On Variant of Regulated Grammar and Regulated Automaton

A Thesis

Submitted in fulfillment of the requirement

for the award of the degree of

DOCTOR OF PHILOSOPHY

IN

COMPUTER SCIENCE AND ENGINEERING

by
Nidhi Kalra
(Registration No. 901403005)

Under the guidance of
Dr. Ajay Kumar
(Asst. Professor, CSED)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

August 2017

DEDICATED WITH EXTREME AFFECTION AND GRATITUDE TO

My *parents* Mr. Ramesh Kalra and Mrs. Urmila Kalra,

My *brother* Paras Kalra, and

My *supervisor* Dr. Ajay Kumar

CERTIFICATE

I hereby certify that work which is being presented in the thesis entitled ***"On Variant of Regulated Grammar and Regulated Automaton"*** in fulfillment of the requirements of **DOCTOR OF PHILOSOPHY** submitted in the **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**, THAPAR UNIVERSITY, PATIALA is an authentic record of my own work carried out under the supervision of Dr. Ajay Kumar and refers work of other researchers which are duly listed in reference section.

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

Nidhi Kalra

(Nidhi Kalra)

Registration No. 901403005

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

A Kumar

(Dr. Ajay Kumar)

Supervisor and Assistant Professor

Computer Science and Engineering Department

Thapar University, Patiala, 147004, India

ACKNOWLEDGEMENT

First and foremost, I would like to thank Almighty God, the most gracious and merciful, for providing me this opportunity and granting me the capability to proceed successfully.

I take this opportunity to express my profound gratitude and deep regards to my respected Supervisor, *Dr. Ajay Kumar, Assistant Professor, Computer Science and Engineering Department, Thapar University, Patiala* for his invaluable guidance, support, and encouragement throughout the course of this thesis. The blessing, help, and guidance given by him shall carry me a long way in the journey of life on which I am about to embark. I feel very lucky to get a lifetime opportunity to work under his knowledgeable supervision. It has been my great pleasure and honor to have worked with him.

I am very grateful to Dr. Alexander Meduna, whose research papers and Regulated Grammar and Automata book inspire me to work carry out this work.

I also express my gratitude to the Doctoral committee comprising of Dr. Maninder Singh (Head, CSED), Dr. A.K. Verma (Associate Professor, CSED), Dr. Parteek Kumar, (Associate Professor, CSED), and Dr. M.D. Singh (Assistant Professor, EIED) for monitoring the progress and providing valuable suggestions for the improvement of this Ph.D. work. I especially thank to respected Dr. Maninder Singh who told me one line, when I joined Thapar University, if you really work hard and give your input, Ph.D. is not at all tough, it is a game of 7-8 months. His one line always inspires and encourages me to do work. I am very thankful to Dean of Research and Sponsored Projects Dr. O.P. Pandey for his valuable inputs and suggestions.

I would also take this opportunity to mention my deep gratitude towards Mrs. Sunita Garhwal, wife of my respected Supervisor, Dr. Ajay Kumar whose presence was always there whenever I needed in both my personal and professional life. Working with both of you was my great honor and I always felt like family support from both of you. I heartily thankful to both of you for supporting during entire duration of my Ph.D.

I would like to acknowledge the most important persons of my life – my parents. I owe my personal regards to them for their love, blessings, care, inspiration, encouragement and motivation which have enabled me to complete my task. My brother Mr. Paras Kalra have also supported me and encouraged me to carry out this work. In spite of being my younger brother, he always stands beside me at every point of my life. I would honestly

say my respected parents and my respected supervisor continuous support and motivation that ultimately made it possible for me. This task would not be possible, if they were not with me.

I am also indebted to my best friend cum Sister Amanjot Kaur not only for all her useful suggestions but also for being there to listen when I needed an ear.

Sincere, thanks to all my friends especially, Gaganpreet Kaur, Parmjit Kaur, Jahnavi, Shailja Sachdeva, Mandeep Singh for their kindness and moral support during my study. I also thank my Ph.D. friends Bharti Saneja, Vandana Bhatia, Sujata Singla, Meenu Singla, Natasha Singh and Navneet Kaur for their support and cooperation throughout the entire process. Thanks for the friendship and memories. I also thankful to my niece Areet whose smile always encouraged me to work.

I would like to thank the members of CSED Department, faculty members, staff members and my seniors who were always there at need of the hour and provided with the help and facilities, which I required for the completion of my thesis.

I gratefully acknowledge the funding received towards my Ph.D. from the Visvesvaraya Ph.D. Scheme fellowship by Ministry of Electronics & Information Technology, Government of India.

Finally, my thanks go to all the people who have supported me to complete the research work directly or indirectly.

(Nidhi Kalra)

ABSTRACT

Context-free grammar plays a vital role in the field of Automata Theory, but context-free grammars are not able to cover all linguistic phenomena. However, there are languages which cannot be represented by context-free grammars. Further, they have classified into context-sensitive and recursively enumerable languages. These languages suffer from a number of problems such as the emptiness problem is undecidable, and only exponential algorithms are known for the membership problem. To overcome the situation, the concept of regulated grammar and automata has been proposed. Regulated Grammar is the class of grammar which on the one hand only uses context-free production rules and on the same hand have a larger generative capacity by some additional mechanisms. Regulated automaton is the formal automaton counterpart of regulated grammars.

This dissertation is devoted to designing and development of a variant of regulated grammar or regulated automata and to apply their concept in RNA and DNA biomolecular structures. Motivated by the similarities between fuzzy finite automata and fuzzy pushdown automata, in this thesis we proposed a novel concept of fuzzy state grammars and fuzzy deep pushdown automata. This concept represents a natural extension of contemporary state grammar and deep pushdown automaton, making them more robust in terms of imprecision, errors, and uncertainty. It has been proved that we can construct fuzzy deep pushdown automata M_{fd} from fuzzy state grammars G_{fs} and vice-versa, and if the fuzzy deep pushdown automaton M_{fd} is constructed from the fuzzy state grammar G_{fs} using the above construction method, then $L(M_{fd}) = L(G_{fs})$.

Most of the regulated automata and the regulated transducer which exist in literature such as deep pushdown automata, parallel deep pushdown automata, deep pushdown transducer and parallel deep pushdown transducer are deterministic with respect to depth but lacks in strict determinism. This thesis also proposes the deterministic models of deep pushdown automata, deep pushdown transducer and their parallel versions named as parallel deep pushdown automata, and parallel deep pushdown transducer. The proposed deterministic models are strictly deterministic and deterministic with respect to depth. Furthermore, based on the deterministic version of these proposed automata and transducer, an infinite hierarchy of languages is explored. Further comparison between proposed deterministic models and existing non-deterministic models has been also done.

In the various existing approaches in the literature, Deoxyribonucleic Acid (DNA) and Ribonucleic Acid (RNA) structure sequences are represented using context-sensitive grammar or mildly context-sensitive grammar with higher time complexities. In this thesis, we have also represented DNA and RNA structure sequences using state grammar and deep pushdown automata. We have represented three commonly found structures in DNA and RNA using state grammar namely: Tandem Repeat, Inverted Repeat, and Pseudoknot. These biological sequences are depicted using formal model deep pushdown automata. Furthermore, we have designed top-down extended LL (1) parser for state grammar. Further, we have extended our work for various biomolecular structures of DNA and RNA using state grammar. Various structures represented are attenuator, extended pseudoknot, kissing hairpin, simple h-type, recursive pseudoknot, and three-knot.

Motivated by the prior work of recognition of imperfect strings in context-free grammar and context-sensitive grammar, this thesis also proposes an algorithm for error tolerance in recognition of faulty strings in a regulated grammar using fuzzy sets**.** The time complexity of the proposed algorithm is $O(|G^2|.|w|^3)$, where $|G|$ represents the number of production rules and $|w|$ is the length of the input string w. We applied the algorithm to regularly controlled and matrix grammar.

Keywords: Regulated grammar, Regulated automaton, State grammar, Deep pushdown automata, Fuzzy state grammar, Fuzzy deep pushdown automaton, Deoxyribonucleic acid, Ribonucleic acid, Deterministic deep pushdown automata, Deterministic deep pushdown transducer, Parallel deterministic deep pushdown automata, Parallel deterministic deep pushdown transducer, Regularly controlled grammar, Matrix grammar.

PREFACE

This manuscript presents my research work done at the Computer Science and Engineering Department, Thapar University, Patiala, India from July 2014 to March 2017. This manuscript mainly focuses on designing and development of a variant of regulated grammar or regulated automata and to apply their concept in RNA and DNA bio-molecular structures. The work presented in this manuscript is as follows:

1. Survey of existing Regulated Grammar and Regulated Automata.
2. Novel concept of Fuzzy State Grammar and Fuzzy Deep Pushdown Automata.
3. Deterministic models of Deep Pushdown Automata, Deep Pushdown Transducer and their parallel versions named as Parallel Deep Pushdown Automata and Parallel Deep Pushdown Transducer.
4. Application of state grammar and deep pushdown automata in biological sequences of Nucleic Acids.
5. Modeling of Biomolecular structures of DNA and RNA using state grammar.
6. An algorithm for error tolerance in recognition of faulty strings in a regulated grammar using fuzzy sets (*RFSRG*).

The list of the main research papers accepted and communicated that I have authored or co-authored during my Ph.D. thesis are as follows:

RESEARCH PUBLICATIONS

1. **N. Kalra and A. Kumar,** Fuzzy state grammar and fuzzy deep pushdown automaton, Journal of Intelligent & Fuzzy Systems, 31 (1), 249-258, 2016. [SCIE Indexed, Impact Factor-1.004]
2. **N. Kalra and A. Kumar,** State Grammar and Deep Pushdown Automata for Biological Sequences of Nucleic Acids, Current Bioinformatics, 11 (4), 470-479, 2016. [SCIE Indexed, Impact Factor-0.77]
3. **N. Kalra and A. Kumar**, Deterministic Deep Pushdown Transducer and its Parallel version, The Computer Journal, 1-11, 2017 [SCIE Indexed, Impact Factor-1.0]

4. **N. Kalra and A. Kumar**, A Survey of Formal Language Theory: Exploration of Various Sub-classes of Chomsky Hierarchy, **Revised Paper Submitted** in IET Software. [SCIE Indexed, Impact Factor-0.595]
5. **A. Kumar and N. Kalra**, Error tolerance for the recognition of faulty strings in a regulated grammar using fuzzy sets, **Communicated** in Lingua. [SCIE Indexed, Impact Factor-0.844]

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

CFG: Context-free grammar

CS: Context-sensitive grammar

PDA: Pushdown Automata

DPDA: Deep Pushdown Automata

PDPDA: Parallel Deep Pushdown Automata

DPDT: Deep Pushdown Transducer

PDPDT: Parallel Deep Pushdown Transducer

DDPDA: Deterministic Deep Pushdown Automaton

DDPDT: Deterministic Deep Pushdown Transducer

PDDPDA: Parallel Deterministic Deep Pushdown Automaton

DPDPDT: Parallel Deterministic Deep Pushdown Transducer

DNA: Deoxyribonucleic Acid

RNA: Ribonucleic Acid

LIST OF FIGURES

LIST OF TABLES

Chapter-1

INTRODUCTION

This chapter aims to introduce the preliminary concepts, basic notations, motivation of the thesis, and organization of the thesis.

1.1 MOTIVATION

Automata theory is one of the protracted established fields of Computer Science. Over the last five decades, it has been evolved in several different directions and levels. The development and exploration of new mathematical models such as regulated grammar [1-3], regulated automaton [3], preautomaton and protoautomaton [4], and their applications stimulated in a variety of deep mathematical theories. Automata theory has a number of applications in the field of neurosciences, biology, linguistics, tomography, cognitive sciences, network security, hardware and software verification, syntax analysis, foundations of XML, and much more. A context-free grammar (CFG) plays a vital role in the field of Automata Theory [5].

CFG's are not able to cover all linguistic phenomena. However, there are languages which cannot be represented by CFG's. Further, they have been classified into context-sensitive (CS) and recursively enumerable languages. These languages suffer from problems such as the emptiness problem is undecidable, and only exponential algorithms are known for the membership problem. Moreover, concepts such as derivation tree, which is an important tool for the analysis of context-free and natural languages, cannot be transformed to context-sensitive and recursive enumerable grammars [1]. However, polynomial time membership algorithms exist for CFG's.

To overcome this situation, the concept of regulated grammar and automata has been proposed [1-3]. Therefore, we are interested in the classes of languages which on one hand only use context-free production rules with a sequential derivation process and on the other hand have a larger generative capacity by some additional mechanisms. Over its history, this theory had modified the CFG's in many ways, including various regulated versions of these grammars. Ginsburg et al. [6] and Cremers et al. [7] proposed regularly controlled and matrix grammar, respectively. Kasai [8] introduced the concept of state grammar. Kuppusamy et al. [9] proposed various variants of regulated grammars such as

maximum depth-first grammars, maximum lengthwise depth-first grammars and absorbing right context grammars. A number of regulated automata have been proposed such as Deep Pushdown Automata (DPDA) [10], Parallel Deep Pushdown Automata (PDPDA) [11], Absolutely Unlimited Deep Pushdown Automata [12], and Jumping Automata [3].

In regulated languages, the input string is either accepted or rejected [13-14], but there is a chance that the user generates some error in the string. Hence, there is a need to introduce a mechanism by which string is not totally accepted or rejected, rather belong to the language with some certainty. It leads to the introduction of the concept of fuzziness in regulated languages.

Asveld [15] proposed fuzzy context-free K-grammar to represent erroneous sentences. Asveld [16] proposed a modified Cocke-Younger-Kasami (CYK) algorithm to recognize fuzzy context-free grammar. Jun et al. [17] introduced the variant of fuzzy sets, named as bipolar fuzzy sets, in which membership degree lies between [-1 to 1] rather than [0-1]. Jacob et al. [18] defined the concept of fuzzy entire sequence space.

Bucurescu et al. [19] proposed the concept of fuzzy pushdown automata and B-fuzzy pushdown automata as an extension to Pushdown Automata (PDA). In fuzzy pushdown automata, weights are assigned to the interval [0-1], and it accepts context-free language. In B-fuzzy pushdown automata, weights are assigned using Boolean algebra, and it accepts context-sensitive language. Moraga [20] defined the relationship between fuzzy CFG and fuzzy PDA, calculating the degree of membership based on weights as determined by a monoid. Hopcroft et al. [21] demonstrate the equivalence between PDA and CFG. Kakoty et al. [22] proposed a fuzzy logic model to evaluate expertise level of learner in an e-learning environment. Arora et al. [23] proposed an approach which converts state chart diagram to finite state automata. Further, regular grammars have been generated from finite state automata.

Motivated by the similarities between fuzzy finite automata [24] and fuzzy pushdown automata [20], there arises the need to investigate a novel fuzzy regulated grammar and fuzzy regulated automata concept. We have proposed a novel concept of fuzzy state grammars and fuzzy deep pushdown automata. This concept represents a natural extension of contemporary state grammar [8] and deep pushdown automaton [10], making them more robust regarding imprecision, errors, and uncertainty. It has been

proved that we can construct fuzzy deep pushdown automata from fuzzy state grammars and vice-versa. Furthermore, it has been proved that if the fuzzy deep pushdown automaton M_{fd} is constructed from fuzzy state grammar G_{fs}, then $L(M_{fd}) = L(G_{fs})$. In other words, for any string $\alpha \in \Sigma^*$, $\mu(\alpha; \alpha \in L(G_{fs})) = \mu(\alpha; \alpha \in L(M_{fd}))$, where μ denotes the membership of a string.

Literature survey reveals that the biological structure sequences have highly expressive language, and it highly correlates with the linguistic theory. Chomsky grammar systems are found to ideal for representing the interactions of nucleotides as there is a similarity between formal languages and biomolecules language [25]. Deoxyribonucleic Acid (DNA) and Ribonucleic Acid (RNA) structure sequences contain duplicate and crossing dependencies; hence these structures cannot be represented using CFG's.

Sung [26] used context-sensitive grammar to represent various forms of RNA secondary structures. Rivas et al. [27] used cross interaction grammar to represent RNA secondary structures, including pseudoknots. Searls [28] represented the structure of nucleic acids using indexed grammar. Searls [29] also proposed a string variable grammar for the representation of DNA. Uemura et al. [30] described a subclass of tree adjoining grammars for the modeling and prediction of RNA structures. Cai et al. [25] represented RNA pseudoknot structures using parallel communicating grammar systems. Mizoguchi et al. [31] used stochastic multiple context-free grammars to represent various classes of pseudoknots. Aligned sequences were computed using stochastic multiple context-free grammars based on probability estimators. Kuppusamy et al. [32] modeled DNA and RNA biomolecular structures of the intermolecular level, intramolecular level and at RNA secondary level using matrix-insertion deletion grammar. These sequences are represented using context-sensitive grammar or mildly context-sensitive grammar with higher time complexities.

Motivated from existing approaches, we have represented deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) structure sequences using state grammar and deep pushdown automata. We have represented three commonly found structures in DNA and RNA using state grammar namely: Tandem Repeat, Inverted Repeat, and Pseudoknot. These biological sequences are depicted using formal model deep pushdown automata. Furthermore, we have designed top-down extended LL (1) parser for state grammar. The major benefit of this approach is that DNA and RNA sequences can be parsed in linear

time $O(n)$, where n is the length of the string, which is a significant improvement over the existing approaches. There are various other biological structures such as extended pseudoknot, recursive pseudoknot, simple h-type, kissing hairpin, three-knot which includes crossing dependencies, which also requires power beyond context-free grammar. Further, we have extended our work for various biomolecular structures of DNA and RNA using state grammar. Various structures represented are attenuator, extended pseudoknot, kissing hairpin, simple h-type, recursive pseudoknot, and three-knot.

Most of the regulated automata and regulated transducer which exist in literature such as deep pushdown automata (DPDA) [10], parallel deep pushdown automata (PDPDA) [11], deep pushdown transducer (DPDT) [33] and parallel pushdown transducer (PDPDT) [33] are deterministic with respect to depth but lacks in strict determinism. We have designed a deterministic deep pushdown automaton (DDPDA), deterministic deep pushdown transducer (DDPDT) and their parallel versions named as a parallel deterministic deep pushdown automaton (PDDPDA) and a parallel deterministic deep pushdown transducer (PDDPDT), as an extension of DPDA, DPDT, PDPDA, and PDPDT, respectively. The proposed models obey the properties of both strict determinism and determinism with respect to depth. Furthermore, based on the deterministic version of these proposed automata and transducer, an infinite hierarchy of languages is explored.

Implementation of DDPDA, DDPDT and their parallel versions (PDDPDA and PDDPDT) require additional states as compared to non-strict DPDA, DPDT and their parallel versions (PDPDA and PDPDT), but it will require less number of moves with respect to the given input string. Hence, DDPDA, DDPDT and their parallel versions (PDDPDA and PDDPDT) 'arrive at the answer' faster in comparison to the equivalent DPDA, DPDT and their parallel versions (PDPDA and PDPDT).

Gao et al. [34] diagnose the motor fault using the concept of fuzzy logic. There is a need to diagnose the fault in linguistic theory. Fuzzy sets deal with the concept of vagueness [35]. Schinder et al. [36] described an approach for recognizing imperfect strings generated by the context-free grammar. Inui et al. [37] extended the work of Schinder et al. [36], describing an approach, for recognizing imperfect string generated by the context-sensitive grammar. Both Schinder et al. [36] and Inui et al. [37] used a number of fuzzy sets to determine the membership values of imperfect strings not belonging to the grammar.

Motivated by the prior work of recognition of imperfect strings in context-free grammar and context-sensitive grammar, an algorithm is designed for error tolerance in recognition of faulty strings in a regulated grammar using fuzzy sets**.** The time complexity of the proposed algorithm is $O(|G^2|.|w|^3)$, where $|G|$ represents the number of production rules and $|w|$ is the length of the input string *w.* We applied the algorithm to regularly controlled and matrix grammar.

1.2 PRELIMINARIES CONCEPT

In this section, we introduce some of the basic notations and definitions which are used throughout the thesis.

Let $\sum$ be an alphabet consisting of a finite set of terminals. G denotes the grammar. $L(G)$ denotes the language generated by the grammar G. $\mu_L(\alpha)$ denotes the membership grade of string α in the language L such that $0 \le \mu_L(\alpha) \le 1$. Operator _ is used to replace a symbol by any arbitrary symbol from the alphabet $\sum$. Let $\sum_D = \{g,c,a,t\}$ and $\sum_R = \{g,c,a,u\}$ be an alphabet of DNA and RNA, respectively. Here Purine is classified into adenine (a) and guanine (g), while pyrimidine is classified into cytosine (c) thymine (t) and uracil (u). Pairing occurs between purines and pyrimidine. The complement of a symbol u is represented by $\bar{u}$. Complement pairing in DNA is $\bar{a} = t,\ \bar{t} = a,\ \bar{g} = c,$ and $\bar{c} = g.$ Complement pairing commonly occurs in RNA is Watson-Crick pairing. In Watson-Crick base pairs, $\bar{a} = u,\ \bar{u} = a,\ \bar{g} = c,$ and $\bar{c} = g$. $\wedge$ denotes empty string. ϕ denotes empty set. $\sum^*$ represents a free monoid generated by the set $\sum$.

Nucleic acids, including DNA and RNA, are important macromolecules that exist in every form of life. RNA and DNA are made from monomers known as nucleotides. Each nucleotide consists of a pentose carbon sugar, a phosphate group, and a nitrogenous base. If the sugar is ribose, then the polymer is RNA. If the sugar is deoxyribose, then the polymer is DNA.

DNA Structure- DNA is double-stranded helical structure. The strands of the double helix are anti-parallel in nature with one being 5’ to 3’ and the opposite strand 3’ to 5’. Each strand of the original DNA polymer serves as a template for the production of the complementary strand. Replication occurs from 5 prime to 3 prime directions.

RNA Structure- RNA structure is classified into the primary, secondary and tertiary structure. The primary structure is the sequence of bases in a linear order that forms the complete molecule [38]. The secondary structure is a two-dimensional structure formed by folding of molecule around itself to do the base pairing. In secondary structure formation, not all the bases contain exact complementary base pairs, some of the bases lack correct complement base pair, which leads to the formation of loops such as hairpin loops, internal loops, bulge loops, etc. Tertiary structure is 3- dimensional structure such as pseudoknot which is formed by the orientation of these secondary structure motifs. RNA tends to form single-stranded 3D structures because of the presence of extra 2'-hydroxyl group present in Ribose part of RNA [26]. Tertiary structures are 3-dimensional structure made by one molecule. Three-dimensional structure formed by more than one molecule is called quaternary structures.

Throughout this thesis, the push operation is represented by $\Rightarrow_e$ and $\Rightarrow_p$ is used to represent the pop operation.

DEFINITION 1.2.1. Grammar [39] is a quadruple $G(N, \sum, P, S)$, where

- N is a finite set of non-terminals,
- $\sum$ is a finite set of terminals,
- P is a set of productions, and
- S is a start symbol, and $S \in N$.

DEFINITION 1.2.2. A context-free grammar *(CFG)* [39] is quadruple $G_f(N, \sum, P, S)$ where $(N, \sum, S)$ are similar as specified in definition 1.2.1 and the production P is of the following form: $\alpha \to \beta \mid \alpha \in N$ and $\beta \in (N \bigcup \sum)^*$.

DEFINITION 1.2.3. Context-sensitive grammar *(CS)* [39] is quadruple $G_s(N, \sum, P, S)$ where $(N, \sum, S)$ are similar as specified in definition 1.2.1 and the production P is of the following forms: $\{\alpha \to \beta \mid \alpha, \beta \in (N \bigcup \sum)^* \wedge \mid \beta \mid \geq \mid \alpha \mid\}$.

Here $\mid \alpha \mid$ and $\mid \beta \mid$ represent the number of symbols (terminal and non-terminal) involved in α and β respectively.

DEFINITION 1.2.4. Regulated grammar is dual tuple $G_R(G, RG)$, where $G(N, \sum, S, P)$ are similar as specified in definition 1.2.2 and *RG* is the restriction applied on the derivations of strings. *RG* depends on the type of regulated grammar [1-3].

DEFINITION 1.2.5. Regularly controlled grammar [6] is dual tuple $G_{RC}(G, r)$, where $G(N, \sum, S, P)$ are similar as specified in definition 1.2.2 and r is a regular expression over the set of production P which controls the derivations of strings.

Language $L(G_{RC})$ represented by regularly controlled grammar G_{RC} consists of all words $w \in \sum^*$ generated by the following sequence:

$$S \overset{pr_0}{\rightarrow} w_1 \overset{pr_1}{\rightarrow} w_2 \ldots \overset{pr_n}{\rightarrow} w \text{ such that } pr_0, pr_1, \ldots, pr_n \in P$$

EXAMPLE 1.1 [1]: Consider the regularly controlled grammar $G_{RC}(G, r)$, where $G(\{S, C, D\}, \{c, d\}, S, P)$ and the production P is of the following forms:

$pr_0 = S \rightarrow CD$

$pr_1 = C \rightarrow cC$ $\quad$ $pr_2 = D \rightarrow cD$

$pr_3 = C \rightarrow dC$ $\quad$ $pr_4 = D \rightarrow dD$

$pr_5 = C \rightarrow c$ $\quad$ $pr_6 = D \rightarrow c$

$pr_7 = C \rightarrow d$ $\quad$ $pr_8 = D \rightarrow d$

$$r = pr_0 (pr_1 pr_2, pr_3 pr_4)^* (pr_5 pr_6, pr_7 pr_8)$$

Consider the sample string $w = cdcd$

$$S \overset{pr_0}{\rightarrow} CD \overset{pr_1}{\rightarrow} cCD \overset{pr_2}{\rightarrow} cCcD \overset{pr_7}{\rightarrow} cdcD \overset{pr_8}{\rightarrow} cdcd$$

The language generated by the above regularly controlled grammar $L(G_{RC}) = \{ww \mid w \in \{c, d\}^+\}$ and it is a non-context-free language.

DEFINITION 1.2.6. Matrix grammar [7] is dual tuple $G_M(G, r_m)$, where $G(N, \sum, S, M)$, $(N, \sum, S)$ are similar as specified in definition 1.2.2 and $M = (m_1, m_2 \ldots . m_n), n \geq 1$

is a finite set of matrices, where each m_i consists of a finite set of context-free production and r_m is a regular expression over the set of matrices m_i.

Matrix Grammar is a rule-based regulated grammar, where restrictions on productions are applied using matrix. Here, we have to apply all of the productions of a matrix in a sequential order before starting another matrix.

EXAMPLE 1.2 [3]: Consider $G_M(G, r_m)$, where $G(\{S, C, D\}, \{c, d\}, S, \{m_0, m_1, m_2, m_3, m_4\})$ be the matrix grammar where

$m_0 = (p_0 = S \to CD),$

$m_1 = (p_1 = C \to cC, p_2 = D \to cD),$

$m_2 = (p_3 = C \to dC, p_4 = D \to dD),$

$m_3 = (p_5 = C \to c, p_6 = D \to c),$

$m_4 = (p_7 = C \to d, p_8 = D \to d),$

The control set is $r_m = m_0(m_1, m_2)^*(m_3, m_4)$.

Consider $w = dcdc$

$$S \overset{m_0:p_0}{\to} CD \overset{m_1:p_1}{\to} cCD \overset{m_1:p_2}{\to} cCcD \overset{m_2:p_3}{\to} cdCcD \overset{m_2:p_4}{\to} cdCcdD \overset{m_3:p_5}{\to} cdccdD \overset{m_3:p_6}{\to} cdccdc$$

The language generated by the above matrix grammar $L(G_M) = \{ww \mid w \in \{c, d\}^+\}$ and it is a non-context-free language.

DEFINITION 1.2.7. Informally, in state grammar, regulation is specified in terms of states. In each derivation, a non-terminal can be rewritten under the present state and during each derivation it changes its state by making a transition from one state to another state and determines which next rule needs to be applied.

Formally, a state grammar [8] is a pen-tuple $G_s(V, Q, \Sigma, P, S)$, where

- V is a finite set of symbols;
- Q is a finite set of states, and $V \cap Q = \phi$;
- $\Sigma \subset V$ is a finite set of terminals;
- $S \in V - \Sigma$ is the start symbol; and

- $P \subseteq (Q \times (V - \Sigma) \times (Q \times V^{+}))$ is a finite relation over the productions.

EXAMPLE 1.3: Consider the state grammar $G_S(\{S, A, B, 0, 1, 2\}, \{p_0, p_1, p_2\}, \{0, 1, 2\}, P, S)$ where the production rules are

$(p_0, S) \rightarrow (p_0, AB)$

$(p_0, A) \rightarrow (p_1, 0A1)$ $\quad$ $(p_1, B) \rightarrow (p_0, 2B)$

$(p_0, A) \rightarrow (p_2, 01)$ $\quad$ $(p_2, B) \rightarrow (p_0, 2)$

Consider the string $s = 000111222$

$S_{p0} \rightarrow AB_{p0} \rightarrow 0A1B_{p1} \rightarrow 0A12B_{p0} \rightarrow 00A112B_{p1} \rightarrow 00A1122B_{p0} \rightarrow 00011122B_{p2} \rightarrow 000111222_{p0}$

The non context-free language generated by G_S is $L(G_S) = \{0^n 1^n 2^n \mid n \geq 1\}$.

DEFINITION 1.2.8. A state translation scheme is an extension of state grammar in which a string is generated from grammar along with output. Formally, a state translation scheme [40] is a sextuple $(V, Q, \sum_I, \sum_O, S, P)$

- V is a finite set of symbols,
- Q is a finite set of states, and $V \bigcap Q = \phi$,
- $\sum_I \subseteq V$ is a set of input symbols,
- $\sum_O \subseteq V$ is a set of output symbols, $N = V - (\sum_I \bigcup \sum_O)$,
- $S \in N$ is a start symbol, $\bar{N} \subseteq N$, $\bar{N}$ are the non-terminals which we obtain on the right hand side, i.e. $\bar{N}$ contains all non-terminals excluding starting non-terminal S.
- $P \in (Q \times (N)) \times (Q \times (\bar{N} \bigcup \sum_I)^* \times (\phi(\bar{N}) \bigcup \sum_O)^*)$ is a finite relation over the productions. $\phi(\bar{N})$ is a bijective function of set $\bar{N}$ permutations.

DEFINITION 1.2.9. DPDA is an extension of conventional PDA. In DPDA, we can push the symbols on the top of the stack as well as on the deeper parts of the stack. On the basis of expansion depth, DPDA coincides with the infinite hierarchy of n-limited state grammar. Formally, DPDA [10] is a septuplet ${}_d M(Q, \Gamma, \sum, R, q_0, S, F)$ where,

- Q is a finite set of states,

- Γ is a finite set of pushdown symbols,
- $\sum$ is a finite set of input symbols, such that $\sum \subseteq \Gamma$ and $\# \in (\Gamma - \sum)$,
- R is a finite relation such that $R \subseteq (I \times Q \times (\Gamma - (\sum \bigcup \{\#\})) \times Q \times (\Gamma - \{\#\})^+)$ $\bigcup (I \times Q \times \{\#\} \times Q \times (\Gamma - \{\#\})^* \{\#\})$ and the elements of $\{I, Q, \Gamma\}$ are pair-wise disjoint where I denotes natural numbers.
- $q_0 \in Q$ denotes the starting state,
- $S \in \Gamma$ is the starting pushdown symbol,
- $F \in Q$ is the set of final states,

d is a maximum depth up to which non-input symbol can be expanded. A configuration of ${}_dM$ is a triple in $Q \times \sum^* \times (\Gamma - \{\#\})^* \{\#\}$.

EXAMPLE 1.4: Consider DPDA ${}_2M(\{q_0, q, p, f\}, \{A, S, \#\}, \{a, b, c\}, R, q_0, S, \{f\})$ with *R* containing rules $1q_0S \to qAA, 1qA \to paAb, 1qA \to fab, 2pA \to qAc, 1fA \to fc$

Given input string $w = aabbcc$, ${}_2M$ makes

$(q_0, aabbcc, S\#) \Rightarrow_e (q, aabbcc, AA\#)$ $[1q_0S \to qAA]$

$\Rightarrow_e (p, aabbcc, aAbA\#)$ $[1qA \to paAb]$

$\Rightarrow_p (p, abbcc, AbA\#)$

$\Rightarrow_e (q, abbcc, AbAc\#)$ $[2pA \to qAc]$

$\Rightarrow_e (f, abbcc, abbAc\#)$ $[1qA \to fab]$

$\Rightarrow_p (f, bbcc, bbAc\#)$

$\Rightarrow_p (f, bcc, bAc\#)$

$\Rightarrow_p (f, cc, Ac\#)$

$\Rightarrow_e (f, cc, cc\#)$ $[1fA \to fc]$

$\Rightarrow_p (f, c, c\#)$

$\Rightarrow_p (f, \wedge, \#)$

Thus, $L({}_2M) = E({}_2M) = \{a^n b^n c^n \mid n \geq 1\}$, where $L({}_2M)$ and $E({}_2M)$ denote the languages accepted by final state and empty pushdown of depth two respectively.

DEFINITION 1.2.10. PDPDA is an extension to DPDA. It is a parallel version of DPDA. In one move of PDPDA, we can expand *n* topmost non-input symbols. It also coincides with the infinite hierarchy of *n*-limited state grammars.

Formally, PDPDA [11] is a septuplet ${}_dM_p(Q, \Gamma, \sum, R, q_0, S, F)$ where,

- Q is a finite set of the states,
- Γ is a finite set of pushdown symbols,
- $\sum$ is a finite set of input symbols, such that $\sum \subseteq \Gamma$ and $\# \in (\Gamma - \sum)$,
- R is a finite set of rules of the type $p(A_1, ..., A_n) \rightarrow q(w_1, ..., w_n)$, where $p, q \in Q$, $A_i \in \Gamma - \sum$, $w_i \in \Gamma^+$, $1 \leq i \leq n$.
- $q_0 \in Q$ denotes the starting state,
- $S \in \Gamma$ is the start pushdown symbol,
- $F \in Q$ is the set of final states,

d is a maximum depth up to which non-input symbol can be expanded. The configuration of PDPDA ${}_dM_p$ is a triplet $Q \times \sum^* \times (\Gamma - \{\#\})^* \{\#\}$.

EXAMPLE 1.5: Consider PDPDA ${}_2M_p(\{q_0, q, p, f\}, \{S, X, \#\}, \{a, b, c\}, R, q_0, S, \{f\})$ with *R* containing rules:

$1q_0S \rightarrow qXX$, $1qX \rightarrow paXb$,

$1qX \rightarrow fab$, $1fX \rightarrow fc$,

$q(X, X) \rightarrow q(aXb, Xc)$, $p(X, X) \rightarrow q(X, Xc)$,

$p(S, X) \rightarrow q(S, Xc)$

For input string $\alpha = aabbcc$,

$(q_0, aabbcc, S\#) \Rightarrow_e (q, aabbcc, XX\#) \Rightarrow_e (q, aabbcc, aXbXc\#) \Rightarrow_p (q, abbcc, XbXc\#)$

$\Rightarrow_e (f, abbcc, abbXc\#) \Rightarrow_p (f, bbcc, bbXc\#) \Rightarrow_p (f, bcc, bXc\#) \Rightarrow_p (f, cc, Xc\#)$

$\Rightarrow_e (f, cc, cc\#) \Rightarrow_p (f, c, c\#) \Rightarrow_p (f, \wedge, \#)$

The $L({}_2M_p) = E({}_2M_p) = \{a^n b^n c^n \mid n \geq 1\}$, where $L({}_2M_p)$ and $E({}_2M_p)$ denotes the language accepted by final state and empty pushdown of depth two respectively.

DEFINITION 1.2.11. The DPDT is the extended version of the DPDA. It works similarly as DPDA except that an output string is generated corresponding to the given input string. Formally, DPDT [33] is an octuplet ${}_dM_T(Q, \sum_I, \Gamma, \sum_O, R, q_0, S, F)$ where,

- Q is the set of states,
- $\sum_I$ is a finite set of input symbols,
- Γ is a finite set of pushdown symbols,
- $\sum_I \subseteq \Gamma$,
- $\sum_O$ is a finite set of output symbols, where $\sum_O \not\subset \Gamma$,
- $\Gamma - \sum_I$, contains a special bottom symbol denoted by #, i.e. $\# \in (\Gamma - \sum_I)$
- $q_0 \in Q$ denotes the starting state,
- $S \in \Gamma$ is the start pushdown symbol,
- $F \in Q$ is the set of final states,
- R is a finite set of rules in a form $mpA \rightarrow qw\alpha'$, where m specifies depth, $m \leq d$, $p, q \in Q$, $A \in \Gamma - \sum_I$, $w \in \Gamma^+$, $\alpha' \in (\sum_O)^*$.

d is a maximum depth up to which non-input symbol can be expanded. A configuration of DPDT ${}_dM_T$ is a 4-tuple $Q \times (\sum_I)^* \times (\Gamma - \{\#\})^* \{\#\} \times (\sum_O)^*$.

EXAMPLE 1.6: DPDT for $L_I = \{a^n b^n c^n \mid n \geq 1\}$ which perform the reversal of the string with the output $L_O = \{c^n b^n a^n \mid n \geq 1\}$.

Meduna [10] designed a DPDA for the language L_I DPDT ${}_2M_T(\{q_0, q, p, f\}, \{a, b, c\}, \{X, S, \#\}, \{a, b, c, \wedge\}, R, q_0, S, \{f\})$ with R containing rules:

$(1q_0S \rightarrow qXX, \wedge)$,

$(1qX \rightarrow paXb, \wedge)$,

$(1qX \rightarrow fab, \wedge)$,

$(2pX \rightarrow qXc, \wedge)$,

$(1fX \rightarrow fc, \wedge)$

Rules for pop operations:

1. If *a* is popped, generate output *c*.
2. If *b* is popped, generate output *b*.
3. If *c* is popped, generate output *a*.

For $w = aabbcc$

$(q_0,\ aabbcc,\ S\#, \wedge) \Rightarrow_e (q,\ aabbcc,\ XX\#, \wedge) \Rightarrow_e (p,\ aabbcc, aXbX\#, \wedge) \Rightarrow_p$
$(p,\ abbcc,\ XbX\#, c) \Rightarrow_e (q,\ abbcc,\ XbXc\#, c) \Rightarrow_e (f,\ abbcc,\ abbXc\#, c) \Rightarrow_p$
$(f, bbcc,\ bbXc\#, cc) \Rightarrow_p (f, bcc, bXc\#, ccb) \Rightarrow_p (f, cc,\ Xc\#, ccbb) \Rightarrow_e (f, cc,\ cc\#, ccbb)$
$\Rightarrow_p (f,\ c,\ c\#, ccbba) \Rightarrow_p (f, \wedge, \#, ccbbaa)$

DEFINITION 1.2.12. The PDPDT is the extended version of the PDPDA. It works similarly as PDPDA except that an output string is generated corresponding to the given input string. Formally, PDPDT [33] is an octuple ${}_dM_{Tp}(Q, \sum_I, \Gamma, \sum_O, R, q_0, S, F)$ where,

- Q is the set of the states,
- $\sum_I$ is a finite set of input symbols,
- Γ is a finite set of pushdown symbols,
- $\sum_I \subseteq \Gamma$,
- $\sum_O$ is a finite set of output symbols, where $\sum_O \not\subset \Gamma$,
- R is a finite set of rules in a form $(p(A_1,...,A_n) \rightarrow q(w_1,...,w_n), \alpha')$ where $n \le d$, $p, q \in Q,\ A_i \in \Gamma - (\sum_I \bigcup \{\#\}),\ w_i \in (\Gamma - \{\#\})^+,\ 1 \le i \le n,\ \alpha' \in (\sum_O)^*$.
- $\Gamma - \sum_I$, contains a special bottom symbol denoted by #, i.e. $\# \in (\Gamma - \sum_I)$
- $q_0 \in Q$ denotes the starting state,
- $S \in \Gamma$ is the start pushdown symbol,
- $F \in Q$ is the set of final states,

d is a maximum depth up to which non-input symbol can be expanded. The configuration of PDPDT ${}_dM_{Tp}$ is a 4-tuple $Q \times (\sum_I)^* \times (\Gamma - \{\#\})^* \{\#\} \times (\sum_O)^*$.

EXAMPLE 1.7: PDPDT for $L_I = \{a^n b^n c^n \mid n \ge 1\}$ which perform the reversal of the string with the output $L_O = \{c^n b^n a^n \mid n \ge 1\}$.

Solar and Meduna [11] designed a PDPDA for the language L_I

PDPDT ${}_2M_{Tp}(\{q_0,\ q,\ p,\ f\},\{a,\ b,\ c\},\{S,\ X,\#\},\{a,b,c,\wedge\},\ R,\ q_0,\ S,\{f\})$ where R is defined as follow:

$(1q_0S \to qXX, \wedge)$,

$(1qX \to paXb, \wedge)$,

$(1qX \to fab, \wedge)$,

$(1fX \to fc, \wedge)$,

$(q(X, X) \to q(aXb, Xc), \wedge)$,

$(p(X, X) \to q(X, Xc), \wedge)$,

$(p(S, X) \to q(S, Xc), \wedge)$

Rules for pop operations:

1. If *a* is popped, generate output *c*.
2. If *b* is popped, generate output *b*.
3. If *c* is popped, generate output *a*.

For $w = aabbcc$, ${}_dM_{Tp}$ will process the string as follows:

$(q_0, aabbcc, S\#, \wedge) \Rightarrow_e (q, aabbcc, XX\#, \wedge) \Rightarrow_e (q, aabbcc, aXbXc\#, \wedge) \Rightarrow_p$
$(q, abbcc, XbXc\#, c) \Rightarrow_e (f, abbcc, abbXc\#, c) \Rightarrow_p (f, bbcc, bbXc\#, cc) \Rightarrow_p (f, bcc, bXc\#, ccb)$
$\Rightarrow_p (f, cc, Xc\#, ccbb) \Rightarrow_e (f, cc, cc\#, ccbb) \Rightarrow_p (f, c, c\#, ccbba) \Rightarrow_p (f, \wedge, \#, ccbbaa)$.

DEFINITION 1.2.13. Fuzzy grammar [41] is quadruple $G_f(N, \sum, P, S)$, where $(N, \sum, S)$ are similar as specified in definition 1.2.1 and P is a set of fuzzy productions of the following form: $\{\alpha \overset{\mu}{\to} \beta \mid 0 \le \mu \le 1 \wedge \alpha, \beta \in (N \bigcup \sum)^*\}$..

DEFINITION 1.2.14. Fuzzy language L[41] is a fuzzy set of Σ^*. $L = \{(x, \mu(x)) \mid x \in \Sigma^*$ $\wedge 0 \le \mu(x) \le 1\}$ where $\mu(x)$ denote the degree of membership of string x.

If a single derivation exists for the string x, $\mu(x)$ is obtained by taking a minimum of all the derivation rules from S to derive x, if there exist more than one derivation for a

particular string x, then max is taken over the minimum of all the derivation rules starting from S to derive x.

1.3 ORGANIZATION OF THE THESIS

This thesis is devoted to designing and development of a variant of regulated grammar or regulated automata and to apply their concept in RNA and DNA bio-molecular structures. The results obtained are discussed in chapter 3 to chapter 7. Complete chapter-wise summary of thesis work is given as follows:

- **In Chapter 1**, the motivation of research work, preliminary concepts which include basic notations and definitions and, the thesis outline has been given.
- **In Chapter 2**, we discuss various types of regulated grammar and regulated automata. Regulated grammar can be classified into rule-based, context-based and parallel based. Survey of various existing regulated automata such as Deep Pushdown Automata, Parallel Deep Pushdown Automata, and Absolutely Unlimited Deep Pushdown Automata has been carried out. Survey of existing regulated transducer such as Deep Pushdown Transducer, Parallel Deep Pushdown Transducer has been discussed. Further, the concept of fuzzy context-free grammar, fuzzy pushdown automata, and grammar formalism for DNA and RNA sequences has also been discussed. In the last section, gaps in the area of regulated grammar and regulated automaton, problem statement, and the objectives of this dissertation are explored.
- **In Chapter 3**, a novel concept of fuzzy state grammars and fuzzy deep pushdown automata has been introduced. This concept represents a natural extension of contemporary state grammar and deep pushdown automaton, making them more robust in terms of imprecision, errors, and uncertainty. We have defined the Instantaneous description of fuzzy deep pushdown automata.

  It has been proved that we can construct fuzzy deep pushdown automata from fuzzy state grammars and vice-versa. The two main results of this chapter are as follows:

  - **Theorem 1:** Let $G_{fs}(V, Q, \Sigma, P, S, E, \mu)$ be a fuzzy state grammar, then we can construct a fuzzy deep pushdown automaton M_{fd} $(Q, \sum, \Gamma, R, q_0, S, F, E, \mu, \pi, \gamma, \eta)$ such that $L(G_{fs}) = L(M_{fd})$ i.e. for any $\alpha \in \Sigma^*$, $\mu(\alpha; \alpha \in L(G_{fs})) = \mu(\alpha; \alpha \in L(M_{fd}))$.

- **Theorem 2:** Let M_{fd} $(Q, \sum, \Gamma, R, q_0, S, F, E, \mu, \pi, \gamma, \eta)$ be a fuzzy deep pushdown automaton, then we can construct be a fuzzy state grammar $G_{fs}(V, Q, \Sigma, P, S, E, \mu)$ such that $L(G_{fs}) = L(M_{fd})$ i.e. for any $\alpha \in \Sigma^*$, $\mu(\alpha; \alpha \in L(G_{fs})) = \mu(\alpha; \alpha \in L(M_{fd}))$ where μ denotes the membership of a string.

- **In Chapter 4**, Deterministic models of deep pushdown automata, parallel deep pushdown automata, deep pushdown transducer and parallel deep pushdown transducer are designed. The proposed deterministic models are strictly deterministic and deterministic with respect to depth. Furthermore, based on the deterministic version of these proposed automata and transducer, an infinite hierarchy of languages is explored. We support this discussion of deep and parallel pushdown automata and transducers by way of numerical examples. The main results of this chapter are as follows:
  - **Theorem 1:** The number of moves (expansion and pop) required for DDPDA is less than the DPDA for language $L = \{a^n b^n c^n \mid n \geq 1\}$.
  - **Theorem 2:** The number of moves (expansion and pop) required for PDDPDA is less than the PDPDA for language $L = \{a^n b^n c^n \mid n \geq 1\}$.
  - **Theorem 3:** The number of moves (expansion and pop) required for DDPDT is less than DPDT for reversal of strings for input $\{a^n b^n c^n \mid n \geq 1\}$.
  - **Theorem 4:** The number of moves (expansion and pop) required for PDDPDT is less than PDPDT for reversal of strings for input $\{a^n b^n c^n \mid n \geq 1\}$.
  - **Theorem 5:** For every $DDPDA_n$ with depth *n*, there exists a $PDDPDA_n$ (i.e., for $n \geq 1$, $DDPDA_n = PDDPDA_n$).
  - **Corollary 1:** For every $DDPDT_n$ with depth *n*, there exists a $PDDPDT_n$ (i.e., for $n \geq 1$, $DDPDT_n = PDDPDT_n$).
  - **Theorem 6** [11]**:** For every $n \geq 1$, $DPDA_n = PDPDA_n \subset CS$.
  - **Theorem 7:** Given $L = \{ww \mid w \in (0|1)^+\}$ cannot be accepted by any deterministic deep pushdown automata or parallel deterministic deep pushdown automata.
  - **Corollary 2:** $DDPDT \subset DPDT$ and $PDDPDT \subset PDPDT$
  - **Corollary 3:** Deterministic pushdown automata cannot be designed for $L = \{ww^R \mid w \in (0|1)^+\}$
  - **Theorem 8:** $DDPDA - CF \neq \phi$

- **Theorem 9:** $CF - DDPDA \neq \phi$
- **Theorem 10:** The number of moves (expansion and pop) required for PDDPDT is less than DDPDT for performing the computation $q_0 w \overset{*}{\rightarrow} q_f ww$.

- **In Chapter 5**, deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) structure sequences are represented using state grammar and deep pushdown automata. We have represented three commonly found structure in DNA and RNA using state grammar namely:
  - **Tandem Repeats:** Tandem repeats occur, whenever the nucleotide sequence is duplicated, and duplication occurs directly adjacent to it.
  - **Inverted repeats:** Inverted repeats occur, whenever nucleotides sequence is followed by its reverse complement.
  - **Pseudoknots:** Pseudoknots are found in tertiary RNA structures. Pseudoknot sequences are formed whenever a pairing occurs between a secondary loop structure and complementary base outside the loop.

  These biological sequences are depicted using formal model deep pushdown automata. We have designed extended LL(1) Parser for state grammar. Extended LL(1) parser works similar to LL(1) parser except for the inclusion of states. The above-designed grammars for tandem repeats, inverted repeats, and interleaved repeats do not contain left-recursion. They contain left-factoring, which is handled by the inclusion of states in the parsing table.

  The major benefit of this approach is that the DNA and RNA sequences can be parsed in linear time $O(n)$, where n is the length of the string, which is a significant improvement over the existing approaches. In various existing approaches in the literature, these sequences are represented using context-sensitive grammar or mildly context-sensitive grammar with higher time complexities.
- **In Chapter 6**, we extended our previous work of Chapter 5 for representing Deoxyribonucleic Acids (DNA) and Ribonucleic Acids (RNA) biological structures with the help of state grammar. The various structures represented are attenuator, extended pseudoknot, kissing hairpin, simple h-type, recursive pseudoknot, and three-knot.
- **In Chapter 7**, an algorithm is designed for error tolerance in recognition of faulty strings in a regulated grammar using fuzzy sets (*RFSRG*)**.** Furthermore, depending upon the errors and certainty, we decide whether the string belongs to the language or not

depending on its membership value. Regulated grammar is converted to Chomsky normal form to avoid multiple error elimination. The time complexity of the proposed algorithm is $O(|G^2|.|w|^3)$, where $|G|$ represents the number of production rules and $|w|$ is the length of the input string *w*. We have provided the numerical examples by applying the algorithm to regularly controlled and matrix grammar. By way of numerical examples for a string, $w \notin L$, the determination of a confidence level for the string, *w*, is also demonstrated.

Finally, conclusions are presented, followed by future scope and references.

Chapter-2

LITERATURE SURVEY

In this chapter, we will discuss various types of regulated grammar. Regulated grammar can be classified into context-based, parallel based and rule-based grammar. In this chapter, survey of various existing regulated automata and regulated transducer has been carried out. Further, the concept of fuzzy context-free grammar, fuzzy pushdown automata and grammar formalism for DNA and RNA sequences has also been explored. In the last section, gaps in the area of regulated grammar and regulated automaton, problem statement, and the objectives of this dissertation are explored.

2.1 REGULATED GRAMMAR

Regulated grammar can be classified into context-based, parallel based and rule-based grammar as shown in Fig. 2.1.

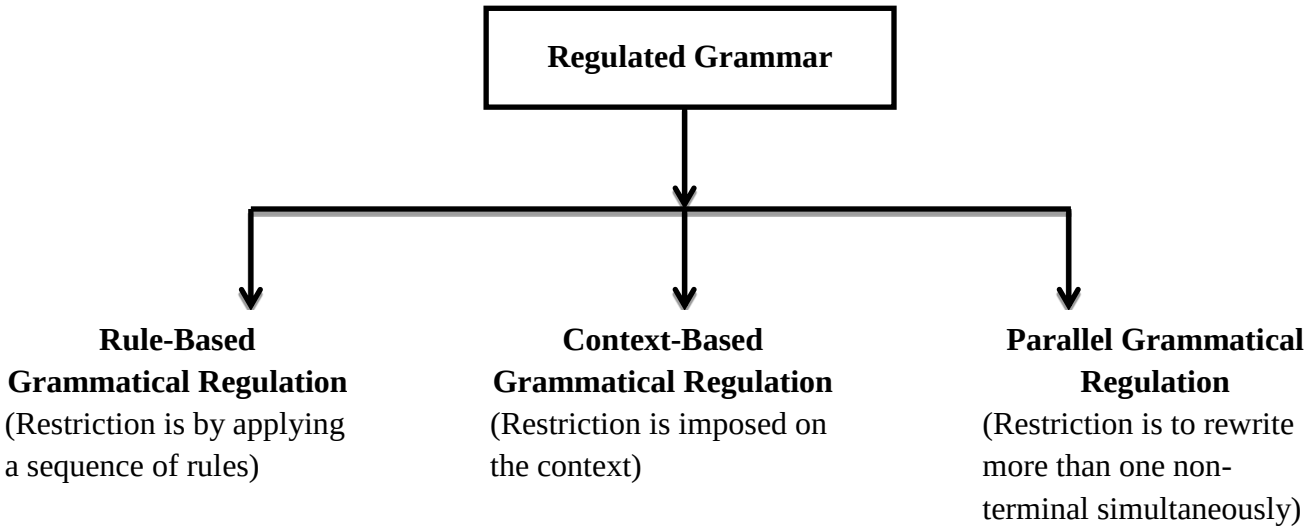


Fig. 2.1: Classification of Regulated Grammar

2.1.1 CONTEXT-BASED GRAMMATICAL REGULATION

In context-based grammatical regulation [3], derivation is regulated by context-related

Results of this chapter are communicated in "IET SOFTWARE" entitled:
A Survey of Formal Language Theory: Exploration of Various Sub-classes of Chomsky Hierarchy

restrictions. The production rules are of the type $(\alpha \to \beta, P_r, F_r)$ where P_r and F_r are permitting and forbidding fields, respectively. A rule in context-based grammar can rewrite a non-terminal if the permitting field occurs in sentential form, and a forbidding field does not occur. The following terminologies are used in context-based grammatical regulation [3]:

- **Propagating:** If there are no production rules of the type $(\alpha \to \wedge, P_r, F_r)$, then grammar is said to be propagating (denoted by superscript P).
- **Degree (*p*, *q*):** Every grammar has a degree (*p*, *q*), where *p* and *q* are positive numbers, and for every rule of the form $(\alpha \to \beta, P_r, F_r)$, $|P_r| \leq p$ and $|F_r| \leq q$ where $|P_r|$ and $|F_r|$ denote the number of productions in permitting and forbidding set.

Fig. 2.2 represents various types of context-based regulated grammars.

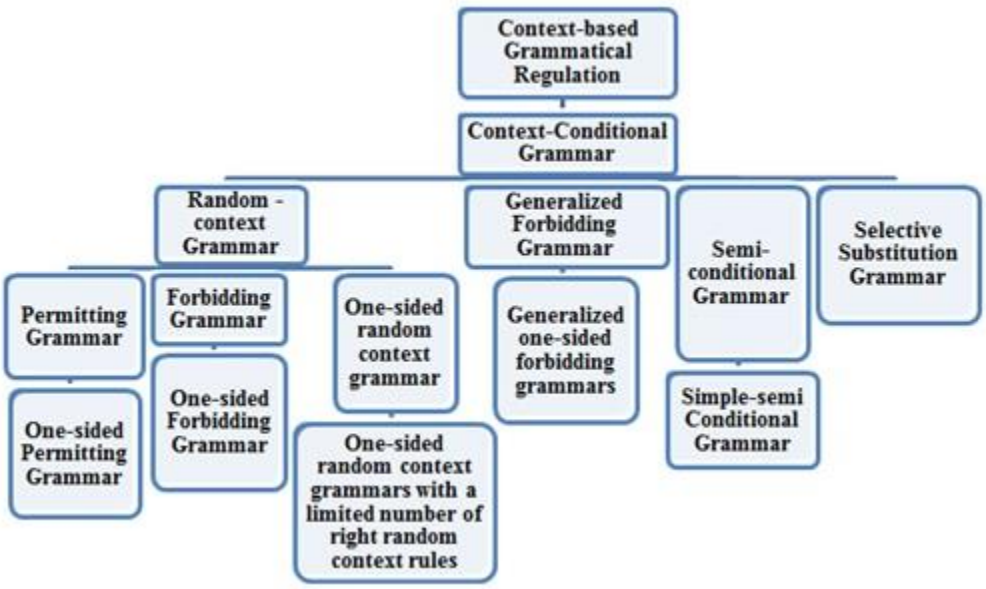


Fig. 2.2: Classification of context-based regulated grammars

Navratil [42] introduced the concept of context-conditional grammar (CC) in which production rules are of the form $(\alpha \to \beta, P_r, F_r)$, where $\alpha \to \beta$ is a context-free rule, P_r is a permitting string and F_r is a forbidden string.

EXAMPLE 2.1: Consider context-conditional grammar $G_{cc}(\{S, C, D, P, Q\}, \{a\}, P, S)$ where *P* is defined by

$\{(S \to a, \phi, \phi), (S \to P, \phi, \phi), (P \to QD, \phi, C), (P \to aD, \phi, C), (Q \to PC, \phi, D),$

$(Q \to aC, \phi, D), (C \to DD, PC, \phi), (D \to CC, QD, \phi), (D \to a, a, \phi)\}$

Consider string $w = aaaaaaaa$

$S \Rightarrow P \Rightarrow QD \Rightarrow QCC \Rightarrow PCCC \Rightarrow PDDCC \Rightarrow PDDCDD \Rightarrow PDDDDDD \Rightarrow aDDDDDDD$
$\Rightarrow aDDaDDDD \Rightarrow^{*} aaaaaaaa$

The language $L(G_{cc}) = \{a^{2^n} \mid n \geq 0\}$

Walt [43] introduced the concept of Random-context Grammar (RC), Permitting Grammar (Per) and Forbidding Grammar (For). Random-context grammar is a context-conditional grammar where production rules are of the form $(\alpha \to \beta, P_r, F_r)$, with the degree (1, 1).

EXAMPLE 2.2: Consider random-context grammar $G_{RC}(\{S, P, Q, R\}, \{a\}, P, S)$ where the production rules *P* are defined by:

$\{(S \to PP, \phi, \{Q, R\}), (P \to Q, \phi, \{S\}), (Q \to S, \phi, \{P\}), (S \to R, \phi, \{P\}), (R \to a, \phi, \{S\})\}$

Consider $w = aaaa$

$S \Rightarrow PP \Rightarrow QP \Rightarrow QQ \Rightarrow SQ \Rightarrow SS \Rightarrow PPS \Rightarrow PPPP \Rightarrow QPPP \Rightarrow QQPP \Rightarrow QQQP \Rightarrow$
$QQQQ \Rightarrow SQQQ \Rightarrow SSQQ \Rightarrow SSSQ \Rightarrow SSSS \Rightarrow RSSS \Rightarrow RRSS \Rightarrow RRRS \Rightarrow RRRR \Rightarrow$
$aRRR \Rightarrow aaRR \Rightarrow aaaR \Rightarrow aaaa$

$L(G_{RC}) = \{a^{2^n} \mid n \geq 0\}$

In permitting grammar [43], the forbidden set of strings is empty. In permitting grammar, rules are of the form $(\alpha \to \beta, P_r)$ with the degree (1, 0).

EXAMPLE 2.3: Consider permitting grammar $G_p(\{S, X, Y, Z, X', Y', Z'\}, \{x, y, z\},$ $P, S)$ where the production rules *P* are defined by

$\{(S \to XYZ, \phi), (X \to xX', \{Y\}), (Y \to yY', \{Z\}), (Z \to zZ', \{X'\}), (X' \to X, \{Y'\}),$

$(Y' \to Y, \{Z'\}), (Z' \to Z, \{X\}), (X \to x, \{Y\}), (Y \to y, \{Z\}), (Z \to z, \phi)\}$

Consider $w = xxyyzz$

$S \Rightarrow XYZ \Rightarrow xX'YZ \Rightarrow xX'yY'Z \Rightarrow xX'yY'zZ' \Rightarrow xXyY'zZ' \Rightarrow xXyYzZ' \Rightarrow xXyYzZ \Rightarrow$
$xxyYzZ \Rightarrow xxyyzZ \Rightarrow xxyyzz$

The language represented by permitting grammar is $L(G_p) = \{x^n y^n z^n \mid n \geq 1\}$

In forbidding grammar [43], the permitting set of strings is empty. In forbidding grammar, rules are of the form $(\alpha \to \beta, F_r)$ with the degree (0, 1).

EXAMPLE 2.4: Consider the forbidding grammar $G_F(\{S, P, Q, R\}, \{a\}, P, S)$ where the production rules P are defined by $\{(S \to PP, \{Q, R\}), (P \to Q, \{S\}), (Q \to S, \{P\}), (S \to R, \{P\}), (R \to a, \{S\})\}$

Consider $w = aaaa$

$S \Rightarrow PP \Rightarrow QP \Rightarrow QQ \Rightarrow SQ \Rightarrow SS \Rightarrow PPS \Rightarrow PPPP \Rightarrow QPPP \Rightarrow QQPP \Rightarrow QQQP \Rightarrow QQQQ \Rightarrow SQQQ \Rightarrow SSQQ \Rightarrow SSSQ \Rightarrow SSSS \Rightarrow RSSS \Rightarrow RRSS \Rightarrow RRRS \Rightarrow RRRR \Rightarrow aRRR \Rightarrow aaRR \Rightarrow aaaR \Rightarrow aaaa$

The language accepted by forbidding grammar is $L(G_F) = \{a^{2^n} \mid n \geq 0\}$.

Meduna et al. [44] proposed one-sided random-context grammar (OSRC) which is also further characterized into one-sided permitting grammar (OSPRG) and one-sided forbidding grammar (OSRFG). In one-sided random context grammar [44], the set of rules is divided into left and right random context rules. Left random-context rules rewrite a non-terminal if in the left-hand side of the non-terminal, the permitting string of a rule is present, and the forbidden string does not occur. Right random-context rules rewrite a non-terminal if in the right-hand side of the non-terminal, the permitting string of the rule is present and the forbidden string does not occur. Production rules are of the form $(\alpha \to \beta, L_{RC}, R_{RC})$.

EXAMPLE 2.5: Let $G_{OSRC}(\{S, P, Q, P', Q'\}, \{p, q, r\}, L_{RC}, R_{RC}, S)$ be a one-sided random-context grammar with L_{RC} is as follows:

$\{(S \to PQ, \phi, \phi), (Q' \to Q, \{P\}, \phi), (Q \to qQ'r, \{P'\}, \phi), (Q \to \wedge, \phi, \{P, P'\})\}$

and R_{RC} is as follows $\{(P \to pP', \{Q\}, \phi), (P' \to P, \{Q'\}, \phi), (P \to \wedge, \{Q\}, \phi)\}$

Consider the string $w = p^n q^n r^n$

$S \Rightarrow PQ \Rightarrow pP'Q \Rightarrow pP'qQ'r \Rightarrow pPqQ'r \Rightarrow pPqQr \Rightarrow^* p^n Pq^n Qr^n \Rightarrow^* p^n q^n Qr^n \Rightarrow^* p^n q^n r^n$

$L(G_{OSRC}) = \{p^n q^n r^n \mid n \geq 0\}$

In one-sided permitting grammar (OSPRG) [44], the forbidden set is null, and the permitting set is divided into left and right permitting random context rules. Left random-context rules rewrite a non-terminal if the permitting string of a rule is present to the left of non-terminal and right random-context rules rewrite a non-terminal if the permitting string of a rule is present to the right of non-terminal.

EXAMPLE 2.6: Let $G_{OSPRG}(\{S, P, Q, R, S\}, \{p, q, r\}, L_P, \phi, S)$ is one-sided permitting grammar, where the right permitting set is null and L_P is as follows:

$\{(S \to PR, \phi, \phi), (P \to p, \phi, \phi), (P \to pQ, \phi, \phi), (Q \to P, \phi, \phi), (R \to qr, \phi, \phi),$

$(R \to qQr, \{Q\}, \phi), (S \to P, \{P\}, \phi)\}$

$L(G_{OSRPG}) = \{p^n q^m r^m \mid n, m \geq 0\}$

In one-sided forbidding grammar (OSRFG) [44], the permitting set is null, and the forbidding set is divided into left and right forbidding random-context rules. Left forbidden random-context rules rewrite a non-terminal if the forbidding string of a rule is absent to the left of non-terminal and right forbidden random-context rules rewrite a non-terminal if the forbidding string of a rule is absent to the right of non-terminal.

EXAMPLE 2.7: Let $G_{OSRFG}(\{S, P, Q, P', Q', \bar{P}, \bar{Q}\}, \{p, q, r\}, L_f, R_f, S)$ is a one-sided forbidding grammar, where

L_f is $\{(S \to PQ, \ \phi), (Q \to qQ'r, \{P, \bar{P}\}), (Q \to \bar{Q}, \{P, P'\}, (Q' \to Q, \{P'\}), (\bar{Q} \to \wedge, \{\bar{P}\})\}$

and R_f is $\{(P \to pP', \{Q'\}), (P \to \bar{P}, \{Q'\}), (P' \to P, \{Q\}), (\bar{P} \to \wedge, \{Q\})\}$

$S \Rightarrow PQ \Rightarrow pP'Q \Rightarrow pP'qQ'r \Rightarrow pPqQ'r \Rightarrow pPqQr \Rightarrow^* p^n Pq^n Qr^n \Rightarrow p^n \bar{P} q^n Qr^n \Rightarrow$
$p^n \bar{P} q^n \bar{Q} r^n \Rightarrow p^n q^n \bar{Q} r^n \Rightarrow p^n q^n r^n$

$L(G_{OSRFG}) = \{p^n q^n r^n \mid n \geq 0\}$

Meduna et al. [45] defined that any recursively enumerable language can be generated using two or less right random context rules. Meduna [46] defined a generalized forbidding grammar (GFG). In generalized forbidding grammar, forbidding set of rules is

formed by finite languages. In GFG, the permitting set of strings is empty and production rules are of the form $(\alpha \to \beta, F_r)$.

EXAMPLE 2.8: Let $G_{GFG}(\{S, P, Q, R\}, \{a\}, P, S)$ in which production rules P are defined by $\{(S \to PP, \{Q, R\}), (P \to Q, \{S\}), (Q \to S, \{P\}), (S \to R, \{P\}), (R \to a, \{S\})\}$ Consider $w = aaaa$

$S \Rightarrow PP \Rightarrow QP \Rightarrow QQ \Rightarrow SQ \Rightarrow SS \Rightarrow PPS \Rightarrow PPPP \Rightarrow QPPP \Rightarrow QQPP \Rightarrow QQQP \Rightarrow$
$QQQQ \Rightarrow SQQQ \Rightarrow SSQQ \Rightarrow SSSQ \Rightarrow SSSS \Rightarrow RSSS \Rightarrow RRSS \Rightarrow RRRS \Rightarrow RRRR \Rightarrow$
$aRRR \Rightarrow aaRR \Rightarrow aaaR \Rightarrow aaaa$

$L(G_{GFG}) = \{a^{2^n} \mid n \geq 0\}$

Meduna et al. [47] introduced the concept of generalized one-sided forbidding grammar (GOSFG). In GOSFG, the forbidden rule set is divided into left and right forbidding rules. A left forbidding rule is applied when the forbidden string is absent to the left of the non-terminal and the right forbidding rule is applied when the forbidden string is absent to the right of the non-terminal.

EXAMPLE 2.9: Let $G_{GOSFG}(\{S, P, Q, P', Q', \bar{P}, \bar{Q}\}, \{p, q, r\}, L_f, R_f, S)$ is a generalized one-sided forbidding grammar, where

L_f is $\{(S \to PQ, \phi), (Q \to qQ'r, \{P, \bar{P}\}), (Q \to \bar{Q}, \{P, P'\}, (Q' \to Q, \{P'\}), (\bar{Q} \to \wedge, \{\bar{P}\})\}$
and R_f is $\{(P \to pP', \{Q'\}), (P \to \bar{P}, \{Q'\}), (P' \to P, \{Q\}), (\bar{P} \to \wedge, \{Q\})\}$

$S \Rightarrow PQ \Rightarrow pP'Q \Rightarrow pP'qQ'r \Rightarrow pPqQ'r \Rightarrow pPqQr \Rightarrow^* p^nPq^nQr^n \Rightarrow p^n\bar{P}q^nQr^n \Rightarrow$
$p^n\bar{P}q^n\bar{Q}r^n \Rightarrow p^nq^n\bar{Q}r^n \Rightarrow p^nq^nr^n$

$L(G_{GOSFG}) = \{p^nq^nr^n \mid n \geq 0\}$

Paun [48] introduced the concept of semi-conditional grammar (SC). Semi-conditional grammar is a context-conditional grammar which satisfying the conditions $|P_r| \leq 1$ and $|F_r| \leq 1$, where $|P_r|$ and $|F_r|$ denote the number of productions in permitting and forbidding set.

EXAMPLE 2.10: Let $G_{sc}(\{S, A, B, C, D, E, F\}, \{a, b, c, d, e, f\}, P, S)$ is a semi-conditional grammar, where production rules P are defined by:

$\{(S, ABC, \phi, \phi), (A \to aDb, B, F), (B \to cEd, D, F), (C \to eFf, D, B), (D, \to A, F, B),$

$(E \to B, A, C), (F \to C, A, E), (A \to \wedge, B, F), (B \to \wedge, C, A), (C \to \wedge, \phi, B)\}$

Consider string $w = aabbccddeeff$

$S \Rightarrow ABC \Rightarrow aDbBC \Rightarrow aDbcEdC \Rightarrow aDbcEdeFf \Rightarrow aAbcEdeFf \Rightarrow aAbcBdeFf \Rightarrow$
$aAbcBdeCf \Rightarrow aaDbbcBdeCf \Rightarrow aaDbbccEddeCf \Rightarrow aaDbbccEddeeFff \Rightarrow aaAbbcc$
$EddeeFff \Rightarrow aaAbbccBddeeFff \Rightarrow aaAbbccBddeeCff \Rightarrow aabbccBddeeCff \Rightarrow aabb$
$ccddeeCff \Rightarrow aabbccddeeff$

$L(G_{sc}) = \{a^n b^n c^n d^n e^n f^n \mid n \geq 0\}$

Meduna et al. [49] introduced the concept of Simple Semi-conditional Grammar (SSC). SSC is a context-conditional grammar satisfying the conditions $|P_r| + |F_r| \leq 1$ where $|P_r|$ and $|F_r|$ denote the number of productions in permitting and forbidding set. Production rules are of the form $(\alpha \to \beta, P_r, F_r)$.

EXAMPLE 2.11: Let $G_{ssc}(\{S, C, D, P, Q\}, \{a\}, P, S)$ is a simple semi-conditional grammar, where *P* is defined as follows:

$\{(S \to a, \phi, \phi), (S \to P, \phi, \phi), (P \to QD, \phi, C), (P \to aD, \phi, C), (Q \to PC, \phi, D),$

$(Q \to aC, \phi, D), (C \to DD, PC, \phi), (D \to CC, QD, \phi), (D \to a, a, \phi)\}$

Consider string $w = aaaaaaaa$

$S \Rightarrow P \Rightarrow QD \Rightarrow QCC \Rightarrow PCCC \Rightarrow PDDCC \Rightarrow PDDCDD \Rightarrow PDDDDDD \Rightarrow aDDDDDDD$
$\Rightarrow aDDaDDDD \Rightarrow^6 aaaaaaaa$

$L(G_{ssc}) = \{a^{2^n} \mid n \geq 0\}$

Kljein [50-51] introduced the concept of Selective Substitution Grammar (SSG). In selective substitution grammars, there is a finite set of the selectors of the form $A^* B' C^*$. The rule $\alpha A \beta \to \alpha x \beta$ is applied if and only if, $\alpha \in A$, $\beta \in C$ and $A \in B$ of the selector $A^* B' C^*$.

Meduna et al. [52] proved the equivalence between one-sided forbidding grammars and selective substitution grammars with and without erasing rules.

Table 2.1 shows the generative power of context-based regulated grammars.

Table 2.1: Comparative table of generative power for various context-based regulated grammars. Here *CC, CFG, CS, Per, For, RCG, RE, OSRC, OSPRG, OSFRG, GFG, GOSFG, SSC, SSCG, SSG* represents Context-conditional grammar, context-free grammar, context-sensitive grammar, permitting grammar, forbidding grammar, random context grammar, recursively enumerable, one-sided random-context grammar, one-sided permitting grammar, one-sided forbidding grammar, generalized forbidding grammar, generalized one-sided forbidding grammar, semi-conditional grammar, simple semi-conditional grammar, and selective substitution grammar. Propagating versions of these grammars are denoted by superscript P.

Ref.	**Property →** / **Type ↓**	**Generative Power**
Navratil [42]	Context-conditional Grammar	$CC^P(0,0) = CC(0,0) = CFG;\ CC^P \subseteq CS;$ $CC \subseteq RE;$
Walt [43]	Random-context Grammar	$CFG \subset Per^P \subset RCG^P \subset CS;$ $Per^P = Per \subset RCG = RE;$ $CFG \subset For^P \subseteq For \subset CS;$
Walt [43]	Permitting Grammar	$CFG \subset Per^P \subset RCG^P \subset CS;$ $Per^P = Per \subset RCG = RE;$ $CFG \subset For^P \subseteq For \subset CS;$
Walt [43]	Forbidding Grammar	$CFG \subset Per^P \subset RCG^P \subset CS;$ $Per^P = Per \subset RCG = RE;$ $CFG \subset For^P \subseteq For \subset CS;$
Meduna et al.[44]	One-sided Random-context Grammar	$OSRC^P = CS; OSRC = RE; OSRC \subseteq CS$
Meduna et al.[44]	One-sided Permitting Grammar	$CFG \subseteq OSPRG^P \subseteq OSPRG; CFG \subset OSPRG^P \subseteq CS$
Meduna et al.[44]	One-sided Forbidding Grammar	$For^P \subseteq OSFRG^P; OSFRG^P = CS; OSFRG = RE$
Meduna et al. [45]	One-sided Random-context Grammar with a limited number of right random context rules	$OSRCG = RE$ where cardinality of right random context rules is not more than 2.
Meduna [46]	Generalized Forbidding Grammar	$GFG(0) = CFG; GFG(1) = ForG; GFG(2) = RE$
Meduna et al. [47]	Generalized one-sided Forbidding Grammar	$GOSFG(n,0) = CFG; GOSFG(0,n) = CFG;$ $CFG \subset GOSFG(1,1); GOSFG(2,1) \subseteq RE$

Table 2.1 (Contd.)		
Paun [48]	Semi-conditional Grammar	$SCG^{p}(0,0) = SCG(0,0) = CFG; CFG \subset SCG^{P}(1,0);$ $CFG \subset SCG^{P}(0,1); SCG^{P}(1,1) \subset CS; SCG^{P}(1,1)$ $\subseteq RCG^{P} \subset CS;$ $CF \subset SCG^{P}(2,1) = SCG^{P}(1,2) = SCG^{P} = CS$ $\subset SCG(2,1) = SCG(1,2) = SCG = RE$
Meduna et al. [49]	Simple Semi-conditional Grammar	$SSCG^{P}(2,1) = CS; SSCG(2,1) = RE$
Kljein [50-51]	Selective Substitution Grammar	$L(SSG) = L(OSFRG); L(SSG)^{P} = L(OSFRG)^{P}$
Meduna et al. [52]	One-sided Forbidding Grammar and Selective Substitution Grammar	$L(SSG) = L(OSFRG); L(SSG)^{P} = L(OSFRG)^{P}$

2.1.2 RULE-BASED GRAMMATICAL REGULATION

In rule-based grammatical regulation, the derivation of a string is regulated by application of rules. Various types of rule-based regulated grammar are depicted in Fig. 2.3.

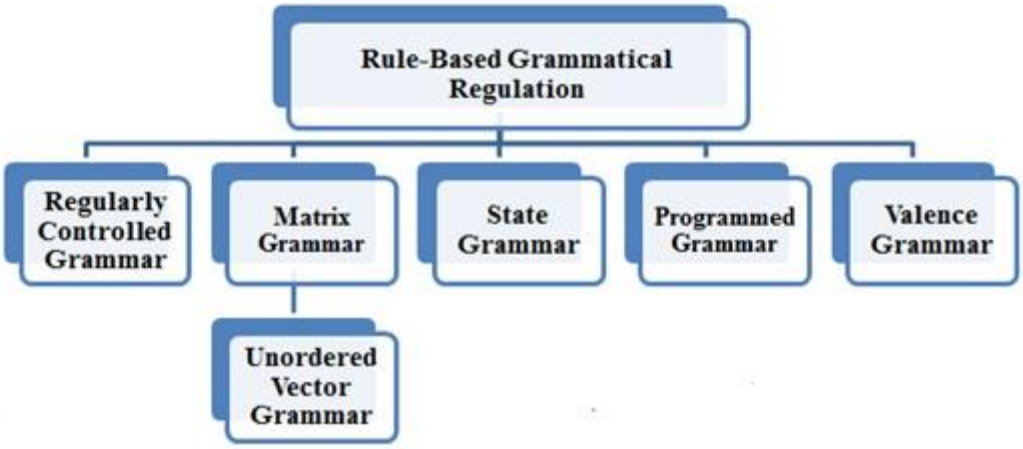


Fig. 2.3: Classification of rule-based regulated grammars

The following terminology is used in the rule-based grammatical regulation:

- **Appearance Checking**: If left-hand side of a production rule is not in the current sentential form and the same rule is in appearance-checking mode then we can skip the application of this rule and move to the next rule in the production set.
- **Without Appearance Checking**: Once a derivation process is started, all the production rules need to be applied as given in the restriction set.

Ginsburg et al. [6] introduced the concept of regularly controlled grammar. In regularly controlled grammar, the restriction is in the form of regular expression.

EXAMPLE 2.12 [3]: Let $G_R(\{S, C, D\}, \{c, d\}, S, P, R)$ be regularly controlled grammar where the production rules *P* are defined by

$\{pr_0 = S \to CD,\ pr_1 = C \to cC,\ pr_2 = D \to cD,\ pr_3 = C \to dC,\ pr_4 = D \to dD,$

$pr_5 = C \to c,\ pr_6 = D \to c,\ pr_7 = C \to d,\ pr_8 = D \to d\}$

$R = pr_0(pr_1pr_2,\ pr_3pr_4)^*(pr_5pr_6,\ pr_7pr_8)$

Consider $w = cdcd$

$S_{pr0} \to CD_{pr1} \to cCD_{pr2} \to cCcD_{pr7} \to cdcD_{pr8} \to cdcd$

$L(G_R) = \{ww \mid w \in \{c, d\}^+\}$

Cremers et al. [7] defined the concept of matrix grammar. In matrix grammar, all production rules of a matrix are applied in a prescribed sequence before starting another matrix.

EXAMPLE 2.13 [3]: Consider the matrix grammar $G_M(\{S, C, D\}, \{c, d\}, S, \{m_0, m_1, m_2, m_3, m_4\})$ without appearance checking where,

$m_0 = (p_0 = S \to CD),$

$m_1 = (p_1 = C \to cC,\ p_2 = D \to cD),$

$m_2 = (p_3 = C \to dC,\ p_4 = D \to dD),$

$m_3 = (p_5 = C \to c,\ p_6 = D \to c),$

$m_4 = (p_7 = C \to d,\ p_8 = D \to d).$

The control set is $m_0(m_1, m_2)^*(m_3, m_4)$.

Consider $w = dcdc$

$S \overset{m_0:p_0}{\to} CD \overset{m_2:p_3}{\to} dCD \overset{m_2:p_4}{\to} dCdD \overset{m_3:p_5}{\to} dcdD \overset{m_3:p_6}{\to} dcdc$.

$L(G_M) = \{ww \mid w \in \{c, d\}^+\}$

Cremers et al. [53] defined unordered vector grammar. Unordered vector grammar can be considered as a matrix grammar, where rules are applied in any arbitrary order.

EXAMPLE 2.14 [1]: Consider the unordered vector grammar without appearance checking $G_{uv}(\{S, C, D\}, \{c, d\}, S, \{m_0, m_1, m_2, m_3, m_4\})$ where,

$m_0 = (p_0 = S \to CD),$

$m_1 = (p_1 = C \to cC,\ p_2 = D \to cD),$

$m_2 = (p_3 = C \to dC,\ p_4 = D \to dD),$

$m_3 = (p_5 = C \to c,\ p_6 = D \to c),$

$m_4 = (p_7 = C \to d,\ p_8 = D \to d).$

$L(G_{uv}) = \{wxw'x : x \in \{c, d\},\ w \in \{c, d\}^*, w' \in Permutation(w)\}$

Kasai [8] proposed state grammar. In state grammar, regulation is specified in terms of states. In each derivation, a non-terminal can be rewritten under the present state and during each derivation it changes its state by making a transition from one state to another state and determines which next rule needs to be applied.

EXAMPLE 2.15: Consider the state grammar $G_S(\{S, A, B, 0, 1, 2\}, \{p_0, p_1, p_2\}, \{0, 1, 2\}, P, S)$ where the production rules are

$(p_0, S) \to (p_0, AB)$

$(p_0, A) \to (p_1, 0A1)$ $\qquad$ $(p_1, B) \to (p_0, 2B)$

$(p_0, A) \to (p_2, 01)$ $\qquad$ $(p_2, B) \to (p_0, 2)$

Consider the string $w = 000111222$

$S_{p0} \to AB_{p0} \to 0A1B_{p1} \to 0A12B_{p0} \to 00A112B_{p1} \to 00A1122B_{p0} \to 00011122B_{p2}$

$\to 000111222_{p0}$

The non-context-free language generated by G_s is $L(G_S) = \{0^n1^n2^n \mid n \geq 1\}$.

Rosenkrantz [54-55] introduced the concept of Programmed Grammar. Programmed grammar is a generalization of phrase structure grammars [56-57]. In programmed grammar, there are two types of sets named as success field and failure field set. Once the derivation is started the next production is chosen from the success field set, but if the

left-hand side is not in the sentential form, then the next production is chosen from the failure field set.

EXAMPLE 2.16 [1]: Consider the programmed grammar (without appearance checking) $G_{pr}(\{S, C, D\}, \{c, d\}, S, \{rp_0, rp_1, rp_2, ..., rp_8\})$, where

$rp_0 = (S \to CD, \{rp_1, rp_3, rp_5, rp_7\}, \phi)$, $rp_1 = (C \to cC, \{rp_2\}, \phi)$

$rp_2 = (D \to cD, \{rp_1, rp_3, rp_5, rp_7\}, \phi)$, $rp_3 = (C \to dC, \{rp_4\}, \phi)$

$rp_4 = (D \to dD, \{rp_1, rp_3, rp_5, rp_7\}, \phi)$, $rp_5 = (C \to c, \{rp_6\}, \phi)$

$rp_6 = (D \to c, \phi, \phi)$, $rp_7 = (C \to d, \{rp_8\}, \phi)$

$rp_8 = (D \to d, \phi, \phi)$

$L(G_{pr}) = \{ww \mid w \in \{c, d\}^+\}$.

Paun [58-59] introduced valence grammar. In valence grammar, restrictions are in the form of a monoid, either additive or multiplicative. Productions are of the form $(\alpha \to \beta, m)$, where m is the additive or multiplicative monoid of integers or rational numbers, respectively.

EXAMPLE 2.17 [1]: Consider $G_{AV}(\{S, C, D\}, \{c, d\}, S, \{p_0, p_1, ..., p_8\}, \{Z+\})$ be the additive grammar, where P is defined by

$p_0 = (S \to CD, 0)$, $p_1 = (C \to cC, 2)$, $p_2 = (D \to cD, -2)$, $p_3 = (C \to dC, 3)$,
$p_4 = (D \to dD, -3)$, $p_5 = (C \to c, 2)$, $p_6 = (D \to c, -2)$, $p_7 = (C \to d, 3)$ $p_8 = (D \to d, -3)$

$L(G_{AV}) = \{w_1 w_2 \mid w_1 w_2 \in \{c, d\}^+, 2_{\#}c(w_1) + 3_{\#}d(w_1) = 2_{\#}c(w_2) + 3_{\#}d(w_2)\}$

EXAMPLE 2.18 [1]: Consider $G_{MV}(\{S, C, D\}, \{c, d\}, S, \{p_0, p_1, ..., p_8\}, \{Q\})$ be the multiplicative grammar where P is defined by

$p_0 = (S \to CD, 1)$, $p_1 = (C \to cC, 2)$, $p_2 = (D \to cD, 1/2)$, $p_3 = (C \to dC, 3)$,
$p_4 = (D \to dD, 1/3)$, $p_5 = (C \to c, 2)$, $p_6 = (D \to c, 1/2)$, $p_7 = (C \to d, 3)$, $p_8 = (D \to d, 1/3)$

$L(G_{MV}) = \{w_1 w_2 \mid w_1 \in \{c, d\}^+, w_2 \in Perm(w_1)\}$

Table 2.2: Classification of rule-based regulated grammars. Here *CFG, RCG, MG, UVG, SG, PG, AVG, MVG, AC,* CS are context-free grammar, regularly controlled grammar,

matrix grammar, unordered vector grammar, state grammar, programmed grammar, additive valence grammar, multiplicative valence grammar, appearance checking and context-sensitive grammar, respectively.

Ref.	Type	Generative Power
Ginsburg et al. [6]	Regularly controlled Grammar	$CFG \subset RCG \subset RCG_{AC} \subset CS$
Cremers et al. [7]	Matrix Grammar	$RCG = MG$ $RCG_{AC} = MG_{AC}$ $CFG \subset MG \subset MG_{AC} \subset CS$
Cremers et al. [53]	Unordered Vector Grammar	$CFG \subset UVG \subset MG$
Kasai [8]	State Grammar	$CFG = SG_1 \subset \cdots \subset SG_\infty = CS$
Rosenkrantz [54-55]	Programmed Grammar	$CFG \subset PG \subset PG_{AC} \subset CS$
Paun [58-59]	Valence Grammar	$CFG \subset AVG \subset MVG$

2.1.3 PARALLEL GRAMMATICAL REGULATION

In parallel grammatical regulation, the derivation is regulated by rewriting more than one non-terminal simultaneously. Fig. 2.4 represents various types of parallel regulated grammar.

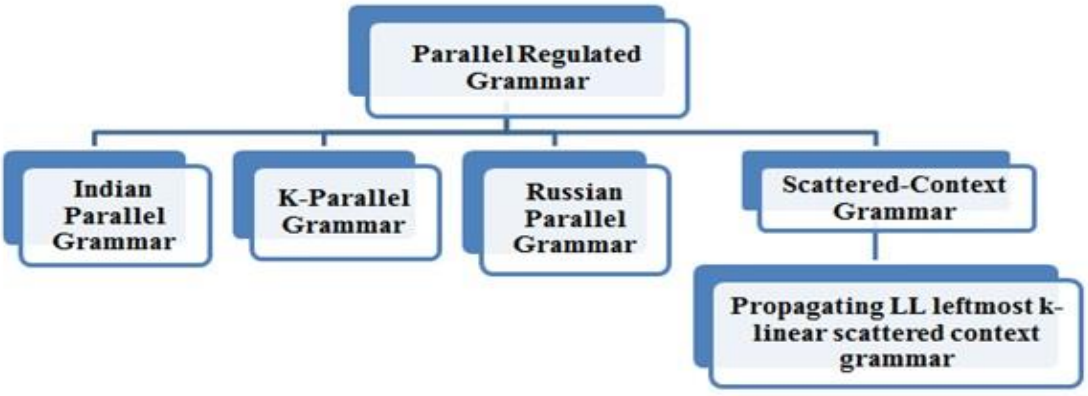


Fig. 2.4: Classification of Parallel regulated grammars

Siromoney [60] introduced the concept of Indian parallel grammar where all occurrences of non-terminals must be written simultaneously.

EXAMPLE 2.19 [60]: Consider the Indian parallel grammar $G_{IP}(\{S\},\{b\}, S, P)$ with the production rules P are $\{(S \to SS), (S \to b)\}$.

For the string $w = bbbbbbbb$,

$S \Rightarrow SS \Rightarrow SSSS \Rightarrow SSSSSSSS \Rightarrow bbbbbbbb$

$L(G_{IP}) = \{b^{2^n} \mid n \geq 0\}$

Saloma [61] introduced the concept of *K*-parallel grammar where *K* numbers of non-terminals should be rewritten simultaneously, except for the starting non-terminal.

EXAMPLE 2.20 [62]: Consider the *K*-parallel grammar $G_{KP}(\{S, P, Q, R\}, \{p, q, r\}, S, P)$ with the production rules *P* are

$\{(S \rightarrow PQR), (P \rightarrow pP), (P \rightarrow p), (Q \rightarrow qQ), (Q \rightarrow q), (R \rightarrow rR), (R \rightarrow r)\}$

For the string $w = pppqqqrrr$,

$S \Rightarrow PQR \Rightarrow pPqQrR \Rightarrow ppPqqQrrR \Rightarrow pppqqqrrr$

$L(G_{KP}) = \{p^n q^n r^n \mid n \geq 0\}$

Dassow [63] defined Russian parallel grammar which lies in between Indian parallel grammar and *K*-parallel grammar. Production rules are of the form: $(A \rightarrow x, K)$, and we apply the rule $A \rightarrow x$ simultaneously to *K* non-terminals. Greibach et al. [64] defined scattered-context grammar. In scattered-context grammar, the matrix includes the production rules that need to be applied to the non-terminals. The rules in a string must be applied in the same order as given in matrix.

EXAMPLE 2.21 [64]: Consider the scattered context grammar $G_{scp}(\{S, X\}, \{p, q, r\}, M, S)$ with the production matrix M is defined by:

$M_1 = (S) \rightarrow (XXX),$

$M_2 = (X, X, X) \rightarrow (pX, qX, rX),$

$M_3 = (X, X, X) \rightarrow (p, q, r)$

For the string $w = pppqqqrrr$,

$S \Rightarrow XXX \Rightarrow pXqXrX \Rightarrow ppXqqXrrX \Rightarrow pppqqqrrr$

$L(G_{scp}) = \{p^n q^n r^n \mid n \geq 0\}$

Meduna et al. [65] defined propagating LL leftmost *k*-linear scattered context grammar. This grammar satisfies the following three conditions: (i) *k*- linear condition (ii) Leftmost condition and (iii) LL condition

Table 2.3 describes different properties of various types of parallel regulated grammar.

Table 2.3: Properties of different types of parallel regulated grammar. Here *CF*, *IPG*, *CS*, *RE*, *KPG*, RPG, SCPG, and LL_kSCPG denote context-free grammar, Indian parallel grammar, context-sensitive grammar, recursively enumerable, *k*-parallel grammar, Russian parallel grammar, scattered-context parallel grammar, and LL leftmost *k*-linear scattered context grammar, respectively. Propagating versions of these grammars are denoted by superscript P.

Ref.	Type	Restriction	Generative Power
Siromoney [60]	Indian Parallel Grammar	All occurrences of non-terminals must be written simultaneously.	$CF \subset IPG^P \subseteq CS \subset IPG = RE$
Saloma [61]	*K*-Parallel Grammar	*K* numbers of non-terminals should be rewritten simultaneously, except for the starting non-terminal.	$CF \subset KPG^P \subseteq CS \subset KPG = RE$
Dassow [63]	Russian Parallel Grammar	Lies in between Indian parallel grammar and *K*-parallel grammar. Production rules are of the form: $(A \to x, K)$, and we apply the rule $A \to x$, simultaneously to *K* non-terminals.	$CF \subset RPG^P \subseteq CS \subset RPG = RE$
Greibach et al. [64]	Scattered-Context Grammar	The matrix includes the production rules that need to be applied to the non-terminals. The rules in a string must be applied in the same order as given in matrix.	$CF \subset SCPG^P \subseteq CS \subset SCPG = RE$
Meduna et al. [65]	Propagating LL leftmost *k*-linear scattered context grammar	This grammar satisfies the following three conditions: i. *k*- linear condition- ii. Leftmost condition iii. LL condition	$LL_{k-1}SCPG \subset LL_kSCPG$, for all $k \geq 2$.

2.2 REGULATED AUTOMATA

In this section, regulated automata counterparts of various regulated grammars have been discussed.

Meduna [10] introduced the concept of DPDA as an extension of conventional PDA. In DPDA, we can push the symbols on the top of the stack as well as on the deeper parts of the stack. On the basis of expansion depth, DPDA coincides with the infinite hierarchy of n-limited state grammar. Solar [11] introduced the concept of PDPDA as an extension to DPDA. It is a parallel version of DPDA. In one move of PDPDA, we can expand *n* topmost non-input symbols. It also coincides with the infinite hierarchy of *n*-limited state grammars.

Krivika et al. [66] proposed the reduced version of the deep pushdown automata. It makes reductions of depth *n* and rewrites the stack substring with the n^{th} topmost non-input symbol in the pushdown. The input string is a part of a pushdown stack in the starting configuration instead of the input tape, and it works similarly as a bottom-up parser.

Leupold et al. [67] proposed the concept of finitely expandable deep pushdown automata. It is an automaton counterpart of matrix grammar and corresponds to the hierarchy of matrix grammars of finite index. Finitely expandable deep pushdown automata contain only a finite number of non-input symbols in the stack.

Meduna et al. [68] proposed the concept of controlled finite automata which coincide with the hierarchy of programmed grammars. It regulates the control of states and transition rules and proved that under certain regulations; regular-controlled finite automata and context-free-controlled finite automata represent regular and context-free languages, respectively. They have specified various conditions using which state-controlled finite automaton can be converted into transition controlled automaton and vice-versa.

Kucera et al. [12] proposed absolutely unlimited deep pushdown automata. Here, push operation can be performed regardless of its depth. Propagating and non-propagating unlimited deep pushdown automata can represent type-1 and type-0 language, respectively.

2.3 REGULATED TRANSDUCER

Solar et al. [33] introduced the concept of a Deep pushdown transducer (DPDT) and Parallel deep pushdown transducers (PDPDT). The DPDT and PDPDT are extended versions of the DPDA and PDPDA, respectively. These regulated automata and transducers are formal models representing some of the context-sensitive languages or unrestricted languages using pushdown automata capabilities with some regulations. It works similarly as DPDA and PDPDA except that an output string is generated corresponding to the given input string.

2.4 FUZZY CONTEXT-FREE GRAMMAR AND FUZZY PUSHDOWN AUTOMATA

Zadeh [41] defines the notion of fuzzy languages which is a fuzzy subset of strings over a finite alphabet. Asveld [15] proposed fuzzy context-free K-grammar to represent erroneous sentences. During each derivation step, they make a finite choice out of the infinity of possible grammatical errors. Formally, a fuzzy context-free *K*-grammar is a fuzzy context-free grammar with a countable *K* number of rules. They proved that on the parameter *K*, the closure properties of a family of languages generated by fuzzy context-free *K*-grammars and family of languages generated by ordinary context-free languages are similar. Asveld [16] proposed a modified Cocke-Younger-Kasami (CYK) algorithm to recognize fuzzy context-free grammar. Jun et al. [17] introduced the variant of fuzzy sets, named as bipolar fuzzy sets, in which membership degree lies between [-1 to 1] rather than [0-1]. Jacob et al. [18] defined the concept of fuzzy entire sequence space.

Bucurescu et al. [19] proposed the concept of fuzzy pushdown automata and B-fuzzy pushdown automata as an extension to Pushdown Automata (PDA). In fuzzy pushdown automata, weights are assigned to the interval [0-1], and it accepts context-free language. In B-fuzzy pushdown automata, weights are assigned using Boolean algebra, and it accepts context-sensitive language. Moraga [20] defined the relationship between fuzzy context-free languages and fuzzy pushdown automata, calculating the degree of membership based on weights as determined by a monoid. Hopcroft et al. [21] demonstrate the equivalence between pushdown automata and context-free grammars. Kakoty et al. [22] proposed a fuzzy logic model to evaluate expertise level of learner in an e-learning environment. Arora et al. [23] proposed an approach which converts state

chart diagram to finite state automata. Further, regular grammars have been generated from finite state automata. Xing [69] proposed one-stack fuzzy pushdown automata and multi-stack fuzzy pushdown automata demonstrating that the languages generated by one stack fuzzy push down automata and multi-stack fuzzy push down automata constituted the language of fuzzy context-free grammars.

Gao et al. [34] diagnose the motor fault using the concept of fuzzy logic. Schinder et al. [36] described an approach for recognizing imperfect strings generated by the context-free grammar. Inui et al. [37] extended the work of Schinder et al. [36], describing an approach for recognizing imperfect string generated by the context-sensitive grammar. Both Schinder et al. [36] and Inui et al. [37] used a number of fuzzy sets to determine the membership values of imperfect strings not belonging to the grammar.

2.5 GRAMMAR FORMALISM FOR DNA AND RNA STRUCTURE SEQUENCES

Sung [26] model various forms of RNA secondary structures, especially pseudoknots such as (hairpins, internal loops, double helixes, and bulge loops) with the help of context-sensitive grammar. Rivas and Eddy [27] represented RNA secondary structure including pseudoknots with the cross interaction grammar. The novel feature of cross-interaction grammar is the use of special non-terminals which helps in specific rearrangements of substrings in a derivation.

Searls [28] represented the structure of nucleic acids with the help of indexed grammar. An indexed grammar uses context-free grammar production rules along with the additional feature of indices that are taken from a special set of symbols, strings of which can be added to non-terminals. Searls [29] proposed a string variable grammar for representing the biological sequences of DNA. Uemura et al. [30] proposed subclass of the tree adjoining grammars to model and predict the RNA structure. A tree adjoining grammar is a kind of formal language that generates a set of trees rather than a set of strings. Cai et al. [25] represented RNA pseudoknot structures with the parallel communicating grammar systems (PCGS). Mizoguchi et al. [31] use the stochastic multiple context-free grammars (SMCFG), to represent various classes of pseudoknots. The probability estimator for the SMCFG is being computed from the aligned sequences.

Sakakibara et al. [70] modeled RNA structure loops using stochastic context-free grammar. Sakakibara [71] also modeled RNA structure loops using pair hidden Markov models. Yuki et al. [72] modeled RNA structure loops using SMCFG. Brown and Wilson [73] modeled RNA pseudoknot structure using the intersection of stochastic context-free grammars. Kuppusamy et al. [32] modeled DNA and RNA biomolecular structures of the intermolecular level, intramolecular level and at RNA secondary level using matrix-insertion deletion grammar.

Anderson et al. [74-75] introduced the concept of stochastic context-free grammar for predicting the RNA structure. Rothemund [76] and Cavaliere et al. [77] proposed a model for defining the action of a restriction enzyme in DNA using a Turing machine and pushdown automata, respectively. Furthermore, Krasinski et al. [78] improved the Cavaliere et al. [77] model using circular pushdown automata in which stack and input tape are on the same circular strand.

2.6 GAPS IN LITERATURE

Based on the literature survey, some of the research gaps identified for proposed research work of regulated grammar and regulated automata are presented below:

1. In regulated languages, the input string is either accepted or rejected, but there is a chance that the user generates some error in the string. Hence, there is a need to introduce a mechanism by which string is not totally accepted or rejected, rather belong to the language with some certainty. It leads to the introduction of the concept of fuzziness in regulated language.
2. Literature study reveals that the biological structure of DNA and RNA has highly expressive language, i.e. DNA and RNA nucleotides, based on the complementary pairs in chemical objects and other biological constraints, form patterns or structures and these structures highly correlates with the linguistic theory. There are some RNA structures like hairpin which matches with language ww^R (the language of palindrome) which requires context-free grammar. There are some other structures like attenuator $\{u\bar{u}^R u\bar{u}^R \mid u \in \sum^*_{DNA}\}$ which matches with ww (copy language) which require power beyond context-free grammar. There are various other biological structures such as extended pseudoknot, recursive pseudoknot, simple h-type, kissing hairpin, three-knot which includes crossing

dependencies, which also requires power beyond context-free grammar. Chomsky grammar systems are found ideal to model interactions of the biological structure of DNA and RNA. In formal language theory, little attention has been given to these sequences. The majority of works related to these sequences are represented using context-sensitive grammar or mildly context-sensitive with higher time complexities.

3. Most of the regulated automata and regulated transducer which exist such as deep pushdown automata, parallel deep pushdown automata, deep pushdown transducer and parallel deep pushdown transducer work non-deterministically.

2.7 PROBLEM FORMULATION

In this section problem statement and objective of our work has been discussed.

2.7.1 PROBLEM STATEMENT

The research proposed in this work is design and development of a variant of regulated grammar or regulated automata and to apply their concept in RNA and DNA bio-molecular structures. It will be an extension of regulated grammar and regulated automata to express uncertainty, imprecision, and vagueness in natural language fragments that will make regulated grammar or regulated automata more robust. The concept of regulated grammar and regulated automata is applied in the interdisciplinary field of biology. It will help to parse the biological structures with lesser time complexity.

2.7.2 OBJECTIVES

Following are the objectives of this dissertation:

1. To explore the various types of regulated grammars and regulated automata.
2. To design and develop a variant of regulated grammar or regulated automaton.
3. To validate the proposed variant of regulated grammar or regulated automaton.
4. To apply the concept of regulated grammar and regulated automaton in the field of biology.

In addition to the above-mentioned objectives, following objectives are also achieved:

1. Novel concept of deterministic deep pushdown automata, deterministic deep pushdown transducer and their parallel versions named as parallel deterministic

deep pushdown automata and parallel deterministic deep pushdown transducer has been proposed.

2. Design of an algorithm for error tolerance in recognition of faulty strings in a regulated grammar using fuzzy sets (*RFSRG*).

Chapter-3

FUZZY STATE GRAMMAR AND FUZZY DEEP PUSHDOWN AUTOMATA

3.1 INTRODUCTION

In formal language [13-14], input string w can either be accepted $(\mu_w = 1)$ or rejected $(\mu_w = 0)$. Fuzziness has introduced into regulated grammar in order to close the gap between the polar accept and reject extremes, and to make the grammar more robust in terms of faulty string acceptance. However, a robust parser may be required in the event of errors being present in the input. The input string will belong to a language with certainty between 0 and 1. The need and importance of developing fuzzy languages and fuzzy automata has been explored previously in the literature [79-82].

Motivated by the similarities between fuzzy finite automata [24] and fuzzy pushdown automata [20], this chapter explores the concepts of fuzzy regulated grammar and fuzzy regulated automata. State grammar is one of the types of rule-based regulated grammar and deep pushdown automaton is the regulated automaton counterpart of state grammars. In this chapter, the concept of fuzzy state grammar and fuzzy deep pushdown automaton are introduced. Fuzzy state grammar is a generalization of state grammar, whereas a fuzzy deep pushdown automaton is a generalization of the deep pushdown automaton. It has been proved that we can construct fuzzy deep pushdown automata M_{fd} from fuzzy state grammars G_{fs} and vice-versa, and if the fuzzy deep pushdown automaton M_{fd} is constructed from the fuzzy state grammar G_{fs} using the above construction method, then $L(M_{fd}) = L(G_{fs})$.

3.2 PROPOSED FUZZY STATE GRAMMAR

In this section, we introduce the concept of fuzzy state grammar.

Results of this chapter has been published in Journal of Intelligent & Fuzzy Systems
N. Kalra and A. Kumar, Fuzzy state grammar and fuzzy deep pushdown automaton, Journal of Intelligent & Fuzzy Systems, 31 (1), 249-258, 2016.

DEFINITION 3.2.1. A fuzzy state grammar G_{fs} is septuple $(V, Q, \Sigma, P, S, E, \mu)$, where

- $(V, Q, \Sigma, P, S\}$ is a state grammar [8] specified in definition 1.2.7,
- E is the set of error productions, and
- $\mu : P \to [0, 1]$ is a weighting function. The set $\{(p, \mu(p)) \mid p \in P\}$ constitutes a fuzzy set of productions.

DEFINITION 3.2.2. A fuzzy state language is given by the triple $(G, \min, \max)$ where G is a fuzzy state grammar. The degree of membership of a string is calculated using the min operation. Min operation is applied to the weights of all productions used to derive a particular string from the initial symbol.

It holds: $\mu_{L(G)}(\alpha) = \min(\mu(p_i), \mu(p_j), ..., \mu(p_z))$ (1)

where $p_i, p_j, ..., p_z \in P$ are productions used to derive the string α.

If the grammar is ambiguous, then different degrees of membership obtained for the string α using different sequences of productions. Max operation is applied to determine the best membership for the string.

$$\mu_{L(G)}(\alpha) = \max\{\min(\mu(p_i), \mu(p_j), ..., \mu(p_z))\} \quad (2)$$

EXAMPLE 3.1: Consider the fuzzy state grammar $G_{fs}(\{S, A, B, a, b\}, \{p_0, p_1, p_2\}, \{a, b\}, P, S, E, \mu)$ where E is the set of error productions, $\mu : P \to [0, 1]$ is a weighting function.

The set $\{(p, \mu(p)) \mid p \in P\}$ constitutes a fuzzy set of productions with the following rules in P:

$(p_0, S) \xrightarrow{1} (p_0, AB)$

$(p_0, A) \xrightarrow{1} (p_1, aAb)$ $\quad$ $(p_1, B) \xrightarrow{1} (p_0, Bc)$

$(p_0, A) \xrightarrow{1} (p_2, ab)$ $\quad$ $(p_2, B) \xrightarrow{1} (p_0, c)$

The non context-free language generated $L(G_{fs}) = \{a^n b^n c^n \mid n \geq 1\}$.

Errors in the input string are classified into following three types:

- **Replacement Error:** In replacement error [36-37], a terminal is substituted with another terminal symbol from the set $\sum$. The certainty of occurrence of replacement error is maximum as tested from different erroneous input dataset. Hence the membership grade of replacement error is 0.6.
- **Addition Error:** In addition error [36-37], an extra new terminal symbol is added from the terminal set $\sum$. An extra new terminal can be placed before or after the terminal. The certainty of addition error occurrence is minimum with membership grade 0.3.
- **Removal Error:** In removal error [36-37], a terminal is replaced by the empty string $\wedge$. The membership grade of removal error is 0.4.

The set of error productions *E* is as follows:

1. On applying replacement error: (Here _ denotes that it can be replaced by any arbitrary symbol from alphabet $\sum$)

$$(p_0,\ A) \overset{0.6}{\rightarrow} (p_1,\ \_Ab)$$

$$(p_0,\ A) \overset{0.6}{\rightarrow} (p_1,\ aA\_)$$

$$(p_0,\ A) \overset{0.6}{\rightarrow} (p_1,\ \_A\_)$$

$$(p_1,\ B) \overset{0.6}{\rightarrow} (p_0,\ B\_)$$

$$(p_0,\ A) \overset{0.6}{\rightarrow} (p_2,\ \_b)$$

$$(p_0,\ A) \overset{0.6}{\rightarrow} (p_2,\ a\_)$$

$$(p_0,\ A) \overset{0.6}{\rightarrow} (p_2,\ \_\ \_\ )$$

$$(p_2,\ B) \overset{0.6}{\rightarrow} (p_0,\ \_)$$

2. On applying addition error:

$$(p_0,\ A) \overset{0.3}{\rightarrow} (p_1,\ \_a\_Ab)$$

$$(p_0,\ A) \overset{0.3}{\rightarrow} (p_1,\ aA\_b\_)$$

$$(p_0,\ A) \overset{0.3}{\rightarrow} (p_1,\ \_a\_A\_b\_)$$

$$(p_1,\ B) \overset{0.3}{\rightarrow} (p_0,\ B\_c\_)$$

$(p_0,\ A) \overset{0.3}{\rightarrow} (p_2,\ \_a\_b)$

$(p_0,\ A) \overset{0.3}{\rightarrow} (p_2,\ a\_b\_)$

$(p_0,\ A) \overset{0.3}{\rightarrow} (p_2,\ \_a\_\_b\_)$

$(p_2,\ B) \overset{0.3}{\rightarrow} (p_0,\ \_c\_)$

3. On applying removal error:

$(p_0,\ A) \overset{0.4}{\rightarrow} (p_1,\ Ab)$

$(p_0,\ A) \overset{0.4}{\rightarrow} (p_1,\ aA)$

$(p_0,\ A) \overset{0.4}{\rightarrow} (p_1,\ A)$

$(p_1,\ B) \overset{0.4}{\rightarrow} (p_0,\ B)$

$(p_0,\ A) \overset{0.4}{\rightarrow} (p_2,\ b)$

$(p_0,\ A) \overset{0.4}{\rightarrow} (p_2,\ a)$

$(p_0,\ A) \overset{0.4}{\rightarrow} (p_2,\ \wedge)$

$(p_2,\ B) \overset{0.4}{\rightarrow} (p_0,\ \wedge)$

Consider unambiguous erroneous string $\alpha = abbbcc$ such that $\alpha \notin L$

$$S_{p_0} \overset{1}{\rightarrow} AB_{p_0} \overset{1}{\rightarrow} aAbB_{p_1} \overset{1}{\rightarrow} aAbBc_{p_0} \overset{0.6}{\rightarrow} abbbBc_{p_2} \overset{1}{\rightarrow} abbbcc_{p_0}$$

$$\mu(abbbcc) = \min(1;\ 1;\ 1;\ 0.6;\ 1) = 0.6$$

Consider ambiguous erroneous string $\alpha = aabacc$ such that $\alpha \notin L$

Derivation 1: $S_{p_0} \overset{1}{\rightarrow} AB_{p_0} \overset{0.6}{\rightarrow} aAaB_{p_1} \overset{1}{\rightarrow} aAaBc_{p_0} \overset{1}{\rightarrow} aabaBc_{p_2} \overset{1}{\rightarrow} aabacc_{p_0}$

$$\mu(aabacc) = \min(1;\ 0.6;\ 1;\ 1;\ 1) = 0.6$$

Derivation 2: $S_{p_0} \overset{1}{\rightarrow} AB_{p_0} \overset{0.3}{\rightarrow} aAbaB_{p_1} \overset{1}{\rightarrow} aAbaBc_{p_0} \overset{0.4}{\rightarrow} aabaBc_{p_2} \overset{1}{\rightarrow} aabacc_{p_0}$

$$\mu(aabacc) = \min(1;\ 0.3;\ 1;\ 0.4;\ 1) = 0.3$$

The membership of erroneous string $\alpha = aabacc$ is:

max (min (Derivation 1, Derivation 2)= max(0.6; 0.3) = 0.6

Hence derivation 1 is the optimal derivation with membership grade $\mu = 0.6$.

3.3 PROPOSED FUZZY DEEP PUSHDOWN AUTOMATON

In this section, we introduce the concept of the fuzzy deep pushdown automaton.

DEFINITION 3.3.1. A fuzzy deep pushdown automaton (FDPDA) M_{fd} is a duodecuple $(Q, \Gamma, \Sigma, R, q_0, S, F, E, \mu, \pi, \gamma, \eta)$, where $(Q, \Gamma, \Sigma, R, q_0, S, F)$ are as usual in deep pushdown automata [10] specified in definition 1.2.9, and

- E is the error set of productions,
- $\mu : P \to [0,1]$ is a weighting function,
- The set $\{p, (\mu(p)) \mid p \in P\}$ consists of fuzzy productions of the form

  $Q \times \Sigma^* \times \Gamma^* \xrightarrow{\mu} Q \times \Gamma^*$
- $\pi : Q \to [0,1]$ is an initial state function defined by

$$\pi(q) = \begin{Bmatrix} 1, & q = q_0 \\ 0, & q \neq q_0 \end{Bmatrix}$$

- $\gamma : \Gamma \to [0,1]$ is a pushdown function defined by

$$\gamma(\Gamma) = \begin{Bmatrix} 1, & \Gamma = S \\ 0, & \Gamma \neq S \end{Bmatrix}$$

- $\eta : Q \to [0,1]$ is a final state function, defined by

$$\eta(F) = \begin{Bmatrix} 1, & f \in F \\ 0, & f \neq F \end{Bmatrix}$$

A configuration of M_{fd} is a triple in $Q \times \Sigma^* \times (\Gamma - \{\#\})^* \{\#\} \xrightarrow{\mu} Q \times \Gamma^*$

EXAMPLE 3.2: Let us consider fuzzy deep pushdown automaton $M_{fd}(\{q_0, q, p\}, \{A, S, \#\}, \{a, b, c\}, R, q_0, S, \{f\}, E, \mu, \pi, \gamma, \eta)$ with R containing rules

$\delta(1, q_0, S) \xrightarrow{1} (q, AA)$, $\delta(1, q, A) \xrightarrow{1} (p, aAb)$,

$\delta(1, q, A) \xrightarrow{1} (f, ab)$, $\delta(2, p, A) \xrightarrow{1} (q, Ac)$,

$\delta(1,\ f,\ A) \overset{1}{\rightarrow} (f, c)$

The set of error productions *E* are:

$\delta(1, q, A) \overset{0.6}{\rightarrow} (p, \_\ Ab)$ $\quad$ $\delta(1, q, A) \overset{0.6}{\rightarrow} (p, aA\_)$

$\delta(1, q, A) \overset{0.6}{\rightarrow} (p, \_\ A\_)$ $\quad$ $\delta(1, q, A) \overset{0.3}{\rightarrow} (p, \_\ a\_\ Ab)$

$\delta(1, q, A) \overset{0.3}{\rightarrow} (p, \_\ a\_\ A\_\ b\_)$ $\quad$ $\delta(1, q, A) \overset{0.3}{\rightarrow} (p, aA\_\ b\_)$

$\delta(1, q, A) \overset{0.4}{\rightarrow} (p, Ab)$ $\quad$ $\delta(1, q, A) \overset{0.4}{\rightarrow} (p, aA)$

$\delta(1, q, A) \overset{0.4}{\rightarrow} (p, A)$ $\quad$ $\delta(2,\ p, A) \overset{0.6}{\rightarrow} (q, A\_)$

$\delta(2,\ p, A) \overset{0.3}{\rightarrow} (q, A\_c\_)$ $\quad$ $\delta(2,\ p, A) \overset{0.4}{\rightarrow} (q, A)$

$\delta(1, q, A) \overset{0.6}{\rightarrow} (f, a\_)$ $\quad$ $\delta(1, q, A) \overset{0.6}{\rightarrow} (f, \_b)$

$\delta(1, q, A) \overset{0.6}{\rightarrow} (f, \_\ \_)$ $\quad$ $\delta(1, q, A) \overset{0.3}{\rightarrow} (f, \_a\_b)$

$\delta(1, q, A) \overset{0.3}{\rightarrow} (f, a\_b\_)$ $\quad$ $\delta(1, q, A) \overset{0.3}{\rightarrow} (f, \_a\_\_b\_)$

$\delta(1, q, A) \overset{0.4}{\rightarrow} (f, a)$ $\quad$ $\delta(1, q, A) \overset{0.4}{\rightarrow} (f, b)$

$\delta(1, q, A) \overset{0.4}{\rightarrow} (f, \wedge)$ $\quad$ $\delta(1,\ f,\ A) \overset{0.6}{\rightarrow} (f, \_)$

$\delta(1,\ f,\ A) \overset{0.3}{\rightarrow} (f, \_c\_)$ $\quad$ $\delta(1,\ f,\ A) \overset{0.4}{\rightarrow} (f, \wedge)$

Pop operation is applied whenever the topmost symbol from the stack and current symbol under read/write (R/W) head matches, keeping the state same and the membership value is 1 for this rule.

Consider unambiguous erroneous string $\alpha = abbbcc \notin L$

$(q_0, abbbcc, S\#) \overset{1}{\Rightarrow}_e (q, abbbcc, AA\#) \overset{1}{\Rightarrow}_e (p, abbbcc, aAbA\#) \overset{1}{\Rightarrow}_p (p, bbbcc, AbA\#) \overset{1}{\Rightarrow}_e$
$(q, bbbcc, AbAc\#) \overset{0.6}{\Rightarrow}_e (f, bbbcc, bbbAc\#) \overset{1}{\Rightarrow}_p (f, bbcc, bbAc\#) \overset{1}{\Rightarrow}_p (f, bcc, bAc\#) \overset{1}{\Rightarrow}_p$
$(f, cc, Ac\#) \overset{1}{\Rightarrow}_e (f, cc, cc\#) \overset{1}{\Rightarrow}_p (f, c, c\#) \overset{1}{\Rightarrow}_p (f, \wedge, \#)$

$$\mu(abbbcc) = \min(1;1;1;1;0.6;1;1;1;1;1;1) = 0.6$$

Consider ambiguous erroneous string $\alpha = aabacc \notin L$

Derivation 1:

$$(q_0, aabacc, S\#) \overset{1}{\Rightarrow}_e (q, aabacc, AA\#) \overset{0.6}{\Rightarrow}_e (p, aabacc, aAaA\#) \overset{1}{\Rightarrow}_p (p, abacc, AaA\#) \overset{1}{\Rightarrow}_e$$
$$(q, abacc, AaAc\#) \overset{1}{\Rightarrow}_e (f, abacc, abaAc\#) \overset{1}{\Rightarrow}_p (f, bacc, baAc\#) \overset{1}{\Rightarrow}_p (f, acc, aAc\#) \overset{1}{\Rightarrow}_p$$
$$(f, cc, Ac\#) \overset{1}{\Rightarrow}_e (f, cc, cc\#) \overset{1}{\Rightarrow}_p (f, c, c\#) \overset{1}{\Rightarrow}_p (f, \wedge, \#)$$

$$\mu(aabacc) = \min(1;0.6;1;1;1;1;1;1;1;1;1) = 0.6$$

Derivation 2:

$$(q_0, aabacc, S\#) \overset{1}{\Rightarrow}_e (q, aabacc, AA\#) \overset{0.3}{\Rightarrow}_e (p, aabacc, aAbaA\#) \overset{1}{\Rightarrow}_p (p, abacc, AbaA\#) \overset{1}{\Rightarrow}_e$$
$$(q, abacc, AbaAc\#) \overset{0.4}{\Rightarrow}_e (f, abacc, abaAc\#) \overset{1}{\Rightarrow}_p (f, bacc, baAc\#) \overset{1}{\Rightarrow}_p (f, acc, aAc\#) \overset{1}{\Rightarrow}_p$$
$$(f, cc, Ac\#) \overset{1}{\Rightarrow}_e (f, cc, cc\#) \overset{1}{\Rightarrow}_p (f, c, c\#) \overset{1}{\Rightarrow}_p (f, \wedge, \#)$$

$$\mu(aabacc) = \min(1;0.3;1;1;0.4;1;1;1;1;1;1) = 0.3$$

The optimal derivation is max (derivation1; derivation 2) = max (0.6; 0.3) = 0.6

3.3.2 INSTANTANEOUS DESCRIPTION

The fuzzy deep pushdown automaton M_{fd} works similarly as deep pushdown automaton [10] except that for every rule some membership value is associated. Configuration or instantaneous description of M_{fd} is described by a triple (q, α, T), where q, α, T denotes the current state, remaining input tape symbols and stack contents respectively.

Initial instantaneous description of M_{fd} is $(q_0, \alpha, S\#)$, where q_0, α, S denotes the starting state, input string and stack starting symbol, respectively. Move from one instantaneous description to another instantaneous description will be carried out by expansion or pop operation.

Consider ID_1 and ID_2 are two successive instantaneous descriptions of M_{fd} and we can move from ID_1 to ID_2 using the following two operations:

1. **Pop operation:** Consider $ID_1(p, b\beta, b\mathrm{T})$ and $ID_2(p, \beta, \mathrm{T})$ are two instantaneous descriptions such that $p \in Q,\ b \in \Sigma, \beta \in \Sigma^*$ and $T \in \Gamma^*$. This operation is represented by $ID_1 \overset{\mu_i}{\Rightarrow}_p ID_2$.
2. **Expansion operation:** Consider $ID_1(p, \beta, \alpha A\mathrm{T})$ and $ID_2(q, \beta, \alpha v\mathrm{T})$ are two successive instantaneous descriptions such that $p, q \in Q, \beta \in \Sigma^*, \alpha, T \in \Gamma^*$ and $(mpA \overset{\mu_i}{\rightarrow} qv) \in R$. It should be noted that α consists of $(m-1)$ non-terminals. This operation is represented by $ID_1 \overset{\mu_i}{\Rightarrow}_e ID_2$.

3.4 RESULTS AND DISCUSSIONS

In this section, we discuss the equivalence between fuzzy state grammar and a fuzzy deep pushdown automaton.

THEOREM 1: Let $G_{fs}(V, Q, \Sigma, P, S, E, \mu)$ be a fuzzy state grammar, then we can construct a fuzzy deep pushdown automaton $M_{fd}\ (Q, \Gamma, \sum, R, q_0, S, F, E, \mu, \pi, \gamma, \eta)$ such that $L(G_{fs}) = L(M_{fd})$ i.e. for any $\alpha \in \Sigma^*,\ \mu(\alpha; \alpha \in L(G_{fs})) = \mu(\alpha; \alpha \in L(M_{fd}))$.

Proof. We can construct the fuzzy deep pushdown automaton M_{fd} from fuzzy state grammar using the following procedure:

The distribution functions [19] are defined by

$$\pi(q) = \begin{Bmatrix} 1, & q = q_0 \\ 0, & q \neq q_0 \end{Bmatrix}$$

$$\gamma(\Gamma) = \begin{Bmatrix} 1, & \Gamma = S \\ 0, & \Gamma \neq S \end{Bmatrix}$$

$$\eta(F) = \begin{Bmatrix} 1, & f \in F \\ 0, & f \neq F \end{Bmatrix}$$

Consider the production of starting symbol in the fuzzy state grammar $(pr_0, S) \overset{\mu_i}{\rightarrow} (pr_0, ABC...)$, its corresponding equivalent fuzzy deep pushdown automaton rule is $1pr_0S \overset{\mu_i}{\rightarrow} pr_0XXX....$ The depth of a rule in the fuzzy deep pushdown automaton depends on how much deep the non-terminal is presenting on the stack. For example,

Consider $(pr_0, S) \overset{\mu_i}{\rightarrow} (pr_0, AB)$ production in the fuzzy state grammar, the corresponding starting state rule in the fuzzy deep pushdown automaton will be $1pr_0S \overset{\mu_i}{\rightarrow} pr_0AA$ and the fuzzy deep pushdown automaton will be constructed of depth 2.

The transition function *R* is defined by:

1. Given the production rule $(q_0, S) \overset{1}{\rightarrow} (p, \alpha)$ in the fuzzy state grammar, add $\delta(1, q_0, S) \overset{1}{\rightarrow} (p, \alpha')$ in R of M_{fd} such that α' is equal to α except that all non-terminals of α are replaced by A in α'.
2. Given the production rule $(q, A) \overset{\mu_i}{\rightarrow} (p, \alpha)$ in the fuzzy state grammar, add $\delta(1, q, A) \overset{\mu_i}{\rightarrow} (p, \alpha)$ in R of M_{fd}.
3. Given the production rule $(q, B) \overset{\mu_i}{\rightarrow} (p, \alpha)$ in the fuzzy state grammar, add $\delta(2, q, A) \overset{\mu_i}{\rightarrow} (p, \alpha)$ in R of M_{fd} (As non-terminals of depth>1, we will replace the all non-terminal *B* of fuzzy state grammar to non-terminal *A* with depth 2 in the fuzzy deep pushdown automaton. Similarly $\delta(3, q, A) \overset{\mu_j}{\rightarrow} (p, \alpha)$ in *R*, if the rule $(q, C) \overset{\mu_j}{\rightarrow} (p, \alpha)$ is in fuzzy state grammar with membership grade μ_j and so on.
4. Pop operation is applied whenever the topmost symbol and the current symbol under read/write head matches, keeping the state same and the membership value is 1 for this rule.

Rigorous Proof. We want to prove that if $S_{q_0} \Rightarrow_*^{\mu} w_1\alpha$ then $(q_0, w_1w_2, S) \Rightarrow_*^{\mu} (q, w_2, \alpha')$ for any w_2, where μ denotes the degree of membership and α' is equal to α except that all non-terminals of α are replaced by A in α'.

$(q_0, S) \overset{1}{=} (q, \gamma_1) \overset{\mu_1}{\Rightarrow} (q, \gamma_2) \overset{\mu_2}{\Rightarrow} \ldots \overset{\mu_{n-1}}{\Rightarrow} (q, \gamma_{n-1}) = w$ such that $\mu(w) = \min(1, \mu_1, \mu_2, ..., \mu_{n-1}) \wedge \gamma_i = w_i\alpha_i$. Here w_i is the string has already generated and $\alpha_i \in V^*$.

Proof by Induction: This theorem is established by mathematical induction of $n \geq 0$, where n represents the number of steps in the derivation of a string of a state grammar.

Basis Step: Consider $n=1,$ on applying the starting symbol production rule of the state grammar, a number of non-terminal are generated, and no input symbol is generated, i.e. $w_1 = \wedge,\ w_2 = w$ and $(q_0, S) \overset{1}{\Rightarrow} (q, \wedge w_2)$ such that $w_2 \in (V-\Sigma)^*$.

Now $(q_0, w, S) \overset{1}{\Rightarrow} (q, w, \alpha')$ such that α' is equal to w_2 except that all non-terminals of α are replaced by A in w_2. Hence it is true for $n=1$.

Induction hypothesis: Assume that if $(q_0, S) \underset{*}{\overset{\mu}{\Rightarrow}} (q, \gamma_n)$ then $(q_0, w_1 w_2, S) \Rightarrow_*^{\mu} (q, w_2, \alpha')$ for any w_2 after n derivation from S, $w_1 \in \Sigma^*$, $\gamma_n = w_1\alpha$, $\alpha \in V^*$, $\alpha' \in \mathrm{T}^*$, $\mu = \min(\mu_1, \mu_2, ..., \mu_n)$ and α' is equal to α except that all non-terminals of α are replaced by A in α'.

Induction step: Now consider $(n+1)^{th}$ derivation, If $(q_0, S) \underset{*}{\overset{\mu}{\Rightarrow}} (q, \gamma_n) \overset{\mu_{n+1}}{\Rightarrow} (p, \gamma_{n+1})$, Here $\gamma_n = w_1\alpha$ such that $w_1 \in \Sigma^*$ and $\alpha \in V^*$.

Following cases can occur:

Case 1: Consider in $(n+1)^{th}$ derivation step, the leftmost non-terminal is replaced by a single terminal.

$\alpha = A\alpha_1$ and $(q_0, S) \underset{*}{\overset{\mu}{\Rightarrow}} (q, \gamma_n) = (q, w_1\alpha) = (q, w_1 A\alpha_1) \overset{\mu_{n+1}}{\Rightarrow} (p, w_1 a\alpha_1) = (p, \gamma_{n+1})$

If the leftmost non-terminal of α is A, then correspondingly in fuzzy deep pushdown automaton rule of 1 depth is applied. Using the 2nd rule of the construction of fuzzy deep pushdown automaton from fuzzy state grammar, we get $(q_0, w_1 w_2, S) \Rightarrow_*^{\mu} (q, w_2, \alpha') \overset{\mu_{n+1}}{\Rightarrow}$ $(q, w_2, a\alpha'')$.

Symbol a is popped from the stack, and the same is read from w_2.

Here $\gamma_{n+1} = w_1 a\alpha_1$ where $w_1 a \in \Sigma^*$ and denote the already consumed string. Similarly, if the leftmost non-terminal *A* gives a number of terminals, these will be popped from the stack and consumed from the input tape. Hence the property holds for $(n+1)$ derivation steps.

Case 2: Consider in $(n+1)^{th}$ derivation step, $(q, A) \to (p, \beta)$ such that $\beta \in V^*$.

$$(q_0, S) \underset{*}{\overset{\mu}{\Rightarrow}} (q, \gamma_n) = (q, w_1\alpha) = (q, w_1 A\alpha_1) \overset{\mu_{n+1}}{\Rightarrow} (p, w_1\beta\alpha_1) = (p, \gamma_{n+1})$$

Case 3: Consider in $(n+1)^{th}$ derivation step, $(q, B) \to (p, \beta)$ such that $\beta \in V^*$.

$$(q_0, S) \underset{*}{\overset{\mu}{\Rightarrow}} (q, \gamma_n) = (q, w_1\alpha) = (q, w_1 A\alpha_1 B\alpha_1) \overset{\mu_{n+1}}{\Rightarrow} (p, w_1 A\alpha_1\beta\alpha_1) = (p, \gamma_{n+1}),$$

Here, $\alpha = A\alpha_1 B\alpha_1$, $\gamma_{n+1} = w_1 A\alpha_1\beta\alpha_1$

If the non-terminal is B in fuzzy state grammar, so using the 3rd rule of the construction of fuzzy deep pushdown automaton from fuzzy state grammar, then correspondingly in fuzzy deep pushdown automaton rule of 2 depth is applied here $\alpha = A\alpha_1 A\alpha_1$ and $\alpha'' = A\alpha_1\beta\alpha_1$

$$(q_0, w_1w_2, S) \Rightarrow_*^{\mu} (q, w_1w_2, \alpha') \overset{\mu_{n+1}}{\Rightarrow} (q, w_1w_2, \alpha'')$$

Case 4: Consider in $(n+1)^{th}$ derivation step, $(q, C) \to (p, \beta)$ such that $\beta \in V^*$.

$$(q_0, S) \underset{*}{\overset{\mu}{\Rightarrow}} (q, \gamma_n) = (q, w_1\alpha) = (q, w_1 A\alpha_1 B\alpha_1 C\alpha_1) \overset{\mu_{n+1}}{\Rightarrow} (p, w_1 A\alpha_1 B\alpha_1\beta\alpha_1) = (p, \gamma_{n+1})$$

Here, $\alpha = A\alpha_1 B\alpha_1 C\alpha_1$, $\gamma_{n+1} = w_1 A\alpha_1 B\alpha_1\beta\alpha_1$

If the non-terminal is C in fuzzy state grammar, so using the 3rd rule of the construction of fuzzy deep pushdown automaton from fuzzy state grammar, then correspondingly in fuzzy deep pushdown automaton rule of 3 depth is applied here $\alpha = A\alpha_1 A\alpha_1 A\alpha_1$ and $\alpha'' = A\alpha_1 B\alpha_1\beta\alpha_1$

$$(q_0, w_1w_2, S) \Rightarrow_*^{\mu} (q, w_1w_2, \alpha') \overset{\mu_{n+1}}{\Rightarrow} (q, w_1w_2, \alpha'').$$

Hence the same process is true for all the non-terminals in fuzzy state grammar. This completes the proof. □

THEOREM 2: Let $M_{fd}\ (Q, \Gamma, \sum, R, q_0, S, F, E, \mu, \pi, \gamma, \eta)$ be a fuzzy deep pushdown automaton, then we can construct be a fuzzy state grammar $G_{fs}(V, Q, \Sigma, P, S, E, \mu)$ such

that $L(G_{fs}) = L(M_{fd})$ i.e. for any $\alpha \in \Sigma^*$, $\mu(\alpha; \alpha \in L(G_{fs})) = \mu(\alpha; \alpha \in L(M_{fd}))$ where μ denotes the membership of a string.

Proof. We shall construct a fuzzy state grammar $G_{fs}(V, Q, \Sigma, P, S, E, \mu)$, where $V = \Sigma \bigcup \Gamma \bigcup S$ and the production rules *P* are defined as follows:

1. If $\delta(1, q, S) \xrightarrow{\mu} (p, AA...)$, then add the production $(q, S) \xrightarrow{\mu} (q, AB...)$ in *P*. The depth of M_{fd} determines the number of different symbol is expanded from the starting symbol of state grammar.
2. If $\delta(1, q, A) \xrightarrow{\mu} (p, \beta)$, then add the production $(q, A) \xrightarrow{\mu} (p, \beta)$ in *P*.
3. If $\delta(2, q, A) \xrightarrow{\mu} (p, \beta)$, then add the production $(q, B) \xrightarrow{\mu} (p, \beta)$ in *P*.
4. If $\delta(3, q, A) \xrightarrow{\mu} (p, \beta)$, then add the production $(q, C) \xrightarrow{\mu} (p, \beta)$ in *P*. The same process is repeated for depth>3.

We want to prove that if $(q_0, w_1 w_2, S) \Rightarrow_*^{\mu} (q, w_2, \alpha)$ for any w_2 then $S_{q_0} \Rightarrow_*^{\mu} w_1 \alpha$, where denotes degree of membership.

Rigorous Proof. This theorem is established by induction of $n \geq 0$, where *n* represents the number of moves made by the fuzzy deep pushdown automata.

Basis Step: The following two cases can occur:

Case 1: For $n = 0$, $(q_0, w, S) \Rightarrow_*^{\mu} (q_0, w, S)$

Here $\alpha = S$, $w_1 = \wedge$ and $S_{q_0} \Rightarrow_*^{\mu} S$ is always true.

Case 2: For $n = 1$, first moves is made by applying the transition function $\delta(1, q_0, S) \xrightarrow{1} (p, \alpha')$

Hence $(q_0, w, S) \xRightarrow{1} (q_0, w, \alpha')$ Here α' is equal to α except that all non-terminals of α are replaced by A in α', $w_1 = \wedge$ and $S_{q_0} \xRightarrow{1} \alpha$ is surely true.

Induction hypothesis: Assume that $(q_0, w_1 w_2, S) \Rightarrow_*^{\mu} (q, w_2, \alpha)$ for any w_2 if $S_{q_0} \Rightarrow_*^{\mu} w_1 \alpha$ works for $n-1$ moves in fuzzy deep pushdown automaton M_{fd}.

Induction step: Consider after n moves $M_{fd} : (q_0, w_1 w_2, S) \Rightarrow_*^{\mu} (q, w_2, \alpha)$, such that $\mu = \min(\mu_1, \mu_2, ..., \mu_n)$ where $\mu_1, \mu_2, ..., \mu_n$ denotes the degree of membership in each respective moves. The following three cases can occur using transition rules 2, 3 and 4 as mentioned above.

Case 1: Consider the n^{th} moves occur due to pop operation.

$$(q_0, w_1 a w_2, S) \Rightarrow_*^{\mu} (q, a w_2, a\alpha) \Rightarrow^1 (q, w_2, \alpha) \text{ where } w = w_1 a$$

Hence $(q_0, w_1 a w_2, S) \Rightarrow_*^{\mu} (q, w_2, \beta)$ then using the induction hypothesis,

$(q_0, w_1 a w_2, S) \Rightarrow_*^{\mu} (q, a w_2, a\alpha)$ means $S_{q_0} \Rightarrow_*^{\mu} w_1 a\alpha$ after *n-1* moves, but $w = w_1 a$ hence $S_{q_0} \Rightarrow_*^{\mu} w\alpha$ with membership $\min(\mu, 1) = \mu$.

Case 2: Consider the n^{th} move occur due to the rule $\delta(1, q, A) \xrightarrow{\mu_n} (p, \gamma)$

$$(q_0, w_1 w_2, S) \Rightarrow_*^{\mu} (q, w_2, A\alpha) \Rightarrow^{\mu_n} (p, w_2, \gamma\alpha) \text{ where } \beta = \gamma\alpha$$

Using induction hypothesis, $(q_0, w_1 w_2, S) \Rightarrow_*^{\mu} (q, w_2, A\alpha)$ means $S_{q_0} \Rightarrow_*^{\mu} w_1 A\alpha$, thus $S_{q_0} \Rightarrow_*^{\mu} w_1 \gamma\alpha = w_1 \beta$ with membership $\min(\mu, \mu_n)$.

Case 3: Consider the n^{th} move occur due to the rule $\delta(m, q, A) \xrightarrow{\mu_n} (p, \gamma)$ where $m = 2, \gamma \in \Gamma^*$.

$(q_0, w_1 w_2, S) \Rightarrow_*^{\mu} (q, w_2, \alpha A\beta) \Rightarrow^{\mu_n} (p, w_2, \alpha\gamma\beta)$ where $w = \alpha\gamma\beta$

Using induction Hypothesis $(q_0, w_1 w_2, S) \Rightarrow_*^{\mu} (q, w_2, \alpha A\beta)$ means $S_{q_0} \Rightarrow_*^{\mu} w_1 \alpha A\beta$, thus on applying the state grammar rule $(q, B) = (p, \gamma)$ will be $S_{q_0} \Rightarrow_*^{\mu} w_1 \alpha A\beta = w_1 \alpha B\beta \Rightarrow$

$w_1 \alpha\gamma\beta = w_1 w$.

Using Theorem 1 and Theorem 2, we have proved that $(q_0, w_1 w_2, S) \Rightarrow_*^{\mu} (q, w_2, \alpha)$ if and only if $S_{q_0} \Rightarrow_*^{\mu} w_1 \alpha$. Now, consider $w_2 = \alpha = \wedge$, then $w = w_1$. It gives, $(q_0, w, S) \Rightarrow_*^{\mu}$ $(q, \wedge, \wedge)$ if and only if $S_{q_0} \Rightarrow_*^{\mu} w$. String w is accepted by M_{fd} and G_{fs} with membership value μ. Hence $L(M_{fd}) = L(G_{fs})$ □

3.5 CONCLUSION

This chapter defines a novel concept of fuzzy state grammar and a fuzzy deep pushdown automaton. We demonstrate increased power, in terms of erroneous string acceptance, using fuzziness in state grammar and a deep pushdown automaton. Furthermore, depending upon the errors and certainty, we decide whether the string belongs to the language or not depending upon its membership value. The three main results demonstrated in this chapter are: (a.) Construction of fuzzy deep pushdown automaton from fuzzy state grammar; (b.) Construction of fuzzy state grammar from fuzzy deep pushdown automaton; and (c.) If Fuzzy deep pushdown automaton M_{fd} is constructed from the fuzzy state grammar G_{fs} using the above construction method, then $L(M_{fd}) = L(G_{fs})$.

Chapter-4

DETERMINISTIC MODELS OF DEEP PUSHDOWN AUTOMATA, TRANSDUCER AND THEIR PARALLEL VERSIONS

4.1 INTRODUCTION

Deep pushdown automata (DPDA) [10] and parallel deep pushdown automata (PDPDA) [11] are types of regulated automata. The aforementioned regulated automata are formal models for representing some of the context-sensitive languages or unrestricted languages by using pushdown automata capabilities with some regulations. The regulated automaton is the formal automaton counterpart of regulated grammars.

In formal language theory, translation schemes and transducers are used to represent translation. In a translation scheme, a string is generated from grammar along with output, whereas a transducer is an automaton device that produces output during each move [83]. The deep pushdown transducer (DPDT) and parallel deep pushdown transducers (PDPDT) [33] are extended versions of the deep pushdown automata (DPDA) and parallel deep pushdown automata (PDPDA), respectively.

Meduna [10] introduced the concept of strict deterministic and deterministic with respect to depth. The DPDA, PDPDA, DPDT and PDPDT are deterministic with respect to depth but lack strict determinism [10-11, 33].

The aim of the chapter is to propose a deterministic deep pushdown automaton (DDPDA), deterministic deep pushdown transducer (DDPDT) and their parallel versions named as parallel deterministic deep pushdown automaton (PDDPDA) and parallel deterministic deep pushdown transducer (PDDPDT), as an extension of DPDA, DPDT, PDPDA and PDPDT, respectively. The proposed models obey the properties of both strict determinism and determinism with respect to depth.

Result of this chapter has been published in The Computer Journal
N. Kalra and A. Kumar, Deterministic Deep Pushdown Transducer and its Parallel version, The Computer Journal, 1-11, 2017.

Implementation of DDPDA, DDPDT and their parallel versions (PDDPDA and PDDPDT) require additional states as compared to non-strict DPDA, DPDT and their parallel versions (PDPDA and PDPDT), but it will require less number of moves with respect to the given input string. Hence, DDPDA, DDPDT and their parallel versions (PDDPDA and PDDPDT) 'arrive at the answer' faster in comparison to the equivalent DPDA, DPDT and their parallel versions (PDPDA and PDPDT).

4.2 PRELIMINARIES CONCEPT

In this chapter, $card(S)$ denotes the cardinality of a set *S*. *CF* and *CS* denote context-free languages and context-sensitive languages, respectively.

DEFINITION 4.2.1. [10]: An automaton or transducer is strictly deterministic, if for every $mqA \to px \in R, card(\{mqA \to ov \mid mqA \to ov \in R, o \in Q, v \in \Gamma^+\} - \{mqA \to px\}) = 0,$ where *m* represents depth.

DEFINITION 4.2.2. [10]: An automaton or transducer is deterministic with respect to depth, if $\forall q \in Q$, $card(\{m \mid mqA \to px \in R, A \in \Gamma, p \in Q, x \in \Gamma^+\} \le 1)$, where *m* represent depth.

4.3 PROPOSED DETERMINISTIC VERSIONS OF DPDA, DPDT AND THEIR PARALLEL VERSIONS (PDPDA AND PDPDT)

For representing the deterministic version of DPDA, DPDT and their parallel versions (PDPDA AND PDPDT), we use the following additional capabilities in their formal definition.

1. Additional capabilities in R/W head: The R/W head can move in the right direction (denoted by 1), or it can remain stationary (denoted by 0).
2. The transition function of DDPDA and its parallel version: We consider the transition function of DDPDA and PDDPDA to be $Q \times (\Sigma \bigcup \{\$, \Delta\}) \times I \times \Gamma \to Q \times \Gamma^* \times D$, where Q denotes the state, Σ denotes the input tape alphabet, Δ, $\$ \in \Sigma$ denotes the left and right end markers of the input string on input tape, I denotes depth (a positive number), Γ denotes the stack content, and $D \in \{0, 1\}$ specifies the direction in which the R/W head marker will

move. In comparison to the formal definition of DPDA and PDPDA, we have included the input tape symbol under the R/W head and direction.

3. The transition function of DDPDT and its parallel version: We consider the transition function of DDPDT and PDDPDT to be $Q\times(\Sigma_I \bigcup\{\$,\Delta\})\times I\times\Gamma \to Q\times\Gamma^*\times D\times\Sigma_O$, where Q denotes the state, Σ_I denotes the input tape alphabet, Δ, $\$\in\Sigma_I$ denotes the left and right end markers of the input string on input tape, I denotes depth (a positive number), Γ denotes the stack content, $D\in\{0,1\}$ specifies the direction in which the R/W head marker will move, and Σ_O denotes the output alphabet. In comparison to the formal definition of DPDT and PDPDT, we have included the input tape symbol under the R/W head and direction.
4. The pop operation is represented explicitly for all states without showing the depth.
5. We have included the capability of null moves as an additional feature in DDPDA, DDPDT, and their parallel versions.
6. Moreover, stack contains a special bottom symbol denoted by Z_0.

Formally, DPDA or PDPDA is said to be strict deterministic and deterministic with respect to depth, If for any $q\in Q$, $a\in\{\sum,\Delta,\$)$, $i\in I$, $X\in\Gamma$, the set $\delta(q,a,i,X)$ has at most one element, and all the transition made are of the same depth, and formally, DPDT or PDPDT is said to be strict deterministic and deterministic with respect to depth, If for any $q\in Q$, $a\in\{\sum_I,\Delta,\$)$, $i\in I$, $X\in\Gamma$, the set $\delta(q,a,i,X)$ has at most one element, and all the transition made are of the same depth.

4.3.1 DETERMINISTIC DEEP PUSHDOWN AUTOMATA

The transition function of DDPDA is $Q\times(\Sigma\bigcup\{\$,\Delta\})\times I\times\Gamma\to Q\times\Gamma^*\times D$.

EXAMPLE 4.1: DDPDA for the language $L=\{a^nb^nc^n \mid n\geq 1\}$.

Deterministic deep pushdown automaton for the language L is ${}_2M(\{q_0, q_1, q_2, q_3, q_4, q_f\},\{A, S, Z_0\},\{a, b, c,\Delta,\$\}, R, q_0, S,\{q_f\})$ where R consists of the following rules:

$\delta(q_0,\Delta,1,S)\to(q_1, AA,1)$ $\qquad$ $\delta(q_1, a,1,A)\to(q_2, Ab,1)$

$\delta(q_2, a, 2, A) \to (q_1, Ac, 0)$ $\delta(q_2, b, 2, A) \to (q_3, c, 0)$

$\delta(q_3, b, 1, A) \to (q_4, \wedge, 0)$ $\delta(q_4, b, b) \to (q_4, \wedge, 1)$

$\delta(q_4, c, c) \to (q_4, \wedge, 1)$ $\delta(q_4, \$, Z_0) \to (q_f, \wedge, 0)$

Consider input string $w = aaabbbccc$.

$(q_0, \underset{\uparrow}{\Delta} aaabbbccc\$, SZ_0) \to (q_1, \Delta \underset{\uparrow}{a} aabbbccc\$, AAZ_0) \to (q_2, \Delta a \underset{\uparrow}{a} abbbccc\$, AbAZ_0)$
$\to (q_1, \Delta a \underset{\uparrow}{a} abbbccc\$, AbAcZ_0) \to (q_2, \Delta aa \underset{\uparrow}{a} bbbccc\$, AbbAcZ_0) \to (q_1, \Delta aa \underset{\uparrow}{a} bbbccc\$,$
$AbbAccZ_0) \to (q_2, \Delta aaa \underset{\uparrow}{b} bbccc\$, AbbbAccZ_0) \to (q_3, \Delta aaa \underset{\uparrow}{b} bbccc\$, AbbbcccZ_0) \to$
$(q_4, \Delta aaa \underset{\uparrow}{b} bbccc\$, bbbcccZ_0) \to (q_4, \Delta aaa \underset{\uparrow}{bb} bccc\$, bbcccZ_0) \to (q_4, \Delta aaabb \underset{\uparrow}{b} ccc\$, bcccZ_0)$
$\to (q_4, \Delta aaabbb \underset{\uparrow}{c} cc\$, cccZ_0) \to (q_4, \Delta aaabbbc \underset{\uparrow}{c} c\$, ccZ_0) \to (q_4, \Delta aaabbbcc \underset{\uparrow}{c}\$, cZ_0)$
$\to (q_4, \Delta aaabbbccc \underset{\uparrow}{\$}, Z_0) \to (q_f, \Delta aaabbbccc \underset{\uparrow}{\$}, \wedge)$

Language, $L(_2M) = E(_2M) = \{a^n b^n c^n \mid n \geq 1\}$, where $L(_2M)$ and $E(_2M)$ denote the languages accepted by final state and empty pushdown of depth two respectively.

DPDA for language *L* is represented in Fig. 4.1.

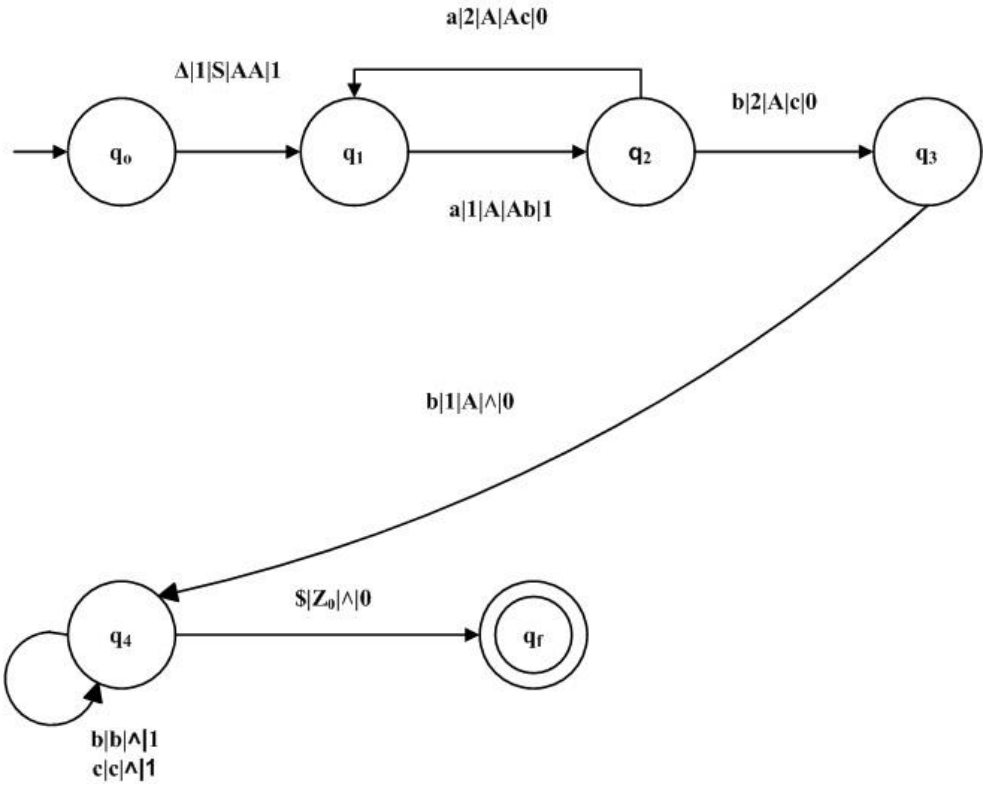


Fig. 4.1: DDPDA for language $\{a^n b^n c^n \mid n \geq 1\}$

4.3.2 DETERMINISTIC DEEP PUSHDOWN TRANSDUCER

The transition function of DDPDT is $Q\times(\Sigma_I \bigcup\{\$,\Delta\})\times I\times\Gamma \to Q\times\Gamma^*\times D\times\Sigma_O$.

EXAMPLE 4.2: DDPDT for performing the computation $q_0 w \overset{*}{\to} q_f ww$ i.e. for a given string *w*, it will generate *ww* as output string.

DDPDT ${}_2M_T(\{q_0, q_1, q_2, q_3, q_4, q_f\},\{0,1,\Delta,\$\},\{A, S, Z_0\},\{0, 1, \wedge\}, R, q_0, S,\{q_f\})$ with *R* containing rules:

$\delta(q_0, \Delta, 1, S) \to (q_1, AA, 1, \wedge)$

$\delta(q_1, 0, 1, A) \to (q_2, A, 0, 0)$

$\delta(q_2, 0, 2, A) \to (q_1, 0A, 1, \wedge)$

$\delta(q_1, 1, 1, A) \to (q_3, A, 0, 1)$

$\delta(q_3, 1, 2, A) \to (q_1, 1A, 1, \wedge)$

$\delta(q_1, \$, 1, A) \to (q_4, \Lambda, 0, \wedge)$

$\delta(q_4, \$, 1) \to (q_4, \Lambda, 0, 1)$

$\delta(q_4, \$, 0) \to (q_4, \Lambda, 0, 0)$

$\delta(q_4, \$, 1, A) \to (q_f, \Lambda, 0, \wedge)$

Deterministic version of DPDT for performing the computation $q_0 w \overset{*}{\to} q_f ww$ is represented in Fig. 4.2.
Consider w=010

$(q_0, \underset{\uparrow}{\Delta}010\$, SZ_0, \wedge) \to (q_1, \Delta\underset{\uparrow}{0}10\$, AAZ_0, \wedge) \to (q_2, \Delta\underset{\uparrow}{0}10\$, AAZ_0, 0) \to (q_1, \Delta0\underset{\uparrow}{1}0\$, A0AZ_0, 0)$
$\to (q_3, \Delta0\underset{\uparrow}{1}0\$, A0AZ_0, 01) \to (q_1, \Delta01\underset{\uparrow}{0}\$, A01AZ_0, 01) \to (q_2, \Delta01\underset{\uparrow}{0}\$, A01AZ_0, 010) \to$
$(q_1, \Delta010\underset{\uparrow}{\$}, A010AZ_0, 010) \to (q_4, \Delta010\underset{\uparrow}{\$}, 010AZ_0, 010) \to (q_4, \Delta010\underset{\uparrow}{\$}, 10AZ_0, 0100)$
$\to (q_4, \Delta010\underset{\uparrow}{\$}, 0AZ_0, 01001) \to (q_4, \Delta010\underset{\uparrow}{\$}, AZ_0, 010010) \to (q_f, \Delta010\underset{\uparrow}{\$}, Z_0, 010010)$

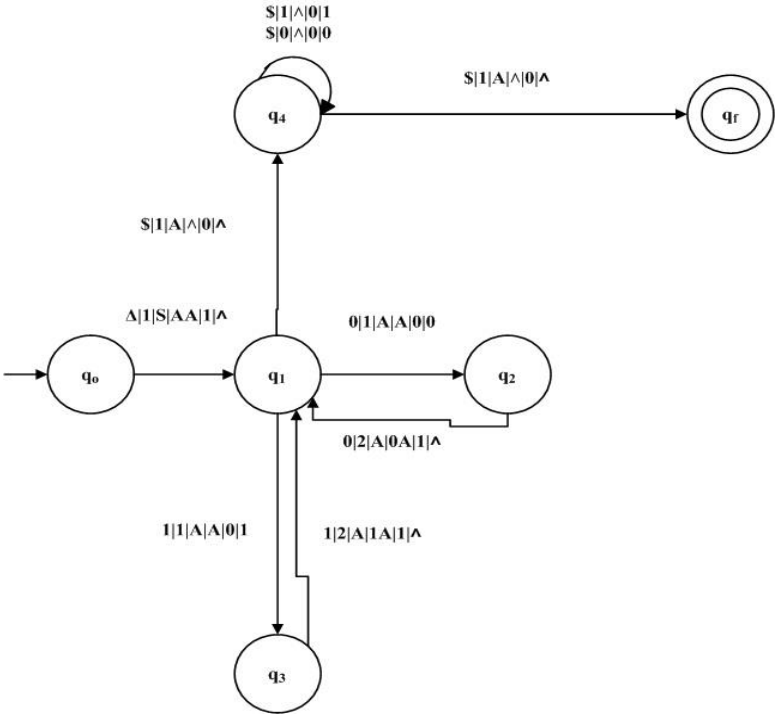


Fig. 4.2: DDPDT for performing computation $q_0 w \overset{*}{\rightarrow} q_f ww$

EXAMPLE 4.3: DDPDT for language $L_I = \{a^n b^n c^n \mid n \geq 1\}$ which produces output $L_O = \{c^n b^n a^n \mid n \geq 1\}$.

DDPDT ${}_2M_T(\{q_0, q_1, q_2, q_3, q_4, q_f\}, \{a, b, c, \Delta, \$\}, \{A, S, Z_0\}, \{a, b, c, \wedge\}, R, q_0, S, \{q_f\})$ with *R* containing rules:

$\delta(q_0, \Delta, 1, S) \rightarrow (q_1, AA, 1, \wedge)$ $\quad$ $\delta(q_1, a, 1, A) \rightarrow (q_2, Ab, 1, c)$

$\delta(q_2, a, 2, A) \rightarrow (q_1, Ac, 0, \wedge)$ $\quad$ $\delta(q_2, b, 2, A) \rightarrow (q_3, c, 0, \wedge)$

$\delta(q_3, b, 1, A) \rightarrow (q_4, \wedge, 0, \wedge)$ $\quad$ $\delta(q_4, b, b) \rightarrow (q_4, \wedge, 1, b)$

$\delta(q_4, c, c) \rightarrow (q_4, \wedge, 1, a)$ $\quad$ $\delta(q_4, \$, Z_0) \rightarrow (q_f, \wedge, 0, \wedge)$

Consider $w = aaabbbccc$, the input tape contains input string *w* followed by end marker \$. The input tape contents are $aaabbbccc\$$. Initially, the stack contains Z_0 followed by *S*.

$(q_0, \underset{\uparrow}{\Delta} aaabbbccc\$, SZ_0, \wedge) \rightarrow (q_1, \Delta \underset{\uparrow}{a} aabbbccc\$, AAZ_0, \wedge) \rightarrow (q_2, \Delta a \underset{\uparrow}{a} abbbccc\$, AbAZ_0, c)$
$\rightarrow (q_1, \Delta a \underset{\uparrow}{a} abbbccc\$, AbAcZ_0, c) \rightarrow (q_2, \Delta aa \underset{\uparrow}{a} bbbccc\$, AbbAcZ_0, cc) \rightarrow (q_1, \Delta aa \underset{\uparrow}{a} bbbccc\$, AbbAccZ_0, cc)$

$\rightarrow (q_2, \Delta aaa\underset{\uparrow}{b}bbccc\$, AbbbAccZ_0, ccc) \rightarrow (q_3, \Delta aaa\underset{\uparrow}{b}bbccc\$, AbbbcccZ_0, ccc) \rightarrow (q_4, \Delta aaa\underset{\uparrow}{b}bbccc\$, bbbcccZ_0,$
$ccc) \rightarrow (q_4, \Delta aaa\underset{\uparrow}{bb}bccc\$, bbcccZ_0, cccb) \rightarrow (q_4, \Delta aaabb\underset{\uparrow}{b}ccc\$, bcccZ_0, cccbb) \rightarrow (q_4, \Delta aaabbb\underset{\uparrow}{c}cc\$, cccZ_0,$
$cccbbb) \rightarrow (q_4, \Delta aaabbbc\underset{\uparrow}{c}c\$, ccZ_0, cccbbba) \rightarrow (q_4, \Delta aaabbbcc\underset{\uparrow}{c}\$, cZ_0, cccbbbaa) \rightarrow (q_4, \Delta aaabbbccc\underset{\uparrow}{\$}, Z_0,$
$cccbbbaaa) \rightarrow (q_f, \Delta aaabbbccc\underset{\uparrow}{\$}, \wedge, ccccbbbaaa)$

Deterministic version of DPDT for performing the reversal of string computation is represented in Fig. 4.3.

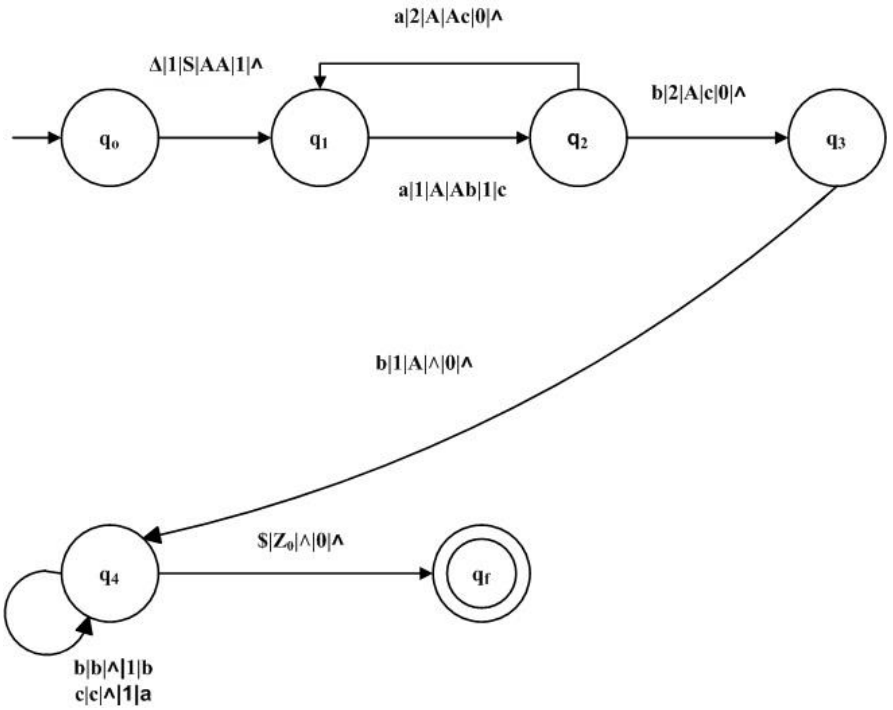


Fig. 4.3: DDPDT for performing string reversal

4.3.3 PARALLEL DETERMINISTIC DEEP PUSHDOWN AUTOMATA

The transition function of PDDPDA is $Q\times(\Sigma\bigcup\{\$,\Delta\})\times I\times\Gamma\rightarrow Q\times\Gamma^*\times D$, and parallel rules are of the type $\delta(p\times a\times(A_1,...,A_n)\rightarrow q\times(w_1,...,w_n)\times d_i)$ where $p,q\in Q$, $A_i\in\Gamma-\sum$, $w_i\in\Gamma^+$, $1\le i\le n$, $a\in\{\sum,\Delta,\$)$, $d_i\in D$.

EXAMPLE 4.4: PDDPDA for the language $L=\{a^nb^nc^n \mid n\ge 1\}$.

The parallel deterministic deep pushdown automaton is as follows: ${}_2M_p(\{q_0, q_1, q_2, q_3, q_f\},\{A, S, Z_0\},\{a, b, c, \Delta, \$\}, R, q_0, S,\{q_f\})$ with R containing rules:

PDDPDA for language L is represented in Fig. 4.4.

$\delta(q_0, \Delta, 1, S) \rightarrow (q_1, AA, 1)$

$\delta(q_1, a, (A, A)) \rightarrow (q_1, (Ab, Ac), 1)$

$\delta(q_1, b, 1, A) \rightarrow (q_2, \wedge, 0)$

$\delta(q_2, b, b) \rightarrow (q_2, \wedge, 1)$

$\delta(q_2, c, 1, A) \rightarrow (q_3, \wedge, 0)$

$\delta(q_3, c, c) \rightarrow (q_3, \wedge, 1)$

$\delta(q_3, \$, Z_0) \rightarrow (q_f, \wedge, 0)$

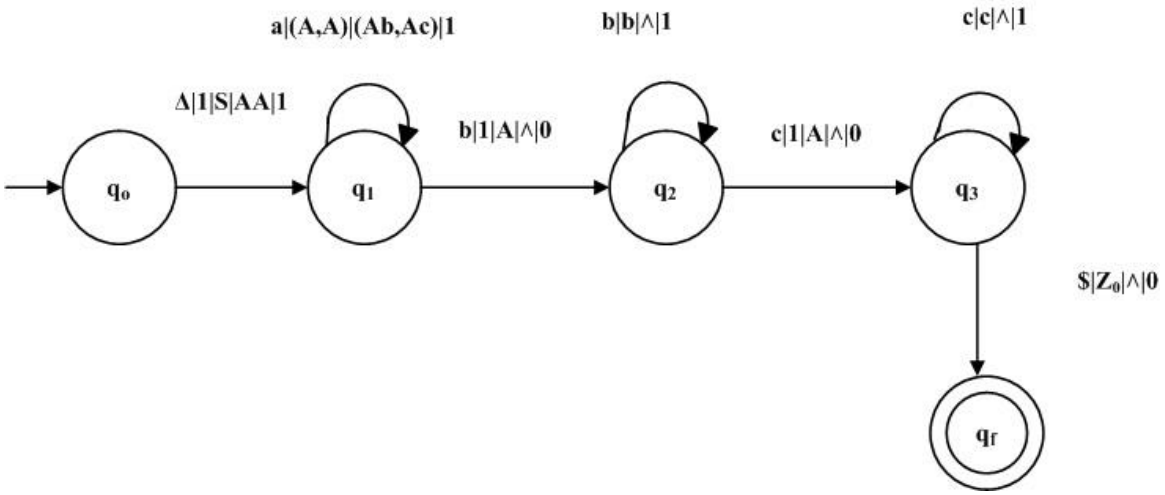


Fig. 4.4: PDDPDA of language $\{a^n b^n c^n \mid n \geq 1\}$

Let us consider $w = aaabbbccc$, the input tape contents are $aaabbbccc\$$. Initially, the stack contains Z_0 followed by S.

$(q_0, \underset{\uparrow}{\Delta} aaabbbccc\$, SZ_0) \rightarrow (q_1, \Delta \underset{\uparrow}{a} aabbbccc\$, AAZ_0) \rightarrow (q_1, \Delta a \underset{\uparrow}{a} abbbccc\$, AbAcZ_0)$
$\rightarrow (q_1, \Delta aa \underset{\uparrow}{a} bbbccc\$, AbbAccZ_0) \rightarrow (q_1, \Delta aaa \underset{\uparrow}{b} bbccc\$, AbbbAcccZ_0) \rightarrow (q_2, \Delta aaa \underset{\uparrow}{b} bbccc\$,$
$bbbAcccZ_0) \rightarrow (q_2, \Delta aaab \underset{\uparrow}{b} bccc\$, bbAcccZ_0) \rightarrow (q_2, \Delta aaabb \underset{\uparrow}{b} ccc\$, bAcccZ_0) \rightarrow (q_2, \Delta aaabbb \underset{\uparrow}{c} cc\$,$
$AcccZ_0) \rightarrow (q_3, \Delta aaabbb \underset{\uparrow}{c} cc\$, cccZ_0) \rightarrow (q_3, \Delta aaabbbc \underset{\uparrow}{c} c\$, ccZ_0) \rightarrow (q_3, \Delta aaabbbcc \underset{\uparrow}{c} \$, cZ_0)$
$\rightarrow (q_3, \Delta aaabbbccc \underset{\uparrow}{\$}, Z_0) \rightarrow (q_f, \Delta aaabbbccc \underset{\uparrow}{\$}, \wedge)$

Thus, $L(_2M_p) = E(_2M_p) = \{a^n b^n c^n \mid n \geq 1\}$, where $L(_2M_p)$ and $E(_2M_p)$ denotes the language accepted by final state and empty pushdown of depth two respectively.

4.3.4 PARALLEL DETERMINISTIC DEEP PUSHDOWN TRANSDUCER

The transition function of PDDPDT is $Q\times(\Sigma_I \bigcup\{\$,\Delta\})\times I\times\Gamma\to Q\times\Gamma^*\times D\times\Sigma_O$, and parallel rules are of the type $\delta(p\times a\times(A_1,...,A_n)\to q\times(w_1,...,w_n)\times d_i\times b)$ where $p,q\in Q$, $A_i\in\Gamma-\sum, w_i\in\Gamma^+, 1\le i\le n, a\in\{\sum_I,\Delta,\$), d_i\in D, b\in\sum_O$.

EXAMPLE 4.5: PDDPDT for performing the computation $q_0 w \overset{*}{\to} q_f ww$.

PDDPDT ${}_2M_{Tp}(\{q_0, q_1, q_2, q_f\},\{0,1,\Delta,\$\},\{A, S, Z_0\},\{0, 1, \wedge\}, R, q_0, S,\{q_f\})$ with R containing rules:

$\delta(q_0,\Delta,1,S)\to(q_1, AA, 1,\wedge)$

$\delta(q_1,0,(A,A))\to(q_1,(A,0A),1,0)$

$\delta(q_1,1,(A,A))\to(q_1,(A,1A),1,1)$

$\delta(q_1,\$,1,A)\to(q_2,\Lambda,0,\wedge)$

$\delta(q_2,\$,1)\to(q_2,\Lambda,0,1)$

$\delta(q_2,\$,0)\to(q_2,\Lambda,0,0)$

$\delta(q_2,\$,1,A)\to(q_f,\Lambda,0,\wedge)$

PDDPDT for performing the computation $q_0 w \overset{*}{\to} q_f ww$ is represented in Fig. 4.5.

Consider w=010

$(q_0, \underset{\uparrow}{\Delta}010, SZ_0,\wedge)\to(q_1, \Delta\underset{\uparrow}{0}10, AAZ_0,\wedge)\to(q_1,\Delta\underset{\uparrow}{1}0, A0AZ_0, 0)\to(q_1,\Delta\underset{\uparrow}{0}10, A01AZ_0, 01)$
$\to(q_1,\Delta\underset{\uparrow}{0}10\$, A010AZ_0, 010)\to(q_2,\Delta\underset{\uparrow}{0}10\$, 010AZ_0, 010)\to(q_2,\Delta\underset{\uparrow}{1}0\$, 10AZ_0, 0100)$
$\to(q_2,\Delta\underset{\uparrow}{0}10\$, 0AZ_0, 01001)\to(q_2,\Delta\underset{\uparrow}{1}0\$, AZ_0, 010010)\to(q_f,\Delta\underset{\uparrow}{0}10\$, Z_0, 010010)$

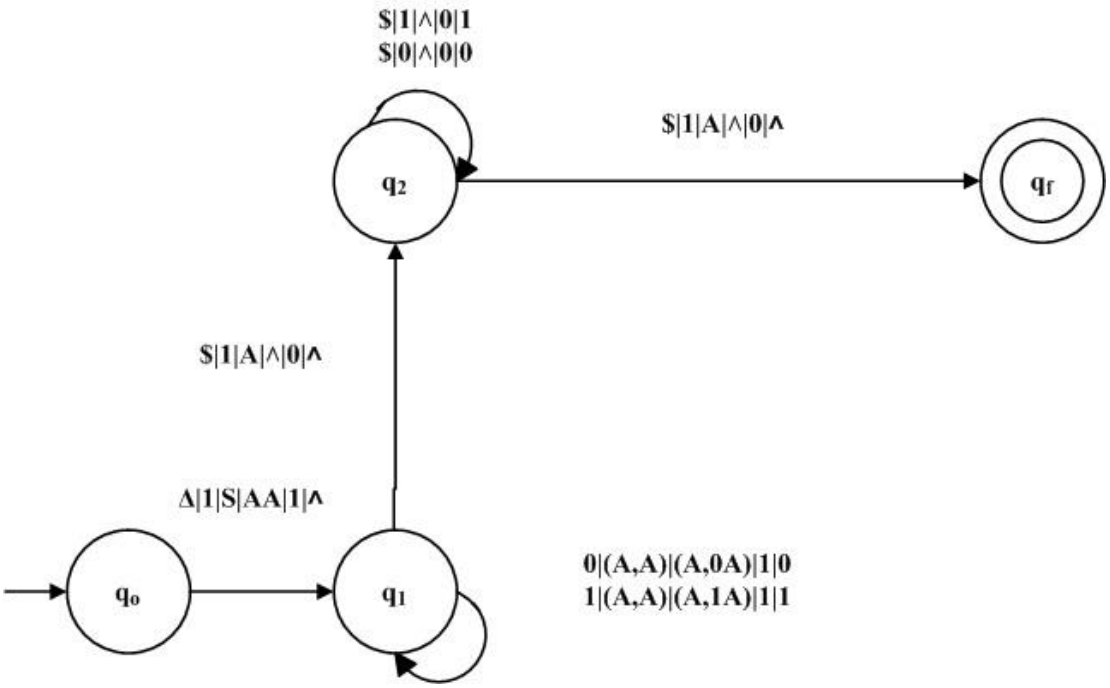


Fig 4.5: PDDPDT for performing the computation $q_0 w \overset{*}{\rightarrow} q_f ww$

EXAMPLE 4.6: PDDPDT for input $L_I = \{a^n b^n c^n \mid n \geq 1\}$ which produces the output $L_O = \{c^n b^n a^n \mid n \geq 1\}$.

PDDPDT ${}_2M_{Tp}(\{q_0, q_1, q_2, q_3, q_f\}, \{a,\ b,\ c, \Delta, \$\}, \{A,\ S, Z_0\}, \{a, b, c, \wedge\}, R,\ q_0, S, \{q_f\})$ with *R* containing rules:

$\delta(q_0, \Delta, 1, S) \rightarrow (q_1, AA, 1, \wedge)$

$\delta(q_1, a, (A, A)) \rightarrow (q_1, (Ab, Ac), 1, c)$

$\delta(q_1, b, 1, A) \rightarrow (q_2, \wedge, 0, \wedge)$

$\delta(q_2, b, b) \rightarrow (q_2, \wedge, 1, b)$

$\delta(q_2, c, 1, A) \rightarrow (q_3, \wedge, 0, \wedge)$

$\delta(q_3, c, c) \rightarrow (q_3, \wedge, 1, a)$

$\delta(q_3, \$, Z_0) \rightarrow (q_f, \wedge, 0, \wedge)$

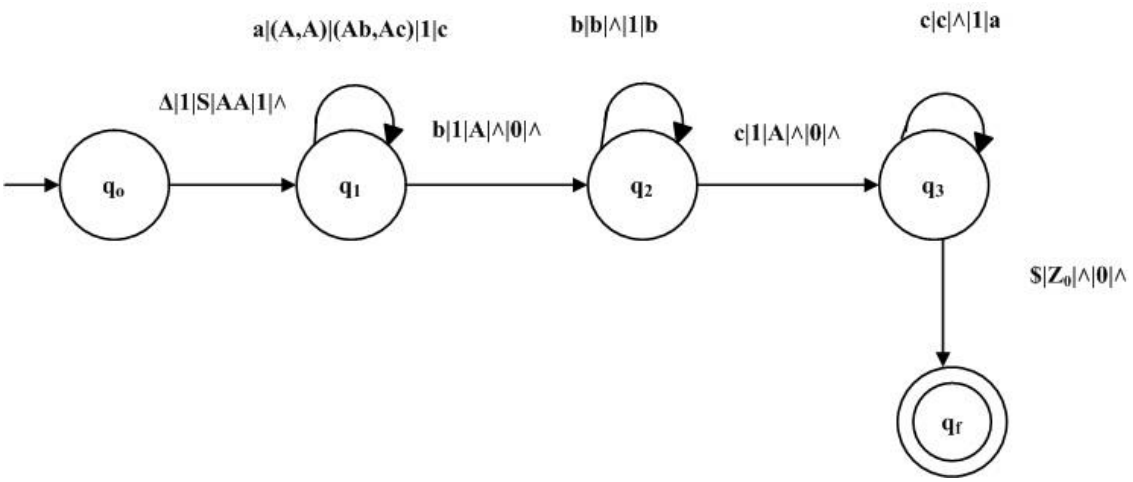


Fig. 4.6: PDDPDT for performing string reversal

PDDPDT for performing the reversal of string computation is represented in Fig. 4.6.

Consider $w = aaabbbccc$, the input tape contains input followed by end marker $. The input tape contents are $aaabbbccc\$$. Initially, stack contains Z_0 followed by S.

$$(q_0, \underset{\uparrow}{\Delta} aaabbbccc\$, SZ_0, \wedge) \rightarrow (q_1, \Delta \underset{\uparrow}{a} aabbbccc\$, AAZ_0, \wedge) \rightarrow (q_1, \Delta a \underset{\uparrow}{a} abbbccc\$, AbAcZ_0, c)$$
$$\rightarrow (q_1, \Delta aa \underset{\uparrow}{a} bbbccc\$, AbbAccZ_0, cc) \rightarrow (q_1, \Delta aaa \underset{\uparrow}{b} bbccc\$, AbbbAcccZ_0, ccc) \rightarrow (q_2, \Delta aaa \underset{\uparrow}{b} bbccc\$, bbbAcccZ_0, ccc)$$
$$\rightarrow (q_2, \Delta aaab \underset{\uparrow}{b} bccc\$, bbAcccZ_0, cccb) \rightarrow (q_2, \Delta aaabb \underset{\uparrow}{b} ccc\$, bAcccZ_0, cccbb) \rightarrow\rightarrow (q_2, \Delta aaabbb \underset{\uparrow}{c} cc\$, AcccZ_0, cccbbb)$$
$$\rightarrow (q_3, \Delta aaabbb \underset{\uparrow}{c} cc\$, cccZ_0, cccbbb) \rightarrow (q_3, \Delta aaabbbc \underset{\uparrow}{c} c\$, ccZ_0, cccbbba) \rightarrow (q_3, \Delta aaabbbcc \underset{\uparrow}{c}\$, cZ_0, cccbbbaa)$$
$$\rightarrow (q_3, \Delta aaabbbccc \underset{\uparrow}{\$}, Z_0, cccbbbaaa) \rightarrow (q_f, \Delta aaabbbccc \underset{\uparrow}{\$}, \wedge, cccbbbaaa)$$

4.4 RESULTS AND DISCUSSIONS

Theorem 1: The number of moves (expansion and pop) required for DDPDA is less than the DPDA for language $L = \{a^n b^n c^n \mid n \geq 1\}$.

Proof: Theoretically, the machine starts from start state, reads the input string character by character, and based on the input read, it makes a transition to another state. A certain set of states are designated accept states; whether the automaton accepts or rejects depends on whether it's in an accept state or reject state after reading the entire string. In DDPDA, at a given state and input, we will have only path, i.e. we will reach to specific state and in case of non-strict DPDA at a given input and a state, we may have multiple paths which leads to different states. In non-strict DPDA, at each state, for a given input string, how to decide from multiple paths is purely random, there is equal probability to choose among various paths. There is multiple paths for the same input

string and at least one path should ends in accept state. In DDPDA, we have to construct a path for every input and every output as compared to non-strict DPDA. In non-strict DPDA, if a wrong path is selected, additional moves may be required in backtracking. Implementation of DDPDA requires additional states as compared to non-strict DPDA but it will require less number of moves with respect to the given input string. Hence, DDPDA, 'arrive at the answer' faster in comparison to the equivalent DPDA.

Proof by Example: Table 4.1 depicts the number of moves (Expansion and pop) used for various strings in DPDA (Example 1.4) and DDPDA (Example 4.1). Clearly, DDPDA made fewer moves than original DPDA.

Table 4.1: Comparative study (in terms of moves) between DPDA and DDPDA

	DPDA			**DDPDA**		
Input String	**Expansion**	**Pop**	**# of Moves**	**Expansion**	**Pop**	**# of Moves**
$w = abc$	3	4	7	4	3	7
$w = aabbcc$	5	7	12	6	5	11
$w = aaabbbccc$	7	10	17	8	7	15
$w = aaaabbbbcccc$	9	13	22	10	9	19
$w = aaaaabbbbbccccc$	11	16	27	12	11	23

Given, any string $w = a^k b^k c^k \mid k \geq 1$, DPDA requires $2k+1$ expansions. Here, *k* expansions are required for generating $a^k b^k$ using depth one. Similarly, *k* expansions are required for generating c^k using depth two. One additional expansion is required for replacing *S* by *AA*. Total number of expansions required is *2k+1*. Number of pop operations required is *3k+1*, where *3k* is the length of the input string for matching each stack symbol with the each input tape symbol and one additional pop operation is required to check whether the stack is empty. Hence, total number of moves are *5k+2*.

In DDPDA, $2k+1$ pop operations are required. *2k* pops are required to pop *b* and *c* from the stack. Additional pop operation is required to check whether the stack is empty. In deterministic deep pushdown automata for language $L = \{a^n b^n c^n \mid n \geq 1\}$, *2k* expansions are required for generating $b^k c^k$ on to stack, one initial expansion is required for replacing *S* by *AA* and one additional expansion is required to replace *A* with null. Hence, in DDPDA, total number of moves is *4k+3*.

Clearly, $5k+2 > 4k+3$ for $k \geq 1$. However, additional moves may be required in DPDA, if a wrong path is selected. □

Theorem 2: The number of moves (expansion and pop) required for PDDPDA is less than the PDPDA for language $L = \{a^n b^n c^n \mid n \geq 1\}$.

Proof: Table 4.2 depicts the number of moves (Expansion and pop) used for various strings in PDDPDA (Example 4.4) and PDPDA (Example 1.5). Clearly, PDDPDA made fewer moves than PDPDA.

Table 4.2: Comparative study (in terms of moves) between PDPDA and PDDPDA

	PDPDA			**PDDPDA**		
Input String	**Expansion**	**Pop**	**# of Moves**	**Expansion**	**Pop**	**# of Moves**
$w = abc$	3	4	7	4	3	7
$w = aabbcc$	4	7	11	5	5	10
$w = aaabbbccc$	5	10	15	6	7	13
$w = aaaabbbbcccc$	6	13	19	7	9	16
$w = aaaaabbbbbccccc$	7	16	23	8	11	19

Given, any string $w = a^k b^k c^k \mid k \geq 1$, PDPDA requires $k+2$ expansions. Here, $(k-1)$ expansions are required for generating $a^{k-1}b^{k-1}c^{k-1}$ in parallel. One expansion is required to generate last a's and b's using $1qA \rightarrow fab$ (See example 1.5 for details). Similarly, one expansion is required to generate last $c's$ using $1fA \rightarrow fc$. One initial expansion is required for replacing *S* by *AA*. Hence, total number of expansion is $k+2$. Number of pop operations required is 3k+1, where 3k is the length of the input string for matching each stack symbol with the each input tape symbol and one additional pop operation is required to check whether the stack is empty. Hence, total number of moves is $4k+3$ in PDPDA.

In PDDPDA, $2k+1$ pop operations are required. *2k* pops are required to pop *b* and *c* from the stack. Additional pop operation is required to check whether the stack is empty. In parallel deterministic deep pushdown automata for language $L = \{a^n b^n c^n \mid n \geq 1\}$, *k* expansions are required for generating $b^k c^k$ on to stack, one initial expansion is required for replacing *S* by *AA* and two additional expansions are required while reading first *'b'* and first *'c'*. Total expansions are $k+3$ Hence, in PDDPDA, total number of moves is $3k+4$.

Clearly, $4k+3 \geq 3k+4$ for $k \geq 1$. However, additional moves may be required in PDPDA, if a wrong path is selected. □

Theorem 3: The number of moves (expansion and pop) required for DDPDT is less than DPDT for reversal of strings for input $\{a^n b^n c^n \mid n \geq 1\}$.

Proof. Table 4.3 represents the number of moves required for various strings of DDPDT (Example 4.3) and DPDT (Example 1.6). Clearly, DDPDT requires fewer moves than DPDT.

Table 4.3: Comparative study (in terms of moves) between DPDT and DDPDT

		DPDT	**DDPDT**
Input String	**Output String**	**# of Moves**	**# of Moves**
$w = abc$	$w = cba$	7	7
$w = aabbcc$	$w = ccbbaa$	12	11
$w = aaabbbccc$	$w = cccbbbaaa$	17	15
$w = aaaabbbbcccc$	$w = ccccbbbbaaaa$	22	19
$w = aaaaabbbbbccccc$	$w = cccccbbbbbaaaaa$	27	23

For any string $w = a^k b^k c^k \mid k \geq 1$, DPDT requires $2k+1$ expansions where

- k expansions are required for generating $a^k b^k$ using depth one,
- k expansions are required for generating c^k using depth two and
- One additional expansion is required for replacing $S \rightarrow AA$.

For each pop operation, some output is generated. Number of pop operation for DPDT is $3k+1$ where

- Numbers of pop operations are required for generating output $w = c^k b^k a^k \mid k \geq 1$ is $3k$, where $3k$ is the length of the input string for matching each stack symbol with the each input tape symbol.
- The additional pop operation is required to check whether the stack is empty.

Hence, total numbers of moves in DPDT is $5k+2$.

For any string $w = a^k b^k c^k \mid k \geq 1$, DDPDT will require $2k+1$ pop operations where

- $2k$ pop are required to generate a and b in the output string.
- Additional pop operation is required to check whether the stack is empty.

DDPDT will require $2k+2$ expansions. DDPDT will require *2k* expansions for generating $b^k c^k$ on to stack, out of which *k* expansions are required for generating output '*c*'. One initial expansion is required for replacing $S \rightarrow AA$ and one additional expansion is required to replace *A* with null.

Hence, in the deterministic deep pushdown transducer, total number of moves are *4k+3*.

Clearly, $5k+2 > 4k+3$ for $k \geq 1$. However, additional moves may be required in DPDT, if a wrong path is selected. □

Theorem 4: The number of moves (expansion and pop) required for PDDPDT is less than PDPDT for reversal of strings for input $\{a^n b^n c^n \mid n \geq 1\}$.

Proof. Table 4.4 depicts the number of moves required for various strings of PDDPDT (Example 4.6) and PDPDT (Example 1.7). Clearly, PDDPDT requires fewer moves than PDPDT.

Table 4.4: Comparative study (in terms of moves) between PDDPDT and PDPDT

		PDPDT	**PDDPDT**
Input String	**Output String**	**# of Moves**	**# of Moves**
$w = abc$	$w = cba$	7	7
$w = aabbcc$	$w = ccbbaa$	11	10
$w = aaabbbccc$	$w = cccbbbaaa$	15	13
$w = aaaabbbbcccc$	$w = ccccbbbbaaaa$	19	16
$w = aaaaabbbbbccccc$	$w = cccccbbbbbaaaaa$	23	19

For any string $w = a^k b^k c^k \mid k \geq 1$, PDPDT requires $k+2$ expansions, where

- $(k-1)$ expansions are required for generating $a^{k-1} b^{k-1} c^{k-1}$ in parallel.
- One expansion is required to generate last *a*'s and *b*'s using $1qA \rightarrow fab$ (See example 1.7 for details).
- One expansion is required to generate last $c's$ using $1fA \rightarrow fc$.
- One initial expansion is required for replacing $S \rightarrow AA$.

PDPDT will require *3k+1* pop operations where

- Numbers of pop operations are required for generating output $w = c^k b^k a^k \mid k \geq 1$ is *3k*, where *3k* is the length of the input string for matching each stack symbol with the each input tape symbol.

- One additional pop operation is required to check whether the stack is empty.

Hence, a total number of moves in PDPDT are $4k+3$.

PDDPDT will require $2k+1$ pop operations where

- *2k* pop is required to generate '*a*' and '*b*' in the output string.
- One additional pop operation is required to check whether the stack is empty.

PDDPDT will require $k+3$ expansion where

- *k* expansions are required for generating $b^k c^k$ on to stack and for producing output '*c*'.
- One initial expansion is required for replacing $S \to AA$.
- Two additional expansions are required while reading first '*b*' and first '*c*'.

Hence, total number of moves in PDDPDT is $3k+4$.

Clearly, $4k+3 \geq 3k+4$ for $k \geq 1$. However, additional moves may be required in PDPDT, if a wrong path is selected. □

Theorem 5: For every $DDPDA_n$ with depth *n*, there exists a $PDDPDA_n$ (i.e., for $n \geq 1$, $DDPDA_n = PDDPDA_n$).

Proof. A deterministic deep pushdown automaton ($DDPDA_n$) can be converted into a parallel deterministic deep pushdown automaton ($PDDPDA_n$) by applying *n* rules in parallel. The proof of this Theorem directly follows from the Lemma 3 in [11] which states that $PDPDA_n = DPDA_n$. □

Corollary 1: For every $DDPDT_n$ with depth *n*, there exists a $PDDPDT_n$ (i.e., for $n \geq 1$, $DDPDT_n = PDDPDT_n$).

Proof. By applying *n* rules in parallel, a DDPDT can be converted into a PDDPDT. The proof of this theorem directly follows from the Theorem 5 which states that $DDPDA_n = PDDPDA_n$, and from the fact that a transducer is an automaton with the additional capability of producing output during each move. □

Theorem 6 [11]**:** For every $n \geq 1$, $DPDA_n = PDPDA_n \subset CS$.

A deep pushdown automaton or parallel deep pushdown automaton can be designed for a language $L = \{ww \mid w \in (0|1)^+\}$. In Theorem 7, we prove that deterministic deep

pushdown automaton or parallel deterministic deep pushdown automaton cannot be designed for the language *L*. □

Theorem 7: Given $L = \{ww \mid w \in (0|1)^+\}$ cannot be accepted by any deterministic deep pushdown automata or parallel deterministic deep pushdown automata.

Proof. DPDA and PDPDA can be designed for input language $L = \{ww \mid w \in (0|1)^+\}$, whereas we are unable to design DDPDA and PDDPDA for L. Fig.4.7 represents the DPDA (depth 2) for language *L*. For any arbitrary string $x \in L$ such that $|x| = 2n$. On reading $n+1^{th}$ symbol (either 0 or 1) from R/W head, DPDA moves to state *f* from state *p*, but it is non-deterministic move as there is a possibility of moving to either state *r* or state *q*.

We are unable to design DDPDA for input language L. During processing of any arbitrary string, *x*, we read the symbols from left to right, as we have the additional feature of depth, but we are not able to determine that R/W head is pointing to $n+1^{th}$ symbol. We have no information regarding whether the midpoint has arrived or not. Therefore, we are unable to design a DDPDA for language L. From Lemma 3 of PDPDA [11], we have DPDA= PDPDA. Therefore $L = \{ww \mid w \in (0|1)^+\}$ cannot be accepted by any deterministic deep pushdown automata or parallel deterministic deep pushdown automata. □

Corollary 2: $DDPDT \subset DPDT$ and $PDDPDT \subset PDPDT$

Proof. DPDT and PDPDT can be designed for input language $L_I = \{ww \mid w \in (0|1)^+\}$, whereas we are unable to design DDPDT and PDDPDT for L_I. Fig.4.7 represents the DPDT (depth 2) for language L_1 considering the output language $L_O = \phi$ .DPDT and PDPDT can be designed for input language $L_I = \{ww \mid w \in (0|1)^*\}$, whereas we cannot design DDPDT and PDDPDT for L_I. The result of this corollary follows directly from Theorem 7 and from the fact that a transducer is an automaton with the additional capability of producing output during each move. Therefore $DDPDT \subset DPDT$ and $PDDPDT \subset PDPDT$ □

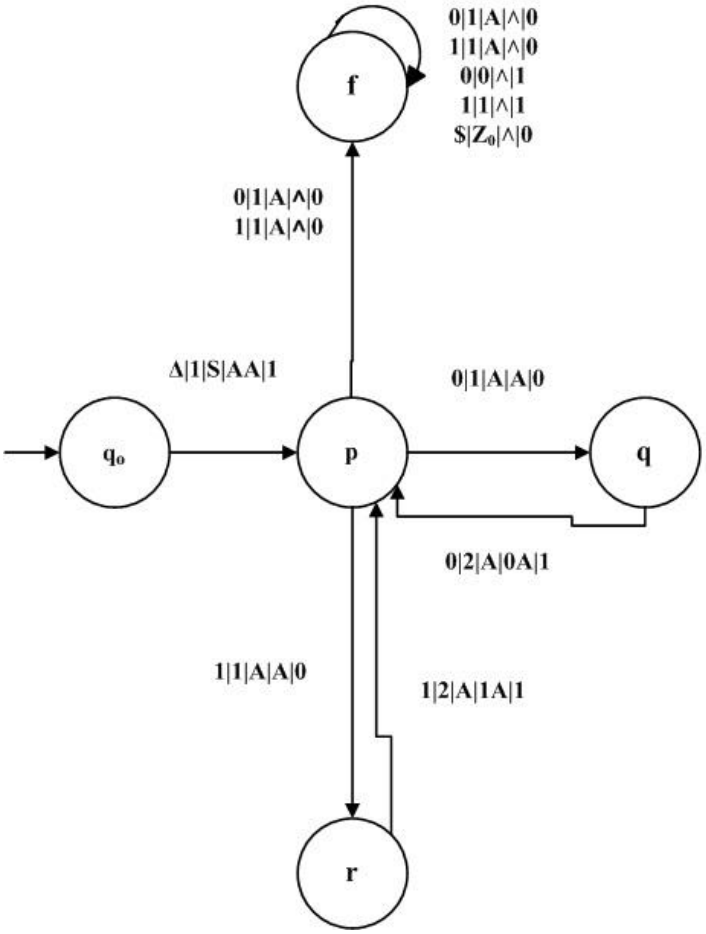


Fig 4.7: Non-strict deterministic deep pushdown automata for the language $L = \{ww \mid w \in (0|1)^+\}$

Corollary 3: Deterministic pushdown automata cannot be designed for $L = \{ww^R \mid w \in (0|1)^+\}$.

Proof. This corollary can be proved analogy to Theorem 7. As we are not able to determine the middle point of the input string as well as how many input symbols are left to process the complete string using the depth concept. □

Next, we will show that deterministic deep pushdown automata accept some non-context-free languages.

Theorem 8: $DDPDA - CF \neq \phi$

Proof. Consider the DDPDA for the language L is ${}_2M(\{q_0, q_1, q_2, q_3, q_4, q_f\}, \{a, b, c, \Delta, \$\}, \{A, S, Z_0\}, R, q_0, S, \{q_f\})$ where R consists of the following rules:

$\delta(q_0, \Delta, 1, S) \rightarrow (q_1, AA, 1)$ $\delta(q_1, a, 1, A) \rightarrow (q_2, Ab, 1)$

$\delta(q_2, a, 2, A) \rightarrow (q_1, Ac, 0)$ $\delta(q_2, b, 2, A) \rightarrow (q_3, c, 0)$

$\delta(q_3, b, 1, A) \rightarrow (q_4, \wedge, 0)$ $\delta(q_4, b, b) \rightarrow (q_4, \wedge, 1)$

$\delta(q_4, c, c) \rightarrow (q_4, \wedge, 1)$ $\delta(q_4, \$, Z_0) \rightarrow (q_f, \wedge, 0)$

Based on our observation in example 4.1, we see that $L(M) = \{a^n b^n c^n \mid n \geq 1\}$

Since $\{a^n b^n c^n \mid n \geq 1\}$ is not a context-free language, the theorem $DDPDA - CF \neq \phi$ holds. □

Theorem 9: $CF - DDPDA \neq \phi$

Proof. This Theorem directly follows from Corollary 3. □

Theorem 10: The number of moves (expansion and pop) required for PDDPDT is less than DDPDT for performing the computation $q_0 w \overset{*}{\rightarrow} q_f ww$.

Proof. Table 4.5 depicts the number of moves required for PDDPDT (Example 4.5) and DDPDT (Example 4.2). Clearly, PDDPDT requires fewer moves than DDPDT.

Table 4.5: Comparative study (in terms of moves) between DDPDT and PDDPDT.

		DDPDT	**PDDPDT**
Input String	**Output String**	**# of Moves**	**# of Moves**
$w = 010, \mid w \mid = 3$	$ww = 010010$	12	9
$w = 0101, \mid w \mid = 4$	$ww = 01010101$	15	11
$w = 01010, \mid w \mid = 5$	$ww = 0101001010$	18	13
$w = 010100, \mid w \mid = 6$	$ww = 010100010100$	21	15
$w = 1011011, \mid w \mid = 7$	$ww = 10110111011011$	24	17

For any input string *w*, we obtain output string ww. DDPDT will require $2 \mid w \mid +3$ expansion where

- $\mid 2w \mid$ moves are required for reading the complete string *w* using depth one, and thereafter after reading each symbol, for generating the same symbol on the stack using depth two. Out of $\mid 2w \mid$, |w| expansion move will generate first *w* portion of the output.
- One expansion move is required for initial expansion $S \rightarrow AA$.

- Two expansions are required for popping *AA*, i.e. one for each *A*, which are generated by the first move.

DDPDT will require |*w*| pop moves, for popping every symbol and generating second *w* portion of the output. Hence, a total number of moves in DDPDT will require $3|w|+3$ moves.

PDDPDT will require $|w|+3$ expansion, where

- $|w|$ expansion moves are required for reading complete string *w* using depth one, and thereafter after reading each symbol, for generating the same symbol on the stack using depth two, owing to the fact that, in parallel variant of deep pushdown transducer, more than one-nonterminal can be expanded simultaneously in one move (See Example 4.5 for more details). The above mentioned |w| expansion move will also generate first *w* portion of the output.
- One expansion move is required for initial expansion $S \rightarrow AA$.
- Two expansions are required for popping *AA*, i.e. one for each *A*, which are generated by the first move.

PDDPDT will require |*w*| pop moves, for popping every symbol and generating second *w* portion of the output. Hence, a total number of moves in PDDPDT will require $2|w|+3$ moves.

Clearly, $3|w|+3 > 2|w|+3$ for |*w*|>1. □

Fig. 4.8 represents the relation between DDPDA, DPDA, PDDPDA, PDPDA, DDPDT, DPDT, PDDPDT, PDPDT, context-free (CF) language and context-sensitive (CS) language.

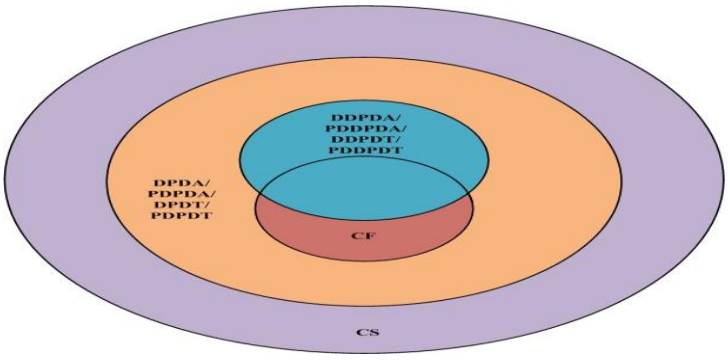


Fig. 4.8: Relation amongst DDPDA, DPDA, PDDPDA, PDPDA, DDPDT, DPDT, PDDPDT, PDPDT, CF and CS

4.5 CONCLUSION

In this chapter, deterministic versions of DPDA, PDPDA, DPDT, and PDPDT have been conceptualized. The proposed models possess the properties of both strict determinism and determinism with respect to depth, which was an open issue addressed in [10-11, 33].

Chapter-5

APPLICATION OF STATE GRAMMAR AND DEEP PUSHDOWN AUTOMATA IN BIOLOGICAL SEQUENCES OF NUCLEIC ACIDS

5.1 INTRODUCTION

Bioinformatics [84] is a field of applied science in which computational and mathematical methods are applied to biology. The analysis of biological structures, such as deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) is a major concern of bioinformatics. The structure of DNA and RNA is determined by the interactions between distantly separated pairs of nucleotides in the molecule. These interactions involve the molecule folding up to form nucleotide sequences in different shapes and forms, such as tandem or interleaved repeats. The grammatical formalism of DNA, RNA, and proteins can be used to solve bioinformatics problems efficiently like multiple alignment calculations, classification, and prediction of biological sequences with their primary and secondary structures and their functions. Hence, there is a growing need to develop a grammatical system in bioinformatics.

Chomsky grammar systems are found to ideal for representing the interactions of nucleotides. In the various existing approaches in the literature, these sequences are represented using context-sensitive grammar or mildly context-sensitive grammar with higher time complexities. However, many fundamental questions regarding the representation of these structures using regulated grammars remain unanswered. In this chapter, we represent deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) biological sequences using state grammar. State grammar specified in definition 1.2.7 is one of the types of rule-based regulated grammar in which restriction are presented in terms of states. These biological sequences are then depicted using formal model deep pushdown automata (DPDA). DPDA specified in definition 1.2.9 is the formal automaton

Results of this chapter has been published in Current Bioinformatics
N. Kalra and A. Kumar, State Grammar and Deep Pushdown Automata for Biological Sequences of Nucleic Acids, Current Bioinformatics, 11 (4), 470-479, 2016.

counterpart of state grammars. Furthermore, we have designed top-down extended LL(1) parser for state grammar. The major benefit of this approach is that the DNA and RNA sequences can be parsed in linear time $O(n)$, where n is the length of the string, which is a significant improvement over the existing approaches.

5.2 STATE GRAMMAR FOR NUCLEIC ACIDS

In this section, we describe the state grammar for biological sequences found in DNA and RNA.

5.2.1 STATE GRAMMAR FOR TANDEM (DIRECT) REPEATS

Tandem repeats [85] occur whenever the nucleotide sequence is duplicated, and duplication occurs directly adjacent to it. Tandem repeats occur in DNA and encompass the feature of copy language. For example, the sequence *ttacg* is duplicated in the string $w = ttacgttacg$.

Grammar $G_1(\{S,X,Y,g,c,a,t\},\{p_0,p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8\},\{g,c,a,t\},P,S)$ be the state grammar where production rules P are as follows:

$1.(p_0,S)\rightarrow(p_0,XY)$

$2.(p_0,X)\rightarrow(p_1,gX)$ $3.(p_1,Y)\rightarrow(p_0,gY)$

$4.(p_0,X)\rightarrow(p_2,cX)$ $5.(p_2,Y)\rightarrow(p_0,cY)$

$6.(p_0,X)\rightarrow(p_3,aX)$ $7.(p_3,Y)\rightarrow(p_0,aY)$

$8.(p_0,X)\rightarrow(p_4,tX)$ $9.(p_4,Y)\rightarrow(p_0,tY)$

$10.(p_0,X)\rightarrow(p_5,g)$ $11.(p_5,Y)\rightarrow(p_0,g)$

$12.(p_0,X)\rightarrow(p_6,c)$ $13.(p_6,Y)\rightarrow(p_0,c)$

$14.(p_0,X)\rightarrow(p_7,a)$ $15.(p_7,Y)\rightarrow(p_0,a)$

$16.(p_0,X)\rightarrow(p_8,t)$ $17.(p_8,Y)\rightarrow(p_0,t)$

A sample derivation for the string, $w = gcatgcat$ is given below:

$S_{p0}\rightarrow XY_{p0}\rightarrow gXY_{p1}\rightarrow gXgY_{p0}\rightarrow gcXgY_{p2}\rightarrow gcXgcY_{p0}\rightarrow gcaXgcY_{p3}\rightarrow gcaXgcaY_{p0}\rightarrow gcatgcaY_{p8}$
$\rightarrow gcatgcat_{p0}$

5.2.2 STATE GRAMMAR FOR INVERTED REPEATS

Inverted repeats [86] occur whenever nucleotides sequence is followed by its reverse complement. Inverted repeats occur in DNA. For example, $5'-aatgc-3'$ is the initial sequence, its complement is $3'-ttacg-5'$ and its reverse complement is $5'-gcatt-3'$. The inverted repeat sequence thus becomes $5'-aatgc\ gcatt-3'$.

Grammar $G_2(\{S,G,C,A,T,g,c,a,t\},\{p_0,p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8\},\{g,c,a,t\},P,S)$ be the state grammar where production rules P are as follows:

1. $(p_0,S)\rightarrow(p_1,gSG)$ 2.$(p_1,G)\rightarrow(p_0,c)$

3.$(p_0,S)\rightarrow(p_2,cSC)$ 4.$(p_2,C)\rightarrow(p_0,g)$

5.$(p_0,S)\rightarrow(p_3,aSA)$ 6.$(p_3,A)\rightarrow(p_0,t)$

7.$(p_{0,}S)\rightarrow(p_4,tsT)$ 8.$(p_4,T)\rightarrow(p_0,a)$

9.$(p_0,S)\rightarrow(p_5,gc)$ 10.$(p_0,S)\rightarrow(p_6,cg)$

11.$(p_0,S)\rightarrow(p_7,at)$ 12.$(p_0,S)\rightarrow(p_8,ta)$

A sample derivation for the string, $w=ttacgcgtaa$ is given below:

$S_{p0}\rightarrow tST_{p4}\rightarrow tSa_{p0}\rightarrow ttSTa_{p4}\rightarrow ttSaa_{p0}\rightarrow ttaSAaa_{p3}\rightarrow ttaStaa_{p0}\rightarrow ttacSCtaa_{p2}\rightarrow ttacSgtaa_{p0}$

$\rightarrow ttacgcgtaa_{p5}$

5.2.3 STATE GRAMMAR FOR INTERLEAVED REPEATS (Pseudoknots)

Pseudoknots are found in tertiary RNA structures. Pseudoknot sequences [87] are formed whenever a pairing occurs between a secondary loop structure and complementary base outside the loop.

Grammar $G_3(\{S,A,B,g,c,a,u\},\{p,q,r,s,t,u',x,x',y,y'\},\{g,c,a,u\},P,S)$ be the state grammar for interleaved repeats, where production rules P are as follows:

1.$(p,S)\rightarrow(q,gAcB)$

2.$(p,S)\rightarrow(q,cAgB)$

3.$(p,S)\rightarrow(q,aAuB)$

$4.(p,S) \rightarrow (q,uAaB)$

$5.(q,A) \rightarrow (r,gAg)$ $6.(r,B) \rightarrow (q,cB)$

$7.(q,A) \rightarrow (s,cAc)$ $8.(s,B) \rightarrow (q,gB)$

$9.(q,A) \rightarrow (t,aAa)$ $10.(t,B) \rightarrow (q,uB)$

$11.(q,A) \rightarrow (u',uAu)$ $12.(u',B) \rightarrow (q,aB)$

$13.(q,A) \rightarrow (x,gg)$ $14.(x,B) \rightarrow (q,c)$

$15.(q,A) \rightarrow (x',cc)$ $16.(x',B) \rightarrow (q,g)$

$17.(q,A) \rightarrow (y,aa)$ $18.(y,B) \rightarrow (q,u)$

$19.(q,A) \rightarrow (y',uu)$ $20.(y',B) \rightarrow (q,a)$

A sample derivation for the string, $w = gacuucacuga$ is given below:

$$S_p \rightarrow gAcB_q \rightarrow gaAacB_t \rightarrow gaAacuB_q \rightarrow gacAcacuB_s \rightarrow gacAcacugB_q \rightarrow gacuucacugB_{y'} \rightarrow$$

$$\rightarrow gacuucacuga_q$$

5.3 DEEP PUSHDOWN AUTOMATA

In this section, a DPDA, state grammar's automaton counterpart, is used to represent the biological sequences found in DNA and RNA.

5.3.1 DPDA FOR TANDEM (DIRECT) REPEATS

DPDA ${}_2M(\{s,p,q,r,s',t,u,v,w,y\},\{g,c,a,t\},\{X,S,\#\},R,s,S,\{u,v,w,y\})$ is depicted in Fig 5.1, where the production rules R are as follows:

$1sS \rightarrow pXX$

$1pX \rightarrow qgX$ $2qX \rightarrow pgX$

$1pX \rightarrow rcX$ $2rX \rightarrow pcX$

$1pX \rightarrow s'aX$ $2s'X \rightarrow paX$

$1pX \rightarrow ttX$ $2tX \rightarrow ptX$

$1pX \rightarrow ug$ $uX \rightarrow ug$

$1pX \to vc$ $\quad vX \to vc$

$1pX \to wa$ $\quad wX \to wa$

$1pX \to yt$ $\quad yX \to yt$

For input accepted string $w = gcatgcat$

$(s, gcatgcat, S\#) \Rightarrow_e (p, gcatgcat, XX\#) \Rightarrow_e (q, gcatgcat, gXX\#) \Rightarrow_p (q, catgcat, XX\#)$
$\Rightarrow_e (p, catgcat, XgX\#) \Rightarrow_e (r, catgcat, cXgX\#) \Rightarrow_p (r, atgcat, XgX\#) \Rightarrow_e (p, atgcat, XgcX\#)$
$\Rightarrow_e (s', atgcat, aXgcX\#) \Rightarrow_p (s', tgcat, XgcX\#) \Rightarrow_e (p, tgcat, XgcaX\#) \Rightarrow_e (y, tgcat, tgcaX\#)$
$\Rightarrow_p (y, gcat, gcaX\#) \Rightarrow_p (y, cat, caX\#) \Rightarrow_p (y, at, aX\#) \Rightarrow_p (y, t, X\#) \Rightarrow_e (y, t, t\#) \Rightarrow_p (y, \wedge, \#)$

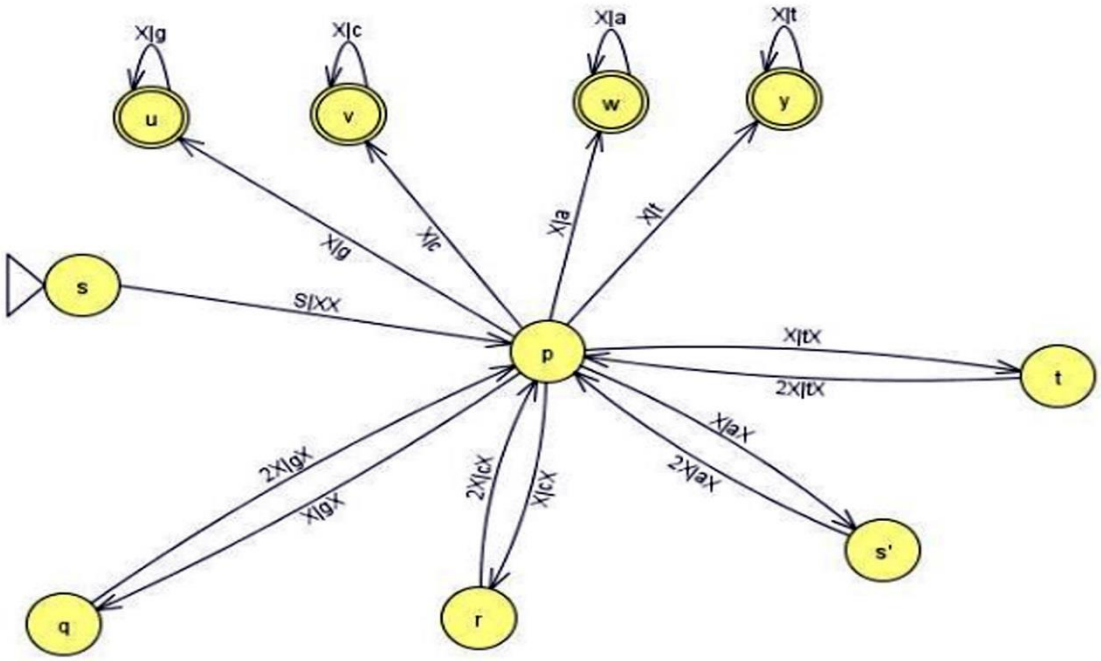


Fig 5.1: DPDA for Tandem (Direct) Repeats

5.3.2 DPDA FOR INVERTED REPEATS

DPDA ${}_2M(\{p,q,r,s,t',u,v,w,x\},\{g,c,a,t\},\{S,\#\},R,p,S,\{u,v,w,x\})$ is depicted in Fig. 5.2, for inverted repeats, and the production rules R are as follows:

$1pS \to qgSS$ $\quad 2qS \to pc$

$1pS \to rcSS$ $\quad 2rS \to pg$

$1pS \to saSS$ $\quad 2sS \to pt$

$1pS \to t't SS$ $\quad\quad$ $2t'S \to pa$

$1pS \to ugc$ $\quad\quad$ $1pS \to vcg$

$1pS \to wat$ $\quad\quad$ $1pS \to xta$

For input accepted string $w = ttacgcgtaa,$

$(p, ttacgcgtaa, S\#) \Rightarrow_e (t', ttacgcgtaa, tSS\#) \Rightarrow_p (t', tacgcgtaa, SS\#) \Rightarrow_e (p, tacgcgtaa, Sa\#)$

$\Rightarrow_e (t', tacgcgtaa, tSSa\#) \Rightarrow_p (t', acgcgtaa, SSa\#) \Rightarrow_e (p, acgcgtaa, Saa\#) \Rightarrow_e (s, acgcgtaa, aSSaa\#)$

$\Rightarrow_p (s, cgcgtaa, SSaa\#) \Rightarrow_e (p, cgcgtaa, Staa\#) \Rightarrow_e (r, cgcgtaa, cSStaa\#) \Rightarrow_p (r, gcgtaa, SStaa\#)$

$\Rightarrow_e (p, gcgtaa, Sgtaa\#) \Rightarrow_e (u, gcgtaa, gcgtaa\#) \Rightarrow_p (u, cgtaa, cgtaa\#) \Rightarrow_p (u, gtaa, gtaa\#)$

$\Rightarrow_p (u, taa, taa\#) \Rightarrow_p (u, aa, aa\#) \Rightarrow_p (u, a, a\#) \Rightarrow_p (u, \wedge, \#)$

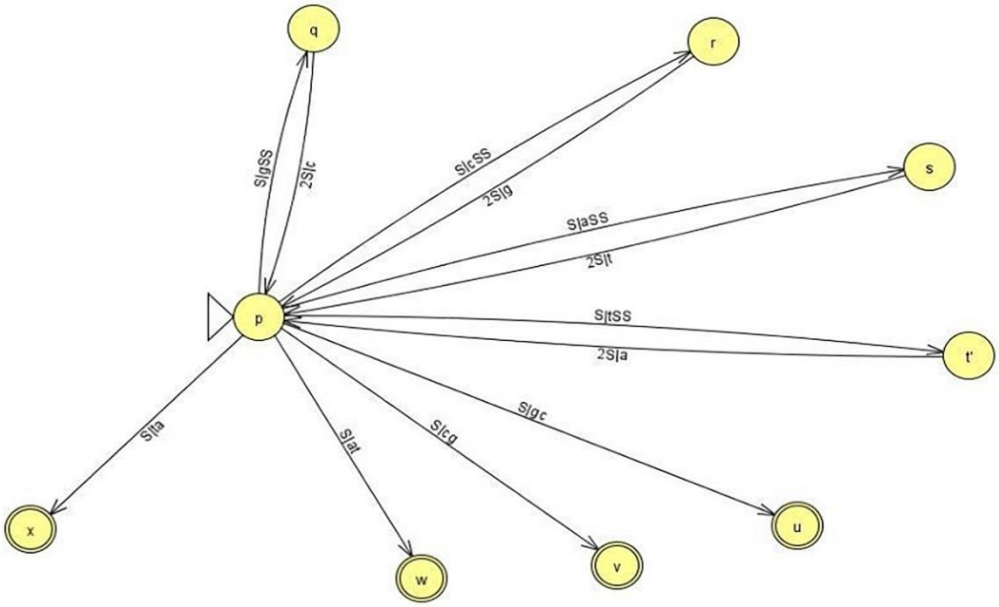


Fig. 5.2: DPDA for Inverted Repeats

5.3.3 DPDA FOR INTERLEAVED REPEATS (Pseudoknots)

DPDA ${}_2M(\{p,q,r,s,t,u',x,x',y,y'\},\{g,c,a,u\},\{S,A,\#\},R,p,S,\{x,x',y,y'\})$ is depicted in Fig. 5.3, for interleaved repeats, and the production rules R are as follows:

$1pS \to qgAcA$ $\quad\quad$ $1pS \to qcAgA$

$1pS \to qaAuA$ $\quad$ $1pS \to quAaA$

$1qA \to rgAg$ $\quad$ $2rA \to qcA$

$1qA \to scAc$ $\quad$ $2sA \to qgA$

$1qA \to taAa$ $\quad$ $2tA \to quA$

$1qA \to u'uAu$ $\quad$ $2u'A \to qaA$

$1qA \to xgg$ $\quad$ $xA \to xc$

$1qA \to x'cc$ $\quad$ $x'A \to xg$

$1qA \to yaa$ $\quad$ $yA \to yu$

$1qA \to y'uu$ $\quad$ $y'A \to ya$

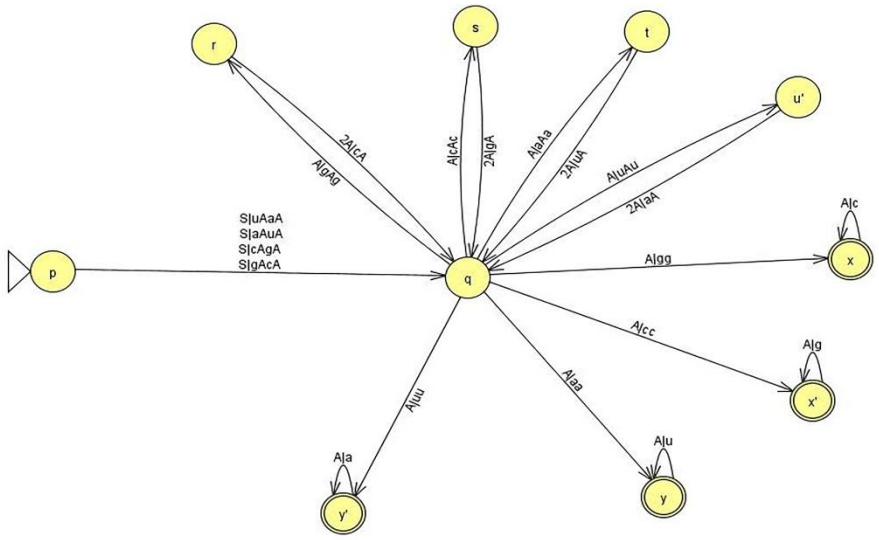


Fig. 5.3: DPDA for Interleaved Repeats (Pseudoknots)

For input string $w = gacuucacuga,$ accepted by ${}_2M_p$.

$(p, gacuucacuga, S\#) \Rightarrow_e (q, gacuucacuga, gAcA\#) \Rightarrow_p (q, acuucacuga, AcA\#) \Rightarrow_e$

$(t, acuucacuga, aAacA\#) \Rightarrow_p (t, cuucacuga, AacA\#) \Rightarrow_e (q, cuucacuga, AacuA\#) \Rightarrow_e$

$(s, cuucacuga, cAcacuA\#) \Rightarrow_p (s, uucacuga, AcacuA\#) \Rightarrow_e (q, uucacuga, AcacugA\#) \Rightarrow_e$

$(y', uucacuga, uucacugA\#) \Rightarrow_p (y', ucacuga, ucacugA\#) \Rightarrow_p (y', cacuga, cacugA\#) \Rightarrow_p$

$(y', acuga, acugA\#) \Rightarrow_p (y', cuga, cugA\#) \Rightarrow_p (y', uga, ugA\#) \Rightarrow_p (y', ga, gA\#) \Rightarrow_p$

$(y', a, A\#) \Rightarrow_e (y', a, a\#) \Rightarrow_p (y', \wedge, \#)$

5.4 EXTENDED LL (1) PARSER FOR STATE GRAMMAR

LL(1) is a top-down parsing that simulates a leftmost derivation for parsing a string. In this section, we examine how LL(1) parsing can be extended for state grammar. A careful examination of the rules in various designed state grammars reveals the following points:

- The above-designed grammars (G_1, G_2, G_3) for tandem repeats, inverted repeats, and interleaved repeats do not contain left-recursion. They contain left-factoring, which is handled by the inclusion of states in the parsing table.
- Construction of Extended LL(1) parsing table: *first* is computed using conventional LL(1) parsing technique [88], except that we consider the state concept in the parsing table. Given a production of the form $P_k : (p_i, A) \rightarrow (p_j, \alpha)$, where P_k denotes the production number, for each alphabet a in $first\ \alpha$, add $P_k : (p_i, A) \rightarrow (p_j, \alpha)$ to $M[A, a]$ in p_i state row and p_j state column.

  In the case of state grammar, we regulate the use of rules by states. To derive a string from the state grammar, the leftmost occurrence of a non-terminal can be rewritten under the current state; and it makes the transition from one state to another, which helps in determining the next rule to be applied. The next rule must be forcefully applied irrespective of the current input tape symbol. Therefore, we replicated the entries for the state which is expanding second non-terminal of start production in LL(1) table as for enforcement (regulation) of the rule which is a necessary condition in case of state grammar.

  As $\wedge \notin first(p_j, \alpha)$, there is no need to compute *follow* in the above-designed grammar.

In LL(1) parsing, a blank entry represents the error. We have omitted some entries of non-terminals and terminals in the LL(1) parsing table for which there is no rule in the corresponding grammar. We have omitted the blank entry to reduce the size of the parsing tables and considered that if an entry does not exist in a table, it indicates a blank entry. The extended LL parser works similarly to an ordinary parser except for the inclusion of state and it can make expansion deeper in the pushdown stack.

Example 5.1: Consider the grammar G_1

Table 5.1 represent the extended LL (1) parsing table for tandem repeats.

Table 5.1: Parsing table for tandem repeat of grammar G_1

		p_0				p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
		g	*c*	*a*	*t*	*g*	*c*	*a*	*t*	*g*	*c*	*a*	*t*
p_0	*S*	1	1	1	1								
	X					2	4	6	8	10	12	14	16
p_1	*Y*	3	3	3	3								
p_2	*Y*	5	5	5	5								
p_3	*Y*	7	7	7	7								
p_4	*Y*	9	9	9	9								
p_5	*Y*	11	11	11	11								
p_6	*Y*	13	13	13	13								
p_7	*Y*	15	15	15	15								
p_8	*Y*	17	17	17	17								

Consider, $w = gcatgcat$ and e_i for $i \geq 1$ denotes the expansion of rule *i* and *p* denotes a pop operation.

$(p_0, gcatgcat, S\#) \Rightarrow_{e_1} (p_0, gcatgcat, XY\#) \Rightarrow_{e_2} (p_1, gcatgcat, gXY\#) \Rightarrow_p (p_1, catgcat, XY\#)$
$\Rightarrow_{e_3} (p_0, catgcat, XgY\#) \Rightarrow_{e_4} (p_2, catgcat, cXgY\#) \Rightarrow_p (p_2, atgcat, XgY\#) \Rightarrow_{e_5} (p_0, atgcat, XgcY\#)$
$\Rightarrow_{e_6} (p_3, atgcat, aXgcY\#) \Rightarrow_p (p_3, tgcat, XgcY\#) \Rightarrow_{e_7} (p_0, tgcat, XgcaY\#) \Rightarrow_{e_{16}} (p_8, tgcat, tgcaY\#)$
$\Rightarrow_p (p_8, gcat, gcaY\#) \Rightarrow_p (p_8, cat, caY\#) \Rightarrow_p (p_8, at, aY\#) \Rightarrow_p (p_8, t, Y\#) \Rightarrow_{e_{17}} (p_0, t, t\#) \Rightarrow_p (p_8, \wedge, \#)$

Example 5.2: Consider the grammar G_2

Table 5.2 represent the extended LL (1) parsing table for the inverted repeat.

Table 5.2: Parsing table for inverted repeat of grammar G_2

		p_0				p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
		g	*c*	*a*	*t*	*g*	*c*	*a*	*t*	*g*	*c*	*a*	*t*
p_0	*S*					1	3	5	7	9	10	11	12
p_1	*G*	2	2	2	2								
p_2	*C*	4	4	4	4								
p_3	*A*	6	6	6	6								
p_4	*T*	8	8	8	8								

Consider, $w = ttacgcgtaa,$

$(p_0, ttacgcgtaa, S\#) \Rightarrow_{e7} (p_4, ttacgcgtaa, tST\#) \Rightarrow_p (p_4, tacgcgtaa, ST\#) \Rightarrow_{e8} (p_0, tacgcgtaa, Sa\#)$
$\Rightarrow_{e7} (p_4, tacgcgtaa, tSTa\#) \Rightarrow_p (p_4, acgcgtaa, STa\#) \Rightarrow_{e8} (p_0, acgcgtaa, Saa\#) \Rightarrow_{e5} (p_3, acgcgtaa, aSAaa\#)$
$\Rightarrow_p (p_3, cgcgtaa, SAaa\#) \Rightarrow_{e6} (p_0, cgcgtaa, Staa\#) \Rightarrow_{e3} (p_2, cgcgtaa, cSCtaa\#) \Rightarrow_p (p_2, gcgtaa, SCtaa\#)$
$\Rightarrow_{e4} (p_0, gcgtaa, Sgtaa\#) \Rightarrow_{e9} (p_5, gcgtaa, gcgtaa\#) \Rightarrow_p (p_5, cgtaa, cgtaa\#) \Rightarrow_p (p_5, gtaa, gtaa\#)$
$\Rightarrow_p (p_5, taa, taa\#) \Rightarrow_p (p_5, aa, aa\#) \Rightarrow_p (p_5, a, a\#) \Rightarrow_p (p_5, \wedge, \#)$

Example 5.3: Consider the grammar G_3 for pseudoknots

Table 5.3 represent the extended LL (1) parsing table for pseudoknots.

Table 5.3: Parsing table for pseudoknots grammar G_3

		q				*r*	*s*	*t*	*u'*	*x*	*x'*	*y*	*y'*
		g	*c*	*a*	*u*	*g*	*c*	*a*	*u*	*g*	*c*	*a*	*u*
p	S	1	2	3	4								
q	A					5	7	9	11	13	15	17	19
r	B	6	6	6	6								
s	B	8	8	8	8								
t	B	10	10	10	10								
u'	B	12	12	12	12								
x	B	14	14	14	14								
x'	B	16	16	16	16								
y	B	18	18	18	18								
y'	B	20	20	20	20								

Consider, $w = gacuucacuga$

$(p, gacuucacuga, S\#) \Rightarrow_{e1} (q, gacuucacuga, gAcB\#) \Rightarrow_p (q, acuucacuga, AcB\#)$
$\Rightarrow_{e9} (t, acuucacuga, aAacB\#) \Rightarrow_p (t, cuucacuga, AacB\#) \Rightarrow_{e10} (q, cuucacuga, AacuB\#)$
$\Rightarrow_{e7} (s, cuucacuga, cAcacuB\#) \Rightarrow_p (s, uucacuga, AcacuB\#) \Rightarrow_{e8} (q, uucacuga, AcacugB\#)$
$\Rightarrow_{e19} (y', uucacuga, uucacugB\#) \Rightarrow_p (y', ucacuga, ucacugB\#) \Rightarrow_p (y', cacuga, cacugB\#)$
$\Rightarrow_p (y', acuga, acugB\#) \Rightarrow_p (y', cuga, cugB\#) \Rightarrow_p (y', cuga, cugB\#) \Rightarrow_p (y', uga, ugB\#)$
$\Rightarrow_p (y', ga, gB\#) \Rightarrow_p (y', a, B\#) \Rightarrow_{e20} (y', a, a\#) \Rightarrow_p (y', \wedge, \#)$

5.5 RESULTS AND DISCUSSIONS

Theorem 1: Consider $G_s(V, Q, \sum, P, S)$ as the state grammar and M as the corresponding deep pushdown automata. The string w can be parsed in $O(n)$ using the Extended LL(1) parser, where n is the length of the input string w.

Proof. Given a string w, the string can be parsed in $O(n)$ using the ordinary LL(1) parser [88]. In the case of state grammar, we can extend it by using the concept proposed by Meduna and Zemek [3] for the parsing of a string represented by k-linear scattered context grammar using deep pushdown automata. We can use a doubly linked list (the 1st element represents the top of the stack) for making expansion deeper in the stack [3]. Operation on a doubly linked list can be performed in $O(1)$ as the depth of the deep pushdown automata is independent of the length of the string. Other operations can be performed similarly as in an ordinary LL parser in $O(n)$ time.

If the current input symbol is g and \$ denotes the end marker of string. The following steps are performed:

- If $g = \$$ and the stack is empty, the string is accepted. This operation can be performed in $O(1)$ [89].
- If the topmost symbol on the stack is a non-terminal (say Y for a state p_i) and the current symbol of the input string is g, then the $(p_i, Y) \rightarrow (p_j, gY)$ rule is applied without moving the read-write head, and it can be performed in $O(1)$.
- If the topmost symbol and next input symbol match, then perform the pop operation, read/write head points to next symbol, and it can be carried out in $O(1)$ [89].

Hence, if the length of the input string w is n, it can be parsed in $O(n)$ using the extended LL parser. □

There exist various approaches for representing DNA and RNA sequences using context-sensitive grammar or indexed grammar. The parsing of context-sensitive grammars and indexed grammar is Exponential and NP-complete, respectively. Attempts have been made to represent these sequences using simple linear tree-adjoining, extended simple linear tree-adjoining and parallel communicating grammar with time complexities $O(n^4)$, $O(n^5)$ and $O(n^6)$, respectively. Table 5.4 depicts the major differences amongst the various existing approaches in the literature and the proposed approach.

Table 5.4: Comparison amongst various existing approaches and proposed approach

Criteria → / Approach ↓	**Proposed grammatical (language) formalism**	**Time Complexity**
Sung Approach [26]	Context-sensitive	Exponential
Searls Approach [28]	Indexed Grammar	NP-Complete
Uemura et al. Approach[30]	Simple Linear Tree Adjoining Grammar and Extended Simple Linear Tree Adjoining Grammar	$O(n^4)$ and $O(n^5)$, respectively
Cai et al. Approach [25]	Parallel Communicating Grammar Systems	$O(n^6)$
Proposed Approach	State Grammar	$O(n)$

We have proposed an approach for representing these sequences using state grammar. The major benefit of representing these sequences using state grammar is that they can be parsed in linear with time complexity $O(n)$, where n is the length of the string.

5.6 CONCLUSION

In this chapter, we have represented DNA and RNA biological sequences using state grammar and deep pushdown automata. The major benefit is that the membership of state grammar can be decided in linear time $O(n)$, where n is the length of the string.

Chapter-6

MODELLING BIOMOLECULAR STRUCTURES OF DNA AND RNA USING STATE GRAMMAR

6.1 INTRODUCTION

DNA molecules sequence is made up of nucleotides, which are of four forms: adenine (a), guanine (g), cytosine (c) and thymine (t). RNA molecules sequence is made up of nucleotides, which are of four forms: adenine (a), guanine (g), cytosine (c) and uracil (u).

The complement of a symbol u is represented by $\bar{u}$. Complement pairing in DNA is $\bar{a}=t,\ \bar{t}=a,\ \bar{g}=c,$ and $\bar{c}=g$. Complement pairing in RNA is $\bar{a}=u,\ \bar{u}=a,\ \bar{g}=c,$ and $\bar{c}=g$. Based on complementary pairs and other biological constraints, these sequences lead to the formation of some patterns (structures). These structures have a number of functional features. The grammatical formalism of biological structures such as DNA, RNA and proteins can be used to solve bioinformatics problems efficiently, such as multiple alignment calculations, classification, and prediction of biological sequences with their primary and secondary structures.

Chomsky grammar systems are found to ideal for representing the interactions of nucleotides as there is a similarity between formal languages and biomolecules language. There are some RNA structures like hairpin which matches with language ww^R (the language of palindrome) which requires context-free grammar. There are some other structures like attenuator $\{u\bar{u}^R u\bar{u}^R \mid u \in \sum^*_{DNA}\}$ which matches with ww (copy language) which require power beyond context-free grammar. There are various other biological structures such as extended pseudoknot, recursive pseudoknot, simple h-type, kissing hairpin, three-knot which includes crossing dependencies, which also requires power beyond context-free grammar.

In this chapter, we are mainly concerned to extend the previous work of our Chapter 5 for representing various biological structures of DNA and RNA using state grammar.

6.2 STATE GRAMMAR FOR BIOMOLECULAR STRUCTURES

This section describes the state grammar for biomolecular structures found in DNA and RNA.

Theorem 1: The attenuator structure language $L_A = \{u\bar{u}^R u\bar{u}^R \mid u \in \sum^*_{DNA}\}$ [32] can be generated by state grammar. Fig. 6.1 represents attenuator structure of the sequence $\{gtcgacgtcgac\}$.

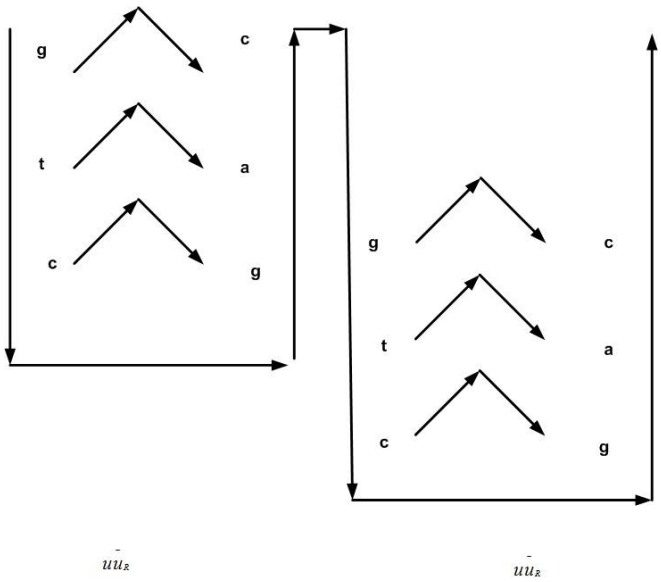


Fig 6.1: Attenuator structure

Proof: Grammar $G_1(\{S,A,B,u\},\{q_0,q_1,q_2,q_3\},\{g,c,a,t\},P,S)$ be the state grammar where production rules P are as follows:

$(q_0,S) \rightarrow (q_1,AB)$

$(q_1,A) \rightarrow (q_2,uA\bar{u})$ $\qquad$ $(q_2,B) \rightarrow (q_1,uB\bar{u})$

$(q_1,A) \rightarrow (q_3,\wedge)$ $\qquad$ $(q_3,B) \rightarrow (q_4,\wedge)$

A sample derivation for the string, $s = gtatacgtatac$

$S_{q0} \rightarrow AB_{q1} \rightarrow gAcB_{q2} \rightarrow gAcgBc_{q1} \rightarrow gtAacgBc_{q2} \rightarrow gtAacgtBac_{q1} \rightarrow gtaAtacgtBac_{q2}$
$\rightarrow gtaAtacgtaBtac_{q1} \rightarrow gtatacgtaBtac_{q3} \rightarrow gtatacgtatac_{q4}$ □

Theorem 2: The extended pseudoknot language $L_{ExP} = \{u_1 v_1 \bar{u}_1^R u_2 \bar{v}_1^R \bar{u}_2^R \mid u_1,u_2,v_1 \in \sum^*_{RNA}\}$ [32] can be generated by state grammar. Fig. 6.2 represents the structure of extended pseudoknot sequence $\{cugcuacagcguuagacg\}$.

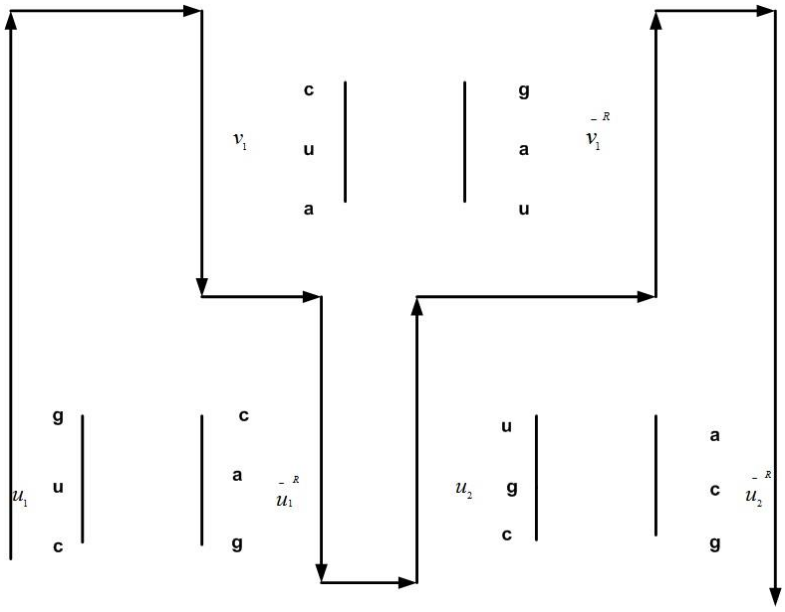


Fig. 6.2: Extended pseudoknot structure

Proof: Grammar $G_2(\{S,A,B,u_1,u_2,v_1\},\{q_0,q_1,q_2,q_3,q_4,q_5\},\{g,c,a,u\},P,S)$ be the state grammar where production rules P are as follows:

$(q_0,S)\rightarrow(q_1,AB)$ $\quad$ $(q_1,A)\rightarrow(q_1,u_1A\bar{u}_1)$

$(q_1,B)\rightarrow(q_1,u_2B\bar{u}_2)$ $\quad$ $(q_1,A)\rightarrow(q_2,A)$

$(q_2,A)\rightarrow(q_3,v_1A)$ $\quad$ $(q_3,B)\rightarrow(q_2,B\bar{v}_1)$

$(q_2,A)\rightarrow(q_4,\wedge)$ $\quad$ $(q_4,B)\rightarrow(q_5,\wedge)$

A sample derivation for the string, $s=gaguucucacga$

$S_{q0}\rightarrow AB_{q1}\rightarrow gAcB_{q1}\rightarrow gaAucB_{q1}\rightarrow gaAucuBa_{q1}\rightarrow gaAucucBga_{q1}\rightarrow gaAucucBga_{q2}$
$\rightarrow gagAucucBga_{q3}\rightarrow gagAucucBcga_{q2}\rightarrow gaguAucucBcga_{q3}\rightarrow gaguAucucBacga_{q2}$
$\rightarrow gaguucucBacga_{q4}\rightarrow gaguucucacga_{q5}$ □

Theorem 3: The simple h-type structure language $L_{SH}=\{u_1v_1u_2\bar{u}_1^R\bar{v}_2\bar{u}_2^R \mid u_1,u_2,v_1,v_2\in \sum^*_{RNA}\}$ [32] can be generated by state grammar. Fig. 6.3 represents the structure of the simple h-type sequence.

Proof: Grammar $G_3(\{S,A,B,u_1,u_2,v_1,v_2\},\{q_0,q_1,q_2,q_3,q_4,q_5,q_6\},\{g,c,a,u\},P,S)$ be the state grammar where production rules P are as follows:

$(q_0,S)\rightarrow(q_1,AB)$ $\quad$ $(q_1,A)\rightarrow(q_1,u_1A\bar{u}_1)$

$(q_1, A) \rightarrow (q_2, A)$ $(q_2, A) \rightarrow (q_2, v_1 A)$

$(q_2, B) \rightarrow (q_3, B)$ $(q_3, B) \rightarrow (q_3, v_2 B)$

$(q_3, A) \rightarrow (q_4, u_2 A)$ $(q_4, B) \rightarrow (q_3, B\bar{u}_2)$

$(q_3, A) \rightarrow (q_5, \wedge)$ $(q_5, B) \rightarrow (q_6, \wedge)$

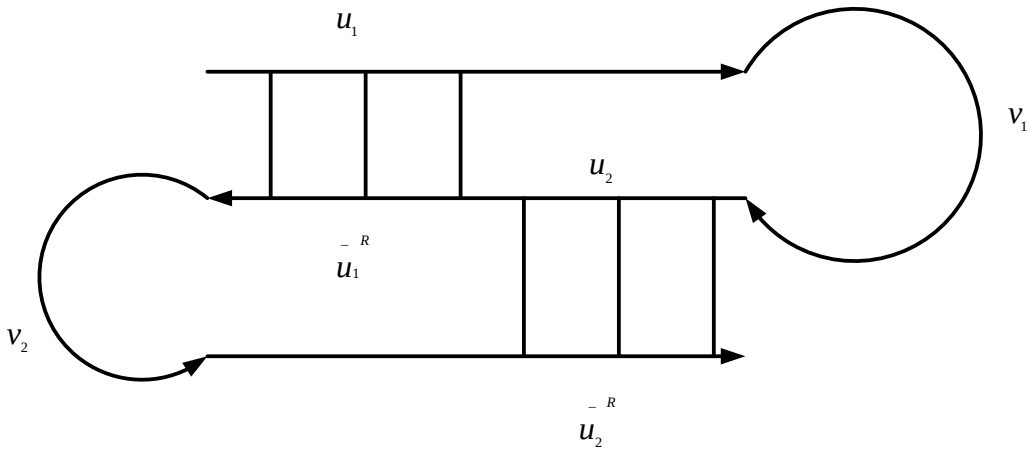


Fig. 6.3: Simple h-type structure

A sample derivation for the string, $s = cugugcucagacag$

$S_{q0} \rightarrow AB_{q1} \rightarrow cAgB_{q1} \rightarrow cuAagB_{q1} \rightarrow cugAcagB_{q1} \rightarrow cugAcagB_{q2} \rightarrow cuguAcagB_{q2} \rightarrow cugugAcagB_{q2}$
$\rightarrow cugugAcagB_{q3} \rightarrow cugugAcagaB_{q3} \rightarrow cugugAcagacB_{q3} \rightarrow cugugcAcagacB_{q4} \rightarrow cugugcAcagacBg_{q3}$
$\rightarrow cugugcuAcagacBg_{q4} \rightarrow cugugcuAcagacBag_{q3} \rightarrow cugugcucagacBag_{q5} \rightarrow cugugcucagacag_{q6}$

□

Theorem 4: The three-knot structure language $L_{TK} = \{u_1 v u_2 u_3 \bar{u}_1^R \bar{u}_2^R \bar{u}_3^R \mid u_1, u_2, u_3, v \in \sum^*_{RNA}\}$ [32] can be generated by state grammar. Fig. 6.4 represents the structure of three - knot sequence.

Proof: Grammar $G_4(\{S, A, B, C, u_1, u_2, u_3, v\}, \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}, \{g, c, a, u\}, P, S)$ be the state grammar where production rules P are as follows:

$(q_0, S) \rightarrow (q_1, ABC)$ $(q_1, A) \rightarrow (q_1, u_1 A \bar{u}_1)$

$(q_1, A) \rightarrow (q_2, A)$ $(q_2, A) \rightarrow (q_2, vA)$

$(q_2, A) \rightarrow (q_3, A)$ $(q_3, A) \rightarrow (q_4, u_2 A)$

$(q_4, B) \rightarrow (q_3, B\bar{u}_2)$ $\quad\quad$ $(q_3, B) \rightarrow (q_5, \wedge)$

$(q_5, A) \rightarrow (q_6, u_3 A)$ $\quad\quad$ $(q_6, C) \rightarrow (q_5, C\bar{u}_3)$

$(q_5, A) \rightarrow (q_7, \wedge)$ $\quad\quad$ $(q_7, C) \rightarrow (q_8, \wedge)$

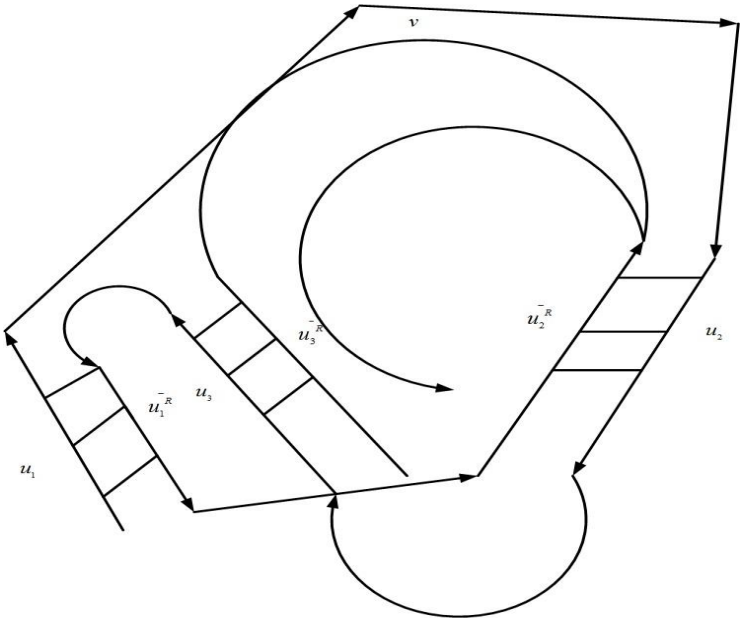


Fig. 6.4: Three-knot structure

A sample derivation for the string, $s = cugcugacacagucug$

$S_{q0} \rightarrow ABC_{q1} \rightarrow cAgBC_{q1} \rightarrow cuAagBC_{q1} \rightarrow cugAcagBC_{q1} \rightarrow cugAcagBC_{q2} \rightarrow cugcAcagBC_{q2}$
$\rightarrow cugcuAcagBC_{q2} \rightarrow cugcuAcagBC_{q3} \rightarrow cugcugAcagBC_{q4} \rightarrow cugcugAcagBcC_{q3}$
$\rightarrow cugcugaAcagBcC_{q4} \rightarrow cugcugaAcagBucC_{q3} \rightarrow cugcugaAcagucC_{q5} \rightarrow cugcugacAcagucC_{q6}$
$\rightarrow cugcugacAcagucCg_{q5} \rightarrow cugcugacaAcagucCg_{q6} \rightarrow cugcugacaAcagucCug_{q5}$
$\rightarrow cugcugacacagucCug_{q7} \rightarrow cugcugacacagucug_{q8}$ □

Theorem 5: The recursive pseudoknot structure language $L_{RC} = \{u_1 u_2 u_3 \bar{u}_2^{R} u_4 \bar{u}_1^{R} \bar{u}_4^{R} u_5 \bar{u}_5^{R} \bar{u}_3^{R} \mid u_1, u_2, u_3, u_4, u_5 \in \sum^*_{RNA}\}$ [32] can be generated by state grammar. Fig. 6.5 represents the structure of recursive pseudoknot sequence.

Proof: Grammar $G_5(\{S, A, B, C, u_1, u_2, u_3, u_4, u_5\}, \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}, \{g, c, a, u\}, P, S)$ be the state grammar where production rules P are as follows:

$(q_0, S) \rightarrow (q_1, ABC)$ $(q_1, B) \rightarrow (q_1, u_4 B \bar{u}_4)$

$(q_1, A) \rightarrow (q_2, A)$ $(q_2, A) \rightarrow (q_3, u_1 A)$

$(q_3, B) \rightarrow (q_2, B\bar{u}_1)$ $(q_2, A) \rightarrow (q_4, A)$

$(q_4, A) \rightarrow (q_4, u_2 A \bar{u}_2)$ $(q_4, B) \rightarrow (q_5, \wedge)$

$(q_5, A) \rightarrow (q_6, u_3 A)$ $(q_6, C) \rightarrow (q_5, C\bar{u}_3)$

$(q_5, A) \rightarrow (q_7, \wedge)$ $(q_5, C) \rightarrow (q_7, u_5 C \bar{u}_5)$

$(q_7, C) \rightarrow (q_8, \wedge)$

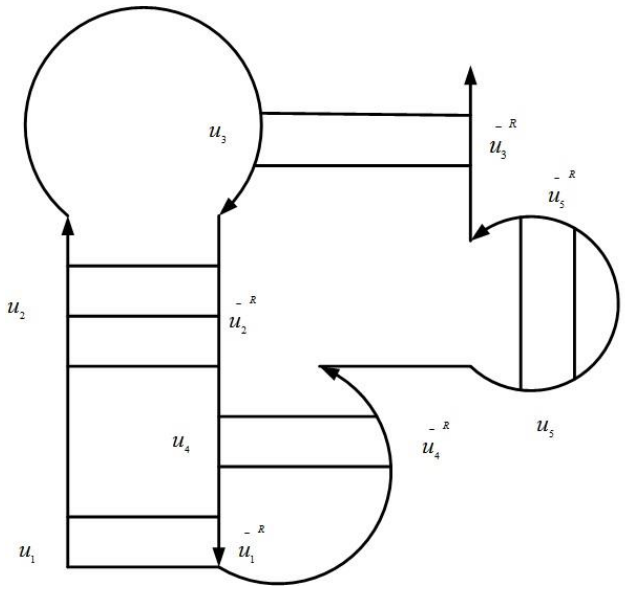


Fig. 6.5: Recursive Pseudoknot structure

A sample derivation for the string, $s = cuacgaguguagacguacuc$

$S_{q0} \rightarrow ABC_{q1} \rightarrow AgBcC_{q1} \rightarrow AguBacC_{q1} \rightarrow AguBacC_{q2} \rightarrow cAguBacC_{q3} \rightarrow cAguBgacC_{q2}$
$\rightarrow cuAguBgacC_{q3} \rightarrow cuAguBagacC_{q2} \rightarrow cuAguBagacC_{q4} \rightarrow cuaAuguBagacC_{q4} \rightarrow cuacAguguBagacC_{q4}$
$\rightarrow cuacAguguagacC_{q5} \rightarrow cuacgAguguagacC_{q6} \rightarrow cuacgAguguagacCc_{q5} \rightarrow cuacgaAguguagacCc_{q6}$
$\rightarrow cuacgaAguguagacCuc_{q5} \rightarrow cuacgaguguagacCuc_{q7} \rightarrow cuacgaguguagacgCcuc_{q7}$
$\rightarrow cuacgaguguagacguCacuc_{q7} \rightarrow cuacgaguguagacguacuc_{q8}$ □

Theorem 6: The kissing hairpin structure language $L_{KH} = \{u_1 v_1 A \bar{A} v_2 u_2 \bar{u}_2^R v_3 B \bar{B} v_4 \bar{u}_1^R \mid u_1, u_2, v_1, v_2, v_3, v_4 \in \sum^*_{RNA}, A, B \in \sum_{RNA}\}$ [32] can be generated by state grammar. Fig. 6.6 represents the structure of kissing hairpin sequence.

Proof: Grammar $G_6(\{S,A,B,C,D,u_1,u_2,v_1,v_2,v_3,v_4\},\{q_0,q_1,q_2,q_3,q_4,q_5,q_6,q_7,q_8,q_9,q_{10},q_{11}\},\{g,c,a,u\},P,S)$ be the state grammar where production rules P are as follows:

$(q_0,S)\rightarrow(q_1,CD)$ $\quad$ $(q_1,C)\rightarrow(q_2,u_1C)$

$(q_2,D)\rightarrow(q_1,D\bar{u}_1)$ $\quad$ $(q_1,C)\rightarrow(q_3,C)$

$(q_3,C)\rightarrow(q_3,v_1C)$ $\quad$ $(q_3,D)\rightarrow(q_3,Dv_4)$

$(q_3,C)\rightarrow(q_4,AC)$ $\quad$ $(q_4,C)\rightarrow(q_5,\bar{A}C)$

$(q_5,D)\rightarrow(q_6,D\bar{B})$ $\quad$ $(q_6,D)\rightarrow(q_7,DB)$

$(q_7,D)\rightarrow(q_7,Dv_3)$ $\quad$ $(q_7,C)\rightarrow(q_7,v_2C)$

$(q_7,C)\rightarrow(q_8,C)$ $\quad$ $(q_8,C)\rightarrow(q_9,u_2C)$

$(q_9,D)\rightarrow(q_8,D\bar{u}_2)$ $\quad$ $(q_8,C)\rightarrow(q_{10},\wedge)$

$(q_{10},D)\rightarrow(q_{11},\wedge)$

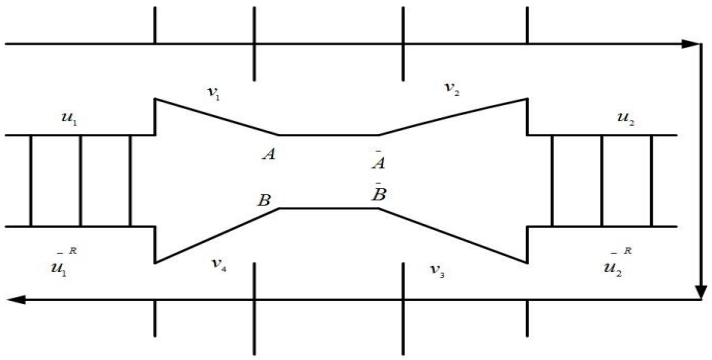


Fig. 6.6 Kissing Hairpin structure

A sample derivation for the string, $s = gaugaucuguaccagcgcuc$

$S_{q0}\rightarrow CD_{q1}\rightarrow gCD_{q2}\rightarrow gCDc_{q1}\rightarrow gaCDc_{q2}\rightarrow gaCDuc_{q1}\rightarrow gaCDuc_{q3}\rightarrow gauCDuc_{q3}\rightarrow gaugCDuc_{q3}\rightarrow gaugCDcuc_{q3}\rightarrow gaugCDgcuc_{q3}\rightarrow gaugaCDgcuc_{q4}\rightarrow gaugauCDgcuc_{q5}\rightarrow gaugauCDcgcuc_{q6}\rightarrow gaugauCDgcgcuc_{q7}\rightarrow gaugaucCDgcgcuc_{q7}\rightarrow gaugaucuCDgcgcuc_{q7}\rightarrow gaugaucuCDagcgcuc_{q7}\rightarrow gaugaucuCDcagcgcuc_{q7}\rightarrow gaugaucuCDcagcgcuc_{q8}\rightarrow gaugaucugCDcagcgcuc_{q9}\rightarrow gaugaucugCDccagcgcuc_{q8}\rightarrow gaugaucuguCDccagcgcuc_{q9}\rightarrow gaugaucuguCDaccagcgcuc_{q8}\rightarrow gaugaucuguDaccagcgcuc_{q10}\rightarrow gaugaucuguaccagcgcuc_{q11}$

□

6.3 CONCLUSION

Some of the DNA and RNA structures include crossing dependencies, so it cannot be represented by regular grammar or context-free grammar; it requires power beyond context-free grammar. In this chapter, we have represented biomolecular structures of DNA and RNA such as an attenuator, extended pseudoknot, kissing hairpin, simple h-type, recursive pseudoknot and three-knot using state grammar.

Chapter-7

AN ALGORITHM FOR ERROR TOLERANCE IN RECOGNITION OF FAULTY STRINGS IN A REGULATED GRAMMAR USING FUZZY SETS

7.1 INTRODUCTION

The regulated grammar consists of production rules similar as context-free grammar, but it has a larger generative capacity than context-free grammar. Regulated grammar lies between context-free and context-sensitive grammar [1-3]. The interleaved dependencies that are captured by context-sensitive or type-0 grammar can be represented using regulated grammar. For example, languages such as ww, $a^n b^n c^n$, a^{2^n} can be represented by the regulated grammar.

Schinder et al.[36] described an approach for recognizing imperfect strings generated by the context-free grammar. Inui et al. [37] expanded upon the work of Schinder et al.[36], describing an approach for identifying imperfect strings generated by the context-sensitive grammar. Both Schinder et al.[36] and Inui et al. [37] used a number of fuzzy sets to determine the membership values of imperfect strings that do not belong to the grammar.

Motivated by the prior work of recognition of imperfect strings in context-free grammar [36] and context-sensitive grammar [37], in this chapter, an algorithm is proposed for the recognition of faulty strings in regulated grammar. As well as, the problem of single and multiple fault occurrences are discussed. Faulty string refers to some imperfection in a string. Single fault refers to imperfection at one position in a string, whereas multiple faults refer to imperfection at more than one position in a string.

Furthermore, depending upon the errors and certainty, it is decided whether the string belongs to the language or not based on its membership value. Here, fuzzy sets are used to compute the membership of strings that do not exactly match with the grammar. The time complexity of the proposed algorithm is $O(|G_R{}^2|.|w|)$, where $|G_R|$ represents the

Results of this chapter are communicated in "LINGUA" entitled:
Error tolerance for the recognition of faulty strings in a regulated grammar using fuzzy sets

number of production rules and $|w|$ is the length of the input string, w. The algorithm is applied to regularly controlled specified in definition 1.2.5 and matrix grammar specified in definition 1.2.6. Regularly controlled and Matrix grammar are one of the types of rule-based regulated grammar. Finally, the proposed algorithm is applied in the Hindi language for the recognition of faulty strings in regulated grammar as a real-life application.

7.2 PRELIMINARIES CONCEPT

DEFINITION 7.2.1. Regular expression [39] over an alphabet $\sum$ is defined using following rules:

- ϕ and $\wedge$ are regular expressions representing the empty set and empty string.
- If $a \in \sum,$ then a is a regular expression represents the singleton set $\{a\}$.
- If r_1 and r_2 are regular expressions, then $r_1 r_2, r_1 | r_2$ and r_1^* are regular expressions represents concatenation, union and Kleene closure.

DEFINITION 7.2.2. A context-free grammar is in Chomsky normal form [5] if the production rules in *P* are of the following form:

- $A \to BC \mid A, B, C \in N$
- $A \to a \mid A \in N$ and $a \in \sum$

DEFINITION 7.2.3. Given $f(x)$ and $g(x)$ are two functions defined on a subset of real numbers. We can write $f(x) = O(g(x))$ if there exists C and N constants such that $|f(x)| \leq C|g(x)|$ for all $x > N$ [89].

7.3 PROPOSED ALGORITHM

In classical automata theory, a string is either accepted or rejected. Fuzzy languages close the gap between natural and formal languages. In the fuzzy set, a generalization of classical set theory, the degree to which an element belongs to the set is referred to as the degree of membership [90]. In this section, an algorithm is proposed for the recognition of faulty strings in regulated grammar using fuzzy sets (*RFSRG*). Given a regularly controlled grammar $G_{RC}(G, r)$. Production rules of G are in context-free grammar. For avoiding multiple errors, in step 1, $G_{RC}(G, r)$ is converted into $G_{RC'}(G', r')$ where G' is Chomsky normal form of G. Every production $p_i \in G$ is converted into productions

$p_i^1, p_i^2, ..., p_i^k \mid k \geq 1$ of G'. Specifically, G is converted into Chomsky normal form G' using the following operations [91]:

- DEL: Elimination of null production $(A \to \wedge)$
- TERM: Elimination of the terminal symbol at the right-hand side, except for length one.
- UNIT: Elimination of unit production $(A \to B)$
- BIN: Reduction of non-terminals at right-hand side and conversion to the form $A \to BC$

In step 2, control derivation r of G_{RC} is converted into r' of $G_{RC'}$ such that $L(G', r') = L(G, r) - \{\wedge\}$.

PROCEDURE 1: Regularly controlled_**c**ontrol_derivation $(G_{RC}, G_{RC'})$

Input: Regularly controlled grammar $G_{RC}(G, r)$ and $G_{RC'}(G', r')$ where G' is Chomsky normal form of G and initially $r' = r$.

Output: Control derivation r' such that $L(G', r') = L(G, r) - \{\wedge\}$.

begin

do $\exists\, p_i \in r$

Replace p_i with $p_i^1, p_i^2, ..., p_i^k$ in r'.

od

end

EXAMPLE 7.1: Consider the regularly controlled grammar of example 1.1 $G_{RC}(G, r)$, where $G(\{S, C, D\}, \{c, d\}, S, P)$ and the production P is of the following forms:

$pr_0 = S \to CD$ $\qquad$ $pr_1 = C \to cC$

$pr_2 = D \to cD$ $\qquad$ $pr_3 = C \to dC$

$pr_4 = D \to dD$ $\qquad$ $pr_5 = C \to c$

$pr_6 = D \to c$ $\qquad$ $pr_7 = C \to d$

$pr_8 = D \to d$

$$r = pr_0 (pr_1 pr_2, pr_3 pr_4)^* (pr_5 pr_6, pr_7 pr_8)$$

Equivalent Chomsky normal form (CNF) for the grammar G_{RC} :

$pr_0 = S \to CD$ $\qquad$ $pr_1^1 = C \to A_1 C$

$pr_1^2 = A_1 \to c$ $pr_2^1 = D \to A_2D$

$pr_2^2 = A_2 \to c$ $pr_3^1 = C \to A_3C$

$pr_3^2 = A_3 \to d$ $pr_4^1 = D \to A_4D$

$pr_4^2 = A_4 \to d$ $pr_5 = C \to c$

$pr_6 = D \to c$ $pr_7 = C \to d$

$pr_8 = D \to d$

r is converted into $r' = pr_0(pr_1^1 pr_1^2 pr_2^1 pr_2^2, pr_3^1 pr_3^2 pr_4^1 pr_4^2)^*(pr_5 pr_6, pr_7 pr_8)$

In step 3, we apply replace, add and remove operators to the productions of the form $A \to a \mid A \in N$ and $a \in \sum$. Hence, there is no possibility in occurrence of multiple errors using one production. Errors in the input string are classified into following three types:

1. **Replacement Error [36-37]:** In replacement error, a terminal is substituted with another terminal symbol from the set $\sum$.
2. **Addition Error [36-37]:** In addition error, an extra new terminal symbol is added from the terminal set $\sum$. An extra new terminal can be placed before or after the terminal.
3. **Removal Error [36-37]:** In removal error, a terminal is replaced by the empty string $\wedge$.

To accumulate the faulty string replace, add and remove operators are considered.

PROCEDURE 2: Error_productions $(G_{RC'}, P')$

Input: Regularly controlled grammar $G_{RC'}(G', r')$ with production set P.

Output: Additional production P' are generated using replace, add and remove operator.

Initialization: $P' = \phi$

begin

do $\exists p_i \in P$

If p_i **is of the form** $A \to a \mid A \in N \wedge a \in \sum$

$P' = P' \bigcup \{A \to \wedge\}$ **//Using Remove operator**

do $\exists b \in \sum$

If $(a \neq b)$

$P' = P' \bigcup \{A \to b\}$ **//Using Replace operator**

$P' = P' \bigcup \{A \to ba\} \bigcup \{A \to ab\}$ **//Using Add operator**

od

od

$P \leftarrow P \bigcup P'$

end

As regularly controlled grammar is ambiguous, in step 4, different derivations for the string, $w \notin L,$ were generated. In one derivation of a string, total $2|w|-1$ productions are used. In each derivation of a string, *w,* the number of productions of the form $A \to BC \mid A, B, C \in N$ can be $|w|-1$ and the number of productions of the form $A \to a \mid A \in N \wedge a \in \sum$ can be $|w|$.

In step 5, fuzzy set E [36] is used for determining the number of erroneous substitutions. Membership of a faulty string is determined by:

$$\mu_E(w) = \frac{|w| - |E_p|}{|w|}$$

where $|w|$ represents the length of the string, *w,* and $|E_p|$ represents the number of error productions used for the generation of string, w.

A certainty factor was defined that can be used for the acceptance and rejection of strings. In step 6, a fuzzy set T_f [36] is used for determining the derivation with minimal production rules utilized to derive the faulty string, w. Membership of a faulty string is determined by:

$$\mu_{T_f}(w) = 1 - \begin{cases} 0, & x \le a \\ 2\left(\frac{x-a}{b-a}\right)^2, & a < x \le (a+b)/2 \\ 1-2\left(\frac{x-b}{b-a}\right)^2, & (a+b)/2 < x < b \\ 1, & x \ge b \end{cases}$$

where *a* and *b* represent the minimum and a maximum number of productions used to derive the faulty string, *w,* in all derivations, respectively. *x* denotes the actual number of productions used in a particular derivation of the faulty string, w. In this *a* and *b* plots the extreme points of the sloped curve, therefore making the s-shape.

In step 7, the optimal derivation for the string w, was chosen. The optimal derivation was found by choosing the highest degree of membership obtained from E-set and T_f-set

---

Algorithm 7.1: $RFSRG(G_{RC}, w, \mu(w))$

Input: A regularly controlled grammar $G_{RC}(G, r)$ and a faulty string w.

Output: Degree of membership of the string w.

Method

begin

1. Convert context-free grammar G into Chomsky normal form G' using the conventional procedure, and $G_{RC'}(G', r')$.
2. Regularly controlled_**c**ontrol_derivation $(G_{RC}, G_{RC'})$.
3. Error_productions $(G_{RC'}, P')$.
4. Generate different derivation for the faulty string w using P'.
5. Compute the degree of membership μ_E using the fuzzy set E [36] for each derivation of the string w.

$$\mu_E(w) = \frac{|w| - |E_p|}{|w|}$$

where $|w|$ represents the length of the string, w, and $|E_p|$ represents the number of error productions used for the generation of string, w.

6. Compute degree of membership μ_T using the fuzzy set T_f [36] for each derivation of the string w.

$$\mu_{T_f}(w) = 1 - S \text{ shaped membership function}$$

$$\mu_{T_f}(w) = 1 - \begin{cases} 0, & x \le a \\ 2\left(\frac{x-a}{b-a}\right)^2, & a < x \le (a+b)/2 \\ 1-2\left(\frac{x-b}{b-a}\right)^2, & (a+b)/2 < x < b \\ 1, & x \ge b \end{cases}$$

where a and b represent the minimum and a maximum number of productions used to derive the faulty string, w, in all derivations, respectively. x denotes the actual number of productions used in a particular derivation of the faulty string, w.

7. Optimal derivation of string w:

   For each derivation of w

   $\sigma_i(w) = \min(\mu_E(w), \mu_{T_f}(w))$, Here $\sigma_i(w)$ denote the minimum degree of membership for i^{th} derivation of a string w using E and T_f sets.

   End of loop

   $\mu(w) = \max(\sigma_i(w))$

End

If a matrix grammar $G_M(G, r_m)$ is given instead of regularly controlled grammar. During, the conversion of G into Chomsky normal form G', every production $p_i \in G$ is replaced by production $p_i^1, p_i^2, ...p_i^k \mid k \geq 1$ of G'. In step 2 of *RFSRG*, we call procedure Matrix_Control_derivation $(G_M, G_{M'})$. The rest of the algorithm remains the same.

PROCEDURE 3: Matrix_Control_derivation $(G_M, G_{M'})$

Input: Matrix Grammar $G_M(G, r_m)$ and $G_{M'}(G', r_m')$ where G' is Chomsky normal form of G, and initially $r_m' = r_m$.

Output: Control derivation rm' such that $L(G', r_m') = L(G, r_m) - \{\wedge\}$.

begin

$r_m' = r_m$

do $\exists (p_i \in m_i \wedge m_i \in r_m)$

Replace entry of $(p_i \in m_i)$ *with* $(p_i^1, p_i^2 ... \in m_i')$ *in* $m_i' \in r_m'$

od

end

EXAMPLE 7.2: Consider $G_M(G, r_m)$, where $G(\{S, C, D\}, \{c, d\}, S, \{m_0, m_1, m_2, m_3, \ m_4\})$ be the matrix grammar of example 1.2 where

$m_0 = (p_0 = S \rightarrow CD)$,

$m_1 = (p_1 = C \rightarrow cC, p_2 = D \rightarrow cD)$,

$m_2 = (p_3 = C \rightarrow dC, p_4 = D \rightarrow dD)$,

$$m_3 = (p_5 = C \to c, p_6 = D \to c),$$

$$m_4 = (p_7 = C \to d, p_8 = D \to d),$$

The control set is $r_m = m_0(m_1, m_2)^*(m_3, m_4)$.

Equivalent CNF for the grammar G_M :

$$m_0^{'} = (p_0 = S \to CD),$$

$$m_1^{'} = (p_1^1 = C \to A_1C, p_1^2 = A_1 \to c, p_2^1 = D \to A_2D, p_2^2 = A_2 \to c),$$

$$m_2^{'} = (p_3^1 = C \to A_3C, p_3^2 = A_3 \to d, p_4^1 = D \to A_4D, p_4^2 = A_4 \to d),$$

$$m_3^{'} = (p_5 = C \to c, p_6 = D \to c),$$

$$m_4^{'} = (p_7 = C \to d, p_8 = D \to d)$$

The control set $r_m^{'}$ is $m_0^{'}(m_1^{'}, m_2^{'})^*(m_3^{'}, m_4^{'})$.

7.4 NUMERICAL EXAMPLES

In this section, we apply *RFSRG* algorithm to regularly controlled grammar and matrix grammar.

EXAMPLE 7.3: Consider the regularly controlled grammar of example 1.1 $G_{RC}(G, r)$, where $G(\{S, C, D\}, \{c, d\}, S, P)$ and the production P is of the following forms:

$$pr_0 = S \to CD \qquad pr_1 = C \to cC$$

$$pr_2 = D \to cD \qquad pr_3 = C \to dC$$

$$pr_4 = D \to dD \qquad pr_5 = C \to c$$

$$pr_6 = D \to c \qquad pr_7 = C \to d$$

$$pr_8 = D \to d$$

$$r = pr_0(pr_1pr_2, pr_3pr_4)^*(pr_5pr_6, pr_7pr_8)$$

Equivalent Chomsky normal form (CNF) for the grammar G_{RC} :

$$pr_0 = S \to CD \qquad pr_1^1 = C \to A_1C$$

$$pr_1^2 = A_1 \to c \qquad pr_2^1 = D \to A_2D$$

$$pr_2^2 = A_2 \to c \qquad pr_3^1 = C \to A_3C$$

$$pr_3^2 = A_3 \to d \qquad pr_4^1 = D \to A_4D$$

$$pr_4^2 = A_4 \to d \qquad pr_5 = C \to c$$

$pr_6 = D \to c$ $pr_7 = C \to d$

$pr_8 = D \to d$

r is converted into $r' = pr_0(pr_1^1 pr_1^2 pr_2^1 pr_2^2, pr_3^1 pr_3^2 pr_4^1 pr_4^2)^*(pr_5 pr_6, pr_7 pr_8)$

Using fuzzy replace operator:

$pr_9 = A_1 \to \_$ $pr_{10} = A_2 \to \_$

$pr_{11} = A_3 \to \_$ $pr_{12} = A_4 \to \_$

$pr_{13} = C \to \_$ $pr_{14} = D \to \_$

$pr_{15} = C \to \_$ $pr_{16} = D \to \_$

Using fuzzy add operator:

$pr_{17} = A_1 \to \_c \mid c\_$ $pr_{18} = A_2 \to \_c \mid c\_$

$pr_{19} = A_3 \to \_d \mid d\_$ $pr_{20} = A_4 \to \_d \mid d\_$

$pr_{21} = C \to \_c \mid c\_$ $pr_{22} = D \to \_c \mid c\_$

$pr_{23} = C \to \_d \mid d\_$ $pr_{24} = D \to \_d \mid d\_$

Using fuzzy remove operator:

$pr_{25} = A_1 \to \wedge$ $pr_{26} = A_2 \to \wedge$

$pr_{27} = A_3 \to \wedge$ $pr_{28} = A_4 \to \wedge$

$pr_{29} = C \to \wedge$ $pr_{30} = D \to \wedge$

$pr_{31} = C \to \wedge$ $pr_{32} = D \to \wedge$

Consider the string $w = ddddcd \notin L$. Table 7.1 depicts some ways for deriving the string *w*.

Table 7.1: Various derivation for the string $w = ddddcd$

Derivation 1	$S \to_{p0} CD \to_{p_3^1} A_3CD \to_{p_3^2} dCD \to_{p_4^1} dCA_4D \to_{p_4^2} dCdD \to_{p_3^1} dA_3CdD$ $\to_{p19} dddCdD \to_{p4'} dddCdA_4D \to_{p20} dddCdcdD \to_{p29} ddddcdD \to_{p30} ddddcd$
Derivation 2	$S \to_{p0} CD \to_{p_3^1} A_3CD \to_{p_3^2} dCD \to_{p_4^1} dCA_4D \to_{p_4^2} dCdD \to_{p_1^1} dA_1CdD$ $\to_{p9} ddCdD \to_{p_2^1} ddCdA_2D \to_{p_2^2} ddCdcD \to_{p7} ddddcD \to_{p8} ddddcd$
Derivation 3	$S \to_{p0} CD \to_{p_3^1} A_3CD \to_{p19} ddCD \to_{p_4^1} ddCA_4D \to_{p20} ddCdcD$ $\to_7 ddddcD \to_8 ddddcd$
Derivation 4	$S \to_{p0} CD \to_{p_3^1} A_3CD \to_{p_3^2} dCD \to_{p_4^1} dCA_4D \to_{p_4^2} dCdD \to_{p_3^1} dA_3CdD$ $\to_{p_3^2} ddCdD \to_{p_4^1} ddCdA_4D \to_{p12} ddCdcD \to_{p7} ddddcD \to_{p8} ddddcD$

Table 7.1 (Contd.)	
Derivation 5	$S \to_{p0} CD \to_{p_1^1} A_1CD \to_{p9} dCD \to_{p_2^1} dCA_2D \to_{p10} dCdD \to_{p_1^1} dA_1CdD$ $\to_{p9} ddCdD \to_{p_2^1} ddCdA_2D \to_{p_2^2} ddCdcD \to_{p7} ddddcD \to_{p8} ddddcd$
Derivation 6	$S \to_{p0} CD \to_{p_1^1} A_1CD \to_{p9} dCD \to_{p_2^1} dCA_2D \to_{p10} dCdD \to_{p_3^1} dA_3CdD$ $\to_{p19} dddCdD \to_{p_4^1} dddCdA_4D \to_{p20} dddCdcdD \to_{p29} ddddcdD \to_{p30} ddddcd$
Derivation 7	$S \to_{p0} CD \to_{p_1^1} A_1CD \to_{p9} dCD \to_{p_2^1} dCA_2D \to_{p10} dCdD \to_{p_3^1} dA_3CdD$ $\to_{p_3^2} ddCdD \to_{p_4^1} ddCdA_4D \to_{p12} ddCdcD \to_{p7} ddddcD \to_{p8} ddddcD$
Derivation 8	$S \to_{p0} CD \to_{p_3^1} A_3CD \to_{p_3^2} dCD \to_{p_4^1} dCA_4D \to_{p_4^2} dCdD \to_{p_3^1} dA_3CdD$ $\to_{p_3^2} ddCdD \to_{p_4^1} ddCdA_4D \to_{p12} ddCdcD \to_{p13} ddddcD \to_{p14} ddddcd$
Derivation 9	$S \to_{p0} CD \to_{p_3^1} A_3CD \to_{p19} ddCD \to_{p_4^1} ddCA_4D \to_{p20} ddCdcD$ $\to_{13} ddddcD \to_{14} ddddcd$

Table 7.2 represents the confidence level for the string w using $E-$set.

Table 7.2: Confidence level for the string w using $E-$set

Derivation No.	1	2	3	4	5	6	7	8	9
Grade of Membership	0.3333	0.8333	0.6667	0.8333	0.5	0	0.5	0.5	0.3333

Table 7.3 represents the confidence level for the string w using T_f-set.

Table 7.3: Confidence level for the string w using T_f-set

Derivation No.	1	2	3	4	5	6	7	8	9
Grade of Membership	1-1 = 0	1-1 =0	1-0 =1	1-1 =0	1-1 = 0	1-1 = 0	1-1 = 0	1-1 = 0	1-0 =1

Table 7.4: Final interpretations with fuzzy confidence for the string $w = ddddcd$

Derivation No.	1	2	3	4	5	6	7	8	9
$\mu(w \in E)$	0.3333	0.8333	0.6667	0.8333	0.5	0	0.5	0.5	0.3333
$\mu(w \in T_f)$	0	0	1	0	0	0	0	0	1
Min	0	0	0.6667	0	0	0	0	0	0.3333

$$\mu(w) = \max(0, 0, 0.6667, 0, 0, 0, 0, 0, 0.3333) = 0.6667$$

By choosing a certain level of confidence λ_c, we say the string is accepted if $\mu(w) \geq \lambda_c$.

EXAMPLE 7.4: Consider $G_M(G, r_m)$, where $G(\{S, C, D\}, \{c, d\}, S, \{m_0, m_1, m_2, m_3, \ m_4\})$ be the matrix grammar of example 1.2 where

$m_0 = (p_0 = S \to CD)$,

$m_1 = (p_1 = C \to cC, p_2 = D \to cD)$,

$m_2 = (p_3 = C \to dC, p_4 = D \to dD)$,

$m_3 = (p_5 = C \to c, p_6 = D \to c)$,

$m_4 = (p_7 = C \to d, p_8 = D \to d)$,

The control set is $r_m = m_0(m_1, m_2)^*(m_3, m_4)$.

Equivalent CNF for the grammar G_M :

$m_0^{'} = (p_0 = S \to CD)$,

$m_1^{'} = (p_1^1 = C \to A_1C, p_1^2 = A_1 \to c, p_2^1 = D \to A_2D, p_2^2 = A_2 \to c)$,

$m_2^{'} = (p_3^1 = C \to A_3C, p_3^2 = A_3 \to d, p_4^1 = D \to A_4D, p_4^2 = A_4 \to d)$,

$m_3^{'} = (p_5 = C \to c, p_6 = D \to c)$,

$m_4^{'} = (p_7 = C \to d, p_8 = D \to d)$

The control set $r_m^{'}$ is $m_0^{'}(m_1^{'}, m_2^{'})^*(m_3^{'}, m_4^{'})$.

Using fuzzy replace operator:

$pr_9 = A_1 \to \_$ $\quad$ $pr_{10} = A_2 \to \_$

$pr_{11} = A_3 \to \_$ $\quad$ $pr_{12} = A_4 \to \_$

$pr_{13} = C \to \_$ $\quad$ $pr_{14} = D \to \_$

$pr_{15} = C \to \_$ $\quad$ $pr_{16} = D \to \_$

Using fuzzy add operator:

$pr_{17} = A_1 \to \_c \mid c\_$ $\quad$ $pr_{18} = A_2 \to \_c \mid c\_$

$pr_{19} = A_3 \to \_d \mid d\_$ $\quad$ $pr_{20} = A_4 \to \_d \mid d\_$

$pr_{21} = C \to \_c \mid c\_$ $\quad$ $pr_{22} = D \to \_c \mid c$

$pr_{23} = C \to \_d \mid d\_$ $\quad$ $pr_{24} = D \to \_d \mid d\_$

Using fuzzy remove operator:

$pr_{25} = A_1 \to \wedge$ $pr_{26} = A_2 \to \wedge$

$pr_{27} = A_3 \to \wedge$ $pr_{28} = A_4 \to \wedge$

$pr_{29} = C \to \wedge$ $pr_{30} = D \to \wedge$

$pr_{31} = C \to \wedge$ $pr_{32} = D \to \wedge$

On considering string, $w = cdcdcccd \notin L$. Table 7.5 depicts few ways for deriving the string $w = cdcdcccd$

Table 7.5: Various derivation for the string $w = cdcdcccd$

Derivation 1	$S \overset{m_0}{\to}_{p_0} CD \overset{m_1}{\to}_{p_1^1} A_1CD \overset{m_1}{\to}_{p_1^2} cCD \overset{m_1}{\to}_{p_2^1} cCA_2D \overset{m_1}{\to}_{p_2^2} cCcD \overset{m_2}{\to}_{p_3^1} cA_3CcD \overset{m_2}{\to}_{p_3^2} cdCcD$ $\overset{m_2}{\to}_{p_4^1} cdCcA_4D \overset{m_2}{\to}_{p14} cdCccD \overset{m_1}{\to}_{p_1^1} cdA_1CccD \overset{m_1}{\to}_{p_1^2} cdcCccD \overset{m_1}{\to}_{p_2^1} cdcCccA_2D \overset{m_1}{\to}_{p_2^2}$ $cdcCcccD \overset{m_4}{\to}_{p7} cdcdcccD \overset{m_4}{\to}_{p8} cdcdcccd$
Derivation 2	$S \overset{m_0}{\to}_{p_0} CD \overset{m_1}{\to}_{p_1^1} A_1CD \overset{m_1}{\to}_{p_1^2} cCD \overset{m_1}{\to}_{p_2^1} cCA_2D \overset{m_1}{\to}_{p_2^2} cCcD \overset{m_1}{\to}_{p_1^1} cA_1CcD \overset{m_1}{\to}_{p9} ccCcD$ $\overset{m_1}{\to}_{p_2^1} cdCcA_2D \overset{m_1}{\to}_{p_2^2} cdCccD \overset{m_1}{\to}_{p_1^1} cdA_1CccD \overset{m_1}{\to}_{p_1^2} cdcCccD \overset{m_1}{\to}_{p_2^1} cdcCccA_2D$ $\overset{m_1}{\to}_{p_2^2} cdcCcccD \overset{m_4}{\to}_{p7} cdcdcccD \overset{m_4}{\to}_{p8} cdcdcccd$
Derivation 3	$S \overset{m_0}{\to}_{p_0} CD \overset{m_1}{\to}_{p_1^1} A_1CD \overset{m_1}{\to}_{p_1^2} cCD \overset{m_1}{\to}_{p_2^1} cCA_2D \overset{m_1}{\to}_{p_2^2} cCcD \overset{m_1}{\to}_{p_1^1} cA_1CcD \overset{m_1}{\to}_{p17}$ $cdcCcD \overset{m_1}{\to}_{p_2^1} cdcCcA_2D \overset{m_1}{\to}_{p18} cdcCcccD \overset{m_4}{\to}_{p7} cdcdcccD \overset{m_4}{\to}_{p8} cdcdcccd$
Derivation 4	$S \overset{m_0}{\to}_{p_0} CD \overset{m_1}{\to}_{p_1^1} A_1CD \overset{m_1}{\to}_{p_1^2} cCD \overset{m_1}{\to}_{p_2^1} cCA_2D \overset{m_1}{\to}_{p_2^2} cCcD \overset{m_1}{\to}_{p_1^1} cA_1CcD \overset{m_1}{\to}_{p17} cdcCcD$ $\overset{m_1}{\to}_{p_2^1} cdcCcA_2D \overset{m_1}{\to}_{p18} cdcCcccD \overset{m_2}{\to}_{p_3^1} cdcA_3CcccD \overset{m_2}{\to}_{p_3^2} cdcdCcccD$ $\overset{m_2}{\to}_{p_4^1} cdcdCcccA_4D \overset{m_2}{\to}_{p_4^2} cdcdCcccdD \overset{m_4}{\to}_{p31} cdcdcccdD \overset{m_4}{\to}_{p32} cdcdcccd$
Derivation 5	$S \overset{m_0}{\to}_{p_0} CD \overset{m_2}{\to}_{p_3^1} A_3CD \overset{m_2}{\to}_{p11} cCD \overset{m_2}{\to}_{p_4^1} cCA_4D \overset{m_2}{\to}_{p_{12}} cCcD \overset{m_2}{\to}_{p_3^1} cA_3CcD \overset{m_2}{\to}_{p_3^2} cdCcD$ $\overset{m_2}{\to}_{p_4^1} cdCcA_4D \overset{m_2}{\to}_{p12} cdCccD \overset{m_1}{\to}_{p_1^1} cdA_1CccD \overset{m_1}{\to}_{p_1^2} cdcCccD \overset{m_1}{\to}_{p_2^1} cdcCccA_2D$ $\overset{m_1}{\to}_{p_2^2} cdcCcccD \overset{m_4}{\to}_{p7} cdcdcccD \overset{m_4}{\to}_{p8} cdcdcccd$
Derivation 6	$S \overset{m_0}{\to}_{p_0} CD \overset{m_2}{\to}_{p_3^1} A_3CD \overset{m_2}{\to}_{p11} cCD \overset{m_2}{\to}_{p_4^1} cCA_4D \overset{m_2}{\to}_{p_{12}} cCcD \overset{m_1}{\to}_{p_1^1} cA_1CcD \overset{m_1}{\to}_{p9} ccCcD$ $\overset{m_1}{\to}_{p_2^1} cdCcA_2D \overset{m_1}{\to}_{p_2^2} cdCccD \overset{m_1}{\to}_{p_1^1} cdA_1CccD \overset{m_1}{\to}_{p_1^2} cdcCccD \overset{m_1}{\to}_{p_2^1} cdcCccA_2D$ $\overset{m_1}{\to}_{p_2^2} cdcCcccD \overset{m_4}{\to}_{p7} cdcdcccD \overset{m_4}{\to}_{p8} cdcdcccd$
Derivation 7	$S \overset{m_0}{\to}_{p_0} CD \overset{m_2}{\to}_{p_3^1} A_3CD \overset{m_2}{\to}_{p11} cCD \overset{m_2}{\to}_{p_4^1} cCA_4D \overset{m_2}{\to}_{p_{12}} cCcD \overset{m_1}{\to}_{p_1^1} cA_1CcD \overset{m_1}{\to}_{p17} cdcCcD$ $\overset{m_1}{\to}_{p_2^1} cdcCcA_2D \overset{m_1}{\to}_{p18} cdcCcccD \overset{m_4}{\to}_{p7} cdcdcccD \overset{m_4}{\to}_{p8} cdcdcccd$

Table 7.5 (Contd.)	
Derivation 8	$S \xrightarrow{m_0}_{p_0} CD \xrightarrow{m_2}_{p_3^1} A_3CD \xrightarrow{m_2}_{p11} cCD \xrightarrow{m_2}_{p_4^1} cCA_4D \xrightarrow{m_2}_{p_{12}} cCcD \xrightarrow{m_1}_{p_1^1} cA_1CcD \xrightarrow{m_1}_{p17} cdcCcD$ $\xrightarrow{m_1}_{p_2^1} cdcCcA_2D \xrightarrow{m_1}_{p18} cdcCcccD \xrightarrow{m_2}_{p_3^1} cdcA_3CcccD \xrightarrow{m_2}_{p_3^2} cdcdCcccD \xrightarrow{m_2}_{p_4^1} cdcdCcccA_4D$ $\xrightarrow{m_2}_{p_4^2} cdcdCcccdD \xrightarrow{m_4}_{p31} cdcdcccdD \xrightarrow{m_4}_{p32} cdcdcccd$
Derivation 9	$S \xrightarrow{m_0}_{p_0} CD \xrightarrow{m_1}_{p_1^1} A_1CD \xrightarrow{m_1}_{p_1^2} cCD \xrightarrow{m_1}_{p_2^1} cCA_2D \xrightarrow{m_1}_{p_2^2} cCcD \xrightarrow{m_2}_{p_3^1} cA_3CcD \xrightarrow{m_2}_{p_3^2} cdCcD$ $\xrightarrow{m_2}_{p_4^1} cdCcA_4D \xrightarrow{m_2}_{p12} cdCcccD \xrightarrow{m_2}_{p_3^1} cdA_3CcccD \xrightarrow{m_2}_{p11} cdcCcccD \xrightarrow{m_2}_{p_4^1} cdcCccA_4D$ $\xrightarrow{m_2}_{p12} cdcCccccD \xrightarrow{m_4}_{p7} cdcdcccD \xrightarrow{m_4}_{p8} cdcdcccd$
Derivation 10	$S \xrightarrow{m_0}_{p_0} CD \xrightarrow{m_1}_{p_1^1} A_1CD \xrightarrow{m_1}_{p_1^2} cCD \xrightarrow{m_1}_{p_2^1} cCA_2D \xrightarrow{m_1}_{p_2^2} cCcD \xrightarrow{m_1}_{p_1^1} cA_1CcD \xrightarrow{m_1}_{p9} ccCcD$ $\xrightarrow{m_1}_{p_2^1} cdCcA_2D \xrightarrow{m_1}_{p_2^2} cdCcccD \xrightarrow{m_2}_{p_3^1} cdA_3CcccD \xrightarrow{m_2}_{p11} cdcCcccD \xrightarrow{m_2}_{p_4^1} cdcCccA_4D$ $\xrightarrow{m_2}_{p12} cdcCccccD \xrightarrow{m_4}_{p7} cdcdcccD \xrightarrow{m_4}_{p8} cdcdcccd$

Table 7.6 represents the confidence level for the string w using E-set.

Table 7.6: Confidence level for the string w using E-set

Derivation No.	1	2	3	4	5	6	7	8	9	10
Grade of Membership	0.875	0.875	0.75	0.5	0.625	0.625	0.5	0.25	0.625	0.625

Table 7.7 represents the confidence level for the string w using T_f-set.

Table 7.7: Confidence level for the string w using T_f-set

Derivation No.	1	2	3	4	5	6	7	8	9	10
Grade of Membership	1-1 = 0	1-1 =0	1-0 =1	1-1 =0	1-1 = 0	1-1 = 0	1-0 = 1	1-1 = 0	1-1 = 0	1-1 =0

Table 7.8: Final interpretations with fuzzy confidence for the string $w = cdcdcccd$

Derivation No.	1	2	3	4	5	6	7	8	9	10
$\mu(w \in E)$	0.875	0.875	0.75	0.5	0.625	0.625	0.5	0.25	0.625	0.625
$\mu(w \in T_f)$	1-1 = 0	1-1 =0	1-0 =1	1-1 =0	1-1 = 0	1-1 = 0	1-0 = 1	1-1 = 0	1-1 = 0	1-1 =0
Min	0	0	0.75	0	0	0	0.5	0	0	0

$$\mu(w) = \max(0, 0, 0.75, 0, 0, 0, 0.5, 0, 0, 0) = 0.75$$

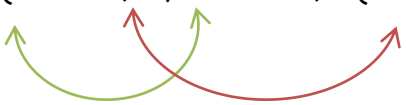
By choosing a certain level of confidence λ_c, we say the string is accepted if $\mu(w) \geq \lambda_c$.

7.5 REAL-LIFE APPLICATION OF *RFSRG* ALGORITHM IN THE HINDI LANGUAGE

Regulated languages are the most appropriate formal method for representing human languages.

In linguistic topology, the Hindi language contains crossing dependency, and its structure is similar to the German and Dutch languages. Consider the following sentence of the Hindi Language spoken in India.

S_1: मोहन ने वो सेब ख़ाया जो खराब हो गया|



Meaning: Mohan that apple ate which rotten was.

English: Mohan ate that apple which was rotten.

S_2: राम ने उस पेन से लेख लिखा जो उसे उपहार मे मिला था|

Meaning: Ram has that pen with essay written which him gifted was

English: Ram has written the essay with that pen which was gifted to him.

In the above sentences, sequential noun phrases मोहन (Mohan), सेब (apple), राम (Ram) , पेन (pen) and the verb phrases ख़ाया (ate), खराब हो गया (rotten), लिखा (written) , उपहार मे मिला था (gifted) both form two separate series of constituents. Let m' be the morphism which maps to मोहन and ख़ाया to c, सेब and खराब हो गया to d and all other words of the sentence to the empty string. Therefore, $S_1 : cdcd$. Therefore, the sentence S_1 is of the form ww, which is a non-context-free language.

Consider m" be the morphism, which maps राम, पेन, लिखा, उपहार मे मिला था to a, b, c and d respectively and the dependency relation between राम (Ram) and लिखा (written) is denoted by m and पेन (pen) and उपहार मे मिला था (gifted) by n and other words of the sentence by empty string. Therefore, the sentence S_2 is of the form $a^m b^n c^m d^n$, which is a non-context-free language. The crossing dependency of the Hindi language can be represented by a regulated grammar.

Now consider the sentence S_1 and If the users unintentionally writes the S_1 sentence as

$S_1^{'}$: मोहन ने वो सेब ख़ाया जो सेब खराब हो गया|

In this sentence, user makes an insertion error of one noun phrase सेब (apple). Now the sentence $S_1^{'}$ maps to $cdcdd$ instead of $cdcd$. In formal grammar, this string is completely rejected. However, in the case of fuzzy regulated grammar, we cannot completely reject the string w. By considering the grammar of example 7.1, Erroneous sentence acceptance $S_1^{'}$ is shown using proposed algorithm ***RFSRG.*** Table 7.9 depicts few ways for deriving the string $w = cdcdd$

Table 7.9: Various derivation for the string $w = cdcdd$

Derivation 1	**Derivation 2**	**Derivation 3**	**Derivation 4**
$pr_0 = S \to CD$	$pr_0 = S \to CD$	$pr_0 = S \to CD$	$pr_0 = S \to CD$
$pr_1^1 = S \to A_1CD$	$pr_1^1 = S \to A_1CD$	$pr_3^1 = S \to A_3CD$	$pr_3^1 = S \to A_3CD$
$pr_1^2 = S \to cCD$	$pr_1^2 = S \to cCD$	$pr_{11} = S \to cCD$	$pr_{19} = S \to cdCD$
$pr_2^1 = S \to cCA_2D$	$pr_2^1 = S \to cCA_2D$	$pr_4^1 = S \to cCA_4D$	$pr_4^1 = S \to cdCA_4D$
$pr_{18} = S \to cCcdD$	$pr_2^2 = S \to cCcD$	$pr_4^2 = S \to cCdD$	$pr_4^2 = S \to cdCdD$
$pr_7 = S \to cdcdD$	$pr_7 = S \to cdcD$	$pr_{23} = S \to cdcdD$	$pr_5 = S \to cdcdD$
$pr_8 = S \to cdcdd$	$pr_{24} = S \to cdcdd$	$pr_8 = S \to cdcdd$	$pr_{14} = S \to cdcdd$

Table 7.10 represents the confidence level for the string w using $E-$set.

Table 7.10: Confidence level for the string w using $E-$set

Derivation No.	Derivation 1	Derivation 2	Derivation 3	Derivation 4
Grade of Membership	0.75	0.75	0.5	0.5

Table 7.11 represents the confidence level for the string w using T_f – set.

Table 7.11: Confidence level for the string w using T_f – set

Derivation No.	Derivation 1	Derivation 2	Derivation 3	Derivation 4
Grade of Membership	1-0=1	1-0=1	1-0=1	1-0=1

Table 7.12: Final interpretations with fuzzy confidence for the string $w = cdcdd$

Derivation No.	Derivation 1	Derivation 2	Derivation 3	Derivation 4
$\mu(w \in E)$	0.75	0.75	0.5	0.5
$\mu(w \in T_f)$	1	1	1	1
Min	0.75	0.75	0.5	0.5

$$\mu(w) = \max(0.75; 0.75; 0.5; 0.5) = 0.75$$

By choosing a certain level of confidence λ_c, we say string is accepted if $\mu(w) \geq \lambda_c$.

7.6 ANALYSIS OF TIME COMPLEXITY

In this section, we analyze the time complexity of the proposed algorithm ***RFSRG.***

Theorem 1: Given a regulated grammar G_R and string, w. The time complexity of the *RFSRG* algorithm is equal to $O(|G_R{}^2|.|w|)$, where $|G_R|$ represents the size of productions and $|w|$ represents the length of the string, respectively.

Proof. In Step1 of *RFSRG* algorithm**,** regulated grammar G_R is converted into Chomsky Normal form $G_{R'}$ using DEL, TERM, UNIT and BIN operations [91]. The size of $G_{R'}$ i.e. $|G_{R'}|$ depends on the order in which DEL, BIN, UNIT, and TERM are carried out. TERM always give a linear increase in the size of the grammar. If we carry out the transformation in BIN $\rightarrow$ DEL $\rightarrow$ UNIT, then $|G_{R'}| = O(|G_R{}^2|)$. In step 2, the control derivation G_R is converted into $G_{R'}$. Each production $p_i \in G_R$ may be converted into a number of productions say $pr_i^1, pr_i^2, ..., pr_i^k \mid k \geq 1$ in $G_{R'}$ and each substitution of p_i in control derivation r gives the corresponding control derivation r'. Each substitution can

be carried out in $O(|1|)$. Therefore, the control derivation r can be converted into r' in $O(|G_R|)$.

In step 3, we add additional productions using replace, add and remove. Thus, the number of productions is of the size $O(|G_R^2|)$. For each production, four faulty productions (one each by replace and remove operation and two productions by add operations) are generated. The addition of each faulty productions can be carried out in $O(|1|)$, thereby giving the overall complexity of the third step is $O(|G_R^2|)$.

In each derivation of a string w, the number of productions of the form $A \to BC \mid A, B, C \in N$ can be $|w|-1$ and the number of productions of the form $A \to a \mid A \in N \wedge a \in \sum$ can be $|w|$. Therefore total $2|w|-1$ productions are used in one derivation of a string. Considering at each step of the derivation, we can choose a production from $O(|G_R^2|)$ productions. Hence the overall complexity is $O(|G_R^2|.|w|)$. Step 5 can be carried out in $O(|1|)$. In step 6, for finding a, b and x values, complexity is $O(|G_R^2|)$ and further computation of $\mu_{T_f}(w)$ can be carried out in $O(|1|)$. So, the overall complexity of Step 6 is $O(|G_R^2|)$. Step 7 can be carried out in $O(|1|)$. Thus the overall complexity of the algorithm is $O(|G_R^2|.|w|)$. □

The proposed algorithm works on all rule-based regulated grammars with little modifications in applications of rules.

7.7 CONCLUSION

In this chapter, an algorithm for deriving a string, $w \notin L$, with a certain level of confidence is described. Regulated grammar is converted to Chomsky normal form to avoid multiple errors. Furthermore, the algorithm is applied to regularly controlled and matrix grammar. The time complexity of the proposed algorithm is $O(|G_R^2|.|w|)$. By the way of numerical examples for a string, $w \notin L$, the determination of a confidence level for the string, w, is also demonstrated. Further, the real life application of a proposed algorithm for the recognition of faulty strings for the natural language, Hindi, is exhibited.

Chapter-8

CONCLUSION AND FUTURE SCOPE

8.1 SUMMARY OF THE MAIN CONTRIBUTION

In this dissertation, we proposed a variant of regulated grammar and regulated automaton and applied their concept in RNA and DNA biomolecular structures. The summary of the main contribution is presented as follows:

1. Survey of various existing regulated grammar, regulated automata and regulated transducer has been carried out. Further, the concept of fuzzy context-free grammar, fuzzy pushdown automata and grammar formalism for DNA and RNA sequences has also been explored.
2. Novel concept of Fuzzy State Grammar and Fuzzy Deep Pushdown Automata has been proposed.
3. Design of deterministic deep pushdown automaton (DDPDA), deterministic deep pushdown transducer (DDPDT) and their parallel versions named as parallel deterministic deep pushdown automaton (PDDPDA) and parallel deterministic deep pushdown transducer (PDDPDT). The proposed models obey the properties of both strict determinism and determinism with respect to depth.
4. Application of state grammar and deep pushdown automata in biological sequences of Nucleic Acids. We have represented three commonly found structures in DNA and RNA using state grammar namely: Tandem Repeat, Inverted Repeat, and Pseudoknot. These biological sequences are depicted using formal model deep pushdown automata. Furthermore, we have extended our work for various biomolecular structures of DNA and RNA using state grammar. Various structures represented are attenuator, extended pseudoknot, kissing hairpin, simple h-type, recursive pseudoknot, and three-knot.
5. Design of an algorithm for error tolerance in recognition of faulty strings in a regulated grammar using fuzzy sets.

The limitation of the proposed work is that, in Chapter 5, we have designed Extended LL (1) parser for state grammar which is an extension of LL (1) parser. The proposed parser handles only left-factoring, by the inclusion of states in the parsing table. Our parser fails

if the grammar contains left recursion. So, there is a scope in future to design a regulated parser which parses the grammar which contains both left-factoring and left-recursion.

8.2 FUTURE SCOPE

Following are a number of promising and stimulating directions for future research building on the proposed work and gleaning of knowledge presented in this dissertation.

8.2.1 REGULATED AUTOMATA

Significant research has been done on regulated grammar, but much less research has been done on its automaton counterpart. A few regulated grammars such as matrix grammar, state grammar, and programmed grammar have their corresponding automata, but other types of grammars still lack their corresponding automata.

8.2.2 PARSING

In this dissertation, we have designed Extended LL (1) parser for structures such as Tandem Repeat, Inverted Repeat and Pseudoknot. In the future, researchers can work on designing of efficient parsing algorithm for the other various types of structures such as attenuator, extended pseudoknot, kissing hairpin, simple h-type, recursive pseudoknot, and three-knot.

8.2.3 DECIDABILITY

Is the family of languages defined by regulated automata such as Deep Pushdown Automata and Controlled Finite Automata closed under various operations such as intersection, concatenation, union, shuffle, Kleene star, reversal?

8.2.4 PARSER FOR THE PROPOSED VARIANT

There is a scope of designing of efficient parsing algorithm for the proposed variant of regulated grammar named as Fuzzy state grammar.

8.2.5 FUZZINESS IN CONTEXT-BASED AND PARALLEL REGULATED GRAMMAR

We have introduced the concept of fuzziness in state grammar which is a rule-based regulated grammar. This will lead to another interesting problem for introducing the concept of fuzziness in context-based and parallel regulated grammar.

8.2.6 BIPOLAR FUZZY SETS IN RFSRG

In this dissertation, we have designed an algorithm for error tolerance in recognition of faulty strings in a regulated grammar using fuzzy sets. We can extend this algorithm by using bipolar fuzzy sets, which is the extension of fuzzy sets, in which membership degree lies between [-1 to 1] rather than [0 to 1].

8.2.7 DESIGNING OF NEW TYPE OF REGULATED GRAMMAR AND REGULATED AUTOMATA

There is a scope in designing a new type of regulated grammar and regulated automata by combining the restrictions of both rule-based and context-based. Further, to explore the language hierarchy of these grammars.

8.2.8 APPLICATIONS

In future, there is a scope of applying the concept of regulated and regulated automata in linguistics, complier design and molecular genetics.

REFERENCES

1. J. Dassow, Grammars with regulated rewriting, In C. Martin-Vide, V. Mitrana and G. Paun, Formal Languages and Applications, Studies in Fuzziness and Softcomputing, Springer Berlin Heidelberg, 148, 249-273, 2004.
2. J. Dassow and G. Paun, Regulated rewriting in Formal Language Theory (1st Ed), Springer-Verlag, Berlin, Heidelberg, 1989.
3. A. Meduna and P. Zemek, Regulated Grammars and Automata (1st Ed), Springer-Verlag, New York, 2014.
4. I. Mikhailova, B. Novikov and G. Zholtkevych, Protoautomata as Models of Systems with Data Accumulation, In: V. Ermolayev et al.(eds.) Proceedings of 9th International Conference on ICT in Education, Research, and Industrial Applications, Kherson State University, Kherson, Ukraine, 582-589, 19-22 June 2013.
5. A. Meduna, Formal Languages and Computation: Models and Their Applications (1st Ed), CRC Press, Boca Raton, Florida, US, 2014.
6. S. Ginsburg and E. H. Spanier, Control sets on grammars, Mathematical Systems Theory, 2(2), 159-177, 1968.
7. A. B. Cremers and O. Mayer, On matrix languages, Information and Control, 23(1), 86-96, 1973.
8. T. Kasai, An hierarchy between context-free and context-sensitive languages, Journal of Computer and System Sciences, 4(5), 492-508, 1970.
9. L. Kuppusamy, S. N. Krishna, R. Raghavan and C. Martin-Vide, Internal Contextual Grammars for Mildly Context-Sensitive Languages, Research on Language and Computation, 5(2), 181-197, 2007.
10. A. Meduna, Deep pushdown automata, Acta Informatica, 42(8-9), 541-552, 2006
11. P. Solar, Parallel deep pushdown automata, In Proceedings of the 18th Conference STUDENT EEICT, Brno University of Technology, Czech Republic, 410-414, 26 April 2012.
12. J. Kucera, A. Meduna and O. Soukup, Absolutely Unlimited Deep Pushdown Automata, In Proceedings of the 10th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2015), Telc, Czech Republic, 36-44, 23-25 October 2015.

13. A. Kumar and A. K. Verma, An Improved Algorithm for the Metamorphosis of Semi-Extended Regular Expressions to Deterministic Finite Automata, The Computer Journal, 58(3), 448-456, 2015.
14. A. Kumar and A. K. Verma, A novel algorithm for the conversion of parallel regular expressions to non-deterministic finite automata, Applied Mathematics & Information Sciences, 8(1), 95-105, 2014.
15. P. R. J. Asveld, Fuzzy context-free languages Part 1: Generalized fuzzy context-free grammars, Theoretical Computer Science, 347(1-2), 167–190, 2005.
16. P. R. J. Asveld, Fuzzy context-free languages Part 2: Recognition and parsing algorithms, Theoretical Computer Science, 347(1-2), 191–213, 2005.
17. Y. B. Jun and J. Kavikumar, Bipolar fuzzy finite state machines, Bulletin of the Malaysian Mathematical Sciences Society, 34(1), 181–188, 2011.
18. J. Kavikumar, A. B. Khamis and R. Kandasamy, Fuzzy entire sequence spaces, International Journal of Mathematics and Mathematical Sciences, 1-9, 2007.
19. I. Bucurescu and A. Pascu, Fuzzy pushdown automata, International Journal of Computer Mathematics, 10(2), 109-119, 1981.
20. C. Moraga, An approach to fuzzy context-free languages, In Proceedings of XIV Spanish Congress on Fuzzy Logic Technologies, Langreo-Mires, 73-78, 17-19 Sep 2008.
21. J. E. Hopcroft, R. Motwani and J. D. Ullman, Introduction to Automata Theory, Languages and Computation (3rd Ed), Pearson Education, USA, 2006.
22. S. Kakoty, M. Lal and S. K. Sarma, Fuzzy expert logic to evaluate learner's expertise level in e-learning environment, In Proceedings of IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), Syed Ammal Engineering College, Ramanathapuram, India, 148-152, 23-25 Aug 2012.
23. D. Arora, B. Hazela and V. Saxena, Semantics for UML model transformation and generation of regular grammar, ACM SIGSOFT Software Engineering Notes, 37(3), 1-5, 2012.
24. Y. Li and W. Pedrycz, Fuzzy finite automata and fuzzy regular expressions with membership values in lattice-ordered monoids, Fuzzy Sets and Systems, 156(1), 68-92, 2005.
25. L. Cai, L. R. Malmberg and Y. Wu, Stochastic modeling of RNA pseudoknotted structures: a grammatical approach, Bioinformatics, 19(1), i66-i73, 2003.

26. K. Y. Sung, The Use of Context-Sensitive Grammar for Modeling RNA Pseudoknots, In Proceedings of International Conference on Bioinformatics and Computational Biology (BIOCOMP'06), Las Vegas, Nevada: USA, 338-344, 26-29 June 2006.
27. E. Rivas and S. R. Eddy, The language of RNA: a formal grammar that includes pseudoknots, Bioinformatics, 16(4), 334-340, 2000.
28. D. B. Searls, The computational linguistics of biological sequences, Artificial intelligence and molecular biology, In American Association for Artificial Intelligence Eds, Menlo Park CA, USA, 2, 47-120, 1993.
29. D. B. Searls, String variable grammar: logic grammar formalism for the biological language of DNA, The Journal of Logic Programming, 24 (1-2), 73-102, 1995.
30. Y. Uemura, A. Hasegawa, S. Kobayashi and T. Yokomori, Tree adjoining grammars for RNA structure prediction, Theoretical Computer Science, 210 (2), 277-303, 1999.
31. N. Mizoguchi, Y. Kato and H. Seki, A grammar-based approach to RNA pseudoknotted structure prediction for aligned sequences, In Proceedings of 1st International Conference on Computational Advances in Bio and Medical Sciences (ICCABS), Orlando, FL, 135-140, 3-5 Feb 2011.
32. L. Kuppusamy and A. Mahendran, Modelling DNA and RNA secondary structures using matrix insertion–deletion systems, International Journal of Applied Mathematics and Computer Science, 26(1), 245-258, 2016.
33. P. Solar, Deep Pushdown Transducers and Parallel Deep Pushdown Transducers, In Proceedings of the 19th Conference STUDENT EEICT, Brno University of Technology, Czech Republic, 207-211, 25 April 2013.
34. X. Z. Gao and S. J. Ovaska, Soft computing methods in motor fault diagnosis, Applied soft computing, 1(1), 73-81, 2001.
35. H. Rana and M. Lal, Rough set based system for effective e-learning, In Proceedings of International Conference on Computing for Sustainable Global Development (INDIACom), Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi, 192-196, 5-7 March 2014.
36. M. Schneider, H. Lim and W. Shoaff, The utilization of fuzzy sets in the recognition of imperfect strings, Fuzzy Sets and Systems, 49 (3), 331-337, 1992.
37. M. Inui, W. Shoaff, L. Fausett and M. Schneider, The recognition of imperfect strings generated by fuzzy context-sensitive grammars, Fuzzy sets and Systems, 62(1), 21-29, 1994.

38. R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchinson, Biological sequence analysis: Probabilistic models of proteins and nucleic acids (1st Ed), Cambridge: Cambridge University Press, 1998.
39. J. C. Martin, Introduction to Languages and the theory of computation (2nd ed), McGraw-Hill, New York, 1991.
40. P. Solar and A. Meduna, Deep Pushdown Transducers and State Translation Schemes, In Proceedings of the 20th Conference STUDENT EEICT, Brno University of Technology, Czech Republic, 264-268, 2014.
41. E. T. Lee and L. A. Zadeh, Note on fuzzy languages, Information Sciences, 1(4), 421-434, 1969.
42. E. Navratil, Context-free grammars with regular conditions, Kybernetika, 6(2), 118–126, 1970.
43. A. P. J. Van der Walt, Random context grammars, In Proceedings of IFIP Congress, North Holland, Amestrdam, 66-68, 1971.
44. A. Meduna and P. Zemek, One-sided random context grammars, Acta Informatica, 48(3), 149–163, 2011.
45. A. Meduna and P. Zemek, One-sided random context grammars with a limited number of right random context rules, Theoretical Computer Science, 516(1), 127–132, 2014.
46. A. Meduna, Generalized forbidding grammars, International Journal of Computer Mathematics, 36(1-2), 31-38, 1990.
47. A. Meduna and P. Zemek, Generalized one-sided forbidding grammars, International Journal of Computer Mathematics, 90(2), 172–182, 2013.
48. G. Paun, A variant of random context grammars: Semi-conditional grammars, Theoretical Computer Science, 41, 1–17, 1985.
49. A. Meduna, and A. Gopalaratnam, On semi-conditional grammars with productions having either forbidding or permitting conditions, Acta Cybernetica, 11(4), 307–323, 1994.
50. H. C. M. Kleijn, Selective substitution grammars based on context-free productions, Ph.D. Thesis, Leiden University, Netherlands, 1983.
51. H. C. M. Kleijn, Basic ideas of selective substitution grammars, Lecture Notes in Computer Science Trends, Techniques, and Problems in Theoretical Computer Science, 75–95, 1987.
52. A. Meduna and P. Zemek, One-sided forbidding grammars and selective substitution grammars, International Journal of Computer Mathematics, 89(5), 586–596, 2012.

53. A. Cremers and O. Mayer, On vector languages, Journal of Computer and System Sciences, 8(2), 158–166, 1974.
54. D. J. Rosenkrantz, Programmed Grammars and Classes of Formal Languages, Journal of the ACM, 16(1), 107–131, 1969.
55. D. J. Rosenkrantz, Programmed Grammars-A new device for generating formal languages. In: Proceedings of 8th Annual Symposium on Switching and Automata Theory (SWAT 1967), USA, 14-20, Oct 1967.
56. N. Chomsky, On certain formal properties of grammars, Information and Control, 2(2), 137–167, 1959.
57. N. Chomsky, Formal properties of grammar, Wiley, New York, 1963.
58. G. Paun, A new generative device: valence grammars, Revue Roumaine De Mathematiques Pures Et Appliquees, 25(6), 911–924, 1980.
59. G. Paun, On a class of valence grammars, Stud. cerc. mat, 42, 255–268, 1990.
60. R. Siromoney and K. Krithivasan, Parallel context-free languages, Information and Control, 24(2), 155–162, 1974.
61. A. Salomaa, Parallelism in rewriting systems, Automata, Languages and Programming Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, 14, 523–533, 1974.
62. Controlled Grammar,https://en.wikipedia.org/wiki/Controlled_grammar,(Accessed March 20 2016)
63. J. Dassow, On some extensions of Russian parallel context-free grammars, Acta Cybernetica, 6(4), 355–350, 1984.
64. S. Greibach and J. Hopcroft, Scattered context grammars, Journal of Computer and System Sciences, 3(3), 233–247, 1969.
65. A. Meduna, L. Vrabel and P. Zemek, LL leftmost k-linear scattered context grammars, In Proceedings of International Conference on Numerical Analysis, Halkidiki, GR, 833–836, 2011.
66. Z. Krivika and R. Schonecker, Reducing Push Down Automata, In Proceedings of MEMICS 2006 Second Doctoral Workshop on Mathematical and Engineering Methods in Computer Science. Mikulov: Faculty of Information Technology, Brno University of Technology, Czech Republic, 214-221, 2006.
67. P. Leupold and A. Meduna, Finitely Expandable Deep PDA's. In: Proceedings of Automata, Formal Languages and Algebraic Systems, Hong Kong University of Science and Technology, Hong Kong, 113-123, 2010.

68. A. Meduna and P. Zemek, Controlled finite automata, Acta Informatica, 51(5), 327–337, 2014.
69. H. Xing, Fuzzy pushdown automata, Fuzzy Sets and Systems, 158(13), 1437-1449, 2007.
70. Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjolander, R. C. Underwood and D. Haussler, Stochastic context-free grammars for tRNA modelling, Nucleic Acids Research, 22(23), 5112–5120, 1994.
71. Y. Sakakibara, Pair hidden Markov models on tree structures, Bioinformatics, 19(1), 232–240, 2003.
72. S. Yuki and T. Kasami, RNA pseudoknotted structure prediction using stochastic multiple context-free grammar, IPSJ Transactions on Bioinformatics, 47, 12–21, 2006.
73. M. Brown and C. Wilson, RNA pseudoknot modeling using intersections of stochastic context-free grammars with applications to database search, In Proceedings of the Pacific Symposium on Biocomputing, Big Island, HI, USA, 109–125, 1995.
74. J. W. J. Anderson, P. Tataru, J. Staines, J. Hein and R. Lyngso, Evolving stochastic context-free grammars for RNA secondary structure prediction, 13(1), 1-10, 2012.
75. J. W. J. Anderson, Stochastic Context-free Grammars and RNA Secondary Structure Prediction. Genome Analysis: Current Procedures and Applications, In M. S. Poptsova Eds. Caister Academic Press, England, 339-366, 2014.
76. P. W. K. Rothemund, A DNA and restriction enzyme implementation of Turing machines: In Proceedings of the DIMACS Workshop in DNA Based Computers, 4 April 1995, Princeton University. In: R. J. Lipton, E. B. Baum Eds. American Mathematical Society, Providence, Rhode Island, 75-119, 1996.
77. M. Cavaliere, N. Jonoska, S. Yogev, R. Piran, E. Keinan and N. C. Seeman, Biomolecular implementation of computing devices with unbounded memory. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 3384, 35-49, 2005.
78. T. Krasinski, S. Sakowski and T. Poplawski, Autonomous push-down automaton built on DNA. International Journal of Computing and Informatics, 36(3), 263-276, 2012.
79. P. R. J. Asveld, Controlled Fuzzy parallel rewriting, in New Trends in Formal Languages-Control, Cooperation, and Combinatorics, Springer-Verlag, 1218, 49–70, 1997.
80. P. R. J. Asveld, The non-self-embedding property for generalized fuzzy context-free grammars, Publicationes Mathematicae Debrecen 54 (Suppl 1), 553–573, 1999.
81. S. Lan, A machine accepted fuzzy context-free languages fuzzy pushdown automata, BUSEFAL, 49, 67–72, 1992.

82. H. Wang and D. Qiu, Computing with words via Turing machines: A formal approach, IEEE Transactions on Fuzzy System, 11(6), 742–753, 2003.
83. A. V. Aho and J. D. Ullman, The theory of parsing, translation, and compiling (Vol.2), Prentice-Hall, Inc., New Jersey, 1973.
84. R. Garcia, Prediction of RNA Pseudoknotted Secondary Structure using Stochastic Context-free Grammars (SCFG). CLEI Electronic Journal, 9 (2), 1-12, 2006.
85. Tandem Repeat, http: //en.wikipedia.org/wiki/Tandem_repeat.(Accessed May 20, 2015).
86. Inverted Repeat, http://en.wikipedia.org/wiki/Inverted_repeat. (Accessed June 10, 2015).
87. Pseudoknots, http://en.wikipedia.org/wiki/Pseudoknot. (Accessed June 10, 2015).
88. A. Meduna, Elements of Compiler Design (1st ed), Auerbach Publications, Boston 2007.
89. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, and C. Stein, Introduction to algorithms (1st ed), MIT Press, United States, 1990.
90. L. A. Zadeh, Fuzzy sets. Information and control, 8(3), 338-353, 1965.
91. M. Lange and H. Leib, To CNF or not to CNF? An efficient yet presentable version of the CYK algorithm. Informatica Didactica, 8, 2008-2010, 2009.