

Grammar Checker for Pashto Language

Mustafa Toufiq
15p-7001

Master of Science

A thesis submitted in partial fulfillment of the requirements for the
degree of *Master of Science (Computer Science)* at the
National University of Computer and Emerging Sciences



Department of Computer Science
National University of Computer and Emerging Sciences,
Peshawar, Pakistan

2019

Plagiarism Undertaking

I take full responsibility of the research work conducted during the MS Thesis titled “Grammar Checker for Pashto Language”. I solemnly declare that the research work presented in the thesis is done solely by me with no significant help from any other person; however, small help wherever taken is duly acknowledged. I have also written the complete thesis by myself. Moreover, I have not presented this thesis (or substantially similar research work) or any part of the thesis previously to any other degree awarding institution within Pakistan or abroad.

I understand that the management of Department of Computer Science of National University of Computer and Emerging Sciences has a zero tolerance policy towards plagiarism. Therefore, I, as an author of the above-mentioned thesis, solemnly declare that no portion of my thesis has been plagiarized and any material used in the thesis from other sources is properly referenced. Moreover, the thesis does not contain any literal citing of more than 70 words (total) even by giving a reference unless I have the written permission of the publisher to do so. Furthermore, the work presented in the thesis is my own original work and I have positively cited the related work of the other researchers by clearly differentiating my work from their relevant work.

I further understand that if I am found guilty of any form of plagiarism in my thesis work even after my graduation, the University reserves the right to revoke my MS degree. Moreover, the University will also have the right to publish my name on its website that keeps a record of the students who plagiarized in their thesis work.

Mustafa Toufiq

Dated:

Verified by Plagiarism Cell Officer

Dated:

Author's Declaration

I, Mustafa Toufiq, hereby state that my MS thesis titled “Grammar Checker for Pashto Language” is my own work and it has not been previously submitted by me for taking partial or full credit for the award of any degree at this University or anywhere else in the world. If my statement is found to be incorrect, at any time even after my graduation, the University has the right to revoke my MS degree.

Mustafa Toufiq

Dated:

Certificate of Approval



It is certified that the research work presented in this thesis, titled: “*Grammar Checker for Pashto Language*”, was conducted by *Mustafa Toufiq* under the supervision of *Dr. Omar Usman Khan*. No part of this thesis has been submitted anywhere else for any other degree. The thesis is submitted to the Department of Computer Science in partial fulfillment of the requirements for the degree of *Master of Science in Computer Science* at the National University of Computer and Emerging Sciences, Peshawar, Pakistan in June, 2019.

Candidate

Mustafa Toufiq

Signature: _____

Supervisor

Dr. Omar Usman Khan

Signature: _____

Internal Examiner

Dr. Muhammad Taimoor

Signature: _____

External Examiner

Dr. Abdul Haseeb Malik

Assistant Professor, Department of Computer
Science, University of Peshawar.

Signature: _____

Dr. Usman Habib

Graduate Program Coordinator

National University of Computer and Emerging Sciences, Peshawar

Dr. Omar Usman Khan

HoD of Computer Science

National University of Computer and Emerging Sciences, Peshawar

I dedicate this thesis to my parents, supervisor, siblings, and friends. Throughout this whole time my family and MS friends were my greatest support and they made this tough journey very easy for me. I thank you all for your constant support.

Acknowledgements

All the praise to ALLAH that induced the man with intelligence, knowledge and wisdom. Peace and blessing of Allah be upon the Holy Prophet who exhort his followers to seek for knowledge from cradle to grave. It is great pleasure to acknowledge individual and organizational help, which made this research possible.

Mustafa Toufiq

15p-7001

Abstract

Pashto is a Semitic language that is rich in its syntax and morphology. The very numerous and complex grammar rules of the language may be confusing for the average user of a word processor. This research aims to build a computational model known as Grammar Checker for Pashto Language. A program which allows user to check the grammatical errors in a Pashto sentence. It can help the average user by checking his/her writing for certain common grammatical errors. The use of this grammar checker can increase productivity and is a small contribution towards the better quality of the Pashto text for anyone who writes Pashto.

Three types of datasets are used for this system. A dictionary, training and testing dataset. Our model uses phrase structure grammar rules (PSGR) in order to check the grammar of a sentence. The system is trained over some base rules and on the basis of those rules, sentence grammar is checked. We have used probabilistic context free grammar (PCFG) and a specific probability is assigned to every base rule. We have also created a tree bank which will play a vital role while checking the grammar of a sentence.

The current implementation covers a well-formed subset of Pashto and focuses on people trying to write in a formal style. Some very successful tests have been performed using a set of Pashto sentences. As it is the first ever attempt to develop a grammar checker for Pashto Language, it is concluded that the approach is promising by observing the results.

→ pos for push to language
→ create free for grammatical
error in the push to sentences

Contents



1	Introduction	1
1.1	Scope	2
1.2	Motivation	2
1.3	Problem Statement	3
1.4	Goals and Objective	3
1.5	Thesis Structure	3
2	Review of Literature	5
2.1	Syntax Based Grammar Checker	5
2.2	Statistics Based Grammar Checker	6
2.3	Rule-Based Grammar Checker	7
2.4	Statistical and Rule-Based Approach	8
2.5	Two Pass Parsing Approach	10
2.5.1	Tokenizer	11
2.5.2	POS Tagger	11
2.5.3	Morphological Classifier	12
2.5.4	Morphological Disambiguator	12
2.5.5	POS Guesser	12
2.5.6	Tag Setter	13
2.5.7	Parser	13
2.5.8	Error Checking and Suggestion Module	13
2.6	The Granska system	13
2.7	Types of Error	15
2.7.1	Mechanic Editing Errors	16
2.7.2	Cognitive Errors	16

2.8	The Chomsky Hierarchy	17
2.8.1	Unrestricted Grammar	17
2.8.2	Context-Sensitive Grammar	17
2.8.3	Context-Free Grammar	17
2.8.4	Regular Grammar	18
2.8.5	PCFG	18
2.9	Parsers	18
2.9.1	Shift Reduce Parser	19
2.9.2	Recursive Decent Parser	19
2.9.3	Char Parser	19
2.9.4	Viterbi Parser	21
2.9.5	Algorithm	22
3	Methodology	25
3.1	Computational Rules	25
3.2	Assigning Probabilities	26
3.3	Generating Parse Trees	28
3.4	Tree Bank	28
3.5	Experimental setup	29
4	Results	33
4.0.1	Evaluation	34
4.0.2	Scoring Points	34
5	Conclusion	37
5.0.1	Future Work	37
	References	41

List of Figures

2.1	Diagrammatic Representation of Syntax-Based Grammar Checker	6
2.2	Example of Statistics Based Grammar Checker	7
2.3	Diagrammatic Representation of Two Path Parsing	14
2.4	An overview of the Granska System	15
2.5	Example of Viterbi Parser	21
2.6	Summary of Literature Review	23
3.1	State Transition Table With Respect to frequency	27
3.2	State Transition Diagram With Respect to Probability	27
3.3	Computational Rule Example	28
3.4	Parse Tree Example	28
3.5	Complete Flow Chart of Pashto Grammar Checker	30
3.6	Complete Flow Chart of Pashto Grammar Checker	31
3.7	Making data into structured form	32
3.8	Stats about the three datasets	32
4.1	Comparison table of different parsers	33
4.2	Result Statistics	34
4.3	Result Plot	35

29/11/21 start
Reading

Chapter 1

Introduction

A grammar checker program offers to correct a mistake while the word is still fresh in our mind [1]. It is one of the most widely used and popular tools within language technology [2]. Grammar-Checker softwares are now available for many languages i.e English, Arabic, Spanish, Indonesian and even for Urdu. They ease the burden of memorizing the rules of the grammar, style and punctuation.

This research offers a **computational model for linguistics-based grammar modeling of Pashto language**. It helps in identifying the **grammatical syntax-based errors** in Pashto sentences. The rule-based grammar checker described in this thesis takes an input from the user and in order to detect errors, the complete sentence is first converted into different **tokens**, **each token of the text is assigned a part-of-speech (POS) tag** [3]. Every sentence is divided into **different units**, e.g. noun phrases and verb phrases. **After that the input text is matched with all the pre-defined error rules**. If a rule matches with the input text, then the sentence is marked as correct but if the rule does not match with the input text, the sentence is supposed to contain an error at a specific position. The rules **describe errors as patterns of words, part-of-speech tags and chunks**. Most of the word-processing programs such as Microsoft office etc have companion grammar checkers [1]. Whether integrated or standalone, **the main purpose of the grammar checker is to try to capture these mistakes** [4].

Rule-Based grammar checkers are incomplete **without proper Part of speech tagging**. POS

tagging has an essential role in many areas of Natural language Processing (NLP). Part-of-speech tagging is known as the process of assigning every word in a corpus a tag which indicates the grammar of that word [5]. There are three main and most commonly used methods for implementing a grammar checker [1].

1. Syntax-Based Grammar Checker
2. Statistics-Based Grammar Checker
3. Rule-Based Grammar Checker

Sentence = [ښه ځي ښه ځي ښه ځي]
Tolron = ښه ځي ښه ځي ښه ځي
Pos = ښه ځي ښه ځي ښه ځي
 e y v o x detection ↑ ↑ ↑ ↑

Another technique

In the techniques discussed above there is also another approach to implement a grammar checker, two pass parsing which is a computational approach to check grammatical errors in a sentence [6]. This approach **parses a sentence in two steps**, first the text will be parsed on regular PSG phrase structure rules once the parsing is failed on these PSG rules then the program will shift to the second step which is **movement rules** [2]. Movement rules are used to reduce the actual number of PSG rules and convert the text or sentence into a required correct format [2].

1.1 Scope

The scope of this research is to :

- Create a part of speech tagger for Pashto Language
- Create a tree bank through which a sentence grammar will be checked

Our main motivate

1.2 Motivation

There are a lot of grammar checkers available for different languages but no grammar checker is available for Pashto language. Our motivation is to make a system which will be known as the **rule-based grammar checker for Pashto language**. The system will automatically detect grammatical errors in a Pashto sentence on the basis of some predefined

and newly generated rules. The main advantage of this system to a common man will be a person can easily check for a mistake in a Pashto sentence.

1.3 Problem Statement

No Rule Based Grammar Checker is available for Pashto Language which uses PCFG (Probabilistic Context Free Grammar) that can identify grammatical mistakes in a Pashto sentence. There are many grammar checkers available which uses different techniques to detect grammatical errors. Even in most efficient rule-based techniques which are used for grammatical error detection in other languages, most of the work is done manually and all the rules are made manually to detect errors.

1.4 Goals and Objective

The main goal of our research is to make a grammar checker for Pashto language which works on rule-based technique. Our objective will be to develop a tree bank through which the grammar of the input sentences will be checked.

1.5 Thesis Structure

In chapter 2, a detailed literature review has been done related to this problem and different techniques has been discussed that are used to solve this type of problems. Also a comparison table is given at the end of the chapter 2 which shows different techniques and their limitations. After literature review, in chapter 3 we have discussed the proposed methodology and overall architecture of our system. The complete flow chart of developing treebank and validation process. We have also discussed some feature of our dataset in the same chapter. Some important results are shown in chapter 4 and finally conclusion and future work is discussed in chapter 5.

→ Main motivation and objective is;

↳ create a POS for pushdown language

↳ Rule based technique to check the grammatical error in pushdown language.

Chapter 2

Review of Literature

Most of the work related to natural language processing (NLP) has been done in different languages like English, Indonesian, Urdu, Arabic and many others [5]. For the past many years no work is done in grammar checking for Pashto language. Although there are no grammar checkers available specifically for Pashto Language, our aim is to make a computational system which can automatically detect grammatical errors from a Pashto sentence. There are basically various approaches to implement a grammar checker such as syntax-based grammar checker, statistics-based grammar checker and rule-based grammar checker [1].

2.1 Syntax Based Grammar Checker

Through syntax-based approach, a sentence is fully analyzed morphologically as well as syntactically [7]. Syntax-based grammar checker has mainly three components.

1. Lexical Database
2. Morphological Analyzer
3. Syntax Analyzer (Parser)

Lexical database can be defined as an organized description of the lexemes of a specific language. Morphological analyzer detects morphemes in a given sentence. Morphologi-

cal analyzer is a program for analyzing the morphology of a text or a word. A parser is a program that splits up the data into smaller units so that it can be easily translated into another language. A parser inputs a text in the form of a sequence of tokens or program instructions and from that sequence it makes a data structure in the form of a parse tree. Parser assigns a syntactic structure to each sentence. If the parsing does not succeed, the input sentence is considered to be incorrect. Every technique has some pros and cons. The main advantage of syntax-based grammar checkers is the availability of large number of lexical databases of different languages, morphological analyzers and parsers. With these advantages there are some disadvantages as well i.e Syntax analyzers only identify that the input sentence is incorrect it does not give any specific explanation regarding the error [8].

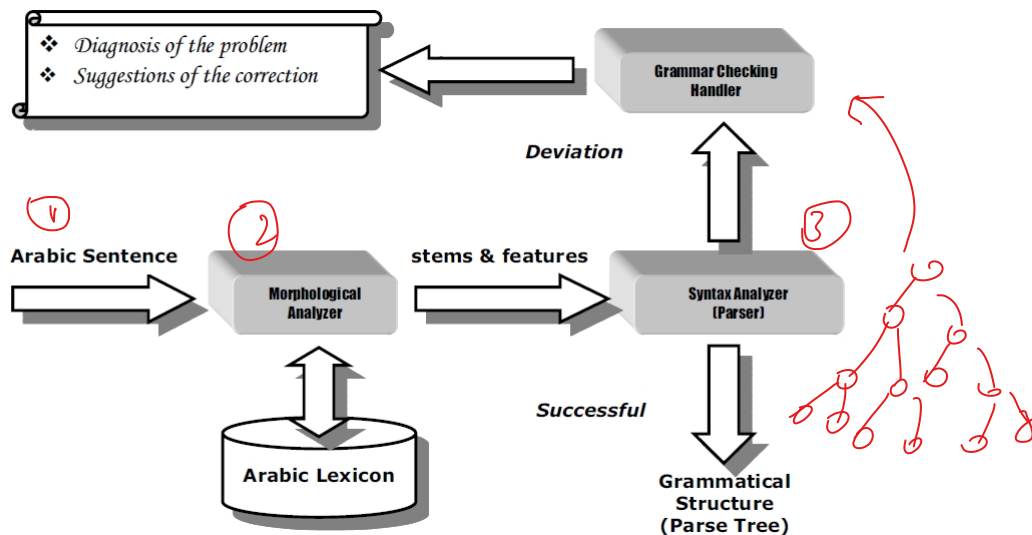


Figure 2.1: Diagrammatic Representation of Syntax-Based Grammar Checker

2.2 Statistics Based Grammar Checker

Researchers are motivated to innovate statistical models due to the huge amount of annotated text which is also known as corpus [9]. The objective behind using annotated corpus is to extract some valuable linguistic information from some specific text or corpus.

Statistical tools are used by many grammar checkers to implement several different tasks

in order to detect grammatical errors. Some of the statistical language tools are parsers (Statistical Parsers) and part of speech taggers.

In statistics-based approach a corpus is used which is part of speech Tagged. From this POS annotated corpus, a list of POS tagged sequence is generated [10]. In the whole corpus some of the sequences will be very common and will repeat frequently but there can be some POS tagged sequences which will not occur at all or they will occur rarely. If the tagged sequence is common so that sentence can be marked as correct in other texts, but if the sequence is uncommon or does not occur in the corpus that sentence will be considered as incorrect.

زه ديره بخښنه غواړم
Verb Adverb Noun Noun (۱, ۱, ۲, ۲, ۱)

زه بخښنه غواړم غواړم
Verb Verb Noun Noun (۱, ۱, ۱, ۱, ۱)

Figure 2.2: Example of Statistics Based Grammar Checker

The first example which is shown is green will be considered as correct in statistics-based approach due to some common POS tagged sequence in the corpus but the next example which is shown in red will be marked as incorrect just because of some uncommon sequence or a sequence which is not occurred in the corpus.

2.3 Rule-Based Grammar Checker

↳ good as compared to other.

Rule-Based approach is almost the same as statistics-based approach, but in this technique rules are developed and sentences are evaluated on the basis of rules [11]. In rule-based approach a corpus is used which is at least POS Tagged. It matches a set of rules (developed manually) against a text. This is the only technique which gives a complete explana-

Combination of (Rule + Statistical)
 ↳ degree of common or not common

tion of any specific error occurred in a sentence. Like any other technique this rule-based checker approach has also many advantages. In this approach a sentence does not have to be complete to be checked, instead if the user is typing a sentence and it is erroneous the program will automatically detect the error while the sentence is not completed yet. Unlike statistics-based grammar checkers error detection in rule-based technique is quite easy because every rule has some expressive description and each rule can be turned on and off. It can give detailed error messages with helpful suggestions and it can even explain grammar rules [5]. This rule-based system is easily extendable from starting just from one rule and then extend it rule by rule.

2.4 Statistical and Rule-Based Approach

Besides these three main approaches to implement a grammar checker which are discussed in the above section, some researchers have developed other techniques as well to implement grammar checkers. These techniques are either with some minor modifications in the existing techniques or they have merged two approaches to form a single technique. These techniques can be a statistical and rule-based approach, two pass parsing and some others which we will discuss in detail in the next section of this chapter.

This system is composed of two parts:

1. The rule matcher

2. The grammar-checker

The Pashto sentence structure is subject-object-verb

Sentence structure for pashto

The main task of the rule matcher module is to identify the error and then correct it [12]. It also provides the details of a specific error whether in punctuation or word choice. All of this work is done by matching the set of rules with the text or sentence. The next module grammar checker determines whether the sequence of elements in the sentence specifies a correct or incorrect structure of an overall sentence.

The rules in this program includes the following set of information and patterns. The patterns are defined as a regular expression. Rules are divided into two categories, "Text

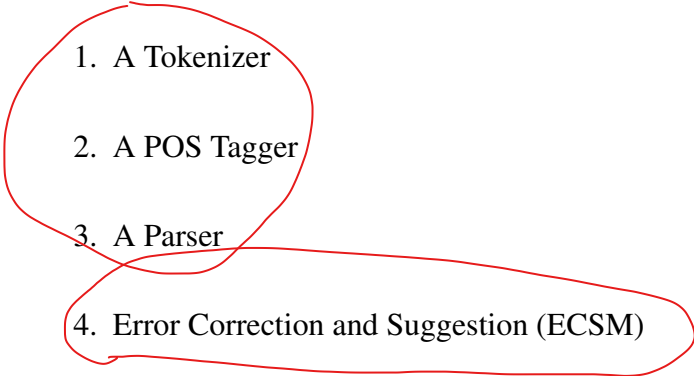
rule set” and “Sentence rule set”. Text rule set are rules which will be matched with the complete corpus, After the corpus is split into different number of sentences then Sentence rule set are used. These rules are matched with every sentence in the corpus [10]. For example, in text rule set a single rule may be used for checking the common mistakes in a corpus which can be checking spaces between the words, which may not be visible when the complete corpus is divided into sentences. Another rule from the set can be used to remove the redundant words in the corpus just like cleaning the corpus or like pre-processing. The rules are made on the basis of some very common mistakes or errors. A special kind of rule is also developed which is used before the corpus is split into sentences. The main objective is to fix the space issues i-e the absence of white spaces after a full stop. Sentences without any punctuation errors are checked by the grammar checker module. A scoring mechanism is used by such kind of techniques are sentence scores are calculated. Scores can be calculated according to three modules which can be chunks, tokens or tags. In most of the cases the longer sentences have low scores which is the main reason that longer sentences are marked as incorrect. A threshold is set and the score is compared to that threshold. If the score of a particular sentence is higher than the threshold the sentence is marked as correct but if the score is less than that sentence is marked as incorrect. The threshold will always be determined from the experiments. In Statistics and rule-based grammar checker two different types of testing sets are used. In each testing set there are fifty correct sentences which are taken from the corpus and fifty incorrect sentences. The first set which is relatively easy to create, incorrect sentences are made by shuffling the word order for every sentence. In the second set, the level of difficulty is much higher for the incorrect sentences. In that 50 set of sentences half of them i-e 25 sentences are shuffled. The remaining 25 sentences are either incomplete, having sentence structures which are unclear or the sentence itself is incomplete. Such type of sentences may seem correct at first glimpse. Both the datasets are tested on three different models which are:

1. The Token
2. The POS tag
3. The Phrase Chunk

The three models discussed above have different accuracies but the maximum accuracy is achieved by POS Tag (N) is 82%, the threshold for this model was -0.4. As discussed earlier that the second dataset have errors which are more difficult to detect therefore its accuracy is less than the previous dataset. The maximum accuracy achieved from this dataset is 70% having POS Tag model with a threshold of -15.21.

2.5 Two Pass Parsing Approach

In another implementation for grammar checker two pass parsing approach is used. Initially when a user inputs a text it is first parsed on some very basic **Phrase Structure Grammar Rules (PSG)**. When a sentence is failed to parse from these PSG rules and it moves to the second pass which is known as **Movement rules**. The main purpose of using movement rules is to convert that erroneous sentence to a desired correct form. Once the movement rules are applied successfully the sentence is again reparsed to check errors. Grammatical errors can be case disagreement, gender or number. The structure of this grammar checker is as follows

- 
1. A Tokenizer
 2. A POS Tagger
 3. A Parser
 4. Error Correction and Suggestion (ECSM)

Every module of this grammar checker is discussed in full detail.

The main question is why using Two pass passing approach. In any language there can be millions of sentences and it is not possible to list all the sentence structures of a language. Therefore, some generalization is required to cover the syntax of a language in some depth. In the first parse which is phrase structure grammar (PSG) it is very difficult to get any generalization of rules. Due to this major drawback even if there are two different sentences having exactly same elements but with just a little varied structure

will be analyzed with different PSG rules. In order to get some generalization in the previous pass (PSG) the next technique which is movement rules are applied. The core objective of movement rules is to reduce the grammar rules for the same sentences which have mostly different sentence structure. As discussed earlier that the two-pass parsing technique have four major components that is a tokenizer, a POS tagger, a parser and an error correction and suggestion module (ECSM).

2.5.1 Tokenizer

Tokenizers can be of several types, but mostly used tokenizers are word tokenizer and sent tokenizer. Sent tokenizer splits the paragraphs into different sentences and similarly word tokenizers split the sentences into different words or tokens. It a short program that identifies a token as well as sentence boundaries. It takes input from a user and then apply word tokenization to split the words into individual tokens [2] [4]. These tokens are then passed to the next module which is POS Tagger.

2.5.2 POS Tagger

The tokens which were generated from the previous module allots a part of speech tags (POS-Tags) to every token. POS tagger looks up the lexical database to collect morpho-syntactic information for every token of a sentence. POS tagger consist of 4 sub modules [4].

1. Morphological Classifier ✓

2. Morphological Disambiguate ✓

3. POS-Guesser ✓

4. Tag-Setter ✓

2.5.3 Morphological Classifier

Morphological classifier labels the words or tokens in a sentence with its morpho-syntactic features. Lookup table usually have all the relevant information of a word/token and is used to find all the labels and morpho-syntactic features of a given word [2] [4]. The information in lookup table regarding a word can be its gender, number or case etc. This information is also known as the features of a word or token. The purpose of a lookup table is to retrieve the information associated with a word or token but there is always a chance of ambiguity for a specific word [4]. For example, there may exists some words in any language which have more than one set of morpho-syntactic information. In such scenarios there is a clear ambiguity and to resolve that another module is used which is **Morphological Disambiguator**.

2.5.4 Morphological Disambiguator

The main purpose of this module is to eradicate the ambiguities which were generated by the previous module [4]. Ambiguities arises when a specific word has more than one morpho-syntactic information. So, by using this module these ambiguities can be easily resolved. The Morphological Disambiguator picks out the most suitable set of morpho syntactic information keeping in view the context of a given sentence and will discard the other one which is irrelevant. There are several ways through which it can remove the ambiguities which includes Neural Networks, Rule based system etc

2.5.5 POS Guesser

③

Lexical database usually contains all possible lexemes of a language but still there is a strong possibility that some of the lexemes or words is missing from the lexical database [4]. So, if user inputs a sentence and an input word or token is not available in the lexical database then it means **it have no morpho-syntactic information**. Now it is the responsibility of POS-Guesser to guess the part of speech of that specific word [4]. There are many erudite POS guessers are available which can even guess the gender of a word.

If the input sentence has no unknown lexemes then this module will be skipped.

2.5.6 Tag Setter

Now when a sentence is completely checked and all the ambiguities are removed the words or tokens are now passed to the next module which is Tag setter. As it is obvious from its name that it will assign a suitable part of speech (POS) tags to every word in a sentence [4].

2.5.7 Parser

The main purpose of a parser is to split the data into different units so that it can be easily transformed into another language [1]. It takes sequence of tokens and builds a data structure usually in the form of parse tree. In the previous module every word was assigned a part of speech tag. These POS-Tagged words are now called terminals nodes and are then fed to the parser. The parser passes these tags to the PSG rules which are loaded from a file. From these PSG rules the parser generates a parse tree only if there are no grammatical errors in the input text [4].

2.5.8 Error Checking and Suggestion Module

This is the final module of the complete structure. If the input sentence is without any error then this module is skipped but if the input sentence is without any error then Error Checking and Suggestion Module (ECSM) comes into action and it suggests the relevant correct version of a sentence [4].

2.6 The Granska system

Granska is a swedish grammar checker program which is a combination of two methods that is probabilistics and rule based methods. It is also know as hybird system due to the combination of two techiniques. The two techiques are used together inorder to

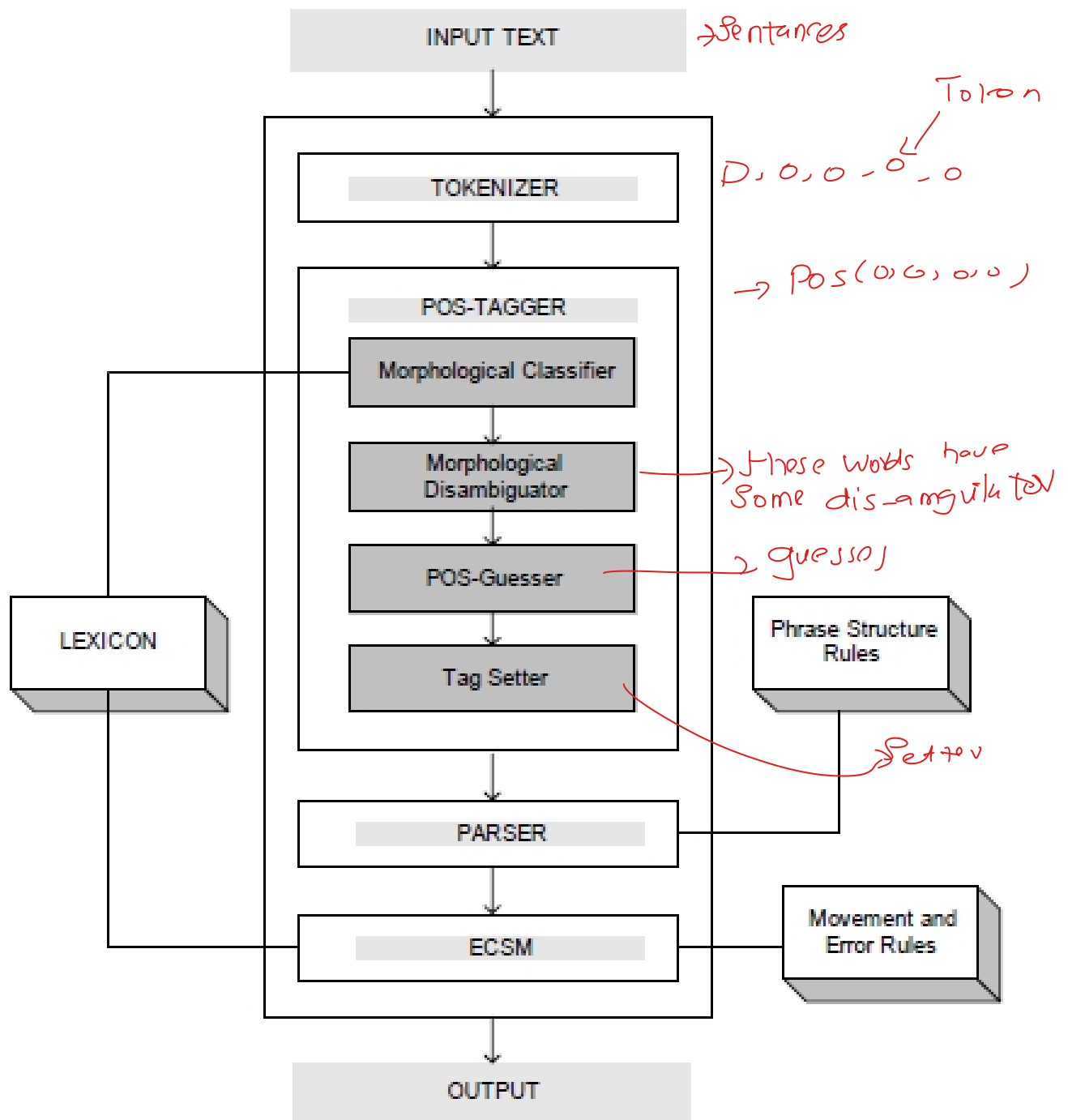


Figure 2.3: Diagrammatic Representation of Two Path Parsing

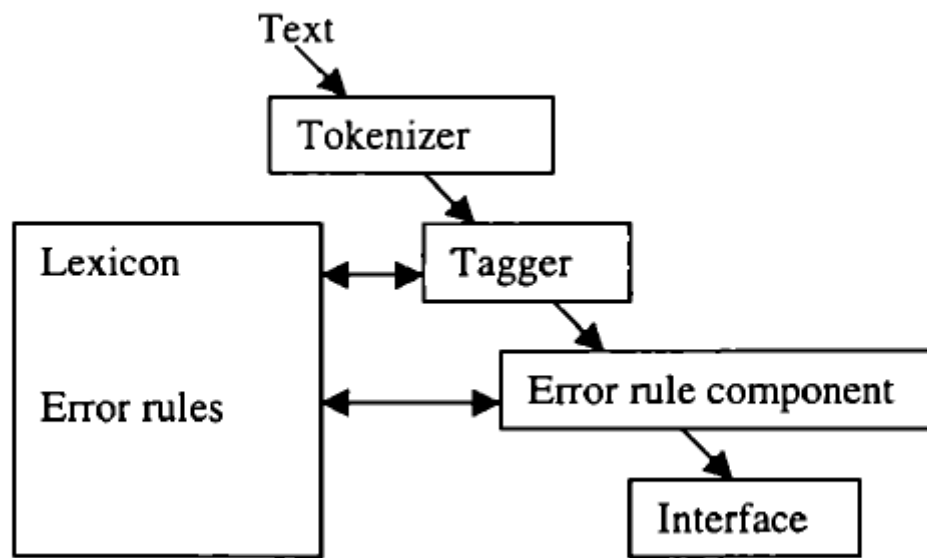


Figure 2.4: An overview of the Granska System

achieve high efficiency and robustness. It detects errors in a Swedish language by using some special error rules explicitly made for Granska. The system also gives the facility of suggestions to the users. The overall structure and working of this module is shown in the figure below.

In the first step which is tokenizer, the overall text and some special characters are split into different tokens or lexemes. In the second step which is known as tagger, the tokens or lexemes which were tokenized in the previous step are assigned a Part of Speech Tag. In the next step the tagged tokens or words are sent to the error rule components. The error rule component checks for the specific grammatical problems in the text. After the grammatical errors are identified then the system gives suggestions for the correction of the specified errors.

2.7 Types of Error → Types of error:

There can be several types of errors which can occur. But the most common ones are classified into two main categories [1].

1. Mechanic editing errors ①
2. Cognitive errors ②

2.7.1 Mechanic Editing Errors

While using a word processor user often use some short cuts or do some operations such as copy paste, cut and paste some of the text or even some insert/delete operations. So, the errors occur due to these operations are known as mechanic editing errors. These types of errors are easy to handle and can be removed by just deleting some part of the text or replacing a text with a correct one [1] [7]. Some of the most frequent errors which can occur from a user while using any word processor are listed below.

- Deleting some part of the old text while inserting new text.
- Misspelled words which accidently produce a real word.
- Selecting a wrong word from spell suggestion module of a word processor.
- Excessive letter or missed letter.

2.7.2 Cognitive Errors

These types of errors are more complicated to deal with. The main reason for these types of errors are the user who are incompetent or completely unfamiliar with the grammar rules of a certain language [1] [7]. Cognitive errors can be removed by simply by just adding a new word or replacing an existing word.

Apart from these two main types of errors grammar checker programs also look for two other type of errors [7]:

1. Structural errors
2. The local errors

For short sentences or short sequence of tags and words the local error rules are applied but in case of more multifaceted or complex errors structural error rules are applied [7]

2.8 The Chomsky Hierarchy

The Chomsky hierarchy has total four levels of grammars [13]:

1. Unrestricted Grammar
2. Restricted Grammar
3. Context Free Grammar
4. Context Sensitive Grammar

2.8.1 Unrestricted Grammar

In every level of grammar there will be a LHS and a RHS. On both the sides of a rule there will be some terminals and non-terminals. Unrestricted grammar is a type 0 grammar in which there is no restriction, means there can be any number of terminals and non-terminals on both side of the rules. Unrestricted grammars are also known as the most powerful grammars in Chomsky hierarchy.

2.8.2 Context-Sensitive Grammar

As the name suggests Context-sensitive grammars are a type of grammar in which some restrictions are applied [13]. It is also known as the subset of Unrestricted Grammar. It's a type 1 grammar in Chomsky Hierarchy. In this grammar the left-hand side LHS rule can never be greater or longer then the right-hand side rules RHS. It can have any number of terminals and non-terminals on the RHS but LHS will always be shorter or equal.

2.8.3 Context-Free Grammar

It's a type 2 grammar in Chomsky hierarchy. As the name suggests that it's a context free grammar means there can never a context with it on the LHS. On the right-hand side there can be terminals, non terminals and a combination of both but on the LHS there

will always be a single nonterminal having no context. This is the most used grammar for linguistics and specially for making phrase grammar structure (PSG). Phrase grammar structure are the production rules which are made with the help of context free grammar.

2.8.4 Regular Grammar

As every level in Chomsky hierarchy is a subset of the previous level. So, we can also say that all the previous states that is restricted grammar, unrestricted grammar, context free grammar are also regular grammars [14]. It can be defined as a grammar in which all rules take one of the following two forms: $A \rightarrow a$; or $A \rightarrow aB$: Here, A and B are the non-terminal and terminal symbol [13].

2.8.5 PCFG

Probabilistic context-free grammars (PCFGs) is a type of CFG where probability $P(A \rightarrow \alpha)$ is associated with every rule. It provides simple statistical models of natural languages [15].

$$\sum_{\alpha:A} P(A \rightarrow \alpha) = 1 \quad (2.1)$$

The probability of each production rule and terminal must be summed up to 1.

2.9 Parsers

We had tried different parsers in order to get better results. Every parser had some pros and cons.

1. Shift Reduce Parser
2. Recursive Decent Parser
3. Chart Parser

4. Viterbi Parser

2.9.1 Shift Reduce Parser

Initially for all the parsing operations Shift reduce parser was used. At each step of this parser, two main operations are performed:

1. Shift operation
2. Reduce operation

Shift operation adds the next word in the sentence to the top of the stack, and the reduce operation removes several elements from the top of the stack and replaces them with a new element [16]. Shift reduce parser is a bottom up parser. Another parser we applied was recursive decent parser. Recursive decent parser is a top down parser [17].

2.9.2 Recursive Decent Parser

Recursive descent parser uses a top-down approach that constructs the parse tree from the top. In this technique every input is read from left to right. Top- down parsers always take start from the root node which is also known as the starting symbol of the grammar, and matches the input string against the production rules. If the input string is matched, that specific rule is then replaced. This type of parsing approach is called recursive because it uses context free grammar which is recursive in nature.

2.9.3 Char Parser

Unlike previous parsers chart parser works in a Dynamic Programming approach [18]. The main goal of dynamic programming is it divides a bigger problem into smaller sub-problems and then solves each smaller problem. Finally, after solving every subset of the problem it all combines to make a one solution. Chart parser works in exactly the same manner. The main advantage of chart parser is that it avoids recursion or repetition which

was the main problem in previous parsers. Chart parser basically stores the partial subtrees into a chart or matrix in order to avoid unnecessary repetitions and use those partial subtrees later.

The dynamic programming chart parser is a top down approach and it parses every sentence from left to right. The parsing process from left to right fills up the chart of the size $N+1$. N is basically the number of input words in a sentence, so the matrix or chart size will be always $N+1$. Every word position in an input sentence contains some states which represent the partial subtree. There are total 3 states of a chart parser. At any time, chart parser can be in any of these states [18].

1. Predicted Constituents
2. In-Progress Constituents
3. Completed Constituents

Predicted Constituents means that the parser has not parsed any single rule yet and it is at the beginning of the rule. When a parser is in In-Progress state it means that some of the rules are parsed successfully and the parser is moving on in order to parse other rules. Where as the final state which is Completed states defines that a rule is completely parsed successfully. There is a very popular concept of dots in chart parser. Dot basically acts as a pointer. The parsing process in chart parser is represented by Dot rules. The main purpose of a dot is it indicates the position of a constituent. e.g

$$NP \rightarrow PN.VP$$

In order to keep track of this dot also known as pointer, chart parser uses a pair of coordinates $[x,y]$. “x” keeps the track of the position from where the state begun and “y” is the actual position of the dot.

State of Chart Parser Example:

- $S \rightarrow . NP VP, [0,0]$ (**Predicted State**)
- $NP \rightarrow PN . VP, [1,2]$ (**In Progress State**)

- $VP \rightarrow V N ., [0,3]$ (Completed State)

Note: the completed state will always contain a fully-grown tree.

2.9.4 Viterbi Parser

Just like chart parser, viterbi algorithm also uses dynamic programming approach which divides the bigger problem into smaller subproblems and combine all those small sub solutions in order to obtain the final result [18]. The main objective of the viterbi algorithm is to find the most likely sequence of states from a given state transition frequencies or (Hidden Markove Model) or we can say the goal of this parser is to sequence of states which has the highest probability. This viterbi algorithm generates a parse tree of an input sentence on the basis of maximum probability. It creates a table and then parses input text [18]. On the basis of those parsing it fills up the table with the most likely constituents. In the next part we have explained the algorithm of the Viterbi parser that how it actually works.

د تادی انجام خراب وي

A	0.1
V	0.2
PN	0.2
N	0.5

	A	V	PN	N
A	0.5	0.2	0.1	0.2
V	0.2	0.5	0.2	0.1
PN	0.4	0.2	0.3	0.1
N	0.3	0.4	0.2	0.1

N	A	PN	V	N
تادی	خراب	د	وي	انجام
0.3	0.3	0.2	0.1	0.1

	وي	خراب	انجام	تادی	د
A	0.00000028	0.000036	0.00024	0.009	0.02
V	0.00000072	0.0000144	0.0006	0.012	0.04
PN	0.00000018	0.0000072	0.00018	0.006	0.04
N	0.00000014	0.0000136	0.00012	0.003	0.1
	V	V	A	V	N

Figure 2.5: Example of Viterbi Parser

2.9.5 Algorithm

1. An input of length T : $o_1 \dots o_T$, and we want to find the sequence of states.
2. Each o_t in the sequence is one of the states in state transition frequency or hidden Markove Model (HMM).
3. For each state q , we initialize our table $V[q, 1] = \pi_q \cdot b_q(o_1)$
4. Then compute for $t = 1 \dots T-1$ for each state q :

$$V[q, t+1] = \max_{q'} V[q', t] \cdot a_{q'q} \cdot b_q(o_{t+1}) \quad (2.2)$$

5. After the loop terminates the best score is $\max_q V[q, T]$.

After all transitions, the winner will be the transition having the maximum probability value. In this method the maximum state will be the last value of the table. This maximum value can also be called the Viterbi score of the state transition frequency for the given sequence. The last node or the maximum value is also the starting index for the backtracking purpose. Backtracking is an essential part of this whole algorithm which visits each node in backward order or reverse order until the first index is reached. The path from which the algorithm has done backtracking is the resulting best state sequence with the highest probability.

Now as we have discussed the literature review and by observing all the limitations in the above discussed techniques now we will move to the next chapter which is methodology of our system.

Name	Technique	Limitations	Ref
Arabic GramCheck	Syntax Based Approach	Only recognize that sentence is incorrect, does not tell the user what the exact problem is. For this, extra rules are needed	1
Two Pass Parsing	Rule Based Approach	Very limited rules. Can be expanded with help from language experts	2
N gram based Statistical Grammar Checker	Statistical Based Approach	Difficult to interpret, as no specific error message is displayed	3
Granska, Swedish grammar checker	Rule Based Approach	The results mentioned for this grammar checker are based on artificial errors	4
Spelling and Grammar Checker for Indonesian Tex	Statistical and Rule-Based	The statistics-based components can be further improved by creating corpuses	5
Towards Grammar Checker Development for Persian Language	Statistical Based Approach	Sometimes the sentence will be correct but will be marked as incorrect due to uncommon POS sequence in the corpus	6

Figure 2.6: Summary of Literature Review

Chapter 3

Methodology

In rule-based grammar checker a part of speech tagged corpus is used [1] [19]. The corpus is tokenized and every word is considered as a separate entity or token. As the corpus is POS tagged so every token is assigned a part of speech tag. Once the part of speech tagging is done the system will go to the next module which is Rule making for Pashto sentences.

*P₁ → Training
P₂ → Testing
P₃ → Creating Rule*
*some base
→ some computational Rule
are made from pashto Sentences*
Pos chunking technique is used

3.1 Computational Rules

We have created three different kinds of datasets. One is for training and the other is for testing and one is for creating some base rules. In this module some computational rules are made from Pashto sentences. We have used **POS chunking technique to generate all the rules**. These rules are known as the **base rules or computational training rules**. There are basically no limitations on how many base rules will be made. Initially we have made some base rules and checked sentence grammar on the basis of these computational rules. We will increased the base rules later depending upon our results. Initially there were a few computational rules but the accuracy was not up to the mark due to which base rules were increased these base rules. When we are done with developing the rules the next major part was to assign probability to each and every rule.

3.2 Assigning Probabilities

We have used PCFG (Probabilistic Context free Grammar) in order to assign probability to each rule. Normally when no probability is assigned with a rule then CFG (Context free grammar) is used [20]. But here as we are assigning probabilities, we are using PCFG. Each Probability will be between 0 and 1. Probabilities are assigned on the basis of POS tags sequences in our training dataset, also known as state transition. In state transition model we see the frequency of every pair of POS tags. That is how many times a sequence of Noun and Noun (N N) is occurred, how many times sequence Noun Adverb (N ADV) is occurred and so on. Example of state transition is shown in the figure below. We have also obtained the actual frequency of every POS tagged pair from the training dataset. The method we have used to obtain the probability of each rule is first the frequency of each pair is calculated and afterword's we have summed up the overall number of tags associated with Nouns, Verbs, Adverbs etc. and then divided it by the total number of tags. Example of this complete method is shown in figure below. The main purpose of adding probabilities to a computational rule is that it will help us generating accurate parse trees and it will also help in making a tree bank which is described in detail in the later module. The next step will be creating parse trees on the basis of pre-defined rules.

We had took sentences from the dataset and made some computational grammar rules against those sentences. The computational grammar rules which were made are known as Base Rules, or training rules. When we got a significant number of rules, we started to test different sentences on to these rules in order to generate parse trees. The sentences used for generating parse tree were from different dataset which was training dataset. Initially we set a threshold of 5 words per sentence means every sentence will have exactly five words or less (threshold was increased to 7 or 8 words depending upon the initial results). If a rule is present against a specific sentence, parser will generate a parse tree for that sentence, but if no rule is present for a specific sentence no parse tree will be generated.

	A	B	C	D
1	N -> N	149	0.620833	
2	N -> V	9	0.0375	
3	N -> ADJ	31	0.129167	
4	N -> AV	26	0.108333	
5	N -> NU	2	0.008333	
6	N -> PP	4	0.016667	
7	N -> PN	19	0.079167	
8		240		
9	ADJ -> ADJ	77	0.57037	
10	ADJ -> N	25	0.185185	
11	ADJ -> V	7	0.051852	
12	ADJ -> AV	14	0.103704	
13	ADJ -> NU	1	0.007407	
14	ADJ -> PP	2	0.014815	
15	ADJ -> PN	9	0.066667	
16		135		
17	V -> V	42	0.56	
18	V -> N	14	0.186667	
19	V -> ADJ	6	0.08	
20	V -> AV	9	0.12	
21	V -> NU	1	0.013333	
22	V -> PP	1	0.013333	
23	V -> PN	2	0.026667	

Figure 3.1: State Transition Table With Respect to frequency

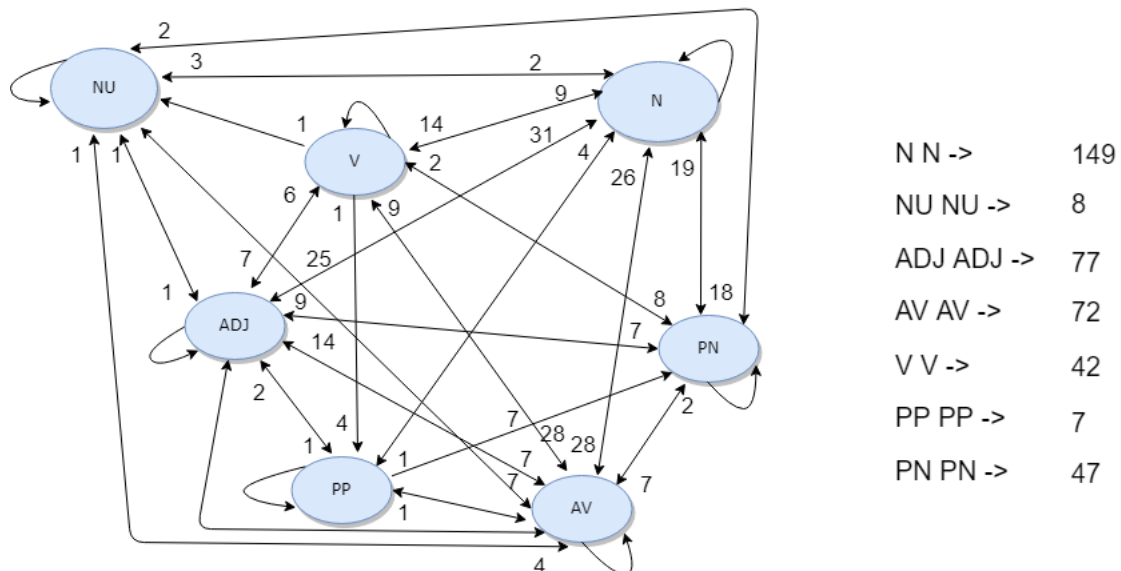


Figure 3.2: State Transition Diagram With Respect to Probability

3.3 Generating Parse Trees

After creating some base-rules we will take different sentences from the testing dataset and input them to our algorithm. The sentence will first be tokenized and every token will be assigned a POS tag. After tokenization those words are then passed to the chart parser where it checks the grammar of the input sentence on the basis of base-rules. If a rule exists for a specific sentence a parse tree will be generated, if not, then no parse tree will be generated. The resulting trees are then stored in a different file for further processing. Once the sentences are all parsed and trees are generated, we will move on to the next module which is developing a tree bank on the basis of the highest probability rule.

```

S -> NP ADVP VP [0.0769230769]
NP -> PRN [0.0294118]
ADVP -> PRP ADV RP [0.333333]
VP -> V [0.05]

```

Figure 3.3: Computational Rule Example

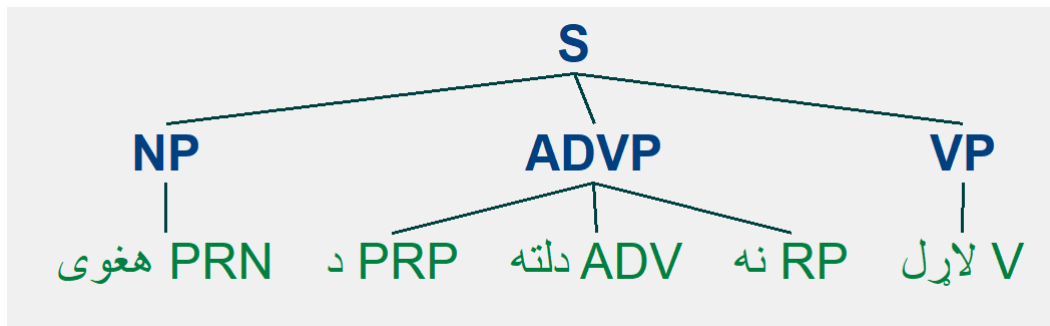


Figure 3.4: Parse Tree Example

3.4 Tree Bank

Now we are in a situation where we have a complete POS tagged corpus (Dataset training and testing), corpus is completely tokenized, every word is assigned a part of speech tag, base rules are created, a number of sentences are parsed and we have got different number

parse trees. Now in order to develop a tree bank in this module we will try to find the maximum probability trees which were obtained previously. In previous module when parse trees were generated for different sentences, a probability was assigned to each parse tree by Viterbi parser. Now we had selected the trees with the maximum probability and added it to the tree bank. The main advantage of a tree bank is we can get rid of some extra rules which are present in our grammar and we can make our computational grammar rules more precise and accurate. This tree bank will also help us in the suggestions modules as well.

User can now check the status of any given sentence that whether its grammatically correct or incorrect. If a certain rule which is generated from training dataset is present grammar or a specific tree is which was generated from the previous training sentences is present in a tree bank or it may be a sub-part of a bigger tree which is present in a tree bank in that case the input sentence is marked as correct. But if a tree which is generated from a sentence does not correspond or match with any of the grammar rules or trees in a tree bank that sentence will be marked as grammatically incorrect.

3.5 Experimental setup

In order to carry out the experiments we have used three different data sets.

1. POS tagged tokens
2. Training Dataset
3. Testing Dataset

The first dataset which is known as Pashto dictionary or POS tagged tokens contains 18249 part of speech tagged Pashto words/tokens. The dataset was initially in raw form containing more than 39,000 tagged words having more than 20 POS tags and some extra information with every word. Obviously we did not need any extra information related to words so we converted the raw text file format into some structured data. In this dataset

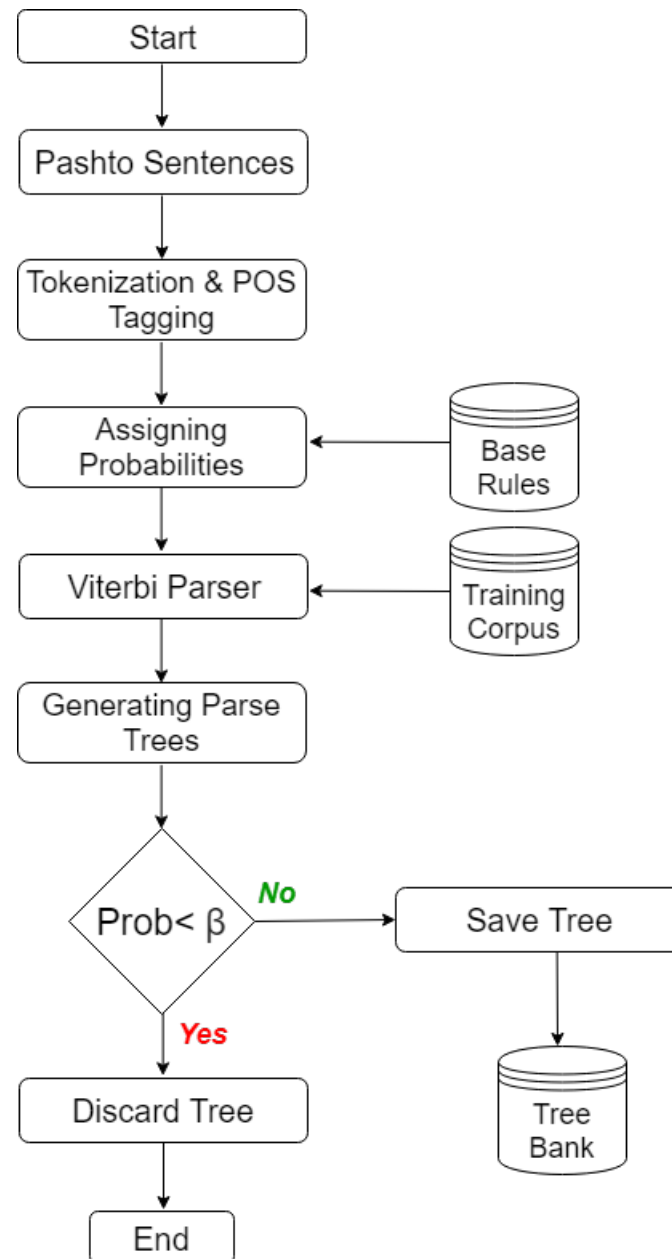


Figure 3.5: Complete Flow Chart of Pashto Grammar Checker

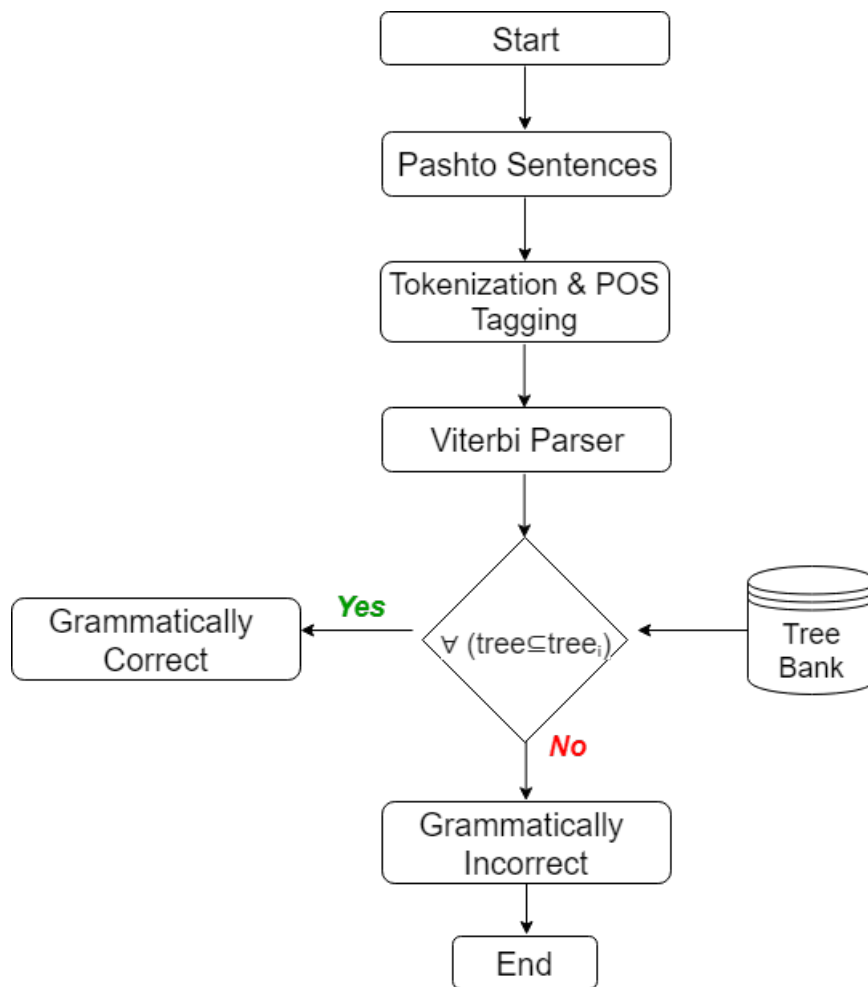


Figure 3.6: Complete Flow Chart of Pashto Grammar Checker

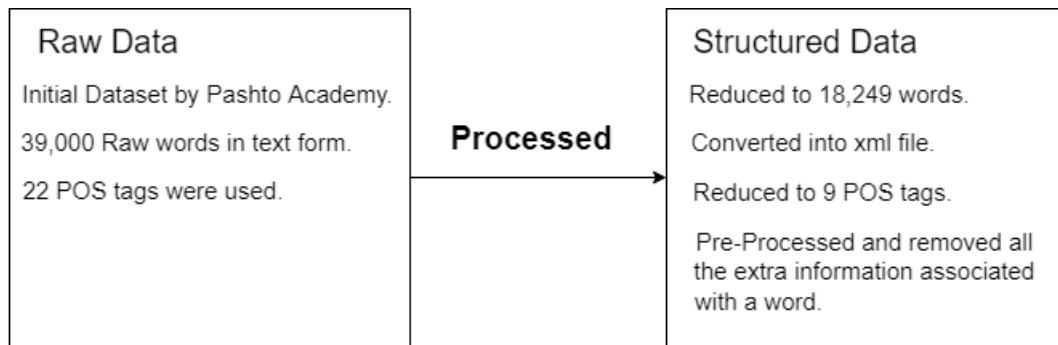


Figure 3.7: Making data into structured form

POS tagged Words	Training Dataset	Testing Dataset
18249	85	152

Figure 3.8: Stats about the three datasets

every word is assigned a POS tag. So, whenever a user inputs a sentence, that sentence breaks down into different tokens and the related part of speech tag is fetched from this dataset.

The other two datasets contains different pashto sentences. Both the datasets training and testing are created manually from different sources. It is specially designed for our system (Pashto Grammar Checker). One set contains all the sentences which are used for training while the other portion of the dataset contains all the sentences which are used for testing our model.

We have obtained some results through this methodology which we will see in the coming chapter.

Chapter 4

Results

In this section, we are going to discuss the key aspect of our research that is the results of the proposed method. We have carried out experiments on large number of sentences selected from test data of our dataset (Pashto Corpus). Initially we tried different parsers but every parser has some limitations. Same was the case when we applied Chart parser which is a dynamic programming approach we ended up with many problems regarding probabilities. The only parser with good results was Viterbi parser, as a result we selected viterbi parser for better accuracy and results. A detailed comparison table is give for beeter understanding.

	Infinite Loop	Exponential Time	Probability Score	Dealing with Complex Sen
Shift Reduce Parser	No	No	No	No
Recursive Decent Parser	Yes	Yes	No	No
Chat Parser	Yes	Yes	No	Yes
Viterbi Parser	No	No	Yes	Yes

Figure 4.1: Comparison table of different parsers

4.0.1 Evaluation

We tested our algorithm on a data set of 152 sentences. This dataset is also known as unseen data because all of these sentences were different from the sentences on which the system was trained. We injected 40 grammatical errors in the test set in order to check the performance of the model. The Pashto grammar checker is tested on an artificial text of errors. These errors are taken from different types of defined errors which are possible in a Pashto sentence. Viterbi parser is applied to each sentence. The results indicate that for most of the wrong sentences, no parse tree was generated. While on the other hand the sentences which were grammatically correct were successfully parsed and a corresponding parse tree was generated.

n= 152	Predicted: NO	Predicted: YES
Actual: NO	TN = 33	FP = 16
Actual: YES	FN = 12	TP = 91

Figure 4.2: Result Statistics

4.0.2 Scoring Points

We also wanted to see the scoring results of our dataset. In order to do that we generated a plot which shows the scores of the dataset sentences. As we can see in the diagram (Plot) that the sentences having maximum length or words greater than 6 in a given sentence are pushed towards the minimum value in the plot and the sentence whose length is less or less than 6 words per sentence are near to the maximum value in the plot. From this result we can easily conclude that the score of the sentences having greater length will be close to zero while the score of the sentences having minimum length will be close to maximum.

4. Results

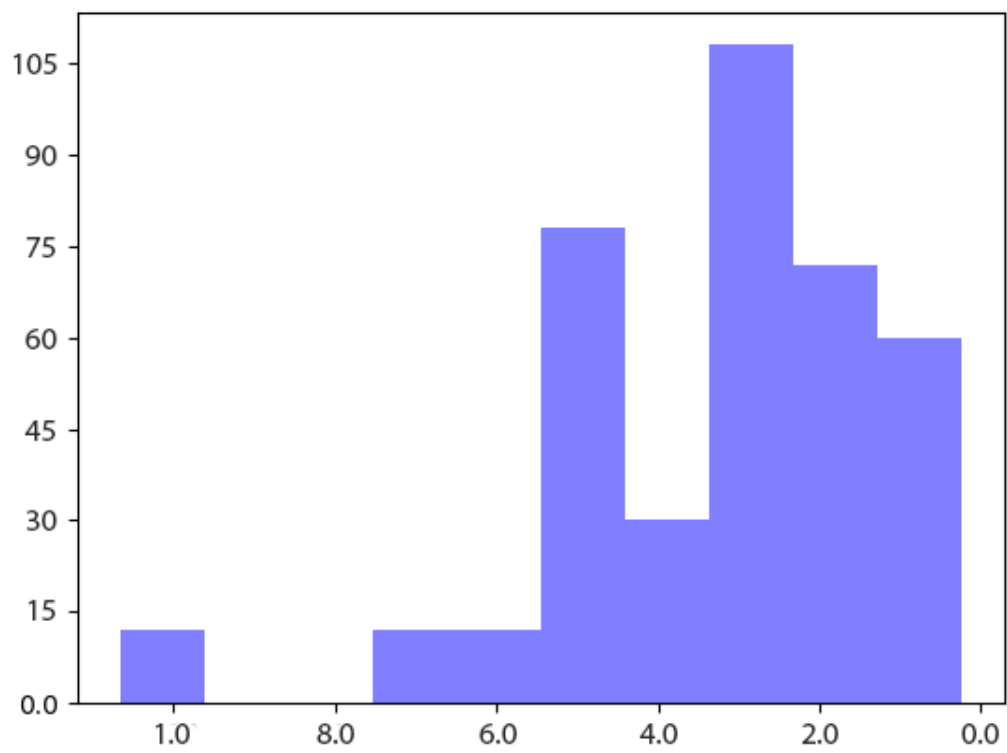


Figure 4.3: Result Plot

Chapter 5

Conclusion

This is the first attempt to develop a Pashto grammar checker system. In the field of NLP, grammar checkers are one of the most commonly used applications in word processors. It is a very important tool for local language computation. In this system, rule-based grammar checking of Pashto texts is implemented. In rule based approach some base rules were used which were taken from the domain or knowledge experts. On the basis of those base rules, different computational rules were generated for different sentences. This useful software is capable of taking input from the user, mark it gramatically correct or detect some specific common grammatical errors. We expect that the presented findings and results will be very useful for the future modifications and improvements in this kind of systems. The system is implemented by using Python 3.7.

5.0.1 Future Work

Since this is the first Pashto grammar checker, there are still a number of tasks required to improve the software. Some additional training and rules would help in better accuracy. Another very important module known as suggestions module can also be added to the sytem. Apart from that the parse trees generated by viterbi parser are in unnormalised form. In future we can make these parse trees in normalised form inorder to increase the accurarcy of our system. The system uses rule based technique for implementation,

5. Conclusion

other than rule based grammar checker, statistical grammar checker can be introduced for Pashto language. The resulting grammar checker can be a hybrid system combining both statistical and rule based approach.

Bibliography

①

[1] K. F. Shaalan, “Arabic gramcheck: A grammar checker for arabic,” *Software: Practice and Experience*, vol. 35, no. 7, pp. 643–665, 2005.

②

[2] H. Kabir, S. Nayyer, J. Zaman, and S. Hussain, “Two pass parsing implementation for an urdu grammar checker,” in *Proceedings of IEEE international multi topic conference*, 2002, pp. 1–8.

③

[3] H.-C. Liou, “Development of an english grammar checker a progress report,” *CALICO journal*, pp. 57–70, 1991.

[4] S. M. J. Rizvi, “Development of algorithms and computational grammar for urdu,” Ph.D. dissertation, Ph. D. thesis, Pakistan Institute of Engineering and Applied Sciences (PIEAS . . . , 2007.

⑤

[5] I. Rabbi, A. Khan, and R. Ali, “Rule-based part of speech tagging for pashto language,” in *Conference on Language and Technology, Lahore, Pakistan*, 2009.

[6] A. Vandeventer, “Creating a grammar checker for call by constraint relaxation: a feasibility study,” *ReCALL*, vol. 13, no. 1, pp. 110–120, 2001.

[7] J. Kinoshita, L. do Nascimento Salvador, and C. E. D. de Menezes, “Cogroo: a brazilian-portuguese grammar checker based on the cetenfolha corpus.” in *LREC*, 2006, pp. 2190–2193.

[8] R. Domeij, O. Knutsson, J. Carlberger, and V. Kann, “Granska—an efficient hybrid system for swedish grammar checking,” in *Proceedings of the 12th Nordic Conference of Computational Linguistics (NODALIDA 1999)*, 2000, pp. 49–56.

- [9] M. Alam, N. UzZaman, M. Khan *et al.*, “N-gram based statistical grammar checker for bangla and english,” 2007.
- [10] N. Ehsan and H. Faili, “Statistical machine translation as a grammar checker for persian language,” in *Proceedings of the Sixth International Multi-Conference on Computing in the Global Information Technology*, 2011, pp. 20–26.
- [11] R. T. Martins, R. Hasegawa, M. d. G. V. Nunes, G. Montilha, and O. N. De Oliveira, “Linguistic issues in the development of regra: A grammar checker for brazilian portuguese,” *Natural Language Engineering*, vol. 4, no. 4, pp. 287–307, 1998.
- [12] G. Adriaens and D. Schreors, “From cogram to alcogram: Toward a controlled english grammar checker,” in *Proceedings of the 14th conference on Computational linguistics-Volume 2*. Association for Computational Linguistics, 1992, pp. 595–601.
- [13] G. Jäger and J. Rogers, “Formal language theory: refining the chomsky hierarchy,” *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 367, no. 1598, pp. 1956–1970, 2012.
- [14] E. Brill, “Some advances in transformation-based part of speech tagging,” *arXiv preprint cmp-lg/9406010*, 1994.
- [15] M. Johnson, “Pcfg models of linguistic tree representations,” *Computational Linguistics*, vol. 24, no. 4, pp. 613–632, 1998.
- [16] S. M. Shieber, “Sentence disambiguation by a shift-reduce parsing technique,” in *Proceedings of the 21st annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 1983, pp. 113–118.
- [17] A. Helfrich and B. Music, “Design and evaluation of grammar checkers in multiple languages,” in *COLING 2000 Volume 2: The 18th International Conference on Computational Linguistics*, vol. 2, 2000.
- [18] D. Klein and C. D. Manning, “A parsing: fast exact viterbi parse selection,” in *Proceedings of the 2003 Conference of the North American Chapter of the Association*

for Computational Linguistics on Human Language Technology-Volume 1. Association for Computational Linguistics, 2003, pp. 40–47.

- [19] A. Fahda and A. Purwarianti, “A statistical and rule-based spelling and grammar checker for indonesian text,” in *Data and Software Engineering (ICoDSE), 2017 International Conference on.* IEEE, 2017, pp. 1–6.
- [20] N. Ehsan and H. Faili, “Towards grammar checker development for persian language,” in *Proceedings of the 6th International Conference on Natural Language Processing and Knowledge Engineering (NLPKE-2010).* IEEE, 2010, pp. 1–8.