5. $F \subseteq Q$ is the subset of Q and known as set of final states.

A language L is accepted by a DFA M $(Q, \Sigma, \delta, q_0, F)$, if and only if $L= \{ w \mid \delta^*(q_0 , w) \in F \}$[25].

Example 1: Let M = $(\{s_1, s_2\}, \{a, b\}, \delta, \{s_1\}, \{s_2\})$ be a DFA shown in Fig 1.2, where δ is represented by the state transition Table 1.1.

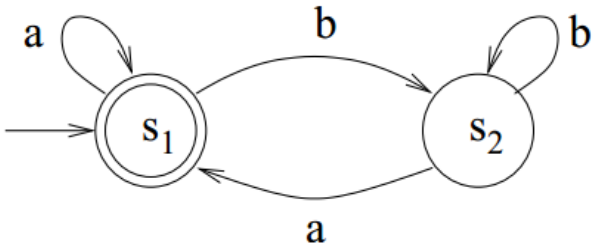

Fig 1.2: State transition diagram for DFA M

DFA M can be represented by transition table 1.2

Table 1.1: Transition table for DFA M

symbols / states	a	b
s_1	s_1	s_2
s_2	s_1	s_2

The language accepted by a DFA is the set of all the strings accepted by the DFA. The DFA in fig1.3 does not accept any string because it has no accepting state.

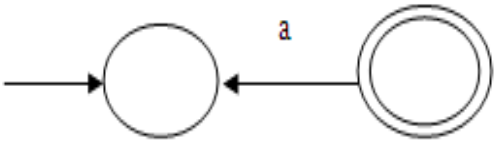

Fig 1.3: Null language

Thus the language it accepts is the empty set Ø.

Def. 6: A Non deterministic finite automata [3] is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where Q, Σ, q_0, and F are similar as in DFA except transition function which is defined by $\delta: Q \times \Sigma \rightarrow Q$.

Example 2: Let $N = (\{s_1, s_2\}, \{a, b\}, \delta, \{s_1\}, \{s_2\})$ be an NFA shown in Fig 1.4, where δ is represented by the state transition diagram in Table 1.2.

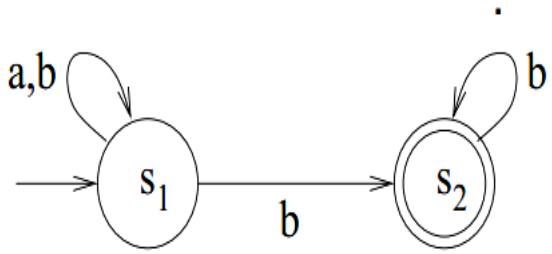

Fig 1.4: State transition diagram for NFA N

NFA M can be represented by transition table 1.2. It is to be noted that $\delta(s_1, b)$ has more than one state therefore N is nondeterministic.

Table 1.2: Transition table for NFA N

symbols / states	a	b
s_1	s_1	$\{s_1, s_2\}$
s_2	-	s_2

1.3 Chomsky Hierarchy

Chomsky hierarchy [2] is a containment hierarchy of classes of formal grammars which is shown from the diagram below Fig 1.5. According to Chomsky hierarchy formal languages can be classified as regular languages, context-free, the context-sensitive and the recursively enumerable languages.

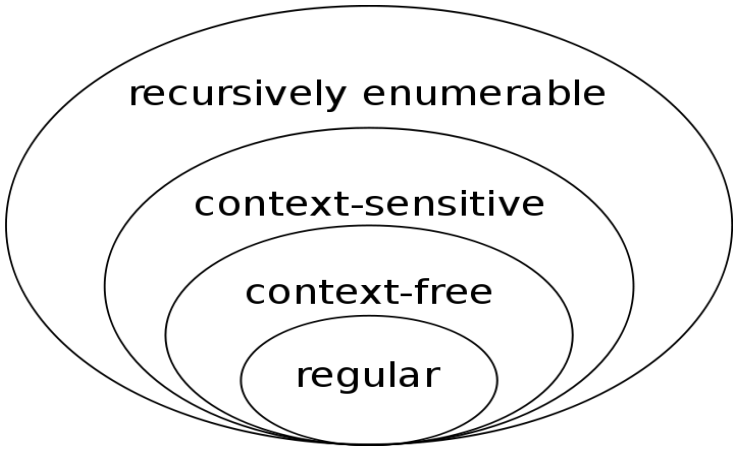

Fig. 1.5: Chomsky hierarchy [2]

1. Type-0 grammar (unrestricted grammars): They generate all languages that can be recognized by a Turing machine [25]. These languages are also known as the recursively enumerable languages [2].
2. Type-1 grammar: They generate the context-sensitive languages. The languages described by these grammars are exactly all languages that can be recognized by a linear bounded automaton [25].
3. Type-2 grammar: They generate context-free languages. These languages are exactly all languages that can be recognized by a non-deterministic pushdown automaton [25]. Context free languages are the theoretical basis for the syntax of most programming languages [2].
4. Type-3 grammar: They generate regular languages. Such a grammar restricts its rules to a single non-terminal on the left-hand side of the terminal or a non-terminal on the right-hand side consisting of a number of terminals. Regular languages [2] are commonly used to define search patterns and the lexical structure of programming languages.

1.4 Types of regular languages

Regular languages can be classified into prefix-free, suffix free and infix free regular languages.

1. Prefix-free regular language: A regular language is prefix-free if and only if its minimal DFA M has only one final state and the final state has no out-transitions whose target state is not a sink state [21]. This property is called non-exiting property. Given two strings x and y over $\sum$, x is a prefix of y if there exists a string $z \in \sum^*$ such that xz = y. Given a set X of strings over $\sum$ is prefix-free, if no string in X is a prefix of any other string in X. For example, string s_1=abba, and string s_2=abbab. Now the language L(M) over $\sum$ where L(M)={abba, abbabaaab} is not prefix-free since abba is prefix of abbabaaab.

2. Suffix -free regular languages: A regular language is suffix-free [17] if and only if its minimal DFA M have a unique sink state and there is no in-transition from the intermediate state to the start state. This is called non-returning property of suffix-free regular languages. Given two strings x and y over $\sum$, x is a suffix of y if there exists a string $u \in \sum^*$ such that y = ux. Given a set X of strings over $\sum$ is suffix-free set if no string in X is a suffix of any other string in X. A regular language L is defined to be suffix-free if L has all suffix-free sets. For example, string s_1 = ab, and string s_2 = aabbab. Now the language L(M) over the NFA where L(M)={ab, aabbab} is not suffix-free since ab is a suffix of aabbab.

3. Infix-free regular languages: A regular language is infix-free [22] if a DFA M is minimal and is non-returning and non-exiting. Given two string x and y over $\sum$, x is an infix of y if there exists two strings u, $v \in \sum^*$ such that y = uxv. Given a set X of strings over $\sum$ is infix-free set if no string in X is an infix of any other string in X. A regular language L is defined to be infix-free if L is an infix-free set. Infix-freeness is used in text searching and computing forbidden words. For example, string s_1=abba, and string s_2 = aabbab. Now the language L(M) over the NFA where L(M) = {bb, aabbab} is not infix-free, since bb is an infix of aabbab.

1.5 Operations on languages

Following are the operations defined on languages:

1. Reversal: Reversal of x is denoted by x^R. Reverse [25] of a string contains all letters of a string in reverse order. For null string (denoted by ε), reverse of the string is ε. For a language L over an alphabet ∑, the reversal of L is denoted by L^R, and $L^R = \{x^R \mid x \in L\}$. For example if L = {001,101,110}, then L^R = {100,101,011}.
2. Concatenation: Consider x and y are strings over an alphabet ∑, concatenation of x and y is denoted by xy. The length of the new string xy is the sum of the length of x and y. For a language L_1 and a language L_2 over alphabet ∑, the concatenation [25] of L_1 and L_2 is denoted by $L_1.L_2$, and $L_1.L_2 = \{xy \mid x \in L_1, y \in L_2\}$. For example, if L_1={01,10,11} and L_2={ε, 01}, then L_1L_2={01,10,11,0101,1001,1101}.
3. Intersection: L_1 and L_2 are languages over an alphabet ∑, the intersection [25] of L_1 and L_2 is denoted by $L_1 \cap L_2$, and $L_1 \cap L_2 = \{x \mid x \in L_1 \text{ and } x \in L_2\}$. For example, if L_1={01,10,11} and L_2={ ε, 01,11}, then L_1L_2={01,11}.
4. Union: L_1 and L_2 are languages over an alphabet ∑, the union [25] of L_1 and L_2 is denoted by $L_1 \cup L_2$, and $L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$. For example if L_1={001,10,111} and L_2={ ε,01}, then $L_1 \cup L_2$={ ε,01,001,10,111}.
5. Kleene closure: Consider a language L over an alphabet ∑, the star of L is denoted by L^*. The operation $L^*$is also called Kleene closure [25]. We can say that $L_0 = \{\varepsilon\}$ and $L^i = L^{i-1}L$, where $i \geq 1$. For example if L= {0, 1}, then Kleene closure of L is the set of all the strings of 0's and 1's i.e. L^* ={ ε, 0, 1, 01, 10, 00, 11, 000.....}.

1.6 Thesis Outline

This thesis is organized into 5 chapters. Chapter 1 describes automata, languages, operations on languages and its applications. Chapter 2 covers state complexity of prefix-

free and suffix-free languages over various operators. Chapter 3 discusses the problem statement, objectives and methodology, motivation behind the thesis and method of approximating state complexities of combined operations on suffix-free regular languages. Chapter 4 describes state complexities of combined operators of suffix-free regular languages. Chapter 5 summarizes the conclusions drawn in the thesis along with the future directions

Chapter 2
State Complexity of Regular Languages

Basically there are three ways to estimate complexity [33] of regular languages: number of states, number of transitions, and the combination of number of states and transitions. State complexity is a type of description complexity [5] on regular languages and it gives a lower and tight bound for the space and the time complexity for many operations.

There are deterministic and nondeterministic models for the state complexity of operations. B.Cui and M.Daley [8, 9] investigated the deterministic state complexity of formal systems. Yu and Y. Gao [32, 10] investigated the state complexity for single and combined operations on regular languages. Shallit [6] investigated the nondeterministic state complexity of finite automata. M. Holzer [22] investigated the nondeterministic state complexity of operations on unary languages. Han et al. [15] studied the state complexity of prefix-free regular languages, Han and Salomaa [16] looked into the state complexity of suffix-free regular languages.

This chapter describes various methods to estimate the state complexity prefix-free and suffix-free regular languages.

2.1 State Complexity of individual operations for Prefix-free Regular Languages

Prefix-free regular languages have been used to define determinism for generalized automata [11] and for expression automata [20]. The state complexity of Prefix-free regular languages is calculated based upon their non-exiting structural property [18].

Glaister and Shallit [12] presented a technique called fooling set theory and proved that a non-trivial prefix-free regular language L for a FA (namely, $L \neq \{\lambda\}$) must have at least 2 states (one start state and one final state).

Fooling set theory says that the language $L = L((a^{n-1})^* b)$ is accepted by an NFA P with n number of states that consists of a cycle of length n-1 of a-transition and b-transition

from the start state to a final state. Now P is the fooling set for L and its cardinality is n, then the nondeterministic State complexity of L is exactly n.

2.1.1 Concatenation of prefix-free regular languages

From Han et al. [15], the deterministic and nondeterministic state complexity is linear in the sizes of the component automata for concatenation of prefix-free regular languages as shown in Fig 2.1.

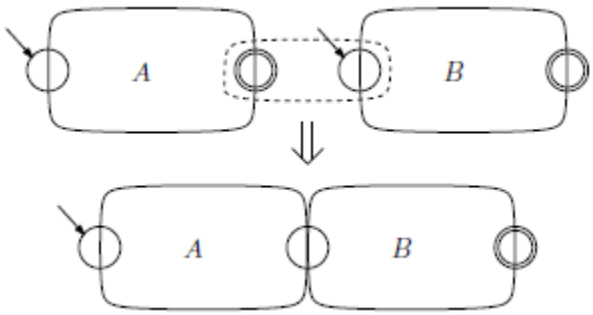

Fig 2.1: Concatenation of prefix-free minimal DFA [15]

Theorem 2.1 Given two prefix-free regular languages L_1 and L_2 accepted by m state and n state NFA, then minimal and sufficient NSC(L_1L_2) is m + n – 1 [18].

Proof: Let A = (Q_1, Σ, δ_1, s_1, f_1) and B = (Q_2, Σ, δ_2, s_2, f_2) be two NFAs recognizing prefix-free regular languages L(A) and L(B). A and B are non-exiting and because of prefix-freeness it must have only one final state. NFA C can be constructed for L(A)L(B) by merging the final state f_1 of A with the start state s_2 of B to give a single state that eliminates all out-transitions from f_1 of A.

Theorem 2.2 Given two prefix-free regular languages L_1 and L_2 with DSC(L_1) = m and DSC(L_2) = n, then the deterministic state complexity DSC(L_1L_2) for L_1L_2 is m+n-2 [15].

Proof: Let A = (Q_1, Σ, δ_1, s_1, f_1) and B = (Q_2, Σ, δ_2, s_2, f_2) be two DFAs recognizing prefix-free regular languages. A and B are non-exiting and assumed to 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 eliminates all out-transitions from the final state of A. Two sink states are also merged to give a single sink state. Hence the resulting automaton is deterministic and, therefore, m + n − 2 states are sufficient.

2.1.2 Union of prefix-free regular languages

Han et al. [22] proved that the union of an m-state and an n-state DFA accepting prefix-free languages is calculated using the Cartesian product of states as shown in Fig 2.2.

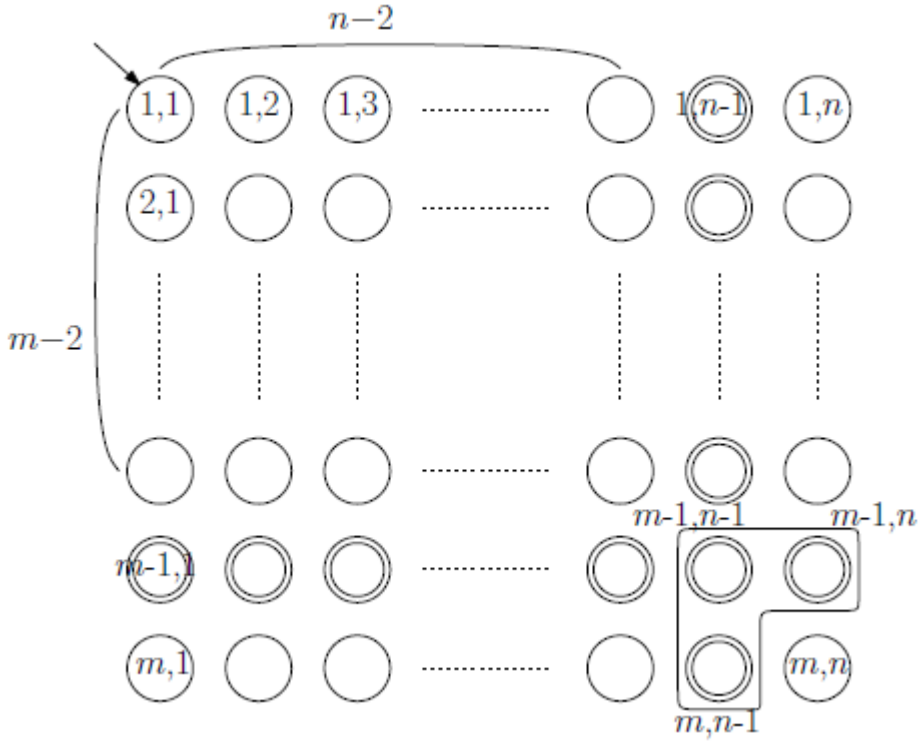

Fig 2.2: Union of two prefix-free minimal DFAs [15]

For the NFA case, state complexity of two prefix-free regular languages is directly constructed without the Cartesian product of states [18].

Theorem 2.3 Given two prefix-free regular languages L_1 and L_2, the nondeterministic state complexity $NSC(L_1 \cup L_2)$ is m + n, where $m = NSC(L_1)$ and $n = NSC(L_2)$ [18].

Proof: 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) can be constructed for $L_1 \cup L_2$ 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 $Q = (Q_1 \cup Q_2 \cup \{s\})$,$q \in Q$, $a \in \sum$, transition function is defined as $\delta(p, a) = \{ (\delta_1(s_1, a) \cup \delta_2(s_2, a)$ if $p = s)$, $(\delta_1(p, a)$ if $p \in Q_1)$, $(\delta_2(p, a)$ if $p \in Q_2) \}$. From the above inference m + n states are sufficient for $L_1 \cup L_2$.

Theorem 2.4 Given two prefix-free regular languages L_1 and L_2, the deterministic state complexity for the union of L_1 and L2 is $mn - 2$, where m is the number of states in DFA accepting L_1 and n is the number of states in DFA accepting L_2 [15].

Proof: For a state (i, j) in $A \cup_c B$, the right language $L_{(i,j)}$ of (i, j) is the union of L_i in A and L_j in B, then all states in the second-last row and all states in the second-last column of $A \cup_c B$ are final states and all states in the last row and all states in the last column of $A \cup_c B$ are not necessarily sink states except for state (n,m). $L_{(m,n-1)} = L_{(m-1,n)} = L_{(m-1,n-1)}$, therefore these three states (m, n−1), (m−1, n) and (m−1, n−1) can be 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.

2.1.3 Intersection of prefix-free regular languages

State complexity of intersection operation of prefix-free regular language is calculated using Cartesian product of the states corresponding to their FAs. mn is the tight [23] bound for the intersection of the regular languages given by either DFAs or NFAs as shown in Fig 2.3 and Fig 2.4.

Theorem 2.5 Given two prefix-free regular languages L_1 and L_2, the nondeterministic state complexity $NSC(L_1 \cap L_2)$ for $L_1 \cap L_2$ is $mn-(m+n)+2$, where $m = NSC(L_1)$, $n = NSC(L_2)$ and $|\sum| \geq 3$ [18].

Proof: Given two FAs $A = (Q_1, \sum, \delta_1, s_1, F_1)$ and $B = (Q_2, \sum, \delta_2, s_2, F_2)$, accepted by the prefix-free regular languages L_1 and L_2. Now A finite automata $M = (Q_1 \times Q_2, \sum, \delta, (s_1, s_2), F_1 \times F_2)$ can be constructed for the intersection of L(A) and L(B) based on the Cartesian product of states [24], where $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ for $p \in Q_1$, $q \in Q_2$ and $a \in \sum$. For intersection, first a unique number from 1 to m is assigned to each state in A and from 1 to n is assigned in B, where $|A| = m$ and $|B| = n$. Let $A \cap_c B$ denote the resulting intersection automaton that is 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 $1 \leq j \leq m-1$, that do not appear in an accepting path in $A \cap_c B$ are useless.

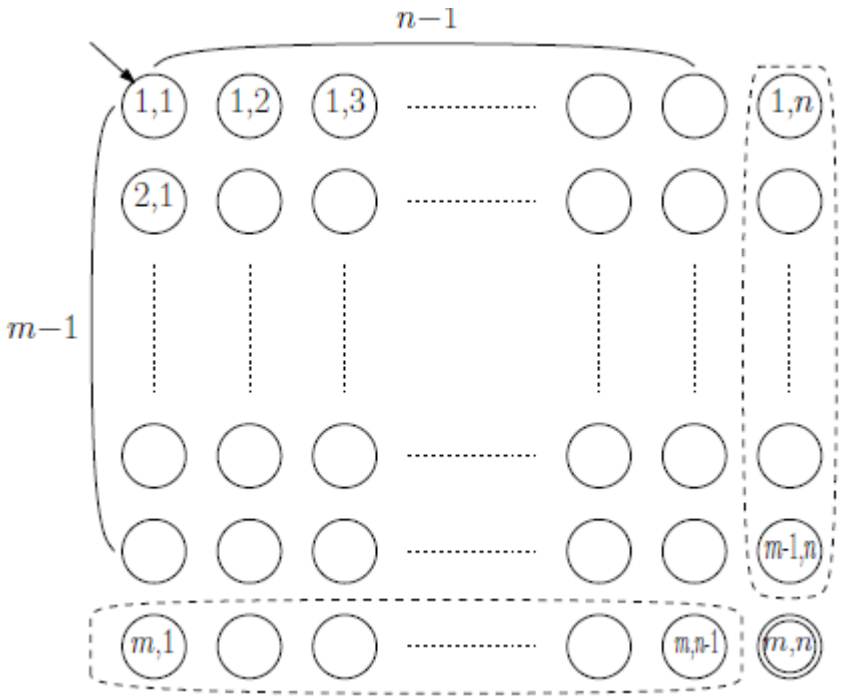

Fig 2.3: Intersection of NFA A $\cap_c$ B [18]

Hence after removing these useless states as shown in Fig 2.3, the resulting automaton is left with mn − (m − 1) − (n − 1) = mn − (m + n) + 2 minimal and sufficient no of states for L(A) ∩ L(B).

Theorem 2.6 Given two prefix-free regular language L_1 and L_2 accepted by m state and n state DfAs , mn−2(m+n)+6 states are necessary and sufficient in the worst-case for DSC($L_1 \cap L_2$) [15].

Proof: Given m state and n state DFAs A and B accepting the prefix-free regular langueges L(A) and L(B). To compute L(A) ∩ L(B), first we need to assign a unique number for each state from 1 to m in A and from 1 to n in B. Assume that m^{th} and n^{th} states are sink states and $(m-1)^{th}$ and $(n-1)^{th}$ states are final states in A and B. Now for right language $L_{(i,j)}$ of (i, j) state, there is an accepting path from (i, j) to (m-1, n-1). Therefore word $w \in L_{(i,j)}$ if and only if $w \in L_i \cap L_j$. Consider last row and last column (m, i) for $1 \leq i \leq n$ and (j, n) for $1 \leq j \leq m$ are equivalent, hence they can be merged. Next consider all states in second last row (m-1, i) for $1 \leq i \leq n-2$ and second last column (j, n-1) for $1 \leq j \leq m-2$ are also equivalent states and can be merged as shown in Fig 2.4.

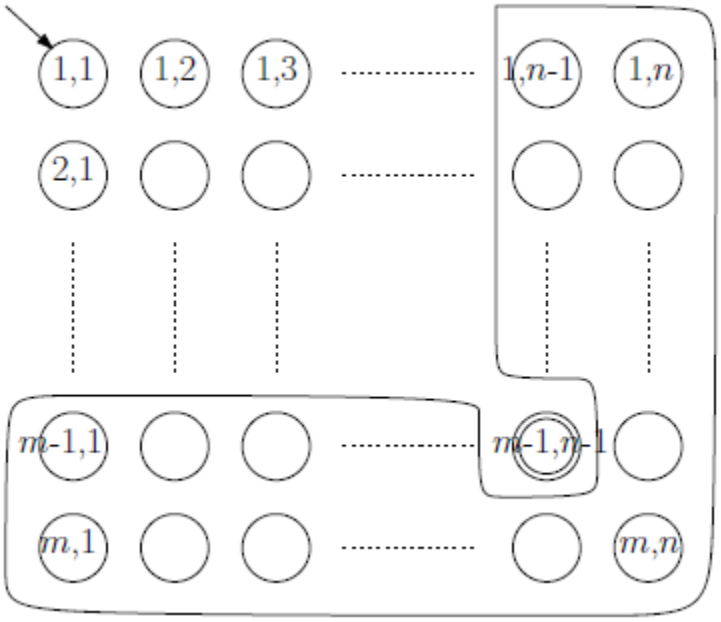

Fig 2.4: Merging of all equivalent states in DFA of A ∩c B [15]

So, after reducing the upper bound, the resulting DFA has nm-2(m+n)+6 minimal and sufficient state complexity.

2.1.4 Kleene star of prefix-free regular languages

For calculating Kleene star of prefix-free regular languages, m is the tight bound for both NFA and DFA case. Fig 2.5 represents kleen star of a language accepted by NFA A

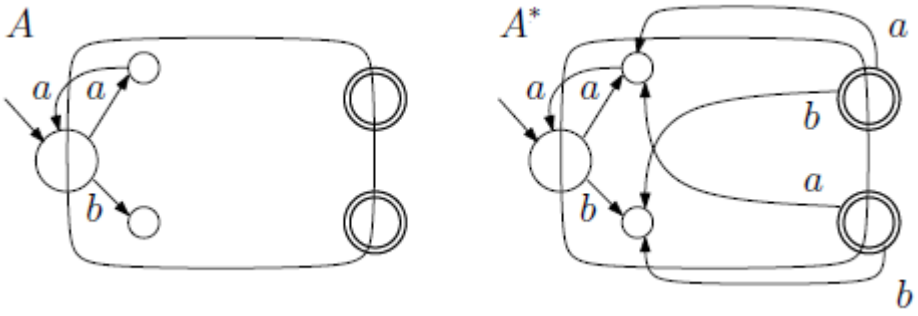

Fig 2.5: Kleene star of L(A) [18]

Theorem 2.7 Given a prefix-free regular language L accepted by NFA, the nondeterministic state complexity NSC(L^*) for L^* is m [26], for |∑|= 2 where m = NSC(L) [18]. □

Theorem 2.8 Given a prefix-free minimal DFA A, m states are necessary and sufficient in the worst-case for the minimal DFA of $L(A)^*$[22] where m = |A| [22]. □

2.1.5 Reversal of prefix-free regular languages

For calculating reversal of prefix-free regular language, the direction of accepting DFA is flipped and start and final states are interchanged.

Theorem 2.9 Given a prefix-free regular language L with m no of states in the NFA, then the nondeterministic state complexity of reversal of language L, NSC (L^R) is m [18].

Proof: Given a minimal NFA A = (Q, ∑, δ, s, f) accepted by prefix-free regular language L, where |Q| = m. A has a single final state and it is non-exiting. Based on this structural property, an NFA A^R = (Q, ∑, δ^R, s, f) is constructed for L^R by flipping the directions of all transitions and interchanging the start state and the final state. Since $L^R = L(A^R)$, and both A and A^R have the same set of states, therefore m states are sufficient for LSC(L^R).

Theorem 2.10 Given a prefix-free regular language L with m no of states in the minimal DFA, then the deterministic state complexity of reversal of language L i.e DSC(L^R) is $2^{m-2} + 1$ [15]. □

Now in case of minimal DFA for $L(A)^R$, an exponential number of states is required. Hence, the state complexity of reversal of a prefix-free minimal DFA is $2^{m-2} + 1$, where m is the number of states.

2.1.6. Complementation of prefix-free regular languages

The calculation of complement of prefix-free regular language is linear for DFA case and exponential for NFA case.

Theorem 2.11 Given an n state NFA accepting L, 2^m states are sufficient for the complementation of the prefix-free regular language NSC(~L) [18].

Proof: For the tight bound, Jir´askov´a [26] recently showed that 2^m states are necessary when |∑| = 2. Let A = (Q, ∑, δ, s, f) be the NFA accepted by the prefix-free regular language L. Now 2^m (some of them may be useless states) states are necessary for the NSC (~L), and the final states are the subsets of Q that contain f. Since L(A) is prefix-

free, L(~A) is also prefix-free and, thus all 2^{m-1} equivalent final states are merged into a single final state f ` in ~A. Therefore, the number of states for an NFA for ~ L(A) is $2^m - 2^{m-1}$ which is equal to 2^{m-1}.

Theorem 2.12 Given prefix-free regular language, with m number of states in minimal DFA, n states are sufficient for the complementation of the language [15].

Proof: The complementation of an n-state DFA does not change the state complexity since it simply interchanges final states and non-final states. Thus, in case of DFA's the state complexity for the complement of a prefix-free regular language is m only if there are m no of states in the minimal DFA for prefix-free regular languages.

2.1.7 Conclusion

The difference between state complexities of nondeterministic and deterministic prefix-free regular languages can be evaluated from the Table 2.1. As can be seen that there is not much difference in some of the state complexities of individual operations, but there is a difference of exponentiation in case of reversal and complementation operation.

Table 2.1: State complexity of individual operations of prefix-free regular languages

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

2.2 State complexities of combined operations for prefix-free regular languages

Yu and his co-authors worked on the following state complexity of combined operations of prefix-free regular languages [19].

2.2.1 Star of Union

According to G.Jir`askov`a [27], subset construction is performed on DFA for star of union of two prefix-free regular languages, and then the final states of the original DFAs are merged. The state complexity of the union of two prefix-free regular languages is mn − 2 [15] and the state complexity of the star of an m-state regular language is $3*2^{m-2}$ [33]. Thus, the composition of two complexities is $3*2^{mn-4}$.

Theorem 2.13 The worst-case state complexity of the star-of-union of an m-state and an n-state, m, n ≥ 3, prefix-free regular languages is precisely $5* 2^{m+n-6}$, where $|\Sigma| \geq 5$ [27].

Proof: For given DFAs A= (Q_1, Σ, δ_1, $q_{0,1}$, f_1) and B = (Q_2, Σ, δ_2, $q_{0,2}$, f_2) recognizing prefix-free regular languages L(A) and L(B), DFA M = (Q, Σ, δ, {q_0, $q_{0,1}$, $q_{0,2}$}, F) is constructed for the language $(L(A_1) \cup L(A_2))^*$. It is to be noted that the start state of M is {q_0, $q_{0,1}$, $q_{0,2}$} which means that the computation begins by simulating both the computation of A and the computation of B as shown in Fig 2.6.

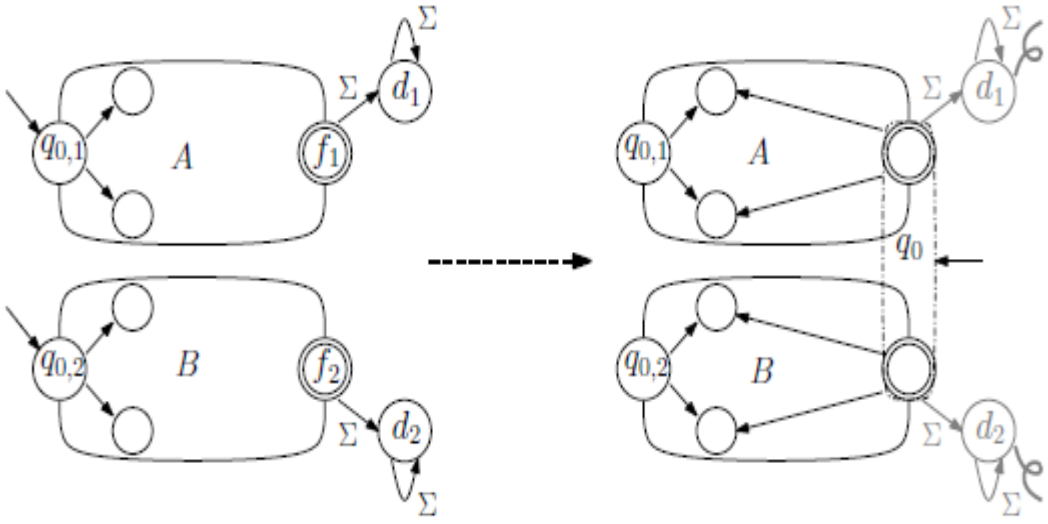

Fig 2.6: Star of union of prefix-free DFA A and B [19]

Whenever one of the simulated computations enters the final state, the computation of M adds both $q_{0,1}$ and $q_{0,2}$ to the current set of states and begins new computations simulating A and B. Namely, q_0 in the current set indicates that the previously simulated computation step is accepted either in A or in B. Note that the presence of $q_{0,1}$ or $q_{0,2}$ in the current state of M is not sufficient to guarantee this property. The choice of the final states guarantees that M recognizes exactly the language $(L(A) \cup L(B))^*$ with $5*2^{m+n-6}$ state complexity.

2.2.2 Star of Reversal

Let A be a minimal DFA for a regular language and |A| = m. Since the state complexity of L(A) is m, the reversal $L(A)^R$ of L(A) can be accepted by an NFA A^R with m states, where A^R can have multiple start states. Then, use of subset construction on A^R and the resulting DFA can have at most 2^m [28] states. Thus, the upper bound for the reversal of regular languages is 2^m.

Theorem 2.14 The worst-case state complexity of the star-of-reversal of an m state prefix-free regular language is precisely $2^{m-2} + 1$, where $|\Sigma| \geq 3$ [19].

Proof: Given a DFA $A = (Q, \Sigma, \delta, q_0, f)$ accepting the prefix-free regular language L(A). Now to construct $A^R = (Q, \Sigma, \delta^R, q_0, f)$, a new final state f ` is introduced and all states in F are connected to f ` with label A^R. The transition function of A consists of all mappings from Q to Q` as shown in Fig 4.7.

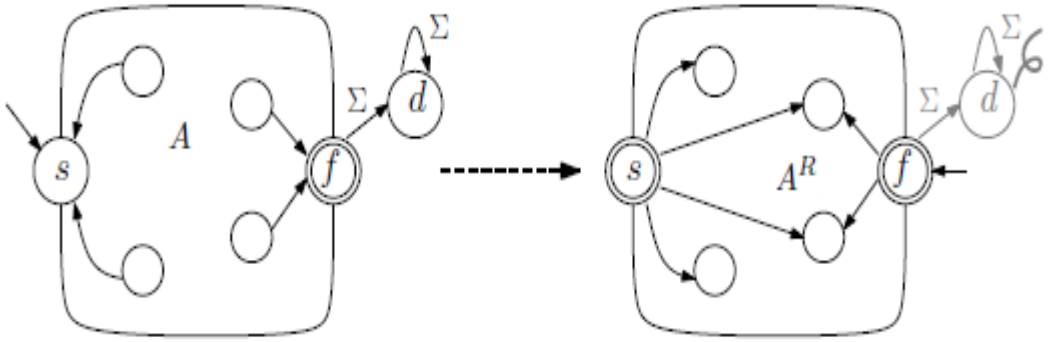

Fig 2.7: Star of Reversal of A [19]

A cannot have a sink state, and consequently, d is not equivalent with any of the states of A. Furthermore, $L(A^R)$ is prefix-free. Thus, A has m − 2 states, then A^R has m states.

Salomaa et al. [32] showed that $2^{m-2} + 1$ states are necessary for the star-of-reversal of an m-state prefix-free regular language L.

2.2.3 Star of concatenation

The state complexity of the star-of-catenation is equal to the state complexity calculated by the composition of state complexities of concatenation. This is due to the fact that prefix-freeness is closed under catenation.

Theorem 2.15 The worst-case state complexity of the star-of-catenation of an m-state and an n-state, m, $n \geq 3$, prefix-free regular languages is precisely $m + n - 2$ [19]. □

2.2.4 Star of intersection

Han et al. [15] gave a construction for the intersection of two prefix-free DFAs based on the Cartesian product of states. In this construction, based on the structural properties of prefix-free DFAs unreachable states are identified and removed and equivalent states are also identified and merged.

Theorem 2.16 The worst-case state complexity of the star-of-intersection of an m-state and an n-state, m, $n \geq 3$, prefix-free regular languages is precisely $mn - 2(m + n) + 6$, where $|\Sigma| \geq 4$ [19]. □

2.2.5 Conclusion

After state complexities of individual operations, researchers have started looking at state complexites of combined operations [13]. They showed that the state complexity of a combined operation is usually not equal to the composition of the state complexities of the participating individual operations [31]. On the other hand, they also observed that in a few cases, the state complexity of a combined operation is very close to the composition of the state complexities of the individual operations. However, in a few cases, the state complexity of combined operations and the composition of state complexities are similar that can be shown from the Table 2.2

Table 2.2: State complexity of combined operations for prefix-free regular languages

Operation	Complexity	Operation	Complexity
$L_1 \cup L_2$	mn-2	$(L_1 \cup L_2)^*$	$5*2^{m+n-6}$
$L_1 \cap L_2$	mn-2(m+n)+6	$(L_1 \cap L_2)^*$	mn-2(m+n)+6
L_1L_2	m+n-2	$(L_1L_2)^*$	m_1+m_2-2
L_1^R	$2^{m-2}+1$	$(L_1^R)^*$	$2^{m-2}+1$
L_1^*	m	$(L_1^*)^* = L^1$	m

2.3 State Complexity of Basic Operations for Suffix-Free Regular Languages

If language L is regular and non-empty, then the start state of a DFA for L^R should not have any in-transitions if L is a suffix-free. However, this condition is necessary but not sufficient. Due to this fact, the state complexity of suffix-free regular languages is not symmetric to the prefix-free case. A FA A is a suffix-free FA if L(A) is suffix-free language. Notice that a suffix-free FA must be non-returning, non-empty and must have a sink state by definition. In a suffix-free regular language it is assumed that a given NFA has no λ -transitions since an n-state NFA can always be transformed with λ -transitions to an equivalent n-state NFA without λ -transitions [24].

2.3.1 Concatenation of suffix-free regular languages

For the prefix-free regular languages, the state complexity is linear in the sizes of the component automata in both DFA and NFA cases. There is an exponential gap between the state complexities of prefix-free regular languages of DFA case and NFA case [19].

Theorem 2.17 Given two suffix-free regular languages L_1 and L_2, the nondeterministic state complexity $NSC(L_1L_2)$ is $m+n-1$, where $m = NSC(L_1)$ and $n = NSC(L_2)$ [14].

Proof: Consider two suffix-free regular languages accepted by m state and n state NFAs A and B. In order to do concatenation on L(A) and L(B), a new start state s is introduced

and s_1 and s_2 are removed such that the transition function $\delta(s_1, i)$ and $\delta(s2, i)$ are replaced with $\delta(s, i)$ for all $i \in \sum$. Hence the resultant state complexity of L(A) L(B) is m+n-1 which is minimal and sufficient.

Now, for state complexity of the catenation of two suffix-free regular languages for minimal DFA's A and B, the upper bound is computed and after that a matching lower bound is presented.

Theorem 2.18 For arbitrary m, $n \geq 3$, $(m-1)2^{n-2} + 1$ states are necessary and sufficient in the worst-case for the catenation an m-state and an n-state suffix-free minimal DFAs accepting L_1L_2 respectively. [16]

Proof: Yu et al. [16] presented a DFA construction for the catenation of two DFAs. Based on their construction, a DFA $C = (Q, \sum, \delta, s, F)$ for L(A) L(B) is computed as follows: $Q = Q_1 \times 2^{Q2} \setminus F_1 \times 2^{Q2\setminus\{s2\}}$, where 2^X denotes the power set of X, Q is a set of pairs such that the first component of each pair is a state from Q_1 and the second component is a subset of Q_2. Q does not have pairs whose first component is a final state of A and whose second component does not contain s_2. Thus, the number of states in C is $m2^n - k2^n - 1$ by construction, where $k = |F_1|$ is the number of final states in A. Now let d_2 denote the sink state of B. Because of d_2, two states in C are equivalent. Thus, all such states are merged and reduced to $(m2^{n-1} - k2^n - 2)$ states. Therefore, the number of remaining states is $m2^n - k2^{n-1} - (m2^{n-1} - k2^{n-2}) = m2^{n-1} - k2^{n-2}$. Since A is non-returning and, thus, s_1 has no in-transitions. It removes $(2^{n-1} - 1) + (m-1-k)2^{n-2}$ states and the current number of states is

$m2^{n-1} - k2^{n-2} - (2^{n-1} - 1) - (m-1-k)2^{n-2} = (m-1)2^{n-2} + 1$ which is sufficient and minimal.

2.3.2 Union of suffix-free regular languages

Now for the union of two suffix-free regular languages, DSC $(L(A) \cup L(B))$ is computed using the Cartesian product of states and for NFA case it is computed linearly.

For nondeterministic state complexity, NFA for the union of two suffix-free regular languages is directly constructed without the Cartesian product. The construction relies

on non determinism and the fact that the computation of a suffix-free FA cannot return to the start state.

Theorem 2.19 Given two suffix-free regular languages L_1 and L_2, the nondeterministic state complexity $NSC(L_1 \cup L_2)$ is $m+n-1$, where $m = NSC(L_1)$, $n = NSC(L_2)$, $m, n \geq 2$ and $|S| \geq 2$ [14]. □

Theorem 2.20 Given two suffix-free minimal DFAs A and B corresponding to the suffix-free regular languages L_1 and L_2, $mn - (m + n) + 2$ is the state complexity for the $DSC(L_1 \cup L_2)$, where $|\sum| \geq 5$ [16].

Proof: Given two minimal DFAs $A = (Q_1, \sum, \delta_1, s_1, F_1)$and $B = (Q_2, \sum, \delta_2, s_2, F_2)$,, let $M = (Q_1 \times Q_2, \sum, \delta, (s_1, s_2), F)$, where for all $p \in Q_1$ and $q \in Q_2$ and $a \in \sum$, $\delta((p, q), a) = (\delta(p, a), \delta(q, a))$ and $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. Then, $L(M) = L(A) \cup L(B)$ and M is deterministic. Let $A \cup_c B$ denote M. Consider the right language of a state (i, j) in $A \cup_c B$, all states (m, i) and (j, n) for $1 \leq i \leq n$ and $1 \leq j \leq m$, in $A \cup c B$ are not necessarily equivalent. Thus, these states cannot be merged. On the other hand, all states (1, i) and (j, 1), for $1 < i \leq n$ and $1 < j \leq m$, are useless since L(A) and L(B) are suffix-free. Therefore $A \cup_c B$ can be minimized by removing these states which results into $2 + (m - 2)(n - 2) + (m - 2) + (n - 2) = mn - (m + n) + 2$ states sufficient for DFA for $(L_1 \cup L_2)$.

2.3.3 Intersection of suffix-free regular languages

Subset construction is done for the intersection of two suffix-free regular languages. There is a complexity gap between NFA and DFA, as NFA does not need to have a sink state

Theorem 2.21 Given two m state and n state NFAs, $|\sum| \geq 3$ for the suffix-free regular languages L_1 and L_2, $mn-(m+n)+2$ is the state complexity of $NSC(L_1 \cap L_2)$ [14].

Since L(A) and L(B) are suffix-free regular languages, all states (s_1, q) and (p, s_2), for $p(\neq s_1) \in Q_1$ and $q(\neq s_2) \notin Q_2$, are unreachable from (s_1, s_2) in M. All unreachable states are

removed and the upper bound is reduced as mn−(m−1)−(n−1) = mn−(m+n)+2 sufficient number of states are required for $L_1 \cap L_2$.

Theorem 2.22 Given two m state and n state minimal DFAs A and B for the corresponding suffix-free regular languages L_1 and L_2, mn − 2(m + n) + 6 is the state complexity for DSC($L_1 \cap L_2$) where $|\sum| \geq 3$ [16].

Proof: Given two DFAs A and B, a DFA M for the intersection of L(A) and L(B) is constructed based on the Cartesian product of states [25] as shown in Fig 2.8.

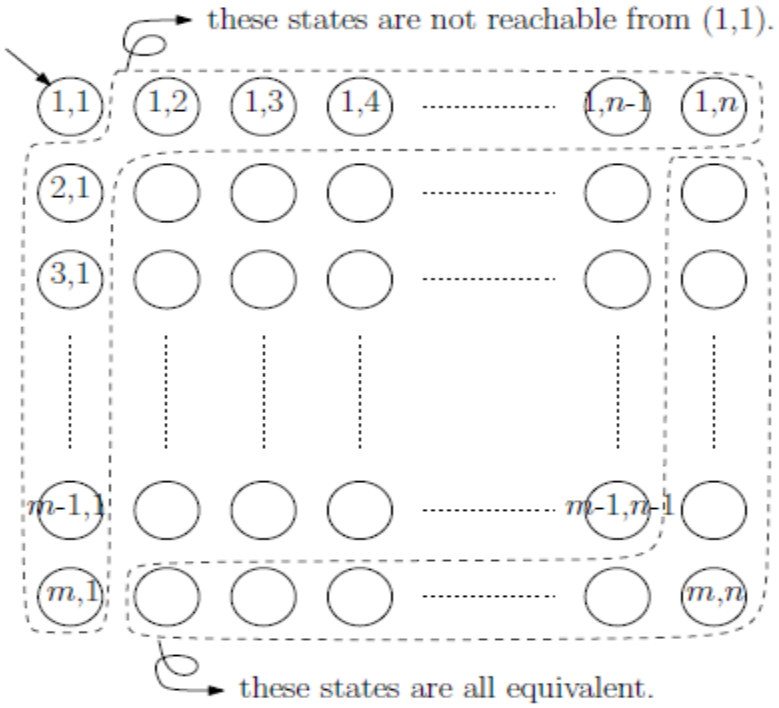

Fig 2.8: Intersection of minimal DFA A and B [16]

Since the automaton M constructed is deterministic, therefore mn states are sufficient for the intersection of L(A) and L(B), where |A| = m and |B| = n, and mn is a tight bound for the intersection of two regular languages [33]. A unique number is assigned for each state from 1 to m in A and from 1 to n in B, where |A| = m and |B| = n. The m^{th} state and the n^{th} state are assumed to be the sink states in A and B, respectively. A DFA for A∩cB is obtained and is minimized by merging all equivalent states and removing unreachable states from the start state. A has the sink state, L(m,i) = Ø, for $1 \leq i \leq n$, therefore, all these states can be merged. All states (j, n), for $1 \leq j \leq m$, of $A \cap_c B$ are equivalent and, therefore, can be merged. Consider all states (1, i), for $1 < i \leq n$, of $A \cap_c B$. Since L(A) is

suffix-free, the start state of A has no in-transitions. It implies that (1, i) is not reachable from (1, 1) in A $\cap$c B and, therefore, these states are useless, a similar result for the the states (j, 1), for $1 < j \leq m$ can be established. Once A $\cap_c$ B is minimized the resulting minimal DFA has 2+(m-2)(n-2) = mn − 2(m + n) + 6 states.

2.3.4 Kleene star of suffix-free regular languages

State complexity of suffix-free regular languages for kleene star operation is calculated linearly in NFA case and exponentially in DFA case.

Theorem 2.23 [19] Given a suffix-free regular language L with NSC(L) = m, then the nondeterministic state complexity NSC(L^*) for L^* is m [14]. □

Theorem 2.24 Given a suffix-free regular language L with m state DFA, then the deterministic state complexity DSC(L^*) is m [16].

Proof: Let A = (Q, $\sum$, δ, s, F) be a minimal DFA for a suffix-free language and k = |Q|. Then, A has a sink state d $\in$ Q and for every string w $\in \sum^+$, $\delta(s, w^k)$ = d. Then an NFA A′ from A that accepts $L(A)^*$ is constructed and then determinized and minimized. Let d be the sink state of A. Now A′ = (Q′,$\sum$, δ', s′, F′) is computed as follows: Q′ = Q, δ'(p, a) = {(δ(p, a) if p $\notin$ F), (δ(p, a), δ(s, a) if p $\in$ F}) , s′ = s, F′ = F $\in$ {s′}. It is easy to verify that A′ accepts $L(A)^*$ from the construction. Note that the two sink states of A and A′ are the same and A′ is also non-returning. Now after applying the subset construction to A′, let A_D denote the resulting DFA from A′. The number of states in A_D is 2^m. Now there are 2^{m-1} states that contain d in A_D, which can be removed. Thus $2^m - 2^{m-1}$ states are left in the resulting DFA. Now a state q in A_D such that s′ $\in$ q and { s′} $\neq$ q is not reachable from { s′} because it is suffix-free language and is non-returning. Hence there are 2^{m-2} + 1 states that contain s` in $A_D$ except {s`} that are unreachable. Therefore $2^{m-1} - (2^{m-2} - 1)$ = 2^{m-2}+1 are sufficient for minimal DFA of $L(A)^*$.

2.3.5 Reversal of suffix-free regular languages

The calculation of reversal of suffix-free regular languages for NFA case is linear and that for DFA case is exponential. If a regular language L is accepted by an m-state

minimal DFA, then its reversal L^R is accepted by an m-state NFA. By the well-known subset argument, it can be conclude that the state complexity of L^R is at most 2^m.

Theorem 2.25 If L is a suffix-free regular language recognized by an NFA with m states, then $NSC(L^R) \leq m+1$. The bound m+1 can be reached by suffix-free languages over a three letter alphabet when $m \geq 4$ [14].

Proof: Given an m-state NFA A, Non deterministic state complexity of a language in general is m+1 [22]. For NFA construction for L^R, the transition directions of A are flipped, the start state is made as the final state and all final states are be made as the start states of A. Now the result is an NFA with multiple start states. Now a new start state is introduced and a λ-transition is done from the new start state to the original start states. Then, the λ -transition removal technique is applied [24], which does not change the number of states. Thus, m+1state are left with the NFA for L^R.

Theorem 2.26 Given an m-state suffix-free minimal DFA A over ∑, $2^{m-2} + 1$ [16] states are necessary and sufficient in the worst-case for the minimal DFA of $L(A)^R$, over three letter alphabet.

Proof: Let A= (Q, ∑, δ, F, s) be the minimal DFA, then A^R = (Q, ∑, δ^R, F, s) be the FA obtained by flipping all transition directions in A and m = |Q|. Since A is non-returning, A^R is non-exiting. Before the subset construction, remove all useless states from A^R. Let d be the sink state in A. Then, all states of F in A^R do not have any out-transitions to d in δ^R. m-1 states are left in A^R after removing useless states. Then subset construction is applied based on all subsets of states of a given FA, which results into 2^{m-1} subsets of states from A^R. Since A^R has $2^{m-2} - 1$ states containing q in A^R is unreachable from s′ in A′ such that $s \in q$ and $\{s\} \neq q$ and all unreachable states are removed such that the minimal and sufficient state complexity for the suffix-free regular language $L(A^R)$ is $2^{m-1} - (2^{m-1}+1) = 2^{m-2}+1$.

2.3.6 Complement of suffix-free Languages

For the complement operation of an m-state suffix-free DFA, it is easy to verify that m states are necessary and sufficient but in NFA [27] case exponentiation is done. 2^m states

are required in transforming an m-state NFA to a DFA. The complementation of an m-state DFA does not require additional states since it simply interchanges final states and non-final states.

Theorem 2.27 Given a suffix-free regular language L having an NFA with m number of states, NSC(L) $\leq 2^{m-1}+1$ for $|\sum| = 2$. There exists a suffix-free regular language L over a three letter alphabet such that NSC(L) = m and NSC(~L) $\geq 2^{m-1}-1$ [14]. □

Theorem 2.28 For an m state DFA for suffix-free regular language L, the minimal and sufficient state complexity of suffix free regular language DSC(~L) is m [16]. □

2.3.7 Conclusion

Based on the structural property that a suffix-free minimal DFA must be non-returning, the table contains the comparison of NFA and DFA state complexities for individual operations.

Table 2.3: State complexity of individual operations for suffix-free regular languages

Operations	Sufix-free DFAs	Sufix-free NFAs
$L_1 \cdot L_2$	$(m-1)2^{n-2} + 1$	$m + n - 1$
$L_1 \cup L_2$	$mn - (m + n) + 2$	$m + n - 1$
$L_1 \cap L_2$	$mn - 2(m + n) + 6$	$mn - 2(m + n) + 2$
L_1^*	$2^{m-2} + 1$	M
L_1^R	$2^{m-2} + 1$	$m + 1$
$\neg L_1$	m	$2^{m-1} \pm 1$

Chapter 3

Problem Statement

This chapter includes the problem statement followed by the motivation behind the thesis and state complexity of combined operations on suffix-free regular languages.

3.1 Problem Statement

State complexities of regular languages like prefix-free and suffix-free have been studied by various researchers from decades. State complexities of prefix-free regular languages over individual operations and combined operations have been studied. The state complexities of suffix free regular languages over individual operations have also been studied. This thesis covers the estimation state complexities of suffix-free regular languages for combined operations.

3.2 Objectives and Methodology

State complexities of operations of various forms of regular languages are based on:

1. Exact state complexity of combined operations.
2. Estimation and approximation of state complexity of combined operations.

In this thesis, we discuss exact state complexity of combined operations on suffix-free regular languages. For following operations state complexity are determined:

1. Combination of star and concatenation
2. Combination of star and union
3. Combination of star and intersection
4. Combination of complement and union
5. Combination of complement and intersection
6. Union of three languages

3.3 Motivation

Since codes are sets of strings over an alphabet, they are closely related to languages. A code is a language defining a class of codes of a corresponding subfamily of a language. If a language L is prefix-free, then its reversal L^R is suffix-free by definition. Yu et al. [33] studied the operational state complexity of general regular languages and examined the deterministic state complexity of combined operations [10] on regular languages. The state complexities of combined operations are different from the compositions of the state complexities of individual operations that form the combinations. The reason for this difference is that the result of the first operation is not among the worst cases of the second operation as shown in Fig 3.1

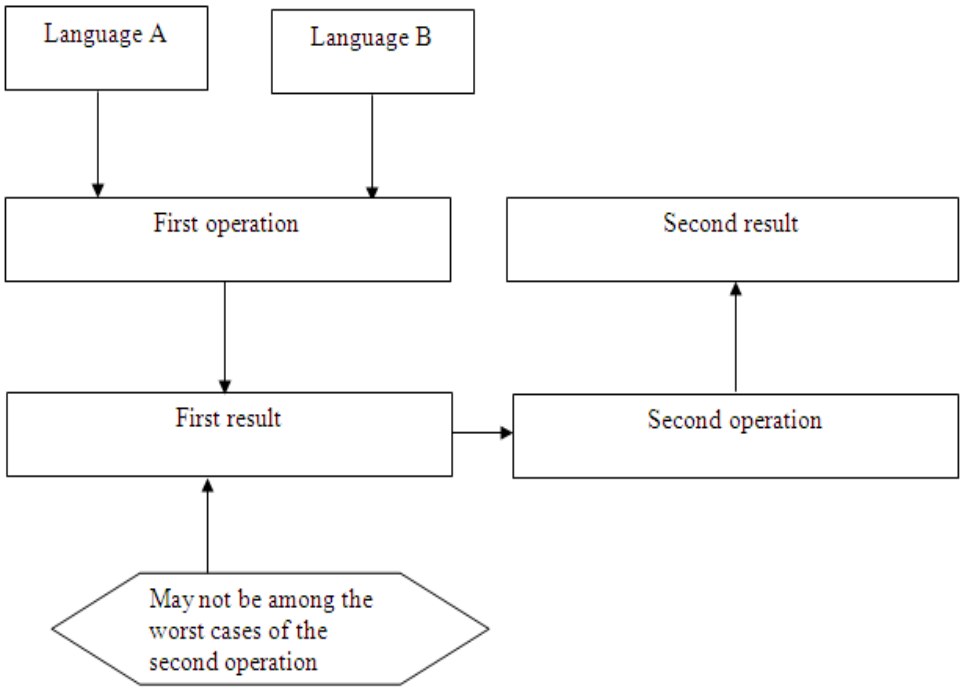

Fig 3.1: Reason for difference in state complexity [19]

Pighizzini and Shallit [12] investigated the deterministic state complexity of unary operations on general languages language. Han et al. [15] studied the deterministic state complexity of prefix-free regular languages and Han and Salomaa [16] looked into the deterministic state complexity of suffix-free regular languages. Y. Han, K. Salomaa [19] studied the deterministic state complexity of combined operations of prefix-free regular languages, so it becomes obvious that state complexity of suffix-free regular languages

for combined operations also need to be discussed. There are only a limited number of individual operations on regular languages. However, the number of combined operations on suffix-free regular languages is unlimited. Thus, it is important to obtain general results that cover not only single combined operations, but also infinite classes of combined operations. A good estimate of the state complexity of a combined operation can be used in many applications.

3.4 State complexities of combined operations for suffix-free regular languages

There seems to be no common method to compute the state complexities of combined operations because each combined operation has its own special features. Although the composition of individual state complexities would give an upper bound to the state complexity of the combined operation. The upper bound is usually too large to be useful. For example, for two regular languages L_1 and L_2 accepted by m-state and n-state deterministic finite automata, respectively, the state complexity of $(L(A) \cap L(B))^*$ is actually $2^{mn}-(2^{2m}+2^{2n})+6$, while the composition of the individual state complexities is $2^{nm-2(m+n)+6}$. So it appears that the state complexity of each combined operation has to be studied specifically.

Approximation of state complexity is helpful because if the exact state complexities have not been obtained, then the approximations with low ratio bounds can be obtained rather easily and they can be used for practical purposes in general. And if the exact state complexities have been proved, the approximations of those results with low ratio bounds can simplify the formulae of the complexities and make them easier to apply. Thus, approximation of state complexity is clearly a useful and important concept in the study of state complexity of combined operations.

Chapter 4

State Complexity of Combined Operations for Suffix-free Regular Languages

This chapter includes the analysis of state complexities of suffix-free regular languages for combined operations.

4.1 State Complexity of $L_1^* L_2$

For estimating the state complexity of $L_1^* L_2$, consider the following examples.

Example 1: DFA A = $(Q_1, \sum, \delta_1, 0, \{m-1\})$ accepting suffix-free regular language L(A) where $Q_1 = \{0, 1, \ldots, m-1\}$, the transitions are given by :

1. $\delta_1 (i, a) = (i + 1) \bmod m$, for $i \in \{Q_1 - (m-1)\}$
2. $\delta_1 (i, x) = i$, for $i \in Q_1$, $x \in \{b, c\}$
3. $\delta_1 (0, x) = m - 1$ where $x \in \{b, c\}$.

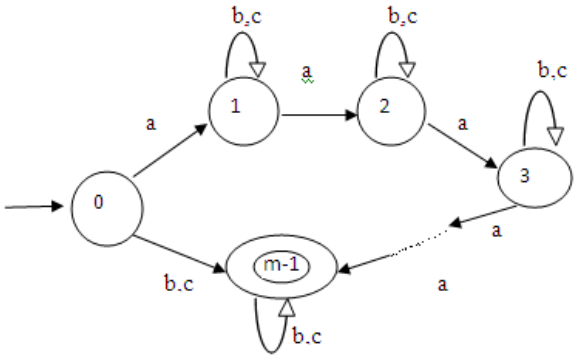

Fig 4.1: DFA for A

Example 2: DFA B = $(Q_2, \sum, \delta_2, 0, n-1)$ accepting suffix-free regular language L(B) where $Q_2 = \{0, 1, \ldots, n-1\}$, and transitions are given by

1. $\delta_2(i, a) = i$, for $i \in \{Q_2 - 0\}$
2. $\delta_2(i, b) = i + 1$, for $i \in Q_2, \delta_2(0, b) = n - 1$

3. $\delta_2(0, a) = 1$, $\delta_2(0, c) = n-1$
4. $\delta_2(i, c) = i + 1 \bmod n$, for $i \in \{1, \ldots, n-1\}$.

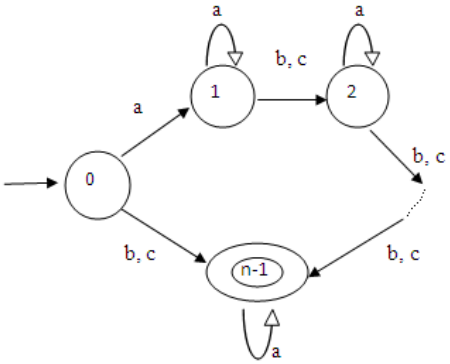

Fig 4.2: DFA for B

Lemma 4.1 For DFA A and B accepting the languages L(A) and L(B), when n=1, the state complexity of $L(A)^*L(B)$ in best caseis 1 for any $m \geq 1$ [19]. □

DFA B has one state and it is initial and final both, therefore B is complete. Which means that $L(A)^*L(B) = \emptyset$ or $\sum^*$ .Thus, $L(A)^*L(B)$ is always accepted by a one-state DFA.

Next, consider the case in lemma 4.2 when DFA A has 2 states, one final state and one initial state. In such a case, $L(A)^*$ is also accepted by A, and hence the state complexity of $L(A)^*L(B)$ is equal to that of L(A)L(B).

Lemma 4.2 For any DFA A of size m = 2 and any DFA B of size $n \geq 2$, the state complexity of $L(A)^*L(B)$ is equal to $2n - 1$ [19]. □

Theorem 4.1 Given two suffix-free regular languages L_1 and L_2 accepted by $m > 2$ and $n > 2$ state DFAs, then $DSC(L_1^* L_2)$ is $(2^{m-2}+1)n - k2^n - 1)$ for $k = |F_1|$.

Proof: Now $L(M) = L(A)^*L(B)$ can be constructed by concatenating the final states of DFA A with the initial state of DFA B as shown in the Fig 4.3. Q is a set of pairs such that the first component of each pair is a state from 2^{Q_1} and the second component is a subset of Q_2. Let $M = (Q, \Sigma, \delta, s, F)$ be the minimal DFA accepting $L(A)^*L(B)$ where, the set of states $Q = 2^{Q_1} \times 2^{Q_2 \setminus F_1} \times 2^{Q_2 \setminus \{s_2\}}$, $s = (s_1, \{\emptyset\})$, $F = \{(q, T) \in Q \mid T \cap F_2 \neq \emptyset\}$ and $\delta((q, T), a) = (q', T')$ for $a \in \sum$, where $q' = \delta_1(q, a)$ and $T' = \{ \delta_2(T, a) \cup \{s_2\}$ if $q' \in F_1$, otherwise $T' = \delta_2(T, a) \}$.

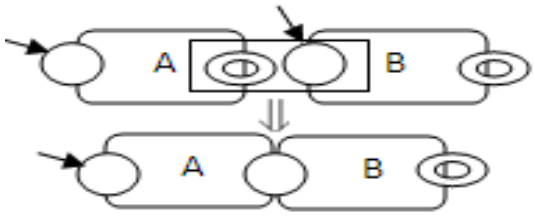

Fig 4.3: Concatenation representation of two DFA A and B

Now Q does not have pairs in which first component is a final state of DFA accepting $L(A^*)$ or whose second component contain s_2. Thus, the number of sufficient and minimal states in M where $k = |F_1|$ is $(2^{m-2}+1)n - k2^n-1)$ by construction.

4.2 State Complexity of $L_1^* \cup L_2$

Given suffix-free regular languages L_1 and L_2 accepted by m-state and n-state DFAs A and B respectively. The state complexity for the operation of $L_1^* \cup L_2$ is computed using the Cartesian product of the states of their accepting DFAs.

Theorem 4.2 The deterministic state complexity for the suffix-free regular languages L_1 and L_2 accepted by the m state and n state DFA for the operation $L_1^* \cup L_2$ is $(n+1)2^{m-2} - 2^{m-1} - n+1$.

Proof: DFA accepting the language $L(M)= L(A)^* \cup L(B)$, can be constructed by applying Cartesian product on the states of DFA constructed for accepting the language $L(A)^*$ and L(B) as shown in fig 4.4. For two suffix-free minimal DFAs $A= (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $B=(Q_2, \Sigma, \delta_2, s_2, F_2)$, FA $M = (2^{Q}_1 \times Q_2, \Sigma, \delta, (s1, s2), F)$ is constructed, where for all $p \in 2^{Q}_1$, and $q \in Q_2$, $a \in \Sigma$, $\delta_1((p, q), a)=(\delta_1(p, a), \delta_2(q, a))$ and $F=(F_1 \times Q_2) \cup (2^{Q}_1 \times F_2)$. The right language $L_{m,n}$ formed from minimal m state and n state DFA A and B, states (i, j) and $L_{(n,j)} = L^*_n \cup L_j \neq \emptyset$ are such that, all states (m, i) and (j, n) for $1 \leq i \leq n$ and $1 \leq j \leq 2^m$, in $(A \cup_c B)$ are not necessarily equivalent. Thus, these states cannot be merged. On the other hand, all states (1, i) and (j, 1), are useless because of the fact that suffix-free languages are non-returning.

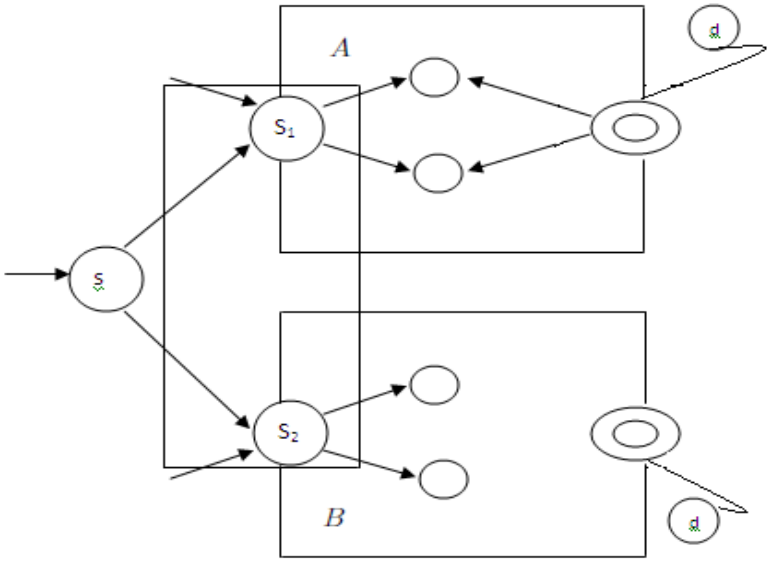

Fig 4.4: DFA for $L(A)^* \cup L(B)$

Therefore the resulting DFA can be minimized by removing the unreachable states, which leads to $(n+1)2^{m-2} - 2^{m-1} - n + 1$ minimal and sufficient state complexity in the worst-case for the DFA of $(L(A)^* \cup L(B))$.

4.3 State Complexity of $L_1^* \cap L_2$

Given suffix-free regular languages L_1 and L_2 accepted by m-state and n-state DFAs A and B respectively. The state complexity for the operation of intersection of the languages $L(A)^*$ and L(B) is computed using the Cartesian product of their states as shown in Fig 4.5.

Theorem 4.3 The deterministic state complexity of m state and n state DFA accepting suffix-free regular language for the operation $L_1^* \cap L_2$ is $(n(2^{m-2} - 2) - 2^{m-1} + 6)$ minimal and sufficient, where $|\Sigma| \geq 3$.

Proof: Now, let m be the no of states for the complete DFA $A = (Q_1, \Sigma, \delta_1, s_1, F_1)$ accepting the language L(A), and n be the complete DFA $B = (Q_2, \Sigma, \delta_2, s_2, F_2)$ accepting the language L(B). Let $M = (2^{Q}{}_1 \times Q_2, \Sigma, \delta, (s1, s2), F)$ be constructed for $L(M) = (L(A)^* \cap L(B))$ where for all $p \in 2^{Q}{}_1$, and $q \in Q_2$ and $a \in \Sigma$, $\delta_1((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ and $F = (F_1 \times Q_2) \cap (2^{Q}{}_1, \times F_2)$.

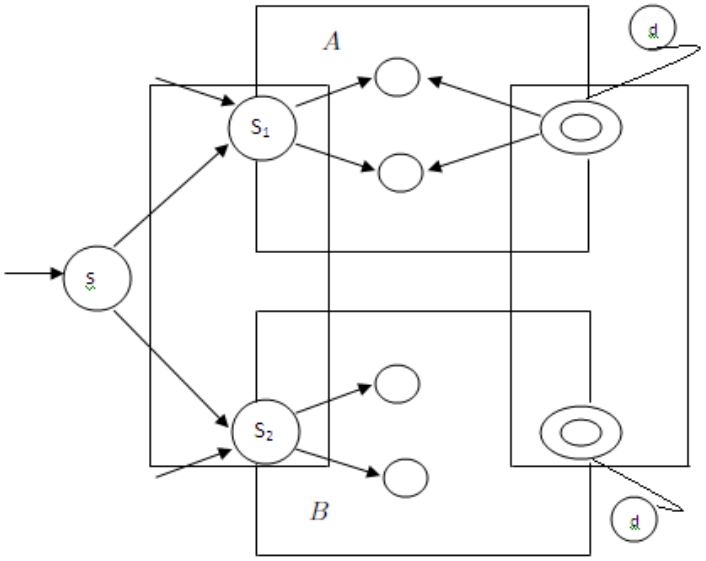

Fig 4.5: DFA for $L(A)^* \cap L(B)$

Since the automaton M constructed is deterministic, we assign a unique number for each state from 1 to 2^m and from 1 to n. A minimized DFA for $L(A)^* \cap L(B)$ is obtained by merging all equivalent states and removing unreachable states from the start state. For a state (i, j) in L(M), the right language $L_{(i,j)}$ of (i, j) is the intersection of L_i in A and L_j in B. Since a suffix-free DFA A has the sink state L(m,i) = Ø, for $1 \leq i \leq n$, where 1 sink state of DFA A is associated with n states of DFA B and one sink state of DFA B is associated with 2^m states in DFA A. Therefore, all these states are equivalent and can be merged together. Similarly, all states (j, n) for $1 \leq j \leq 2^m$ are equivalent.

Next in order to find out all the unreachable states, consider all states (1, i) for $1 < i \leq n$ and (j, 1), for $1 < j \leq 2^m$ states are useless resulting in the simplification calculation equal to $2+(2^{m-2}-2)(n-3)+ (2^{m-2}-2)+(n-3)$.

After removing the useless and unreachable states from $n2^m$, the optimized necessary and sufficient worst-case state complexities for the minimal DFA of $(L(A)^* \cap L(B))$, is $(n(2^{m-2} – 2) – 2^{m-1} + 6)$, where $|\Sigma| \geq 3$.

4.4 State complexity of $\sim(L_1 \cup L_2)$

For applying complement operation on some DFA accepting the suffix-free regular language, final states and non final states are interchanged.

Theorem 4.4 The deterministic state complexity of m state and n state DFA accepting suffix-free regular language for the combined operation $\sim(L_1 \cup L_2)$ is sufficiently mn+1 but need not necessarily be minimum.

Proof: Let two suffix-free languages accepted by the m state minimal DFAs $A = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and n state minimal DFA $B = (Q_2, \Sigma, \delta_2, s_2, F_2)$. Let DFA $M = (\sim(Q_1 \times Q_2), \Sigma, \delta, (s_1, s_2), F)$ be constructed accepting the language $\sim (L(A) \cup L(B))$, where for all $p \in Q_1$ and $q \in Q_2$ and $a \in \Sigma$, $\delta_1((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ and the final state $F= \sim(F_1 \times Q_2) \cup \sim(Q_1 \times F_2)$ with the worst case upper bound state complexity is equal to mn.

It is important to note that for doing complementation, final and non final states need to be interchanged. Hence those states which are previously equivalent might not be equivalent anymore and those states which are unreachable might be reachable otherwise. So, the exact complexity cannot be determined for this operation but an upper bound of mn+1 states can be achieved which might be sufficient but need not necessarily be minimum.

4.5 State complexity of $\sim(L_1 \cap L_2)$

For doing complement operation on intersection operation, Cartesian product is applied on the DFAs accepting suffix-free languages as shown in Fig 4.6 and then final and non final states are interchanged.

Theorem 4.5 The state complexity $\sim(L_1 \cap L_2)$ will be an upper bound of $mn - 2(m + n) + 6$ which is sufficient but not minimal for the suffix-free regular languages accepted by an m state and an n state DFAs.

Proof: Given for two suffix-free minimal DFAs $A = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $B = (Q_2, \Sigma, \delta_2, s_2, F_2)$, let $M = (\sim (Q_1 \times Q_2), \Sigma, \delta, (s_1, s_2), F)$ be constructed where for all $p \in Q_1$ and $q \in Q_2$ and $a \in \Sigma$, $\delta_1((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ and $F= \sim(F_1 \times Q_2) \cup \sim(Q_1 \times F_2)$. Then, $L(M)= \sim(L(A) \cap L(B))$ and for a state (i, j) in L(M), the right language $L_{(i,j)}$ of (i, j) is the intersection of L_i in A and L_j in B.

Since a suffix-free DFA A has the sink state, L(d, i) = Ø, for 1≤ i ≤ n, where 1 sink state of DFA A is associated with n states of DFA B and one sink state of DFA B is associated with m states in DFA A. Therefore, all these states can be merged. Similarly, all states (j, d), for 1 ≤ j ≤ m, of ~(L(A) $\cap_c$ L(B)) are equivalent and, therefore, can be merged. Now for suffix-free minimal DFAs A and B, all states (m, i) for 1 ≤ i ≤ n and all states (j, n) for 1 ≤ j ≤ m of ~(L(A) $\cap_c$ L(B) are equivalent.

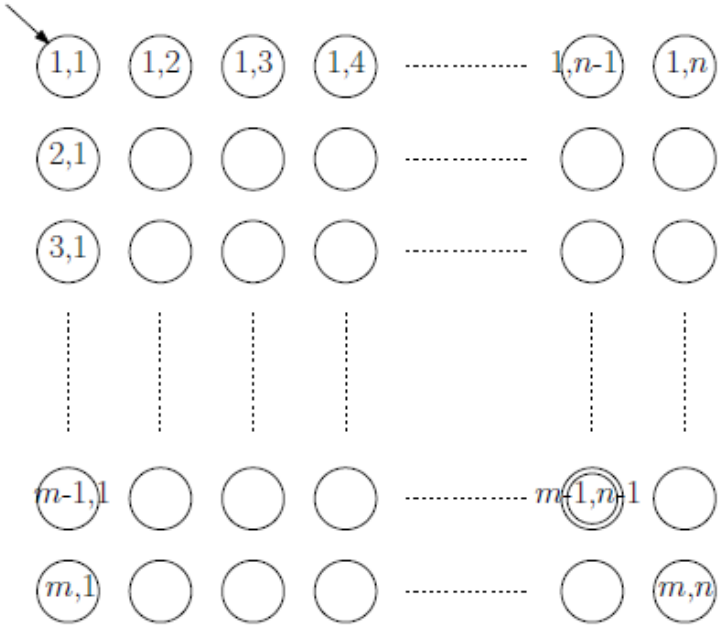

Fig 4.6: Cartesian product of m state and n state FA

Like in intersection operation all states in the first row and in the first column are unreachable from (1, 1). Now (1, i) for 1< i ≤ n, is not reachable from (1,1) in (A$\cap_c$B) and, therefore, these states are useless. A similar result for the states (j, 1), for 1< j ≤ m can be established. Now for doing complementation, we need to interchange final and non final states [4], but the number of states remain same because the final states are not equivalent states, so the resulting state complexity of ~(L(A) $\cap_c$ L(B) will be an upper bound of m – (m + n) + 2 which is sufficient but not minimal.

4.6 State complexity of $(L_1 \cup L_2)^*$

Given suffix-free regular languages L_1 and L_2 accepted by m-state and n-state DFAs A and B respectively. For estimating the star of union, the exponentiation on Cartesian

product of states is done on the DFAs accepting their corresponding languages as shown in Fig 4.7.

Theorem 4.6 The worst-case state complexity of the star-of-union of an m state and an n-state, m,n ≥ 3, prefix-free regular languages is precisely $2^{mn} - (2^m + 2^n) + 2$.

Proof: Given two DFA's A= $(Q_1, \Sigma, \delta_1, s_1, F_1)$ and B= $(Q_2, \Sigma, \delta_2, s_2, F_2)$ accepting two suffix-free regular languages L(A) and L(B). Let M=$(2^{Q_1 \times Q_2}, \Sigma, \delta, (s_1, s_2)$, F) be constructed for $(L(A) \cup L(B))^*$ where for all $p \in Q_1$ and $q \in Q_2$ and $a \in \Sigma$, $\delta_1((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ and $F=(F_1 \times Q_2) \cup (Q_1 \times F_2)$.

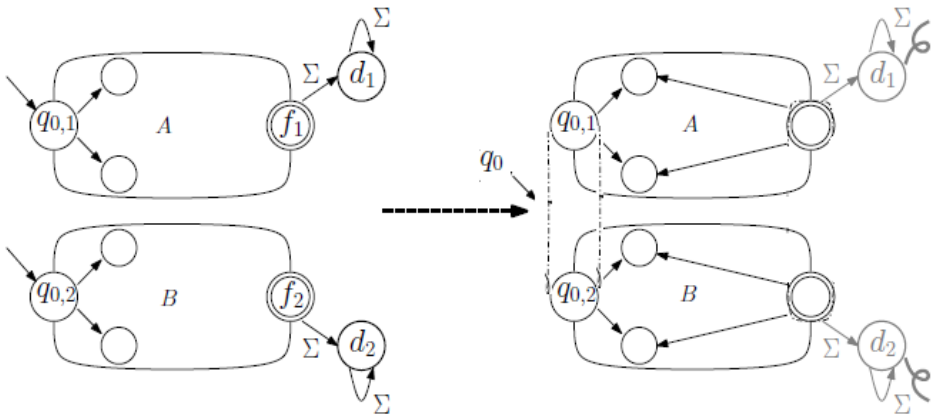

Fig 4.7 Kleene star of union of two suffix-free languages

Lemma 4.3 Given two m > 2, n > 2 state minimal DFA, $2^m + 2^n - 2$ state are useless when creating the resulting DFA of $(A \cup B)^*$.

Proof: For two DFAs A and B accepting suffix-free regular languages L(A) and L(B). Some equivalent states are produced when kleene star on the subset construction is done. Now these unreachable states need to be discovered and removed. Now, all set of states that contain start state from DFA A and DFA B are those equivalent states $2^m + 2^n - 2$ that needs to be removed from the resulting DFA for $L(M)^*$. We know that 2^{mn} is the upper bound for kleene star on subset construction when individual complexities of both the operations on suffix-free reagular are considered. So, after removing these unreachable states from the constituting complexity, $2^{mn} - (2^m + 2^n) + 2$ number of states are left.

4.7 State complexity of $(L_1 \cap L_2)^*$

For calculating the state complexity of star of intersection operator the same order is followed that have been followed to find out the state complexity of star of union operator as shown in the fig 4.8.

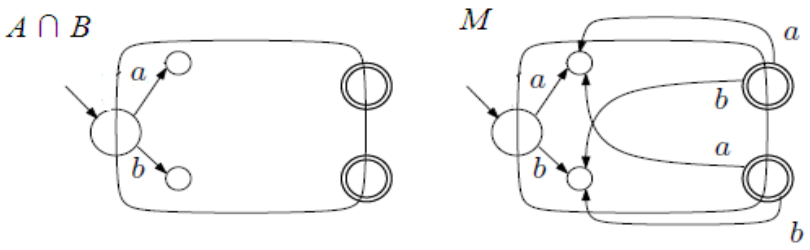

Fig 4.8 Kleene star of DFA M

Theorem 4.7 The worst-case state complexity of the star-of-intersection of an m state and an n-state, m, $n \geq 3$, prefix-free regular languages is precisely $2^{mn} - (2^{2m} + 2^{2n}) + 6$.

Proof: Now given two DFA's A= $(Q_1, \Sigma, \delta_1, s_1, F_1)$ and B=$(Q_2, \Sigma, \delta_2, s_2, F_2)$. First subset construction is done which results into state complexity of mn no of states. Let M=$(2^{Q1\times Q2}, \sum, \delta, s, F)$, where $p \in Q_1$ and $q \in Q_2$ and a, $b \in \Sigma$, $d \in Q$, k=|Q| $\delta((p, q), a) = (\delta_1(p, a) \cap \delta_2(q, a))$, $\delta((p, q), b) = (\delta_1(p, b) \cap \delta_2(q, b))$, and $F = (F_1 \times Q_2) \cap (Q_1 \times F_2)$ and for every string $x \in \sum^{+,} \delta(s,x^k)=d$.

Now exponentiation is done to compute L(M) which results into 2^{mn} no of states. But since DFA is used for the construction of L(M) , so there are no equivalent states, but unreachable states are generated which need to be removed. Consider all states that contain start state except the start state itself. Since suffix-free language is non returning, the no of states that contain s is $2^{2m} + 2^{2n} - 6$. And after removing these states the resulting complexity remains $2^{mn} - (2^{2m} + 2^{2n}) + 6$.

4.8 State complexity of $(L_1 \cup L_2 \cup L_3)$

Now the union of three suffix-free regular languages is investigated by applying Cartesian product of states of DFA's A, B and C accepting L(A) and L(B) and L(C).

Theorem 4.6 The worst-case state complexity of the operation ($L_1 \cup L_2 \cup L_3$) of an m state, an n-state and p state, m, n, $p \geq 3$, prefix-free regular languages is precisely $2^{mn} - (2^m + 2^n) + 2$.

Proof: Given three suffix-free minimal DFAs $A = (Q_1, \sum, \delta_1, s_1, F_1)$ and $B = (Q_2, \sum, \delta_2, s_2, F_2)$, and $C = (Q_3, \sum, \delta_3, s_3, F_3)$, L(M) = (L(A) ∪ L(B) ∪ L(C) is constructed where DFA $M = ((Q_1 \times Q_2 \times Q_3), \sum, \delta, (s_1, s_2, s_3), F)$, for all $t \in Q_1$, $q \in Q_2$, $r \in Q_3$ and $a \in \sum$, $\delta((t, q, r), a) = (\delta(t, a), \delta(q, a), \delta(r, a))$ and $F = (F_1 \times Q_2 \times Q_3) \cup (Q_1 \times F_2 \times Q_3) \cup (Q_1 \times Q_2 \times F_3)$.

Now consider the right language of a state (i, j, k) in $L(A \cup_c B \cup_c C)$. For a state (i, j, k), where $i \geq 3$, $j \geq 3$ and $k \geq 3$ the right language L(i, j, k) of (i, j, k) is the union of L_i in A and L_j in B and L_k in C. A unique number for each state from 1 to m in A, from 1 to n in B and from 1 to p in C is assigned, where |A| = m, |B| = n, |C| = p. Assume that the m^{th}, n^{th} and p^{th} states are the sink states in A, B and C respectively. Now all states (1, j, k) and (i, 1, k) and (i, j, 1), for $1 < i \leq m$ and $1< j \leq n$, $1< k \leq p$ are useless since L(A), L(B) and L(C) are suffix-free and has no in transitions. Therefore, useless or unreachable states of $(A \cup_c B \cup_c C)$ needs to determined and removed. Now (mnp) is the tight bound for the union of (L(A), L(B), L(C)), and it is concluded that ((n-1) + (p-1)+ (m-1) are the useless or unreachable states since there cannot be any in-transition to the starting state of suffix-free regular languages from any intermediate state, so now (mno – (m (n – 1) + n(p – 1) + p(m – 1) + 1) state complexity for L(M) is left, which is minimum and sufficient.

Chapter 5
Conclusions and Future Work

This thesis work describes the exact state complexities of the 8 combined operations listed in Table 5.1. The state complexities of most of these combined operations are smaller than the compositions of the state complexities of individual operations that form the combinations.

5.1 Conclusions

The state complexity of an operation for regular languages is the number of states that are necessary and sufficient for the minimal DFA that accepts the language obtained from the operation. In this paper, some recent results of the study of state complexity are summarized.

Table 5.1: State complexity of combined operations for suffix-free regular languages

Single operation	State complexities	Combined operations	State complexities
$L_1 \cup L_2$	$mn - (m + n) +$	$L_1^* \cup L_2$	$(n+1)2^{m-2} - 2^{m-1} - n + 1$
$L_1 \cap L_2$	$mn - 2(m + n)$	$L_1^* \cap L_2$	$(n(2^{m-2}-2) - 2^{m-1}+6)$
$L_1 L_2$	$(m -1)2^{n-2} + 1$	$L_1^* L_2$	$(2^{m-2}+1)n - k2^n - 1$
$\sim L$	m	$\sim(L_1 \cup L_2)$	mn+1
L^*	$2^{m-2} +1$	$\sim (L_1 \cap L_2)$	mn-(m+n)+2
-	-	$(L_1 \cup L_2)^*$	$2^{mn}-(2^m+2^n)+2$
-	-	$(L_1 \cap L_2)^*$	$2^{mn}-(2^{2m}+2^{2n})+6$
-	-	$(L_1 \cup L_2 \cup L_3)$	(mnp-(m(n-1)+n(p-1)+p(m-1)+1)

5.2 Future Research

Based on the structural property that a suffix-free minimal DFA must be non-returning, we have combined the operations like Kleene star with union and intersection, complementation with concatenation, intersection and union and union combined with union. There are still many interesting combined operations that have not yet been studied and can be studied in future. The compositions may not necessarily be restricted to two operations.

References

[1] Wikipedia, Automata Theory: “http://en.wikipedia.org/wiki/Automata_theory”.

[2] Wikipedia, Chomsky Hierarch:“http://en.wikipedia.org/wiki/Chomsky_hierarchy”

[3] Wikipedia, Nondeterministic finite automaton: “http://en.wikipedia.org/wiki /Nondeterministic_finite_automaton”

[4] Wikipedia, Deterministic finite automaton: “http://en.wikipedia.org/wiki/ Deterministic_finite_automaton”

[5] Brzozowski. J: Quotient complexity of regular languages. Descriptional Complexity of Formal Systems 11th Workshop (DCFS 2009), 17–28, Magdeburg Germany, 2009.

[6] Calude. C, Salomaa. K and Roblot.T: Finite state complexity and the size of transducers. Descriptional Complexity of Formal Systems 12th Workshop (DCFS 2010), 50–61, Saskatoon, Canada, 2010.

[7] Cui. B, Gao.Y, Kari.L and Yu.S: State complexity of catenation combined with star and reversal. Descriptional Complexity of Formal Systems 12th Workshop (DCFS 2010), 74–85, Saskatoon, Canada, 2010.

[8] Daley.M, Domaratzki.M, Salomaa.K: State complexity of orthogonal catenation. Descriptional Complexity of Formal Systems 10th Workshop (DCFS 2008), 134–144 Charlottetown, Canada, 2008.

[9] Domaratzki.M, Salomaa.K: Lower bounds for the transition complexity of NFAs. 31st International Symposium on Mathematical Foundations of Computer Science (MFCS 2006), 315–326 Springer, LNCS 4162, 2007.

[10] Gao.Y: State complexity of combined operations on regular languages. Masters's Thesis University of Western Ontario, Canada, 2006.

[11] Giammarresi.D and Montalbano.R: Deterministic generalized automata. Theoretical Computer Science, 215:191–208, 1999.

[12] Glaister. I and Shallit. J: A lower bound technique for the size of nondeterministic finite automata. Information Processing Letters, 59(2):75–77, 1996.

[13] Gao. Y, Salomaa. K, S. Yu: The state complexity of two combined operations: Star of catenation and star of reversal. Fundamenta Informaticae 83(1-2), 75–89 (2008).

[14] Han.Yo.Sub and Salomaa. K: Non deterministic State Complexity of Suffix-free regular Languages. 12th International Workshop on Descriptional Complexity of Formal Systems (DCFS'10), 189--196, 2010.

[15] Han.Yo.Sub and Salomaa. K. and Wood: D. State complexity of prefix-free regular languages, Descriptional Complexity of Formal Systems DCFS'06, 165–176, 2006.

[16] Han.Yo.Sub and Salomaa. K: State complexity of basic operations on suffix-free regular languages. Descriptional Complexity of Formal Systems DCFS'07 501–512, Springer, Heidelberg (2007).

[17] Han.Yo.Sub and Salomaa. K: State complexity of basic operations on suffix-free regular languages. Theoretical Computer Science, 410(27-29), 2537–2548, 2009.

[18] Han.Yo.Sub and Salomaa. K. and Wood. D: Nondeterministic state complexity of basic operations for prefix-free regular languages. Fundamenta Informaticae, 90(1-2), 93–106, 2009.

[19] Han.Yo.Sub and Salomaa. K. and Yu. S: State complexity of combined operations for prefix-free regular languages. 3rd International Conference on Language and Automata Theory and Applications (LATA 2009), Springer-Verlag, LNCS 5457, 2009, 398–409.

[20] Han.Yo.Sub and Wood. D: The generalization of generalized automata: Expression automata. International Journal of Foundations of Computer Science, 16(3), 499–510, 2005.

[21] Han. Yo-Sub, Wang. Y, and Wood. D: Prefix-free regular languages and pattern matching. Theoretical Computer Science, 389(1-2):307–317, 2007.

[22] Holzer. M and Kutrib. M: Unary language operations and their nondeterministic state complexity. Developments in Language Theory, DLT'02, Lecture Notes in Computer Science 2450, 162–172, 2002.

[23] Holzer. M and Kutrib. M: Nondeterministic descriptional complexity of regular languages. International Journal of Foundations of Computer Science, 14(6), 1087–1102, 2003.

[24] Hopcroft. J and Ullman. J: Introduction to Automata Theory. Languages, and Computation, Addison-Wesley, Reading, MA, 2 edition, 1979.

[25] Hopcroft. J, Motwani. R and Ullman. J. Introduction to Automata theory, Languages, and computation, Pearson Education Inc, ISBN 0-201-44124-, Addison-Wesley, 2001

[26] Jir´asek. J, Jir´askov´a. G, and Szabari. A: State complexity of concatenation and complementation. International Journal of Foundations of Computer Science, 16(3), 511–529, 2005.

[27] Jir´askov´a. G, Okhotin. A: On the state complexity of star of union and star of intersection. Turku Center for Computer Science, TUCS Technical Report 825, 2007.

[28] Liu. G, Martin-Vide. C, Salomaa. A, Yu. S: State complexity of basic language operations combined with reversal. Information and Computation, 206, 1178–1186, 2008.

[29] Mohri. M: On some applications of finite-state automata theory to natural language processing, Natural Language Engineering, 2, 61–80, 1996.

[30] Pereira. M, Riley. M: Speech recognition by composition of weighted finite automata, Finite-State Language Processing, MIT Press, 1996, pp. 431–453, 1996.

[31] Salomaa. K, Yu. S On the state complexity of combined operations and their estimation, International Journal of Foundations of Computer Science 18, 683–698, 2007.

[32] Salomaa. A, Wood. D, and Yu. S: On the state complexity of reversals of regular languages. Theoretical Computer Science, 320(2-3), 315–329, 2004.

[33] Yu. S, Zhuang, Q, Salomaa. K. The state complexities of some basic operationson regular languages. Theoretical Computer Science 125(2), 315–328, 1994.

[34] Yu. S. State complexity of regular languages, Journal of Automata, Languages and Combinatorics, 6(2), 221–234, 2001.

List of Publications

ACCEPTED

Darvena Rohilla and Ajay Kumar. "State Complexity of combined operations on Suffix-free regular Languages ". International journal of recent development in engineering management and social sciences (IJRDEMSS), ISSN 2278-1528 June, 2012.

PUBLISHED

Darvena Rohilla and Ajay Kumar, "Base for state complexities of combined operations on suffix-free regular languages". International Conference On "Advanced Computing Technologies", Paper ID: ICACT/043 8th and 9th June, 2012.