Addis Ababa University

College of Natural Sciences

Development of Automatic Parser for Tigrigna Sentences Using Bottom-Up Probabilistic Chart Parser

Yaynshet Medhin Asefa

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia
Oct, 2017

Addis Ababa University

College of Natural Sciences

Yaynshet Medhin Asefa

Advisor*: Yaregal Assabie (PhD)*

This is to certify that the thesis prepared by *Yaynshet Medhin Asefa*, titled: "*Development of Automatic Parser for Tigrigna Sentences using Bottom Up Probabilistic Chart Parser"* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

	Name	Signature	Date
Advisor:			
Examiner:			
Examiner:			

Dedication

I dedicate this thesis to my mother cheye, who has thought me how to successfully come up with the challenges I faced during the completion of the thesis.

Acknowledgment

There are lots of people without whom this work might not have been completed, and to whom I am greatly grateful. Firstly I am very grateful to my advisor, Yaregal Assabie for his close direction and constructive recommendation during my work. He has been devoting his time and providing ideas to carry out the research. I thank you for your encouragement from the beginning.

I would like to thank to all of my friends, who played such vital roles along the journey, as we equally engaged in encountering various challenges we faced and in providing encouragement to each other at those times when it seemed impossible to continue. I really thank to the linguist Gebrekidan and Tekie. Besides, colleague, researchers, sisters and brothers, who knowingly and unknowingly directed me to an understanding of some of the more obvious challenges to our ability to succeed. Thanks to God the Divine who stays there to make the impossible possible.

Table of contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

ADJ	Adjective
ADV	Adverb
ADVP	Adverb Phrase
ART	Article
ATN	Augmented Transition Network
AUX	Auxiliary
CADJP	Complex Adjective Phrase
CADVP	Complex Adverb Phrase
CARDN	Cardinal followed by a name
CARDPREP	Cardinal with Preposition
CFG	Context Free Grammar
CNF	Chomsky Norm Form
CNP	Complex Noun Phrase
CVP	Complex Verb Phrase
DET	Determiner
DBM	Dependency and Boundary Models
FSM	Finite State Machine
HMM	Hidden Markov Model
HPSG	Head-driven Phrase Structure Grammar
MLC	Most likely Constituent
MT	Machine Translation
N	Noun
NER	Named Entity Recognition
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NLU	Natural Language Understanding
NP	Noun Phrase
NPREP	Nouns with a Preposition
PCFG	Probabilistic Context Free Grammar
POS	Part of Speech
POST	Part of Speech Tagger
PP	Prepositional Phrase
PREP	Preposition
QA	Question and Answering
RTN	Recursive Transition Networks
S	Sentence
SADJP	Simple Adjective Phrase
SADVP	Simple Adverb Phrase

SNP	Simple Noun Phrase
SVP	Simple Verb Phrase
V	Verb
VP	Verb Phrase
VPREP	Verb headed by a Preposition
VREL	Relative verb
WFST	Well Formed Substring Table
WIC	Walta Information Center

ABSTRACT

Automatic parsing is the process of dividing a given sentence to its grammatical structure. Parsing is useful for improving the performance of many NLP applications. There are many research works done on automatic parsing for different languages. The aim of this research work is to design and develop automatic parser for Tigrigna sentences using bottom-up probabilistic chart parser.

We proposed the architecture of the designed system to the identified problem. The architecture has two parts: The learning and parsing. The learning part contains components from which the supervised learning is accomplished. The corpus collected from the different sources is preprocessed by developing simple preprocessing component. The preprocessed sample corpus is manually tagged by two language experts in the language. The tagged corpus is then parsed manually by the linguists. From the parsed sentences Probabilistic Context Free Grammar (PCFGs) are extracted. From the tagged corpus, lexicon was generated using the lexicon generation component. The parsing part contains components which perform the task of parsing given an input sentence such as sentence tokenization, morphological analysis and the PCFG parsing. The first two components make the input sentence suitable to the PCFG chart parsing component.

We then conducted several experiments for both simple and complex Tigrigna sentences. Experimental findings were attained and the solution to the identified problems was addressed and suggested. The experiments were conducted in three parts. The first test was from the training set and the second test was done on test sets from the sample corpora. The third set was different from the two sets which was not from the sample corpora used in the study. The accuracy found on the first test set, second test set and third test set was 95%, 94% and 85%, respectively for the simple Tigrigna sentences. For the complex Tigrigna sentences the result achieved on the three test sets was 91%, 90% and 80%, respectively.

Keywords: Tigrigna; PCFG; Inside algorithm; Viterbi algorithm; PCFG chart parser; bottom-up parsing

Chapter 1: Introduction

1.1 Background

Natural language is a language used by humans for communication purposes and is an important aspect of our lives. In written form, it serves as a continuing record of information from one generation to the next. In oral form, it serves as our primary means of coordinating our daily activities with others in the community. Natural Language Understanding (NLU) is a system of understanding natural language input that deals with machine reading ability (i.e., the ability to read text, process it and understand its meaning). Accordingly, to understand a text in a language, the morphemes within the text need to be identified and divided in to their correct sentence structure or part of speech [1].

Parsing is the process of analyzing a string in a language in an orderly and organized way. In linguistics, to parse is to divide words and phrases into different parts in order to understand relationship and meaning. Syntactic parsing is generally considered as a structural prediction problem. Simply it is nothing but mapping some input of sentences to an output of syntactic representations [2].

Automatic parsing is the process of dividing a sentence given as input to its correct grammatical structure as an output by the system. Its input is text in the form of a sequence of morphemes (i.e., words) and punctuation marks. Its output is a syntactic representation containing information about the relations in which morphemes in the sentence stand to each other and what syntactic functions they fulfill. The syntactic structure of a sentence indicates the way that words in the sentence are related to each other. This structure indicates how the words are grouped together into phrases, what words modify other words, and what words are of central importance in the sentence [3]. In addition, this structure may identify the types of relationships that exist between phrases and can store other information about the particular sentence structure that may be needed for later processing. A. Tayal et al [4] stated that representing a sting in a clear manner is vital for successful understanding of a natural language. Besides, the authors suggested that the most commonly used representation in natural language understanding is the syntactic processing (parsing) which deals with the syntactic structure of a sentence.

The formal representation of natural language processing consists of three components source language, target representation together with a mapping (parsing) process between elements of these components. Generally, automatic parsing is an important task in automatic language understanding and natural language processing applications (e.g., machine translation, question answering, grammar checker, semantic analysis, etc.[5].

Natural Language Processing (NLP) is a field of computer science, Artificial Intelligence (AI), and computational linguistics concerned with the interaction between computers and human (natural) languages. Thus, natural language processing is connected to the area of human-computer interface. The advent of computers has become important for the development of many NLP applications; common NLP tasks include sentence segmentation, Part of speech tagging and parsing, Named Entity Recognition (NER), Machine translation, Co-reference resolution, etc. [4].

Over the past few years, there has been growing interests to develop NLP applications for Tigrigna language. Tigrigna is the official language of Tigray Region of Ethiopia. It is also one of the official languages of Eritrea. A total of 6 million people speak the language and it uses Ethiopic script for writing [6].

Many natural language research works on Tigrigna have been carried out so far, like stemming [7], Part of Speech Tagging (POST) [8], text categorization [9], speech recognition [10], etc. These applications require automatic parsing to improve their performance.

This research work aims to design and develop automatic parser for Tigrigna sentences. In our study, an effort is made on how to develop Tigrigna automatic parser using Probabilistic Context Free Grammar (PCFG). Context-free grammars are widely used robust models of natural language syntax and are suitable for grammar rule induction and play great role in resolving ambiguity in parsing.

1.2 Motivation

Parsing a given input sentence manually is a labor intensive and time taking task. Automatic sentence parsing is important in order to reduce the various costs and to have a model of the syntactic structure of sentences in fast, accurate and consistent way. Moreover, automatic parsing is important for improving the performance of many

Tigrigna Natural Language Processing (NLP) applications like grammar checker, question answering, semantic analysis and machine translation. Thus, the importance of parsing in Tigrigna NLP has motivated us to carry out this research work.

1.3 Statement of the Problem

Automatic parsing is useful for improving the performance of many NLP applications like machine translation, question and answering, semantic analysis, grammar checker, etc. Over the past few years extensive research works have been conducted on automatic parsing and currently there are systems that extract syntactic information from natural language input texts for Arabic language [11, 12]. Abiyot Bayou [13] designed and implemented a word parser for Amharic verbs and their derivation. Besides, Atalech Alemu **[**14] developed an automatic sentence parser for Amharic language. Daniel Gochel [15] integrated previous natural language processing studies on Amharic and used the sum total of the ideas or outputs of these works to fill the gaps identified. Deriba Megersa [16] developed an automatic sentence parser for Afaan Oromo language

However, to our best knowledge, no research work has been done on automatic parsing for Tigrigna sentences and automatic parsing is language specific due to difference in grammatical structure, language characteristics and morphology. This in turn means that the research works done so far need special consideration in the design in order to be applied for Tigrigna language. Thus, automatic parsing should be developed for Tigrigna language by taking its linguistic features into consideration. Therefore, the purpose of this study is to develop an automatic parser for Tigrigna sentences.

1.4 Objectives

General objective

The general objective of this research work is to design and develop an automatic sentence parser for Tigrigna sentences.

Specific objective

In order to achieve or meet the general objective, the research attempts to address the following specific objectives.

- Review related literature in the area of parsing and review the different strategies and approaches used in parsing.
- Review literatures in Tigrigna language.
- Study the basic Ethiopic alphabet and punctuation marks, Tigrigna word categories, word order and grammar to identify properties which are useful for sentence parsing.
- Adapt some existing tools, techniques and approaches in parsing
- Collect sample corpus for the purpose of the study.
- Adapt a parsing algorithm.
- Develop a prototype Tigrigna sentence Parser.
- Evaluate the system.

1.5 Methods

Developing an automatic sentence parser for Tigrigna language which is the general objective of this research work requires one to investigate and identify useful methods. The methods employed to carry out the research are listed below.

Literature Review

A review of literature about Tigrigna language is made. Related works in area of parsing are also reviewed for this study to acquire a deeper understanding of the research area and its problem domains. The different strategies and approaches to automatic parsing are also assessed to identify and point direction in providing solution to identified problems.

Corpus Collection

After assessing the literatures on the research domain, sample corpora to be used for experimentation are collected based on non-probability sampling criteria set for the purpose of this study. The corpora are then analyzed in a way that is suitable for processing.

Design and Development

Design of the overall architecture of the system is proposed and the system is developed. Tools and programming languages that can accomplish the task are stated during the implementation of the prototype system.

Testing

To see the how well the system performs, testing and evaluation is done. To do this some evaluation criteria and techniques were considered. Cross validation is a model of validation used in this study. Finally, the results attained from the experiment were discussed and solution are identified and suggested.

1.6 Scope and Limitations

The scope of this research work is confined to standard Tigrigna language. Dialect languages (i.e., a form of a language that people speak in a particular part of a region, containing some different words and grammar) are not considered in this domain. Since there is lack of large corpora annotated with POS and syntactically parsed in the language, this study uses a small sample corpus for experimentation.

1.7 Application of Results

The output of the current work can be applicable to higher level NLP applications in the language. There are a number of NLP applications that uses parsing techniques such as machine translation, information retrieval and information extraction. Applications of parsing may comprise everything from simple phrase finding e.g., for proper name recognition, to full semantic analysis of text [17]. Tigrigna automatic parser is used to improve the performance of many applications some of them are listed below:-

- **Grammar Checker:** In grammar checking, parsing is used to detect words that fail to follow proper grammar usage. Automatic parsing is required after the grammar checker found each word in a text, look up each word in the dictionary, and then attempt to parse the sentence into a form that matches a grammar rule in the language.
- **Machine Translation:** On a basic level, Machine Translation (MT) performs simple substitution of words in one language (source language) for words in another language (target language). However, that alone usually cannot produce a good translation of a text because recognition of whole phrases and their closest counterparts in the target language is needed. Thus, parsing is used in Machine Translation (MT) to divide the given input in to its correct grammatical structure for ambiguity resolution.

- **Question Answering:** Assigning a question type to the question is crucial task in question answering systems. In question and answering (QA) syntactic parsing techniques can be used to determine the question and answer type.
- **Semantic Analysis:** Syntactic parsing is essential to semantic analysis for relating syntactic structure, from the level of phrases, clauses, sentence and paragraphs to the level of writing as a whole to their language independent meanings.

Besides, this work in parsing Tigrigna language can be applicable for end users of the system who wants to get some basics of Tigrigna language. Moreover, those who are interested in generating the syntactic structure of simple and complex sentences in the language can use it.

1.8 Organization of the Rest of the Thesis

The rest of the thesis is organized as follows. Chapter Two assess literatures in the field and it presents a brief description of Tigrigna language, grammar formalisms, approaches and strategies to parsing. Chapter Three reviews the different related works in automatic parsing. This chapter gives a short and brief review of the papers related to the current study. Chapter Four presents the overall architecture of the designed system along with some general overview of the interconnection between various components of the framework. In Chapter Five we present the implementations, experimentations and experimental results obtained. Chapter Six provides a conclusion of the research work along with future works.

Chapter 2: Literature Review

2.1 Introduction

The goal of this research work is to design and develop an automatic parser for Tigrigna sentences. Thus, it is significant to point out and review Tigrigna language with respect to its grammar, rules of formation and the approaches and strategies that are dedicated to automatic parsing. Thus, this chapter assesses literature in the field and gives a general background concerning the area of study as well as some background information about the tools, strategies and approaches to be used in order to give us the fundamental concepts needed.

2.2 Tigrigna Language

Tigrigna is the official language of Tigray region of Ethiopia. It is also one of the official languages of Eritrea. A total of 6 million people speak the language and it uses Ethiopic Alphabet for writing. It is the third most spoken language in Ethiopia and it is the member of the Ethio-Semitic languages which belongs to Afro-Asiatic super family [6, 18]. Due to its morphological complexity, Tigrigna exhibits the root and stem pattern morphological phenomenon [7, 19]. As the morphological variation is resulted through adding affixes to the root verbs or nouns to indicate number, gender, tense, possession, etc., it is essential to recognize the behavior of Tigrigna language stems and roots. As stated in [7, 20], a root is a verb that indicates a third person singular masculine, such as በልዐ/he eat ፣ ሰተየ/he drink ፣ ሰርሐ/he work, etc. or the noun that expresses a singular noun whereas a stem is a verb in which its ending letter is 'sads' (6^{th} order) or a noun that indicates a singular number. Thus, a stem may or may not have a meaning if the word is a verb [20].

Tigrigna language is one of the Ethiopian languages which are written in Ethiopic (Ge'ez) script. The writing is done from left to right and has no lower and upper case letters. The writing system has syllabic letters, numerals and punctuation. The punctuation and numerals are almost similar to the foreign writing practices. In this study, we have been using both Latin and Ethiopic alphabet interchangeably in order to give some examples of sentences or words in the language. A list of the Ethiopic alphabets and punctuation marks is found in Appendix A and B which is taken from the authors in [18].

2.3 Tigrigna Grammar

2.3.1 Tigrigna Word Classes

Kassa Gebrehiwet [21] grouped Tigrigna parts of speech into eight. These are nouns, adjectives, verbs, adverbs, pronouns, prepositions, conjunctions and interjection (exclamation). These parts of speeches cannot stand alone by themselves. Since pronouns have almost the same role and behavior as nouns it can be seen by concatenating to nouns. Besides conjunctions have the same role and behavior as prepositions due to this reason it is concatenated under preposition. In addition exclamation is used only to express feelings and cannot stand alone as a part of speech. So the only parts of speech that can stand alone as a word class are nouns, adjective, verb, adverb, prepositions [22].

The above five word classes are also categorized in to two as higher and lower classes. The higher classes are noun, adjective, verb and the lower classes are the rest two which are adverb and prepositions. The higher classes are huge in number. Their number increases from time to time and they can be pluralized by adding suffix and can also create new words. On the other hand the lower classes do not exhibit the properties of the higher classes and small in number and their number cannot increase from time to time or cannot take inflectional morphology in order to be pluralized [20].

A brief description of Tigrigna word classes according to Daniel Teklu [20], Kassa Gebrehiwet [21], J. Mason [23] and language expert understanding in the language are reviewed as follows.

Nouns

Noun is one of the higher classes of part of speeches which is used to represent for person, animal, place, action, situation, phenomena and others. Tigrigna nouns are either masculine or feminine and are inflected for number. Tigrigna nouns have plural as well as singular forms, though the plural is not obligatory when the linguistic or pragmatic context makes the number clear. Tigrigna noun plurals may be formed through internal changes ("broken" plural) as well as through the addition of suffixes. For example, ፈረስ/horse, ኣፍራስ /horses.

Like in English a small change is made to denote a plural in Tigrigna, a lot of nouns just need ታት/es/s as a suffix e.g., ክባቢ /region - ክባቢታት/ regions, however, here are some additional rules:

1. If the noun ends in the 3rd order (ጻዕሪ /effort), then the last letter should be changed to the 6th order and "ታት" should be added (ጻዕርታት /efforts).

2. If the noun ends in the 6th order (ፊደል/ letter), then the last letter should be changed to the 4th order and "ት" should be added (ፊደላት/ alphabet).

Many researchers for example Baye Yimam [22] distinguished Tigrigna language with other languages and identified points that made Tigrigna language different. One of the points is Tigrigna language has irregular system of pluralization of nouns. Hence, as in English there are many nouns which form the plural irregularly with the noun gaining/losing a prefix/suffix, and/or small changes within the noun itself.

Adjectives

Adjective is one of the higher classes of word classes and takes the properties of these classes. Since one can form adjectives in many different ways adjectives are huge in number and can take inflectional morphology. They usually precede the nouns that they modify or describe.

Here are examples: ፅቡቕ ከዳን/ nice cloth, ሓፃር ገዛ/ Short house

Verbs

Verbs are based on consonantal roots most consisting of three consonances for example ሰብር/break, ሰበረ/he broke, ይሰብር /he breaks, ምስባር /to break. A verb is one of the higher classes of words which explain the situation or phenomena. One complete sentence cannot stand alone without a verb and it cannot give a complete thought.

Tigrigna has unusual complex tense system, with many nuances achieved using combinations of the three basic tense forms (perfect, imperfect, gerundive) and various auxiliary verbs including the copula እዩ/is, the verb of existence ኣሎ/allo, and the verbs ነበረ/exist, live, ኮነ/become, ጸንሐ/stay.

The verb ኣሎ/there is to be used when you are describing the location a person or object. This is different from the verb እዩ/is which is used for describing a person or object. In English both

meanings are conveyed with the single verb 'to be' while in Tigrigna there are two separate verbs for these two separate meanings. This verb is divided by person, number and gender.

Adverbs

In short explanation, an adverb decides and describes the verb and is part of the word class which can stand alone, however, as everything that describes a noun is not an adjective, the same thing is true for adverbs that is everything that describes a verb is not an adverb. Here is an example which is taken from the reference book.

ወለገሪማ ትማሊ ናብ ሃገረሰላም ከይዱ።።
Weldegerima has gone to hagereselam yesterday.

The underlined word ትማሊ/yesterday describes to the word ከይዱ/gone in terms of time.

Prepositions

Words which are categorized under prepositions are small in number. This is one criterion that we differentiate prepositions and the number of words on this class cannot increase from time to time. This means they cannot be derived or inflected.

Words categorized under preposition can be single or double word. Prepositions like ብ/by and ን/to can be categorized as single word and prepositions like ናብ/to, ካብ/from, ምስ/with, ከም/as and others are categorized under double word preposition. However, there are also prepositions which have words above two.

Pronouns

A pronoun is a word which is regarded as a subclass of a noun. It can be taken as noun because it can function as a noun and can take the position of a noun. Examples of Tigrigna pronouns are: ንሱ/He, ኣነ/I ንስኻ/you (masculine), ንስኺ/you (feminine), ንሳ/she, etc.

Conjunctions

Conjunctions are words that link words, phrases and clauses to make larger grammatical units. However, conjunctions are treated also as a subclass of prepositions.

Examples: ሰብኣይ-ን ሰበይት-ን/ሰብኣይ/husband ን/and ሰበይት/wife

Determiner

Determiners are the constructions used as demonstratives to show definiteness. Besides, they are called demonstrative pronouns.

Examples: እቲ ቆልዓ/the boy እታ ሰበይቲ/the woman

In the above examples the words እቲ/the and እታ/the are determiners

Interjections

Interjections are words that are used to show short sudden expression of emotions, surprise, annoyance, often unexpected, emotions while saying or suggesting something. Tigrigna interjections can stand-alone by themselves outside a sentence or can appear anywhere in a sentence.

Examples: እሰይ/ooh ወይ ኣነ/oops

Numerals

Numerals are words representing numbers and can be classified as ordinal numbers and cardinal numbers. The cardinal numbers are numbers like ሓደ/one, ክልተ/two and the corresponding ordinal numbers are ቀዳማይ/first, ካልኣይ/second, መበል ዓስራይ/tenth, etc. Besides, there are special numerals in Tigrigna like ፍርቂ/half, ርብዒ/quarter and ሲሶ /one third.

2.3.2 Tigrigna Phrases

A phrase is part of a sentence in a language, which is constructed from one or more words in the language. This construct does not give full information. The one thing that should be clear is that it doesn't mean any unstructured collection of words can be a phrase. Size of words can be seen relatively by the number of words they are built. Thus, some phrases are short by the number of words they contain and others can be long based on the number of words they contain.

From the previously stated five basic word classes one can decide the types of phrases that can exist. After discussions with the language experts and according to the Tigrigna grammar book of Daniel Teklu [20] the following types of phrases are suggested and reviewed. Noun Phrase (NP), Verb Phrase (VP), Adjective Phrase (ADJP), Adverbial Phrase (ADVP), Prepositional Phrase (PP). These phrases are investigated one by one with their unique behavior relatively.

Noun Phrase

A noun phrase is phrase proceeded by the word class noun. In noun phrase the noun comes in the right side as a head. Noun phrase is phrase which is right headed by noun. Any sentence is first divided into two large phrases namely, Noun Phrase (NP) and Verb Phrase (VP). Verb phrase is explained in the next section. Thus, Noun Phrase (NP) is taken as one large part of a sentence and it can be found as a verb in two ways one is as noun head alone with no other structure, and with other structural components.

A noun phrase built with noun head only

‘ለምለም ናብ መቐለ ከይዳ/Lemlem has gone to mekelle’

This sentence contains two phrases ‘ለምለም/lemlem’ is the noun phrase of the sentence headed by a noun ‘ለምለም/lemlem’ and ‘ናብ መቐለ ከይዳ/has gone to mekelle’ is verb phrase of the sentence. ለምለም/lemlem as a noun phrase is built by noun head only but it can add other structures.

For example: consider the grammar S$\rightarrow$ NP VP

NP$\rightarrow$N

NP$\rightarrow$N PREP

In the above grammar a noun phrase can be built with noun only (NP $\rightarrow$N) or it can be built from a noun and a preposition (NP$\rightarrow$N PREP).

The tree structure of the NP used as an example above is demonstrated in Figure 2.1, where CNP stands for Complex Noun Phrase and SNP stands for Simple Noun Phrase.

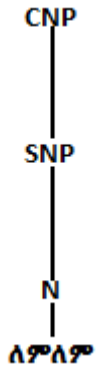

Figure 2. 1 The Structure of Noun Phrase

Here is an illustration of Noun Phrase (NP) built by a head and other structural components

‘እታ ነዋሕ ለምለም ናብ መቐለ ከይዳ/The tall girl lemlem has gone to mekelle’

In this sentence እታ ነዋሕ ለምለም/the tall girl lemlem is a noun phrase (NP) and the head noun is ለምለም/ lemlem and this noun is not alone it has two other structures እታ ነዋሕ/the tall.

Verb Phrase

The phrase that is formed by heading a verb is called Verb Phrase (VP). In verb phrase the head is a verb. Verbs can be classified based on their role, as verbs of movement, verbs to be, verbs of action, verbs of perception.

Verbs to be in general express the status of something. Besides they express the present condition of something as become, be and seem are one of the verbs in this category.
Here is an example:

‘ለምለም [የመና ነዋሕ ኮይና]/Lemlem became very tall’.

ኮይና/became is a verb to be and take an adjective phrase የመና ነዋሕ/very tall. Verb to be are inflected to number, gender and person. The structure to the above sentence is illustrated in Figure 2.2, where CVP stands for Complex Verb Phrase, SVP stands for Simple Verb Phrase, CADJP stands for Complex Adjective Phrase and SADJP stands for Simple Adjective Phrase.

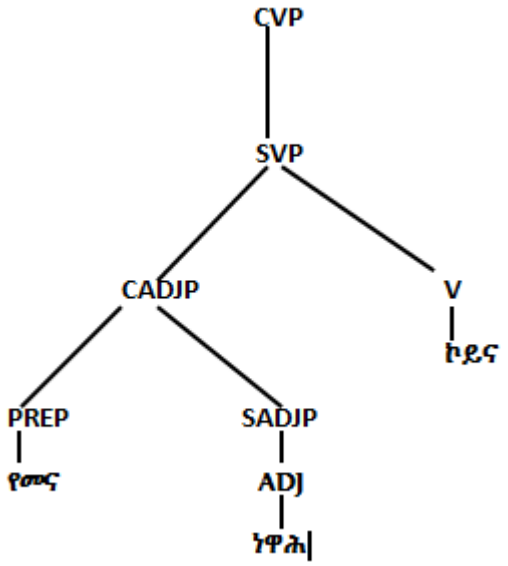

Figure 2.2 The Structure of Verb Phrase

Adjective Phrase

An Adjective Phrase (ADJP) is a phrase whose head is an adjective word class. An adjective has the property of expressing the noun. Thus, most of the time an adjective comes before the noun to express it. Below is an example of adjective phrase.

ቀራን ላሕሚ/Horned cow.

The word ቀራን/horned is an adjective and this adjective is used to express the noun ላሕሚ/cow and it is giving additional information about the noun. So the hierarchical structure looks like the tree structure shown in Figure 2.3, where CADJP stands for Complex Adjective Phrase and SADJP stands for Simple Adjective Phrase.

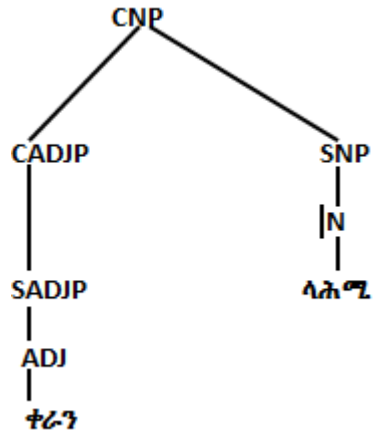

Figure 2. 3 The Structure of Adjective Phrase

Adverbial Phrase

Adverbial phrase is a phrase headed by a word class adverb. As shown previously in Section 2.3.1 an adverb is a word class which gives additional information for actions presented by the verb. In the class of phrases the verb phrase decides the head by giving additional explanation for the action. Thus, adverbial phrase gives information like time, reason of happening for the action on the verb. Since Tigrigna phrases are small in number, the phrases headed by these words are also small relatively. Thus, most of the actions of adverb phrases are expressed by sentence, noun phrase and prepositional phrase. Here is an example of simple adverb phrase.

ʻሓልሓሊፉ ዝናበ ካፉዩ/ it rained rarely.

As any other sentence, the above sentence has a NP and VP. In the VP there is an adverb phrase (ADVP) [ሓልሓሊፉ/rarely]. This phrase expresses the head verb [ካፉዩ/rains]. Before drawing the tree structure let's change the surface structure into deep structure as follows.
ዝናብ ሓልሓሊፉ ካፉዩ።/Rain rarely rained.

The tree representation of the above example is drawn in Figure 2.4, where CADVP stands for Complex Adverb Phrase, SADVP stands for Simple Adverb Phrase and ADVP stands for Adverb Phrase.

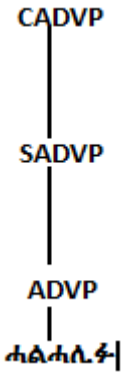

Figure 2. 4 The Structure of Adverbial Phrase

The head verbs in adverbial phrase don not take modifier and complements. In forming Tigrigna phrases there are two main points that should be remembered. These are phrasal identity and phrasal role concepts. A phrase can be different in identity and role.
Example:'ብተበተበ ዝተስርሐ ገዛ/a house built hastily'.

In this noun phrase the phrase: ብተበተበ/ hastily is Prepositional Phrase (PP) in its identity but in its role it is Adverb Phrase (ADVP). It is covering the place of the adverb phrase.

Prepositional Phrase

Prepositional Phrase (PP) is a phrase headed by a preposition and it is different from other phrases in two ways. First, apart from the previously seen four phrases the head alone cannot produce a phrase. This means Prepositional Phrase (PP) cannot stand alone without any modifier or complement it needs a complement with the head phrase. Second, apart from other phrases this phrase left headed. The head preposition operates being on the left side. Here are examples of Tigrigna prepositions.

‘ናብ ርባ/to river’

‘ካብ ገዛ/from home’

In the above listed examples the words ናብ/to, ካብ/from are prepositions. These prepositions create prepositional phrase by adding the nouns ርባ/river and ገዛ/home. The parse tree structure of the prepositional phrase used above is demonstrated in Figure 2.5, where CPP stands for Complex Prepositional Phrase, SPP stands for Simple Prepositional Phrase.

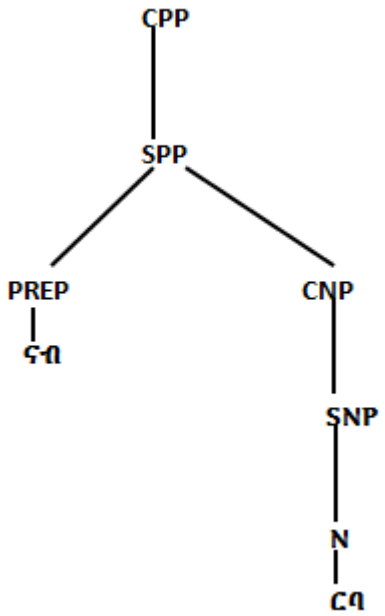

Figure 2. 5 The Structure of Prepositional Phrase

2.3.3 Tigrigna Sentence

This section discusses how the previously seen phrases come together to form full sentence to give meaningful information. According to Daniel Teklu [20] Tigrigna sentences are categorized into five main categories. But here we are going to concern only on the two types of sentences namely simple and complex sentences.

Simple sentence is a sentence which is composed of single Noun (N) and single Verb (V) that gives full information but it doesn’t mean it couldn’t contain the complements and modifiers in the verb. A complex sentence is a sentence which is composed of two or more nouns and verbs. Here is an example of simple sentence

‘እቲ ተምሃራይ ናብ ቤት ትምህርቲ ከይዱ/The student has gone to school’.

The example above is a simple sentence in that it is composed of a noun ተምሃራይ/student and a verb ከይዱ/gone. The tree structure of the sentence is shown in Figure 2.6 taken from Daniel Teklu [20].

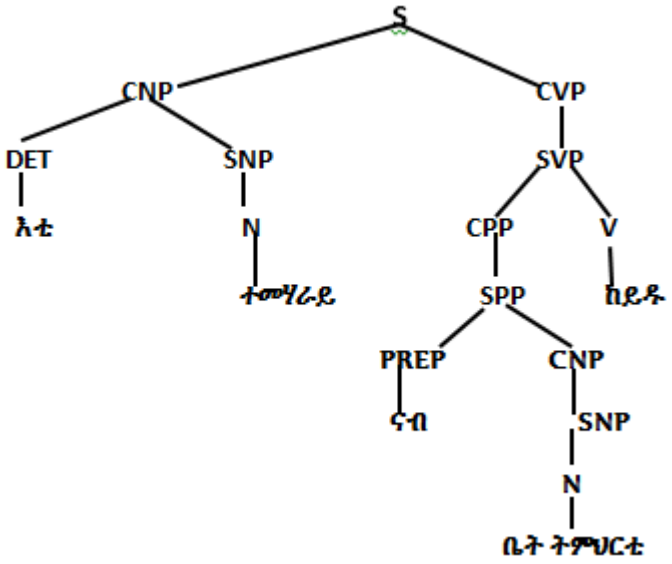

Figure 2. 6 The Structure of Simple Sentence

In linguistics, word order is the study of the order of the syntactic constituents of a language. A verb is a very simple idea to describe and it is an action word. An example of a verb in a sentence would be: ጆን ኑቲ ቑልፊ ጨቢጥዎ/*John grabbed the keys*, in which case ጆን/ john is subject, the one ‘doing something’. ኑቲ ቑልፊ/the keys is the object, the one that is grabbed. In English, the verb ጨቢጥዎ/grabbed always goes after the subject ጆን/John. Thus, the word order becomes Subject-Verb Object, abbreviated as SVO using the first letter of each word. This is different in Tigrigna, where the verb ጨቢጥዎ/grabbed always goes after the object ኑቲ ቑልፊ/the keys. Thus, the word order becomes: Subject-Object-Verb, abbreviated as SOV using the first letter of each word. All verbs go at the end of a statement unlike in English where verbs are placed in the middle of a statement ([Subject] [Verb] [Object]), Tigrigna places its verb at the end ([Subject] [Object] [Verb]) and Tigrigna verbs are conjugated to agree with the subject in person, number, and gender.

2.4 Grammar Formalisms

A parser need to contain a grammar that specifies how sentences are built from their parts, and how the information related to the sentence derives from the information associated to its fragments. When the parser accepts an input word or sentence in one language it produces a summary description of possible structural relations that may hold between words or sequences of words in the language, and produces structural descriptions of the input sentence or word as permitted by the grammatical rules. In few parsers the production of a meaning representation is carried out in parallel with the derivation of a structural analysis according to the grammar. But it is believed that grammatical structure contributes to construction of meaning and that discovering the grammatical structure of a natural language word sequence is a significant or basic step in determining the meaning of the sentence. But many sentences in any language even Tigrigna are ambiguous, having more than one meaning. A grammar formalism which greatly handles such ambiguity is important in any natural language processing. The ambiguity can be handled in a probabilistic framework that can tell us not only which analyses are possible, but how much more likely one is than another [24].

There are diverse grammatical formalisms proposed so far as these are context free grammar [25], transformational grammar [26], transition network grammars [27], unification based grammar [28] and probabilistic context free grammars [29]. Each of these formalisms is discussed one by one in detail in the following sub sections.

2.4.1 Context Free Grammar Rule (CFG)

A Context Free Grammar (CFG) is a set of production rules used to produce patterns of strings. A CFG consists of components such as a set of terminal symbols, which are the characters of the alphabet that appear in the strings generated by the grammar, a set of non-terminal symbols, which are placeholders for patterns of terminal symbols that can be generated by the non-terminal symbols, a set of productions, which are rules for replacing non terminal symbols (on the left side of the production) in a string with other non-terminal or terminal symbols (on the right side of the production) and a start symbol, which is a special non terminal symbol that appears in the initial string generated by the grammar.

CFGs are a very vital class of grammars as the formalism is influential enough to explain most of the structure in natural languages and yet it is restricted enough so that efficient parsers can be built to analyze sentences. In constructing a grammar for a language you are interested in generality, the range of sentences the grammar analyzes correctly, the range of non-sentences it identifies as problematic and the simplicity of the grammar itself. However, as you attempt to extend a grammar to cover a wide range of sentences you often find that one analysis is easily extendable while the other requires complex modification. The analysis that retains its simplicity and generality as it is extended is more desirable [3].

Table 2.1 shows sample CFGs for the two types of sentences.

Table 2. 1 Sample CFGs

CFG for the 110 Simple Sentence	CFG for the Complex 100 Sentence
S→NP VP	S→NP VP
S→PP VP	S→PP VP
NP→ADJ N	PP→CARDPREP N
VP→PP V	PP→PREP ADJ
PP→PREP N	NP→N
PP→NPREP N	NP→V N
ADJP→ADV ADJ	VP→NP V

2.4.2 Transition Network Grammar

Another formalism that is useful in a wide range of applications is transition network grammar which is based on the notion of a transition network consisting of nodes and labeled arcs. One of the nodes is specified as the initial state, or start state. Simple transition networks are often called Finite State Machines (FSMs). Finite state machines are equivalent in expressive power to regular grammar and thus are not powerful enough to describe all languages that can be described by a CFG. To get the descriptive power of CFGs, you need a notion of recursion in the network grammar [3].

Recursive Transition Network

A Recursive Transition Network (RTN) is like a simple transition network, except that it allows arc labels to refer to other networks as well as word categories. RTN systems

integrate some additional arc types that are useful but not formally necessary. A recursive transition network is a directed graph with labeled states and arcs, a distinguished state called the start state, and a distinguished set of states called final states. It looks essentially like a nondeterministic finite state transition diagram except that the labels on the arcs may be state names as well as terminal symbols [27].

Augmented Transition Network Grammars

ATNs build on the idea of using finite state machines (Markov model) to parse sentences. According to adding a recursive mechanism to a finite state model, parsing can be achieved much more efficiently. Instead of building an automaton for a particular sentence, a collection of transition graphs are built. A grammatically correct sentence is parsed by reaching a final state in any state graph. Transitions between these graphs are simply subroutine calls from one state to any initial state on any graph in the network. A sentence is determined to be grammatically correct if a final state is reached by the last word in the sentence [27].

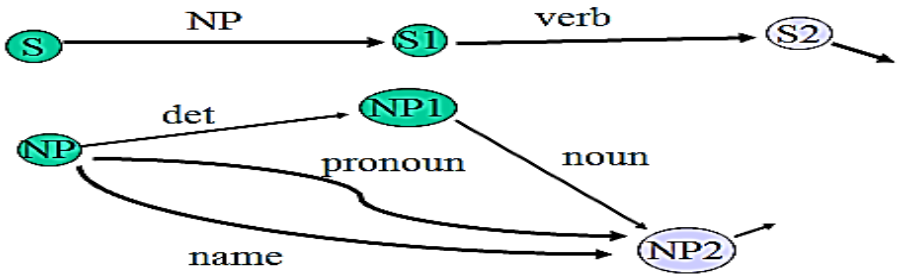

Figure 2. 7 Augmented Transition Network

2.4.3 Context Sensitive Grammar

A Context Sensitive Grammar (CSG) is a formal grammar in which the left-hand sides and right-hand sides of any production rules may be surrounded by a context of terminal and non-terminal symbols. Context-sensitive grammars are more general than context-free grammars, in the sense that there are languages that can be described by CSG but not by context-free grammars. Context-sensitive grammars are more powerful than CFGs though the former kinds of grammars are much harder to work with than the latter [30].

2.4.4 Unification Based Grammar

Unification-based grammars is the grammar formalism that makes extensive use of feature structures (such as case, gender, tense) including their values represented in the lexical

entries of words. These characteristic structures are manipulated by the operation of unification by specifying the entire grammar as a set of constraints between feature structures. The other rest grammar formalisms discussed earlier can provide as the support or strengthen for the unification-based grammars, CFGs being the most common. As stated in [31] the main reason for the excessive power of the unification-based grammars is that recursion can be encoded in the feature structures.

2.4.5 Probabilistic Context Free Grammar (PCFG)

A Probabilistic Context Free Grammars (PCFGs) is nothing but Context Free Grammar (CFG) with associated probabilities. The probabilistic version of CFG is determined by CFG rules and statistical properties of the rules. As grammars get larger and larger they become more and more ambiguous. In such cases, a PCFG give a suggestion of the most probable parse. It can also be programmed to give the n most probable structures, which can be useful when sentences are ambiguous to the reader. Compared with the CFG, the PCFG is especially useful for sentence parsing and they have been used in a variety of applications like for the selection of the most probable parse of an ambiguous sentence [32], in selecting the best rule to do the task of parsing [33], In pattern recognition [34], audio modeling [35].

PCFGs are defined formally as follows PCFG = <W, N, S, R>

Where W is set of words which are terminals $w_1 \ldots w_n$, N stands for set of non-terminals $N_1 \ldots N_n$, S is the start symbol N_1 and R stands for rule probability.

As stated in [36], context-free grammars can be generalized to the probabilistic case by having some statistics on rule use. This is done by counting the number of times each rule is used in a corpus containing parsed sentences and use this to estimate the probability of each rule being used, for instance, consider a category C, where the grammar contains m rules R, with the left-hand side C. You could estimate the probability of using rule R to derive C by (1) [3].

$$\text{PROB}(R_j \mid C) = \text{Count}(\#\text{ times } R_j \text{ used}) \Big/ \text{sum } i = 1, m(\#\text{ times } R_j \text{ used}) \quad (1)$$

where C is a category and R is a rule.

For the parameter: q (a → ß). For each rule a → ß ∈ R. The parameter q(a → ß) can be interpreted as the conditional probability of choosing rule a → ß given that the non-terminal being expanded is a. For any X ∈ N, where N is element of non-terminal we have the constraint shown in (2) [37].

$$\sum_{\alpha \to \beta \epsilon R: \alpha = X} q(\alpha \to \beta) = 1 \quad (2)$$

In addition we have q (a → ß) ≥ 0 for any a → ß ∈ R.

Given a parse-tree t containing rulesα1 → β1, α2 → β2…… αn → βn the probability of t under the PCFG is given in (3).

$$P(t) = \prod_{i=1}^{n} q(\alpha i \to \beta i) \quad (3)$$

Table 2.2 demonstrates an example of simple probabilistic context free grammar. This example is from the sample corpora to demonstrate how the PCFG works. See Appendix G for the full PCFG extracted from the sample sentences for both simple and complex sentences.

Table 2. 2 Sample showing how PCFGs are extracted

Rule	Count for Rule	Count for LHS	Probability
S→NPVP	69	100	0.69
S→ PP VP	31	100	0.31
NP→DET N	200	225	0.889
VP→NP AUX	50	50	1.0
NP→PP V	25	225	0.111
PP→ PREP N	30	30	1.0

The technique involves making certain independence assumptions about rule use. In particular you must assume that the probability of a constituent being derived by a rule R is independent of how the constituent is used as a sub constituent with this assumption, a formalism can be developed based on the probability that a constituent C generates a sequence of wordsW_i, W_i+1,...,W_jThis type of probability is called the inside probability because it assigns a probability to the word sequence inside the constituent. It is written as

PROB ($W_{i,j}$| C). Let's assume that the number of nouns (count (N)) in the given corpus is 10, and among these 5 of them goes to the nouns 'Wetaderat', count (N→ Wetaderat') = 5. Thus, the probability of wetaderat goes to noun, P (N -> 'wetaderat') is given by the formula below.

$$P\ (N \text{ -> 'wetaderat'}) = \frac{\text{count (N} \text{->Wetaderat)}}{count(N)} = {}^{5}/_{10} = 0.5$$

Where N is the word category for noun, the word 'wetaderat/soldiers' is under word category N (noun).

Thus, as stated by V. Alfred [38] PCFG gives a better probabilistic model for syntax analysis for statistical properties of natural languages. PCFGs give clue about what constitutes a likely parse of a string based on evidence from corpora, are good for grammar induction, are robust to grammatical mistakes and have better predictive power than HMM.

Here we are going to illustrate an example showing how the probability of a tree is calculated, assuming the simple PCFG rules shown in Table 2.3. In order to find the probability of a tree in a PCFG model, one just multiplies the probabilities of the rules that built its local sub trees. E.g., the Tigrigna sentence:

ኣብ ወረዳ ወልቃይት ጽርየት ትምህርቲ ተገሊጹ። /^ab wereda Welqayt ^Sryet tmhrti tegeli^Su.

According to the grammar in Table 2.3, nonterminal nodes in the trees have been subscripted with the probability of the local tree that they head. Using these conditions we can justify the calculation of the probability of a tree in terms of just multiplying probabilities attached to rules.

Table 2. 3 A Simple PCFG and lexical rules

PCFG	Lexical Rules
S→ PP VP [1.0] PP→PREP N [1.0] NP→N [1.0] VP→NP VP [0.3] VP→N V [0.7]	N→ ^Wereda [0.25] N→ Welqayt[0.25] N→ ^Sryet [0.25] N→ Tmhrti [0.25] V → glS[1.0] PREP→ ^ab [1.0]

The sample sentence used above has the parse tree as show in Figure 4.3 and its probability is calculated as follows.

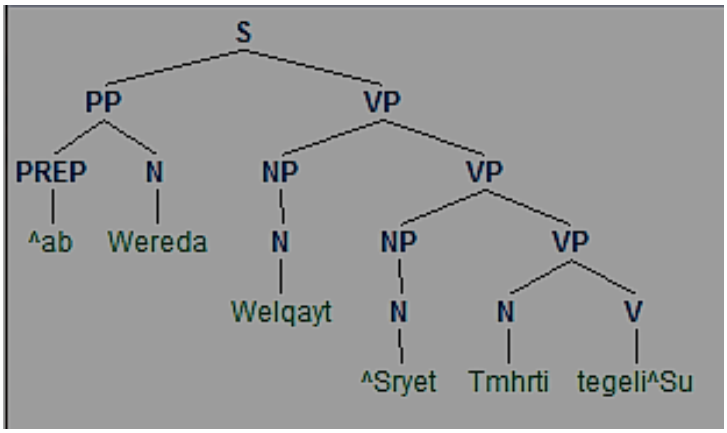

Figure 2. 8 Parse Tree Structure

2.5 Parsing

Parsing refers to the process dividing a given sentences into part of speech i.e., noun, adjective, verb, preposition, etc., to each word in a given sentence of a particular language, and it can also group the words into phrases. Besides it is a derivation process which identifies the structure of sentences using a given grammar rule. Even so, in any natural language processing the task of parsing may be taken place at a word or sentence level. Word level parsing is a system of tokenizing a word into its smallest bit of language that has its own meaning which is known as morpheme [15].

Before going any further to convey how sentence parsing really works by taking some sample Tigrigna text let's see how the parser works in general. While an input string (or

word) is given as input to the parser token by token after tokenized by the sentence tokenization, then the parser splits the tokens and finds each token in the grammar and if the tokens exist then the parser parses them based on the productions rules derived. Morphological Analyzer (MA) can be used to divide the words into roots and affixes according to the morphological rules of Tigrigna language. The lexicon can be generated using lexicon generation after the morphological analyzer returns the root of the words. The parse of the tokens depends on the word class for each word in the result of Part of Speech Tagging. Accordingly, a parse tree is built for each input sentence given to it. The parse tree is nothing but a graphical means to illustrate a derivation of the input sentence or string from a grammar and it records the rules of the language and how they are matched. Every node of a parse tree corresponds either to an input word or to a non-terminal in the grammar [3].

Sentence parsing which is very popular as syntactic parsing is a system that explores different ways of combining grammatical rules to find a combination that generates a tree that could represent the structure of the input sentence. In other words, it is a task in NLP in which a flat input sentence is converted into a hierarchical structure that corresponds to the units of meaning in the sentence [3].

Here is sample Tigrigna sentence taken from the corpus.

ፕረዚደንት ኢራን ምስ ፕረዚደንት ኣፍጋኒስታን ተራኺቦም/prEzdant ^iran ms prEzdant ^afganistan teraKibom/president Iran met with president Afghanistan'.

S((NP(ADJ prEzdant)(NP(N ^iran)))
VP((NP(CONJ ms)(ADJ prEzdant)(VP(N ^afganistan)(V teraKibom))))

For the purpose automatic parsing there are lots of tools used in order to accomplish the task of parsing. These tools and necessary components are discussed below.

2.5.1 Tools for Parsing

Here are some of the most necessary tools that are at least required to do the work of parsing.

Lexicon

Languages have a set of vocabulary items called lexicon and before a grammar is specified one must define the lexicon. The lexicon must contain information about all the different words that can be used, including all the relevant feature value restrictions. When a word is ambiguous, it may be described by multiple entries in the lexicon, one for each different use.

Lexicon is used to efficiently store the possible categories for each word in a language. Items in the lexicon are called lexemes or lexical items. With a lexicon specified, a grammar need not contain any lexical rules. Rather than having a separate rule to indicate the possible syntactic categories for each word, a structure called the lexicon is used to efficiently store the possible categories for each word [3]. Besides it is a list of words that associates each word with its syntactic properties as the category such as, whether the word is a verb, a noun, an adjective, etc., subcategory of the word (such as, whether a particular verb is transitive or intransitive), other syntactic properties (such as, the gender of a noun), and perhaps also morphological and semantic information. Thus, it contains lexical rules, the part of speech category of the lexeme on the left hand side and the lexeme on the right hand side separated by an arrow. In this study the lexical information doesn't contain the subcategory of the word. Sample lexical items which are taken from the study with their attributes are shown in Figure 2.9.

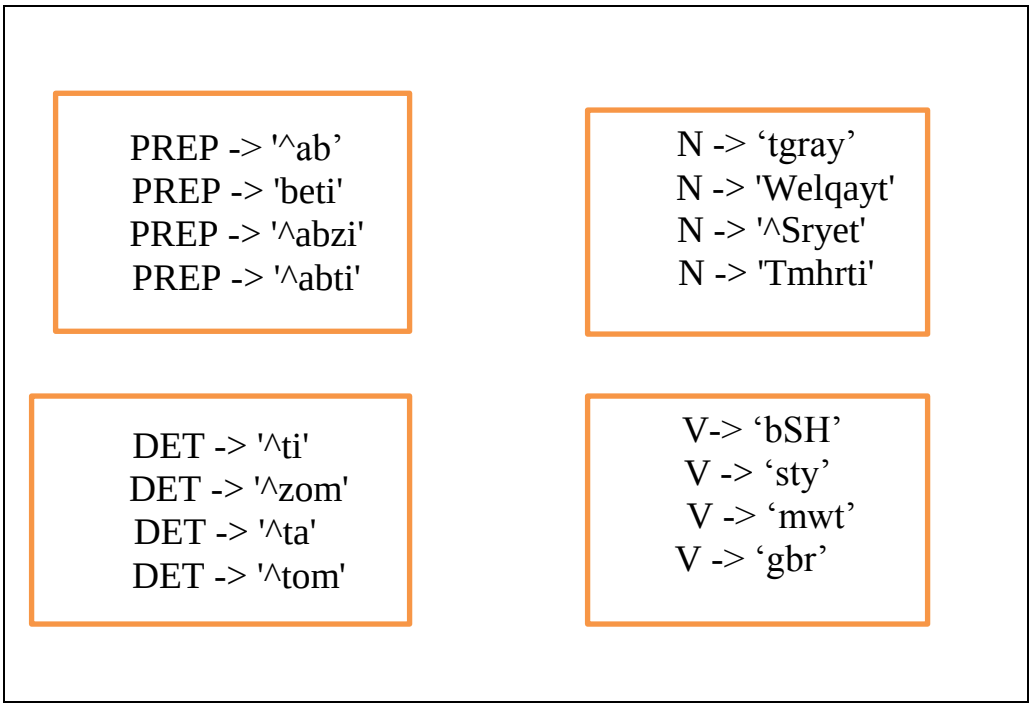

Figure 2. 9 Sample Lexical Items

Grammar

Grammar is set of rules that govern how words are concatenated together to form a meaningful sentence. Any languages have lexicon and a grammar describing how such items may be combined to form larger items and for a system to be able to correctly process a

language it must obey precise lexical and syntactic rule. A grammar can be regarded as a device that enumerates the sentences of a language. Putting it in another way a grammar is a finite list of rules defining a language [38]. A Context Free Grammar (CFG) is derived for both simple and complex Tigrigna sentence separately. Hence, the grammar rule should represent all of these sentences. Figure 2.10 shows the types of sentences that the grammar rules represents.

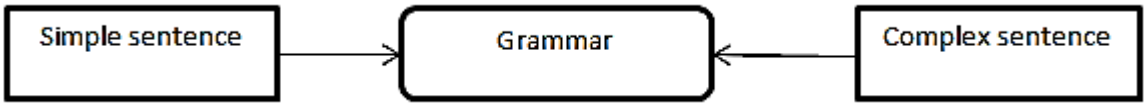

Figure 2.10 Sentences Recognized by the Grammar

2.5.2 Approaches to Sentence Parsing

A distinction is frequently made among different methods of syntax analysis proposed before and based on their parsing and reasoning mechanism, these attempts can be generally classified into two categories namely, rule-based and stochastic approach associated with statistical probabilities.

Rule Based Approach

The Rule-based approach to automatic sentence parsing is entirely based on the information from the knowledge base and some kind of learning technique (if any) to handle ambiguity and guess unknown words. A rule based approach system learns a set of rules automatically based on a given list of tokens and then parses sentences accordingly from these rules. In this approach, the task of parsing can be done in two different ways namely top-down and bottom-up parsing strategies [3]

Stochastic Approach

Stochastic approaches use probability (i.e., statistics) in evaluating the parsing problem of a given sentence. It is based on the ideas of Bayes network theorem, independent events and the Markov assumption in process of parsing a given string. Thus, this approach uses these ideas to determine the most likely lexical sequence of each word in a given sentence. Based on the type of statistics it uses, stochastic approach is further divided in to two sub approaches (i.e., supervised and unsupervised) [14].

Supervised: The supervised statistical parsing provides learning of parsers from tree-banks of parse trees or annotated text provided by human linguists. In supervised stochastic approach there are two core components to be considered, the lexicon and list of contextual probability which provides contextual information and indicates the particular lexical category that is appropriate for a particular context. One of the problems in developing supervised parsers is that there is lack of annotated corpora in the language. But with the presence of pre-tagged corpora supervised statistical parsers can be adopted in a language easily [3].

Unsupervised: In unsupervised stochastic approach there is no need of any pre-tagged corpora in the training process. Besides, they do not require pre-processed corpus in the learning processes. And they use Baum-Welch algorithm other than the Viterbi algorithm used by supervised approaches. But both sub approaches involve estimation of the accuracy of the parsers to improve performance and acquire improved results. In both sub approaches disambiguation can be attained either by using statistical, hybrid or rule based approaches [16]. The following statistical approaches are reviewed.

a. Probabilistic Context Free Grammar (PCFG) Parsing

The probabilities of particular parse trees can be found using a standard chart parsing algorithm, where the probability of each constituent is computed from the probability of its sub constituents and the probability of the rule used. Particularly, when entering an entry E of category C using a rule i with n sub constituents corresponding to entries **E**, then its probability is given as in (4).

$$\text{PROB}(E) = \text{PROB}(\text{Rule}_i|C) * \text{PROB}(E_{11} \ldots E_n) * \text{PROB}(E_n) \quad (4)$$

where E is an entry and C is a category. Unfortunately, a parser built using these techniques turn out not to work as well as you might expect, although it does help. Several researchers have found that these techniques identify the correct parse about 50 percent of the time. It doesn't do better because the independence assumptions that need to be made are too radical. Problems arise in many ways, but one serious issue is the handling of lexical items, for instance, the context- free model assumes that the probability of a particular verb being used in a VP rule is independent of which rule is being considered. This means lexical

preferences for certain rules cannot be handled within the basic framework. This problem then influences many other issues, such as attachment decisions. Thus, pure probabilistic context-free grammars have drawbacks [39].

PCFGs tend to be robust and they produce a model of a language based on real data, and therefore do not have to worry about things like grammatical mistakes, which occur in real-life situations. Although PCFGs have many advantages, a critical disadvantage is that context is not taken into account at all [40].

Even structurally a PCFG is deficient and it takes no account of where a phrase appears in a sentence, for instance, the expanding of a subject NP is often different to the expanding of an object NP. Many attempts have been made to combine other theories and PCFGs to achieve better results and the probabilistic method of parsing gives a rank to the parses of some ambiguous text based on their probabilities from the given corpora [41].

b. Best First Parsing

Algorithms can be developed that attempt to explore the high-probability constituents first. These are called best-first parsing algorithms. The hope is that the best parse can be found quickly and much of the search space containing lower-rated possibilities is never explored. So adopting a best-first strategy makes a significant improvement in the efficiency of the parser. Even though it does not consider every possible constituent, the best-first parser is guaranteed to find the highest probability interpretation [3].

c. Simple Context Dependent Best First Parsing

The best-first algorithm leads to improvement in the efficiency of the parser but does not affect the accuracy problem. An alternative method of computing rule probabilities that uses more context-dependent lexical information is that of context -dependent best-first parser. The idea of this technique is in that the first word in a constituent is often the head and thus has a dramatic effect on the probabilities of rules that account for its complement. This suggests a new probability measure for rules that is relative to the first word, PROB(R | C, w). This is anticipated in (5) J. Allen [3].

$$\text{PROB}(R|C,W) = \frac{Count(\#\ times\ the\ rule\ R\ used\ for\ Cat\ C\ starting\ with\ W)}{Count(\#\ times\ cat\ C\ starts\ with\ W)} \qquad (5)$$

where R stands for the rule, C stands for category and W is word.

2.5.3 Parsing Strategies

On the previous section we discussed the different approaches that have been proposed to apply a set of grammar rules to the analysis of a sentence. These various approaches can be split up on a number of strategies, each strategy concerning different priorities in the derivation of a parse tree. This section looks at three such strategies.

Top-Down vs. Bottom-Up

A top-down parser starts with the S symbol and attempts to rewrite it into a sequence of terminal symbols that matches the classes of the words in the input sentence. The state of the parse at any given time can be represented as a list of symbols that are the result of operations applied so far, called the symbol list. Here is an example; the parser starts in start state (S), after applying the grammar rule S → NP VP, the symbol list will be (NP VP). It then applies the rule NP →ART N and the symbol list will be (ART N VP), and so on. The parser might recursively continue in this way until it arrives completely at terminal symbol states, and then it might check the input sentence to see if the classes of words in it matched with the rewritten sequence of terminal symbols [3].

In a bottom-up strategy, you start with the words in the sentence and use the rewrite rules backward to reduce the sequence of symbols until it consists solely of S. The left-hand side of each rule is used to rewrite the symbol on the right-hand side [10]. That is, it begins from each word and assigns its grammatical category until it reaches the start symbol. This operation is repeated, at each state, using the sequence of highest-level labels as the new string to operate on. On the other hand this technique attempts to group words into their respective categories together (e.g., take a sequence ART ADJ N and identify it as a NP) in a manner allowed by the rule [15].

Plus and Minus: Top-down methods have the advantage of being highly predictive. A word might be ambiguous in isolation, but if some of those possible categories cannot be used in a legal sentence, then these categories may never even be considered. On the other hand the top-down parser would rewrite S to (NP VP) and then rewrite the NP to produce three possibilities, (ARTADJ N VP), (ART N VP), and (ADJ N VP). In contrast, the bottom-up parser would have considered all three interpretations of some word from the start that is; all three would be added to the chart and would combine with active arcs. Given this argument,

the top-down approach seems more efficient. On the other hand Reduplication of effort is very common in pure top-down approaches and becomes a serious problem, and large constituents may be rebuilt again and again as they are used in different rules. In contrast, the bottom-up parser only checks the input once, and only builds each constituent exactly once. So by this argument, the bottom-up approach appears more efficient [3].

Depth-First vs. Breadth-First Strategies

A depth-first approach travels as speedily as possible between root and leaves (in a top-down approach) or leaves and root (in a bottom-up approach). A breadth-first approach travels all branches at every level before going up/down a level. A depth-first approach strains the vertical dimension of a tree structure of a sentence, breadth-first stresses its horizontal dimension, by taking into consideration all nodes at the same level in the tree structure [42].

Combinations of the Strategies

The above discussion has defined two strategic alternatives in the design of a parsing algorithm. Accordingly the following combinations of these strategies are defined [43].

- Top-Down/Depth-First
- Top-Down/Breadth-First
- Bottom-Up/Depth-First
- Bottom-Up/Breadth-First

Node Selection Strategies

The strategy combinations defined in the previous section do not bound all the possibilities in the derivations of a parse tree for a given sentence, for instance, when parsing bottom-up and depth-first, these strategies do not say which word in the input string one should start with. One could start with the first, but this is only one possibility. When parsing top-down, any of the possible constituents of the present unit could be expanded first. An additional limitation on the parsing process is needed to suggest which node of the parse tree should be expanded first. There are several possible strategies proposed [42].

Left-to-Right Expansion: in such approach the left-most node is expanded first (i.e., incorporating words of the input string starting from the leftmost and proceeding until the right-most is incorporated (in bottom-up parser) and expanding the left-most elements of the present unit's constituents before those to the right (in top-down parser).

Head-Driven Expansion: this parser designed using this approach is sometimes known as keyword parser, some key item at each level of structure (the "head") is expanded first, since this unit is seen as the controller of all other units at this level, for instance, a bottom up key-search parser would start by parsing the verb, since this item controls the presence and nature of other structural elements (e.g., a transitive verb expects a complement, while an intransitive one does not). Nouns may be selected as a second-level key, since they control much of the structure of the nominal group.

Chart Parsing

Simple parsers experience limitations in both completeness and efficiency. To come up with these problems, we will apply the algorithm design technique of dynamic programming to the parsing problem. Dynamic programming stores partial results and reuses them when appropriate, achieving significant efficiency gain. Such approach can be applied to syntactic parsing for various purposes and is known as chart parsing. The chart is a data structure for representing partial results of parsing process in such a way that they can be reused later on. Dynamic programming allows us to build a partial constituent just once. It will then maintain the results in a table, then it looks it up when it need to use it as a sub constituent of some object. The table that is used to save the partial results is called Well-Formed Substring Table (WFST). The substring table contains to adjacent sequence of words within a sentence. After the table is constructed is will systematically record what syntactic constituents have been found [39]. Simple illustration of the data structure is shown in Figure 2.11 for the simple Tigrigna sentence ‘ወታደራት ናብ ጦርነት ከይዶም’/soldiers gone to war.

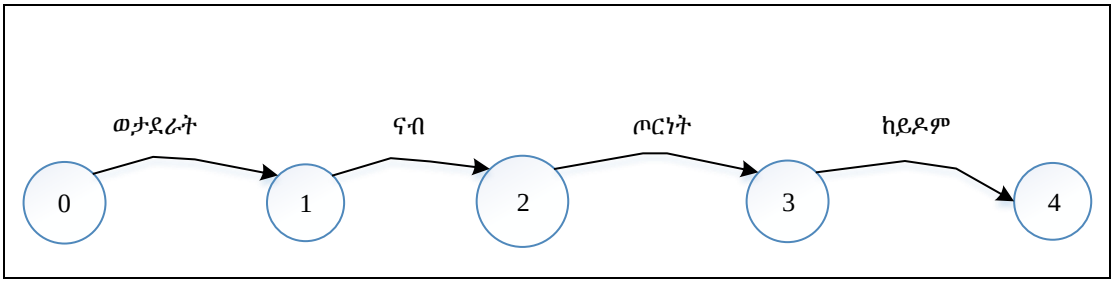

Figure 2. 11 The Chart Data Structure

In a well formed substring table, we maintain the position of the words by filling in cells in a triangular matrix. The vertical axis will denote the start position of a substring, while the

horizontal axis will denote the end position. In the example above the word ናብ/'to' will appear in the cell with coordinates (1, 2). Each cell in the matrix will contain the lexical category of the words not the words themselves. Accordingly, cell (1, 2) will contain the entry PREP (Preposition). For an input string$a_1 a_2 ... a_n$, and if the grammar contains a production of the form A→ a_i, then we add A the cell (i-1, i). Thus, for every word in sentence, we look up in our grammar rule or lexicon what category it has. I the previous example above if we have N in cell (0, 1), and PREP in cell (1, 2), cell (0, 2) will have a lexical category of NP (since we have the rule saying NP→N PREP). The general formula to do this is we can enter A in (i, j) if there is a production rule saying A→B C, and we find nonterminal B in (i, k) and C in (k, j). After constructing an S (start symbol) node that covers the whole input, we conclude that there is a parse for the whole input string i.e., S⇒*$a_1 a_2 \ldots a_n$ [39]. Figure 2.12 illustrates the work for the simple Tigrigna sentence we used above.

WFST	1	2	3	4
0	N	NP	-	-
1	-	PREP	-	-
2	-	-	N	-
3	-	-	-	V

Figure 2. 12 Sample Well Formed Substring Table

The above information can be represented in a directed acyclic graph, which is sometimes called a chart as shown in Figure 2.13

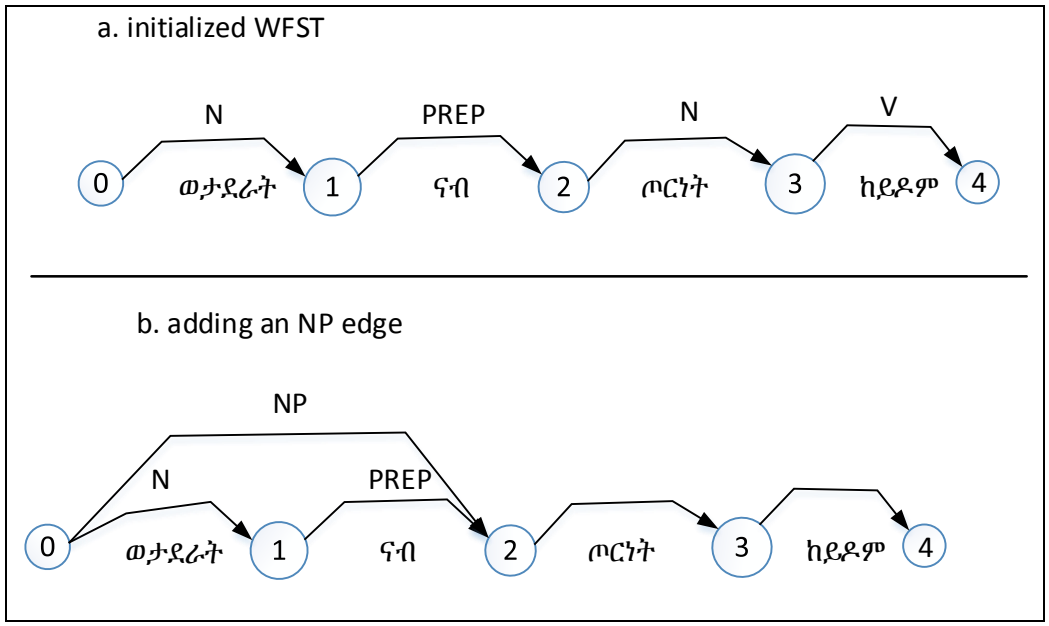

Figure 2. 13 Chart for the WFST

As one can see from the above example charts can hold multiple hypotheses for a given span. Thus, they are more general than WFST. A WFST is unable to encode a relation of immediate dominance. To come up with such problem, we can label edges with the whole production that justified the addition of the edge instead of labeling them with a non-terminal category. This is illustrated as shown in Figure 2.14.

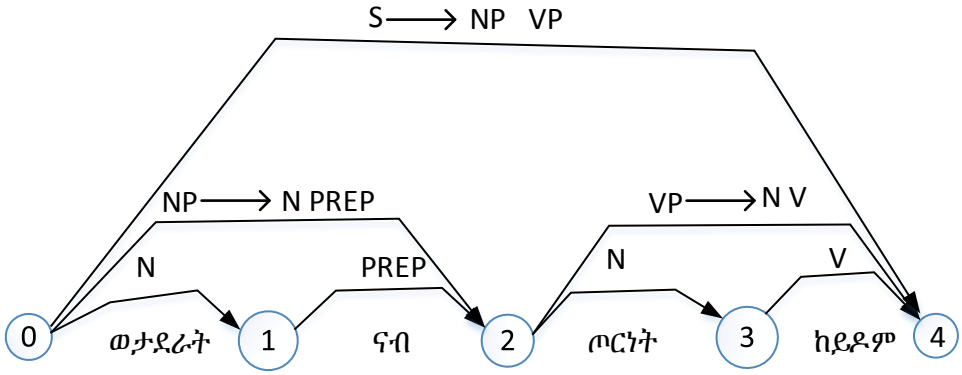

Figure 2. 14 Chart Labeled with Productions

Chart parser hypothesizes constituents by adding edges based on the grammar rule, the tokens and the constituents already found. Any constituent that is compatible with the current knowledge can be hypothesized. Many of the hypothetical constituents may not be

used in the final result but WFST just saves these hypotheses. Hypothesizing edges that are incomplete is vital for example the work done by the parser in processing the production NP→PREP VP ADJP can be reused in processing NP→PREP VP. Therefore, the hypothesis the PREP constituent which is the beginning of a NP the will be record. Such hypothesis is recorded by adding a dot to the edge's right hand side. Accordingly, the edge to the left of the dot states what the constituent starts with and edge to the right side of the dot states what is still required for that constituent to be complete the constituent. The edge that records the hypothesis saying a NP starts with PREP ናብ/'to', but still requires a VP to become a complete edge is illustrated in Figure 2.15.

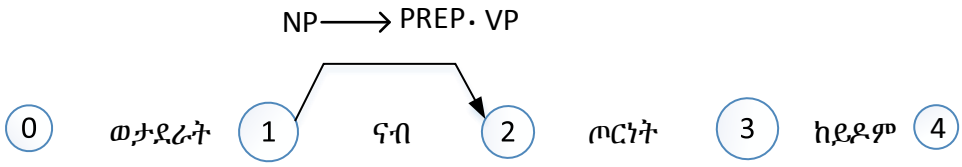

Figure 2. 15 Chart with Incomplete Edge

This is to mean that the dotted edges are used to record the entire hypothesis that the chart parser makes about the constituents in a sentence. Generally, a dotted edge [A→ $c_1 \ldots c_d . c_{d+1} \ldots c_n$,(i, j)] records the hypothesis that a constituent of type A that spans (i,j) starts with children $c_1 \ldots c_d$, but it still requires children $c_{d+1} \ldots c_n$ to become a complete. $c_1 \ldots c_d$ and c_{d+1} may be empty. If d = n, then $c_{d+1} \ldots c_n$ is empty and the edge represents a complete constituent and is called a complete edge. If not, the edge represents an incomplete constituent and is called an incomplete edge. If d = 0, then $c_1 \ldots c_n$ is empty and the edge is called a self-loop edge [39].

The main idea behind chart parsers is that the importance of improving parsing efficiency rather than the simple parsers. There are three points to keep in mind for efficiency consideration of chart parsing. It is advisable not to do twice what can be done once, not to do once what can be avoided altogether and not to represent distinctions if that is not the concern of the study [44].

The chart parsing is a kind of parsing which efficiently parse an ambiguous text using algorithmic technique of dynamic programming. Since dynamic programming stores intermediate results and re-uses them when needed it results in achieving significant

efficiency gains. We can apply this method letting us store partial solutions to the parsing task and then look them up as required in order to efficiently reach at a whole solution. Chart parser stores the intermediate results and maintains the record of rules that have matched but are not completed. For example in the example above, a dynamic programming allows us to build the VP 'ናብ ጦርነት ከይዶም' just once. Then store it in a table then looks it up when it is necessary as a sub constituent. It is recommended to record that result in a data structure known as a chart. Recording of intermediate results is a form of dynamic programming that avoids repeated work [45].

2.6 Summary

In this Chapter we began by discussing Tigrigna language. Tigrinya is one of the official languages of both Eritrea and Ethiopia. Then we discussed the Tigrigna grammar. Under this we covered Tigrigna word classes, phrases and sentences. Tigrigna places its verb at the end ([Subject] [Object] [Verb]) and Tigrigna verbs are conjugated to agree with the subject in person, number, and gender. We then identified the required components in developing a sentence parsing. In developing a parser lexicon, grammar formalism and parsing methods are needed. We have discussed the different grammar formalisms, parsing strategies and approaches that are proposed so far by some researchers. From the literature review made, we observed that chart parser is a type of parser suitable for ambiguous grammars. It uses the dynamic programming approach. Partial hypothesized results are stored in a structure called a chart and can be re-used. This eliminates backtracking and prevents a combinatorial explosion. Chart parsers are widely used in most NLP tasks as approaches to parsing. Besides, the probabilistic parser play great role in avoiding structural ambiguity in syntactic parsing by assigning a probability to each rule derived from the corpora. Thus, we hypothesize that if these two independent parts are joined together to do a single job they give a better result of the parser. Finally, we proposed to use PCFGs and bottom up chart parsing algorithm in our study.

Chapter 3: Related Work

3.1 Introduction

Currently, there are much works done on NLP for different languages. Sentence parsing is used to improve the performance of many applications such as Machine Translation, Grammar Checker, Question Answering, Semantic analysis and many other applications. Besides it is also vital for recognition of phrasal categories, study correct grammatical structure, and to see the relationship between words in a sentence. Automatic parsing is one of the most widely studied topics in NLP and many research works were dedicated to automatic parsing in the past few decades.

This chapter makes a review on papers that are similar to the area of the research. For better understanding of the current study, relevant works that have been carried out for various Ethiopian and non-Ethiopian languages are discussed.

3.2 Parsers Developed for Ethiopian Languages

Abiyot Bayou [13] has designed and implemented a word parser for Amharic verbs and their derivation. A knowledge-based system that parses verbs, and nouns derived from verbs was designed in the study. Root pattern and affixes were used to determine the lexical and inflectional category of the words. The study did not include the property of words at syntactic level. Experiments were done on a limited number of words in which 200 verbs and 200 nouns. The result of the experiment showed a promising result (i.e., 86% of the verbs and 84% of the nouns were recognized correctly).

Atalech Alemu [14] developed automatic sentence parsing for Amharic texts. An effort was made to describe how to develop an Automatic sentence parser for Amharic language using the probabilistic Inside Outside algorithm supplemented with chart parsing algorithm that implemented Probabilistic Context Free Grammars. The research work adopted PCFG parsing to develop the Amharic parser. A small sample corpus was prepared as part of the study from Amharic book entitled “Amharic grammar” to generate phrase structure rules and to estimate probability values. The study considered very small corpus (only 100 simple sentences) and composed of four word only and the sentences were from one type of sentence (i.e., simple declarative). The selections of the sentences were in such a way that the sentence should composed of two or more of the phrase classes of the language. Only

one parameter was used to measure the performance of the parser. A POS tagger developed by Mesfin [46] was used to automatically tag the words in a sentence and assign a part of speech like Noun, Verb, Adverb, Adjective and so on. The output of the tagger was used by the parser developed as an input. Manual parsing was used by the researcher after the corpus has passed through the POS tagger. The author selected 80 sentences randomly from the sample to conduct the probability calculations for the words in the sentences, grammar rule inductions and the assignment of the probability to the grammar rules. The rest 20 sentences were used as a test set. Experiments were organized in three phases, one on the training set, the second on the test set, and the third one on a set of four word sentences, different from the ones used as training and testing set, obtained from different people. On the first set of sample sentences the results achieved was very high, 100% on the training set and approximately 96% on the test set. The result obtained on the third set was 77%.

The other work dedicated to Amharic sentence parser reviewed in the current study is the work done by Daniel Gochel [15]. The author developed an automatic complex sentence parser for Amharic, The Inside-Outside algorithm together with a chart parse module that was originally proposed by Yao and Lua [38] were used to implement the Probabilistic Context Free Grammars PCFGs parsing. The study is similar to the work of Atelach Alemu [14] but focuses on complex Amharic sentence. A total of 350 complex sentences were collected from two widely used Amharic grammar books in the language (i.e., Amharic grammar and Amharic for beginners). The sample corpus is given to the morphological analyzer and POS tagger for the purpose of pre-processing. Among these 280 of them were used to calculate the probability of words in the sentences, grammar rule formation and probability assignment to the grammar rules. The rest 70 sentences were taken as a test set. Experiments were conducted in two phases, one on the Training set and the other on the test set. The results achieved based on the small sample were high, 89.6% on the training set and approximately 81.5% on the test set. In the study, the percentage of correctly tagged and parsed words and sentences in the sampled text was used to measure the performance boost of the original purely statistical part-of-speech tagger, and a simple sentence parser, and a newly developed complex sentence Amharic parser. On the experiments conducted using the training set and test set to see the POS tagger's performance when the morphological analysis module was embedded, 98.7% and 94% accuracies were observed on the two

sample corpus subsets, respectively. The result suggest that most of the identified errors were human made errors during the preprocessing of the parser's input, conflicting PCFG rules, low probability and absence of some rules.

Diriba Megersa [16] developed an automatic sentence parser for Oromo language using the chart algorithm and made an effort to develop a prototype. The study has been conducted using the chart algorithm with the grammar formalism Head-driven Phrase Structure Grammar (HPSG) compiled into left to right table. Beside with the chart algorithm, the author has also used the supervised learning algorithm to enable the parser predict unknown and ambiguous words and a morphological analyzer to split words into root form and their corresponding morpheme. For the purpose of the study a sample narrative text of two pages consisting of 352 sentences in Oromo was obtained from unpublished handout for Oromo syntax. Among these sentences 85% of them were used as a training set and the rest 15% were used as a test set. Experiments were conducted in two phases, one on the training set and the other on the test set. The Parser was first trained on the training set. The result obtained on the training set was then evaluated by comparing it with the manually parsed text used as a training set. To improve the accuracy of the Parser, error were identified and corrected and the training was repeated again. These processes were done repeatedly until the result obtained was found to be efficient. In the study, only one parameter, the percentage of correctly parse sentences in the sampled text was used to measure the performance of the parser. The results achieved based on the small sample were high, 95% on the training set an approximately 88.5% on the test set. The supervised learning algorithm endorsed by Eric Brill [47, 48] was implemented to enable the parser to guess unknown words in an input sentence. Similarly to the previous papers, the author suggested that errors identified were due to human made errors rather than the algorithms used.

3.3 Parsers Developed for non-Ethiopian Languages

Al-Taani et al [11] developed top-down chart parser for Arabic sentences. The work was designed for parsing simple Arabic sentences, which includes small and verbal sentences within specific domain Arabic grammar. The author derived the grammar rules which give the precise description of grammatical sentences first. Then, they implemented the parser which assigns grammatical structure of the input sentences. The corpus used for the purpose

of the study contains a total of 70 sentences collected from Arabic documents composed of words ranging from 2 to 6. The parser designed by the researcher has three main steps to follow in order to parse Arabic sentence which are word classification, grammar identification and finally parsing the sentences. First the user might initially use the automatic categorization to break the sentence into its main components. After that the system identifies those grammars which are used to analyze and determine the structure of the given grammar.

On the other hand B. Bataineh and E. Bataineh [12] developed an efficient recursive transition network parser for Arabic language. The work has the goal of analyzing and extracting the attributes of Arabic words. In the implementation of the parser, top-down parsing technique with recursive transition network grammar was used (i.e., Recursive Transition Network). Morphological analyzer is also used to consider words that are not found in lexicon directly. For the purpose of experimentation, the researchers collected a sample of 90 sentences composed of 6 words length from grade 6 Arabic text book. The sentences are made to have gender and number agreement to ensure the correction of syntax structure of the Arabic sentences. The result of the designed parser, it is found that some sentences are unparsed totally and some other sentences are parsed incorrectly because of the following reasons; first when the parser was unable to get the word in the lexicon, second because of the incorrect input sentence by the user and third reason was suggested due to lack of rules (i.e., when the parser is unable to produce a rule for the input sentences and when the syntactic form of the sentences is not included in the grammar).

E. Charniak [49] presented a statistical parser that induces its grammar and probabilities from a hand-parsed corpus of the peen tree-bank [50]. Parsers induced from corpora are of interest both as simply exercises in machine learning and also because they are often the best parsers obtainable by any method. That is, if one desires a parser that produces trees in the tree-bank style and that assigns some parse to all sentences thrown at it, then parsers induced from tree-bank data are currently the best. The work illustrated a parsing system based upon a language model for English that is, in turn, based upon assigning probabilities to possible parses for a sentence. The model used in the study used in a parsing system by finding the parse for the sentence with the highest probability. The researcher evaluated the performance of the system by training the parser on about one million words of the Peen

Wall Street Journal Tree bank and testing on 50,000 words. The work showed that the system outperforms previous schemes as different authors have also ranked it as the third in a series of parsers. An experiment was described which suggests that its superiority in performance to other parsers in the area mainly from unsupervised learning plus the more extensive collection of statistics it uses, both more and less detailed than those in previous systems.

3.4 Summary

In this Chapter, we discussed related works on automatic parsing of various languages. Each language has unique characteristics, which requires special considerations in the development of automatic parser. Although several researchers proposed various techniques for natural language text, the proposed techniques are language specific and need special consideration in the design in order to be applied for Tigrigna language. Accordingly, it can be understood that to develop automatic parser for Tigrigna language, one has to deal with the unique features of the languages. To our best knowledge, the development of automatic parser for Tigrigna language has not been attempted so far. Thus, this research aims at developing automatic parser for Tigrigna text by dealing with the grammatical features of the language.

Chapter 4: Design of Tigrigna Parser

4.1 Introduction

This Chapter presents the system architecture to the identified problem. The system architecture comprises system components and the interaction between the components.

We begin by presenting the architecture and giving an overview of the system model. Then we proceed to discuss each components of the architecture. During the explanation of the components what each component does, how each component performs and in what way it is interrelated with other components is reported. Besides the various algorithms or pseudo-codes adapted in the study to accomplish the work are stated.

4.2 System Architecture

The architecture has two parts: the training and parsing. The training part contains components where training is accomplished (i.e., training of PCFG rules and lexical rules). The parsing part contains components which perform the task of parsing given the input sentence. In the training part of the architecture, first the sample corpora collected from the sources is preprocessed by developing simple preprocessing component. The preprocessed sample corpus is manually tagged and stored in a container called POS tagged sentences. The tagged sentences are then parsed manually by two linguists in the language. The parsed sentences are later used for comparison purposes with the system parse. Beside, from the parsed sentences a Context Free Grammar (CFG) induction is made. Using the formula to calculate the probability of a given rule and constituent, Probabilistic Context Free Grammars (PCFGs) are extracted by assigning probability to each rule based on evidence from the corpora. The tagged corpus is also used to prepare the lexicon using the lexicon generation component. The lexicon generation module is used to prepare the lexicon automatically from the tagged corpus. Morphological analysis is helpful for morphologically rich languages like Tigrigna. The morphological analysis is used to return root word of the given word by the lexicon generator. The parsing part contains components such as sentence tokenization, morphological analysis and the PCFG chart parsing. The first two components make the input sentence suitable to the parsing component. The sentence tokenization component is used to split the input sentence into single units (words) so that the morphological analyzer later gets units of the given input sentences at a time. For the

purpose of the study the morphological analyzer developed by M. Gasser [51] is integrated in to our system to enhance the performance of the designed parser. Then the PCFG chart parser parses the given input according to the rules. The overall architecture of the system with the commonly used components is presented in Figure 4.1.

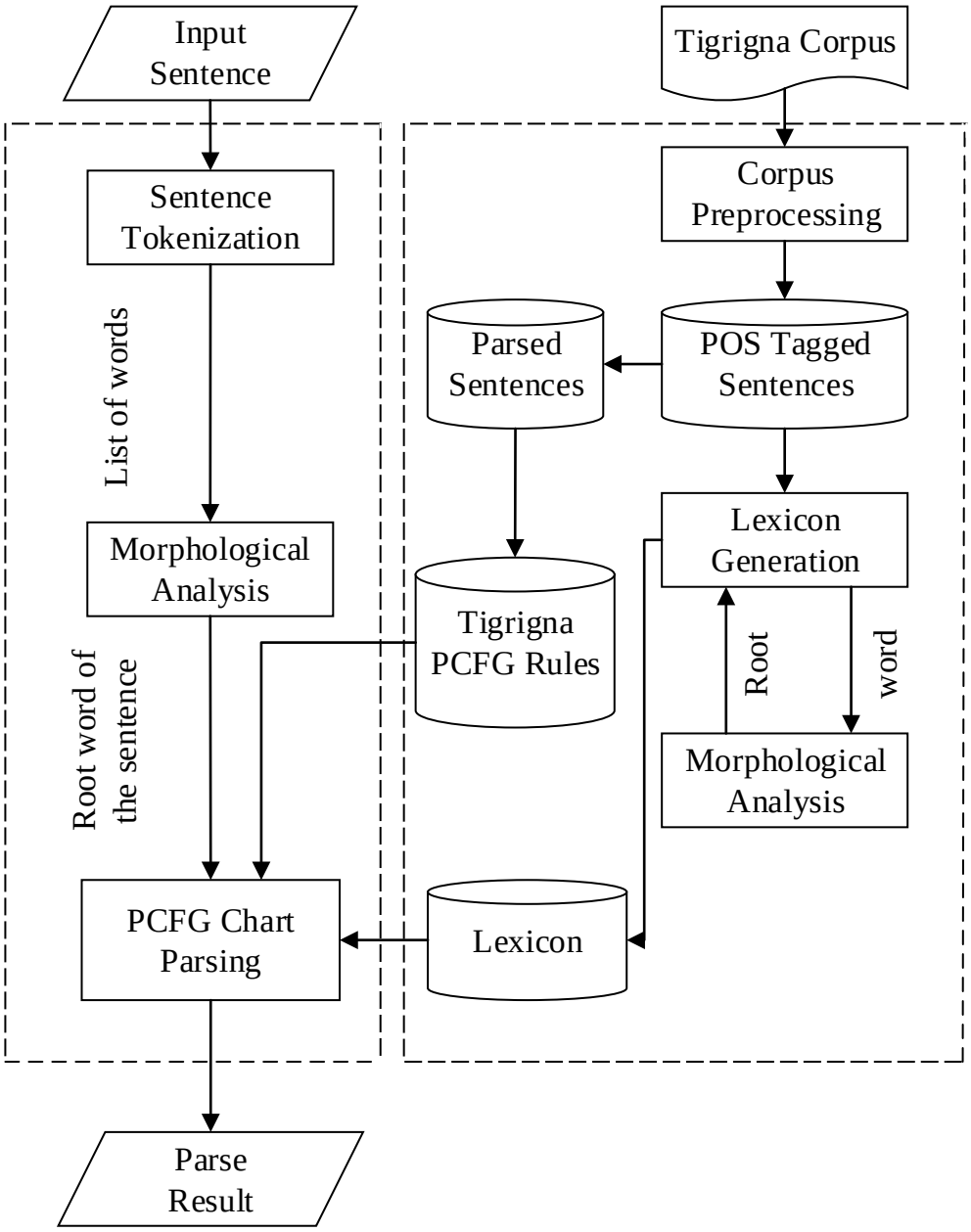

Figure 4. 1 The Architecture of the Parser

4.3 Corpus Preprocessing

Sentences for this study are collected from different sources. The corpus from the different sources contains some unnecessary words and characters. Some of the corpus contains some tags which are out of the language. It should have to be made suitable for next step. The original corpus from the sources cannot be used directly due to the existence of needless scripts (such as <s>, <doc>), characters (like < >). In addition to this the corpus contains some unwanted characters which are not in the language characteristics. Since these characters can affect the next process, they should be removed first. The first thing to be done before going any further is to remove any useless texts and characters in the corpus. However, doing this task by hand is a tedious and time taking. Thus, in our current study these unnecessary things are removed by developing simple preprocessing component. Accordingly, the preprocessing component scans the corpus and whenever it gets the above declared unnecessary texts it deletes them and writes the text that is necessary for use and free from any unwanted texts in to a file. The algorithm used to preprocess the characters in the corpus is shown in Algorithm 4.1.

```
Input: Tigrigna corpus
Read the file from corpus
For each string in the corpus
  If the string is '<s>' or '</s>' or
 '<doc>' or'</doc>'or '<p>' or '</p>'or
 '<PUNC>'or '<body>'
     Remove the string
     For each character in the file
       Look for the characters'<'or'>'or'።
       'or'፣'or '?'or'!'
         If the above mentioned characters
         appear
           Remove the characters
         End if
      End for
   End if
End for
Write the file of the corpus
Output: preprocessed corpus
```

Algorithm 4. 1 Corpus Preprocessing Algorithm

Besides the simple Tigrigna sentences taken from the source book were written in Ethiopic or Ge'ez script. The complex sentences corpus taken from other sources like Walta Information Center, Ethiopian Broadcasting Corporation Tigrigna program and Woyen were given in Latin characters. To make the corpus taken from the source book consistence with the other sources provided in Latin, we converted them into Latin characters. Thus, to do the conversion we developed a simple corpus reader which uses a dictionary in order to convert each Tigrigna Fidel to its equivalent Latin character. The algorithm used for the corpus reader is given in Algorithm 4.2.

```
Input: Tigrigna preprocessed corpus
Define the dictionary
Read the corpus from a file
For each word in the corpus:
   For each character in the word:
       If the character is in dictionary:
          Replace the character in the
          dictionary
          Write the character
       End if
       If the character is space character:
          Write the space character
       End if
   End for
End for
Close the written file
Close read file
Output: a paragraph of text in Latin
```

Algorithm 4. 2 Algorithm for the Corpus Reader

The output of the corpus reader is a paragraph of texts containing the given corpus in Latin characters. The paragraph of texts should be separated in sentential form so that they can be suitable for later process. Accordingly the corpus reader provides the paragraph of text to the sentence splitter. The sentence splitter sometimes called sentence segmenter, splits text up into individual sentences with unambiguous delimiters. Thus, the sentence splitter module in our study splits the corpus based on Tigrigna sentence delimiters which can be one from the characters '።', '?' and !, after the transliteration process they will be like '.', '?' and '!' respectively**.** The algorithm used for splitting the paragraph into sentence is illustrated in Algorithm 4.3.

```
Input: A paragraph of text containing the corpus
Accept the paragraph of text
Scan throughout the corpus
For each text in the corpus
 If the corpus comprises characters '.' or'?' or '!'
   Get text before the characters
  Mark it as the end of a sentence
  Break the sentence and add end marker after it
 Else
  Scan until the above characters appear
 End if
End for
Output: list of segmented sentences by an end marker
```

Algorithm 4. 3 Algorithm for Sentence Splitting

Part of Speech Tagging and Parsing

Part of Speech Tagging (POST) is the process of labeling a word in a text (corpus) as equivalent to a particular part of speech category (Noun, Adjective, Preposition, Adverb,

etc.). There is a work done by Teklay Gebregzabher [8] on Tigrigna Part of Speech Tagger (POST). But since it is not a freely available open source we cannot integrate it in to our system. Even though it is time taking and tiresome task, manual tagging is done on the processed text corpora by some language expert in the language for the 142 simple sentences. Then the tagged sentences from the different sources are stored in a container called POS tagged sentences. The tagged corpus is later parsed manually by the linguist according to the phrase structure rule of Tigrigna language to be used for comparison purposes with the designed parser and for the extraction of CFG and PCFG rules. The parsed sentences are kept in the parsed sentences part of the architecture. The parsed sentences are found under Appendix H.

4.4 Lexicon Generation

After the corpus is processed and tagged manually the next task to be done is to prepare the lexicon. As discussed in Chapter Two of this study a lexicon is the backbone of most Natural Language Processing (NLP) system. It is the knowledge about individual words in the language. Words with grammatical attributes are very important for many applications. In representing the lexicon for sentence parser, it would only need to store one entry for a word that can have a number of derivations and inflectional word. We know that preparing the lexicon manually is time taking, tedious task and even it is error prone, mainly when the size of the corpus is getting larger and larger. Thus, for this purpose, we developed a lexicon generator that gives the lexical rules from POS tagged sentences automatically. The morphological analyzer is required in the preparation of the lexicon to get the word's root. The POS tagged sentences have the form: the word followed by a forward slash and a POS tag set like N, V, ADJ, DET, etc. given nearly after it. Accordingly the lexicon generator reads the POS tagged sentences one by one and outputs the result as POS tag set followed by an " →" symbol and then lastly the root word enclosed in double quotation mark. To return the root words we have used the morphological analyzer. This is done in such a way that the lexicon generator requests the morphological analyzer for the word's root then the morphological analyzer returns the root of that word. The algorithm used for generating a lexicon is given in Algorithm 4.4.

```
Input: POS tagged corpus
Define lexical rules
Read the file from a corpus
Recognize the POS tag and the word
For each word in the file
   Invoke the HONMORPHO ()
   See the result of HONMORPHO
   If the HONMORPHO finds the root
     Select the root
     Return the root
   Else
     Return the word it self
   End if
Write the results based on the lexical rules
End For
Write the file of the corpus
Output: lexical items
```

Algorithm 4. 4 Lexicon Generation Algorithm

4.5 Sentence Tokenization

The morphological analyzer integrated in our system takes one word at a time. To make the input sentence suitable the sentence should be splitted into words before given to the morphological analyzer. For the purpose of this work we developed a sentence tokenization component so that the sentences are divided in to morphemes. After the tokenization process, the words are provided to the morphological analyzer. Accordingly, the tokenizer searches for the sentences and when it gets some word delimiter it identifies it as a separate word. The tokenization module developed for this purpose has limitations. A problem faced during the tokenization process is that since the delimiter used to separate words is a while space character, words like compound words or multiword could not be recognized as a

single word rather they are treated as a distinct words. To come up with such kinds of problems, some rearrangement is made manually. The sentence tokenization algorithm is shown in Algorithm 4.5.

```
Input: the user input sentence
For each character in the sentence
   If the character is white space
     Print the characters before the white space
   Else
     Append the character to a string variable
   End If
End For
Output: segmented words of the user input sentence
```

Algorithm 4. 5 Algorithm for Sentence Tokenization

4.6 The Morphological Analysis

The morphological analyzer is also used before the input sentence is given to the parser. the input sentence should be a root word sentence. The reason is the prepared lexicon consists of words that are morphologically analyzed in their root word. For this reason, the parser inputs sentences whose words are morphologically analyzed otherwise it is impossible to get the tag set of the input sentence words in the prepared lexicon. So for the purpose of this study we integrated a freely available source morphological analyzer which is the HORNMORPHO developed by M. Gasser [51]. It is a Python program that analyzes Amharic, Oromo, and Tigrigna words into their constituent morphemes (meaningful parts) and generates words, given a root or stem and a representation of the word's grammatical structure. It is part of the $\boldsymbol{L^3}$project at Indiana University which is dedicated to developing

computational tools for under-resourced languages. The algorithm used to integrate the HORNMORPHO to our system is given in Algorithm 4.6.

```
Input: word
Accept a word
Invoke the HornMorpho ()
    If the input word has a root
        Select the root of the word
        Return the root of the word
    Else
       Return the input word itself
    End If
Output: root of the input word or the input word
itself
```

Algorithm 4. 6 Morphological Analysis Algorithm

4.7 The PCFG Chart Parsing

This parsing component implements bottom-up PCFG chart parsing algorithm. As stated in Chapter Two of this study simple parsers have several drawbacks and limitations in that they lack both completeness and efficiency. In order to deal with these drawbacks we need to have other methods to derive the parse of an ambiguous sentence. The parser module implements two independent methods to deal with the parses of an ambiguous sentence (i.e., the chart parsing and probabilistic parsing). These are methods to deal with the problem of ambiguity in the parses of a given text. In this study we used the combinations of these two methods together, probabilistic chart parsing using the bottom up strategy.

Chart parsers use a set of rules to heuristically decide when an edge should be added to a chart. This set of rules, along with a specification of when they should be applied, forms a strategy.

a. Fundamental Rule

The fundamental rule is important which is used by every chart parser and it is used to combine an incomplete edge that's expecting a nonterminal B with a following, complete edge whose left hand side is B. Appendix I shows the fundamental rule.

Note that the dot has moved one place to the right, and the span of this new edge is the combined span of the other two. Note also that in adding this new edge we do not remove the other two, because they might be used again. If there is a chart of the form shown in Figure 4.2 (a), then we can add a new complete edge as shown in Figure 4.2 (b). in order to create a bottom-up chart parser, we add two new rules namely the Bottom-Up Initialization Rule and the Bottom-Up Predict Rule to the Fundamental Rule.

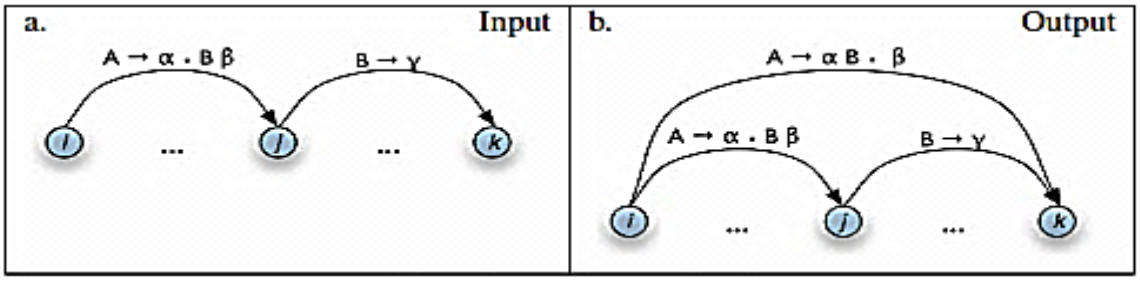

Figure 4. 2 The Fundamental Rule

b. Bottom-up Initialization Rule

The bottom-up initialization is stated under Appendix I. Figure 4.3 shows the bottom-up initialization rule for the sentence 'ወታደራት ናብ ጦርነት ከይዶም/Wetaderat nab Tornet keydom/soldiers have gone to war'.

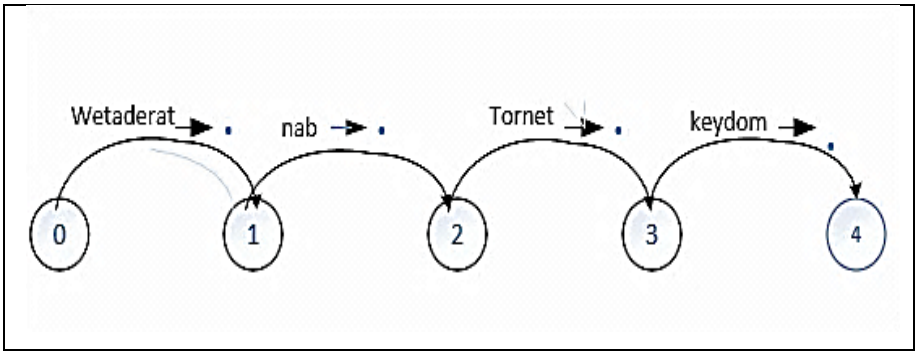

Figure 4. 3 Example of Bottom-up Initialization Rule

The dots on the right hand side of these productions are showing that we have complete edges for the lexical units.

c. Bottom-Up Predict Rule

To show the rule graphically assume that if the chart looks as in Figure 4.4(a), then the Bottom-Up Predict Rule tells the parser to augment the chart as shown in Figure 4.4(b). The bottom-up predict rule is shown under Appendix I.

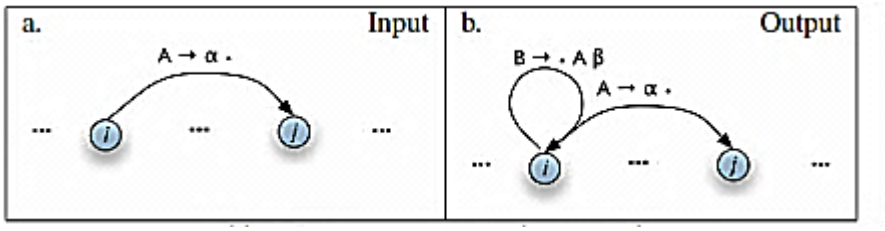

Figure 4. 4 Bottom-Up Predict Rule

The algorithm used in our study keeps a queue of edges, and adds them to the chart one at a time. The ordering of this queue is based on the probabilities associated with the edges allowing the parser to expand more likely edges before less likely ones. Each time an edge is added to the chart, it may become possible to insert new edges, so these are added to the queue. The bottom-up chart parser continues adding the edges in the queue to the chart until enough complete parses have been found, or until the edge queue is empty. Like an edge in a regular chart, a probabilistic edge is consisted of a dotted production, a span, and a partial parse tree. The parse tree contains an associated probability with it. Its probability is the product of the probability of the production that generated it and the probabilities of its children. For example, the probability of the edge [Edge: S → NP . VP, 0:2] is the probability of the PCFG production S → NP VP multiplied by the probability of its NP child. The edge's tree only includes children for elements to the left of the edge's dot. Thus, the edge's probability does not include probabilities for the constituents to the right of the edge's dot. To parse a sentence, a chart parser first creates an empty chart spanning the sentence. It then finds edges that are licensed by its knowledge about the sentence, and adds them to the chart one at a time until one or more parse edges are found. The algorithm used to implement the parser is shown in Algorithm 4.7 which is adapted from S. Bird et al [52].

```
Create an empty chart spanning the sentence.
Apply the Probabilistic Bottom-Up Initialization
Rule to each word.
Until no more edges are added:
 Apply the Probabilistic Bottom-Up Predict Rule
 Everywhere it applies.
 Apply the Probabilistic Fundamental Rule
 Everywhere it applies.
Return the parse trees similar to the parse edges
in the chart.
```

Algorithm 4. 7 Bottom-Up PCFG Chart Parsing Algorithm

The bottom-up PCFG parser uses a dynamic programming to find the single most likely parse for a string. It parses string by iteratively filling in a table called most likely constituents table. The most likely constituent table stores the most likely tree for each span and node value. After the table is completed, the parser returns the entry for the most likely constituent that spans the entire text and whose node value is the start symbol S. what we are supposed to record is the most likely constituent for any given span and node value. The PCFG suggests the ranks the tree structures based on their probability. Thus, the parser discards all of the tree structures other than the structure which is ranked first. Since the type of grammar that we are implementing is PCFGs, we can calculate the probability of each constituent from the probabilities of its children. The bottom-up PCFG parser tries edges in descending order of the inside probabilities of their trees. The inside probability of a tree is simply the probability of the entire tree. A constituent's children cannot cover a larger span than the constituent itself. Based on this assumption, each entry of the most likely constituents table depends only on entries for constituents with shorter or equal spans. Accordingly, the Viterbi parser fills in the most likely constituent table incrementally. The parser starts by filling in all entries for constituents that span one element of text. After that it fills in the entries for constituents that span two elements of text. Until the entire table has

been filled, it continues to fill in the entries for larger constituents. Lastly, it yields the table entry for a constituent spanning the entire text, whose node value is the grammar's start symbol. The Viterbi algorithm which is adapted from E. Loper and S. Bird [53] is illustrated in Algorithm 4.8

```
Create an empty Most Likely Constituent (MLC) table
For any given span of the text:
 For each start index in the span:
  For each production in the grammar:
   For each sequence of sub trees in MLC:
    Old production is the probability of the
    Current entry in the table
    New production is the probability of the tree
    formed by applying the production to the
    children
    If the new production is greater than the Old
    production:
      Update the table with this new tree
      Insert new tree in to the MLC
Return table entry for a constituent spanning the
entire text whose node value is grammar's start
symbol
```

Algorithm 4. 8 Viterbi Parsing Algorithm

Chapter 5: Experiment

5.1 Introduction

In this chapter, the implementation, experimentation, the experimental results and the evaluation of the system are discussed.

First we briefly described the sample corpus used for the purpose of this study. Second the implementation is explained. Beneath that the development environment is highlighted stating the experimental setting such as capacity of the computer, operating system used to develop the system. Then the tools and programming languages used in the experiment and reasons why they were chosen is covered. Afterwards the development of the parser is illustrated. Third the evaluation metrics and techniques are stated. Fourth the experimental results and discussions are explained. Finally, the solution and suggestion to the recognized problems was discussed.

5.2 Corpus Collection

In this study, we collected both simple and complex sentences from different sources. The simple Tigrigna sentences are taken from the Amharic grammar book of Baye Yimam [22] which was later converted to Tigrigna by two linguist experts while the complex sentences are taken from the tagged corpus of different sources like Walta Information Center (WIC) corpus, Ethiopian Broadcasting Corporation Tigrigna program and Woyen newspaper. Besides, the simple Tigrigna sentences were written in Tigrigna Fidel then we converted them into Latin characters so that they can be readable to international researchers out of the language domain as well as to be compatible with the other corpus taken from other sources. Thus, to do the conversion we developed a simple corpus reader which uses a dictionary in order to convert each Tigrigna Fidel to its equivalent Latin character. This is developed using python programming language. The corpus reader has a key, the Tigrigna character in one side which is separated by colon and on the other hand there is a Latin character which is equivalent to the Tigrigna Fidel. Below is simple illustration.

Example : 'ሀ' : 'he' , 'ሁ' : 'hu' , 'ሂ' : 'hi' , 'ሃ' : 'ha' , 'ሄ' : 'hE', 'ህ' : 'h' , 'ሆ' : 'ho'

The detail dictionary used for transliteration is listed under Appendix C taken from the work of M. Gasser [19]. The simple and complex Tigrigna sentences with their transliterated

version are listed under Appendix D. The transliterated corpus is used for implementation and testing the system. The corpus reader scans the contents of the corpus and changes each character accordingly. Thus, for the purpose of this study we used a total of 262 sentences based on the distribution of the different phrase structures, using a non-probability sampling (i.e., judgment sampling technique). Judgmental sampling is a non-probability sampling technique where the researcher selects units to be sampled based on their knowledge and professional or expert judgment**.** These sentences are later used for experimentation. The number of sentences was limited to 262 because of the time constraint and the complexity of grammar rule induction and probability calculation. The number of simple Tigrigna sentences is 142 and the rest 120 are complex Tigrigna sentences which are taken from tagged corpus of the previously mentioned sources. The criterions set in the selection of the simple Tigrigna sentence is that the presence of two or more words in the noun phrase and verb phrase. In the case of complex sentence the criterion set is the presence of a total of eight or lesser words in a sentence, three or more words in the noun phrase and verb phrase.

Apart from the parts of speeches identified from the different literatures we made and other previous works, there are number of special tags identified by the researcher based on guidance from the language expert and other previous works. In addition the abbreviation used for the tags in the examples we used in Chapter Two are slightly modified. Thus, some unusual tags are added to be applicable in our study (e.g., NPREP, CARDN, CARDPREP, VPREP, etc.). The detailed special tags used in our study are listed under Appendix E and the tagged sentences are found under Appendix F.

5.3 Implementation

5.3.1 Development Environment

It is advisable to have an environment which is high in memory and processor which is capable of running all of the NLTK packages and other open-source Python Integrated Development Environment (IDE). The designed system is implemented and tested on an available environment with Intel Core i7 CPU of 2.20 GHZ speed, a 4 GB RAM, and Windows 7 Operating system.

5.3.2 The Tools and Programming Languages

The system is developed using Natural Language Toolkit (NLTK) and Python 3.4 programming language. The reason for choosing this programming language is visible in that they are suitable for processing different NLP tasks. Python is easy to use and Python code is extremely short, takes much less time to develop a program in Python and it has a large repository of built-in functions which allow for many tasks to be performed easily. Besides python programming language has the capability of using Unicode representations easily than other programming languages [52]. The other probable reason is that since we are going to integrate the HornMorpho which is written in python we need to use this language for ease of integration of the analyzer with our system. And the reason behind selecting NLTK is that it is an open source tool that contains open source python modules, linguistic data corpus and documentation for research and development in the area of natural language processing. It supports many NLP tasks such as tokenizer, stemmer, sentiment, chunking, clustering, part of speech tagger with distributions for Windows, Mac, and Linux for some languages such as English and German. Moreover, it contains different corpora for these two languages with their respective corpus reader module [54].

5.3.3 The Development of the Parser

A prototype has been developed which prompt the user to provide the sentence that the user wants to be parsed. After the user provides the sentence, the system parses the input sentence according to the PCFG rules based on the mentioned parsing method. The parser shows while the edges are added to the chart and also it displays the process of inserting constituents into the Most Likely Constituent (MLC) table. The system asks if the user wants to draw the parse, if so the parser generates the most probable tree representation of the input sentence. Then the system again prompts if the user wants to print parses, if so, the system provides the parses of the input string and parse tree probability to the user. The system asks if the user wants to continue and lets the user provide additional sentences to be parsed.

A sentence is given as an input to the parser for parsing. For example, if the user inputs the following Tigrigna sentence:

‘ስደተኛታት በርማ ኣብ ባንግላዲሽ ንጥምየት ተቓሊዖም(sdeteNatat berma ^ab bangladx nTmEt teqali`om)’.

Figure 5.1 shows edges as they are added to the chart, and shows the probability for each edges’ tree

```
|[-] . . . . .| [0:1] 'sdeteNatat'          [1.0]
 |. [-] . . . .| [1:2] 'berma'              [1.0]
 |. . [-] . . .| [2:3] '^ab'                [1.0]
 |. . . [-] . .| [3:4] 'bangladx'           [1.0]
 |. . . . [-] .| [4:5] 'nTmEt'              [1.0]
 |. . . . . [-]| [5:6] 'teqali`om'          [1.0]
```

Figure 5. 1 Adding an Edge to the Chart

The output of the parser shows how each constituent is added to the chart, and also it shows the process while the system is trying to find the most likely constituents spanning each portions of the text until the entire table has been filled. The designed parser also outputs the probability of the selected parse structure and the parse result. For the sentence above, the output of the parser looks as follows.

```
  Inserting tokens into the most likely constituents table...
     Insert: |=.....| sdeteNatat
     Insert: |.=....| berma
     Insert: |..=...| ^ab
     Insert: |...=..| bangladx
     Insert: |....=.| nTmEt
     Insert: |.....=| ^aqali`om
Finding the most likely constituents spanning 1 text
elements...
   Insert: |=.....| N -> 'sdeteNatat' [0.019]
0.0190000000
   Insert: |=.....| NP -> N [0.315]
0.0059850000
   Insert: |.=....| N -> 'berma' [0.01]
0.0100000000
   Insert: |.=....| NP -> N [0.315]
0.0031500000
   Insert: |..=...| PREP -> '^ab' [0.777]
0.7770000000
   Insert: |...=..| N -> 'bangladx' [0.019]
0.0190000000
```

```
   Insert: |...=..| NP -> N [0.315]
0.0059850000
   Insert: |....=.| NPREP -> 'nTmEt' [0.261]
0.2610000000
   Insert: |.....=| V -> 'teqali`om' [0.053]
0.0530000000
```

Figure 5. 2 Filling in the Most Likely Constituent Table

The probability of the parse structure: (p=2.6495742470194837e12)

The parse result:

```
S((NP (N sdeteNatat))
  (VP(NP (N berma))
   (VP(PP (PREP ^ab) (N bangladx))
      (VP (NPREP nTmEt) (V ^aqali`om)))))
```

The parse tree structure for the sentence ‘ስደተኛታት በርማ ኣብ ባንግላዲሽ ንጥምየት ተቓሊያም(sdeteNatat berma ^ab bangladx nTmEt ^aqali`om)’.

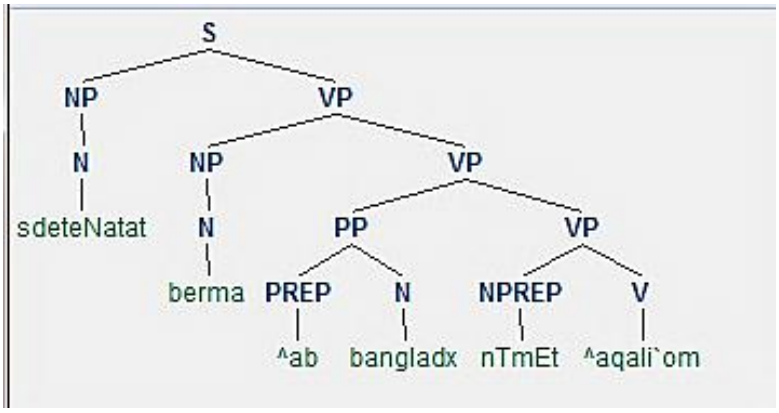

Figure 5. 3 Parse tree Structure of a Sentence

5.4 Evaluation Metrics and Techniques

Cross validation is a model evaluation method used in this study. The type of cross validation selected for the purpose of this study is random permutation cross-validation. This method generates a user defined number of independent training and testing data set splits. The sample corpus is first mixed and then split into a pair of train and test sets. This method

provides a finer control on the proportion of samples on each pair of train/test [55]. Accordingly, the probability calculations for the words in the sample, grammar rule induction, and probability assignment to the grammar rules, which were discussed in Chapter Four, are conducted first on the 110 simple sentences and finally on the 100 complex sentences from the total of 262 sample corpus. The remaining 52 sentences from the original corpus are used as a test corpus. The first test is done on the 210 sentences which were used in the PCFG induction. Three more tests are made using the training set, a randomly chosen set from the sample corpus and a set that was not included in the sample. The output of the parser for these sets is listed under Appendix H.

There is several evaluation techniques proposed for parsers. The evaluation technique used for the purpose of this study is the fraction of the constituents that match between the system's parse tree and human (language expert) parse. This is done by counting the number of correctly parsed sentences out of the whole test set divided by the total number of test set provided. The correctness of the parsed sentences was decided by comparing the parsed sentences of the designed parser to the hand parsed sentences by two linguist expert. The reason to select such technique is to reduce the complexity of the evaluation time. So we have found this technique to be easier than the other proposed techniques.

5.5 Experimental Results and Discussion

Since all sentences on the first test set were not correctly parsed, the first training set was found to have an accuracy of **95.5%**. This is due to the fact that 105 of the 110 simple Tigrigna sentence were correctly parsed when compared to the hand parsed sentences by the linguist. The error on the five miss parsed was due to wrongly tagged word in the lexicon of the training set. Here are the wrongly parsed sentences. The detail parsed results are listed under Appendix H.

The simple Tigrigna sentence ካሕሳይ ምስሑ ይበልዕ ኣሎ(kaHsay msHu ybel` alo)(kahsay is eating his lunch) was wrongly parsed as:

S (NP (N kaHsay)
VP((NP(N msHu)(V ybel`))(AUX ^alo))

The correct parse is given as the parse below after rule replacement

S (NP (N kaHsay)

VP((NP(N msHu)(VREL ybel`))(AUX ^alo))

The manually replaced rules is NP→N VREL instead of the rule NP→N V

These are the miss tagged words in some of the miss parsed sentences. The word ‘alo’is tagged as word tag **V** instead of **AUX** in some sentence, The word ‘neyre’/’I was ’is given the tag set **V** instead of **AUX,** the word **‘**ybel`’/’who was eating’ is given the tag set **V** in instead of **VREL** and ‘des’/’fine’ is in some of the sentence tagged as **V** instead of **ADJ.** After having such errors the corpus was reviewed again and again by both linguist experts and was suggested be valid according to the tags identified. After this error was manually corrected, the result was found to be perfect on the first set, which was 100%.

The result obtained on the remaining 32 test set for the simple sentences was found to be promising, it was found to be 94% on the second test sets.

$$\text{Accuracy} = \left(\frac{30}{32}\right) \times 100 \sim 94\ \%$$

The two miss parsed sentences were due to an error occurred on the lexicon. The prepared lexicon contains error in the part of speech tag of the two words. the errors were corrected and results were attained.

These are the words Hadani(hunter) is given the tag set **N** instead of **ADJ** and Sekaram(drunk) is given the tags set **N** instead of **ADJ.** The miss tagged words were later correctly tagged and again the test sets were given to the designed parser and the result found was ideal that is 100%.

This encouraging result was found since the rules collected from the training corpus appeared to be perfect enough to support all the sentences in the test set, the sentences have close type of construct and probabilities were found to be precise enough to determine the true parse tree structure by calculating the inside probability and Viterbi PCFG parser to find the most likely parse among the possible parses for the given sentence.

Another 20 set was collected by the researcher and other people out of the corpus. These sentences were of the same construct with the training set and the words of the sentence were formed from the words of the lexicon which was used earlier. These sentences were given to the parser and the parser parsed 17 of them correctly, attaining an accuracy of 85%. The two wrongly parsed sentences were due to the error in the lexicon, which they were given wrong part of speech tag. The mistagged are words in the lexicon that caused the sentence to be misparsed are listed below.

N→ Sekaram

N→ ztegod^u' and N→ Hadani'

Besides an experiment was held on the complex Tigrigna sentences and a promising result was found by using the same evaluation technique used for evaluating the simple Tigrigna sentences.

Even though the sentences taken were not uniform in their construct, the accuracy found on the first training set was good, which was 91%. This result was found since 91 of the 100 complex sentences were correctly parsed compared to the hand parsed sentences by the linguist.

Accuracy = $(\frac{91}{100}) \times 100 = 91\%$

This result was achieved due to the fact that most of the words in the sentences were not tagged correctly and lexicon was somehow not fully prepared for each word in the corpus with associated probability in addition to the grammar rules. Thus, the reason for the nine wrongly parsed sentences was due to lack of grammar rule that could generate the correct parse and miss tagged words in the lexicon. This missed grammar rules were added manually and the error was corrected. Besides the miss tagged words were also corrected by hand. So the result found after adding the grammar rule and correcting the tag sets was ultimate, which was 100%.

One of the wrongly parsed sentence and the missed grammar rules are demonstrated below.

The sentence ኣብ ወረዳ ጸገዴ ትርከብ ዩንየን ማሕበር ልቓሕ ረኺባ(^ab wereda ^SegedE trkeb yunyen maHber lQaH reKiba) was wrongly parsed as.

S (PP (PREP ^ab) (N wereda)

VP((NP(N ^SegedE)(V trkeb)(NP(N yunyen))(VP(N maHber)(VP(N lQaH)(V reKiba))))

The reason for miss parse of this sentence was due to the lack of a probabilistic context free grammar rule (PCFG) NP → V N [0.004]. After adding this rule (PCFG) the parser parsed the sentence correctly as follows.

S (PP (PREP ^ab) (N wereda)

VP((NP(N ^SegedE)(NP(V trkeb)(N yunyen))(VP(N maHber)(VP(N lQaH)(V reKiba))))

Other missparsed sentences arose due to error in the tag of the lexicon (e.g., the word tmali(yesterday) was wrongly tagged as N in some sentence of the sample corpus).

The sentence ሓላፊ እቲ ቦርድ ኣቶ ያይንሸት ትማሊ ሞይቶም(Halafi ^ti bord ato yaynxet tmali moytom)(head of the bord Mr yaynshet died yesterday) was wrongly parsed as

S(NP(N Halafi)(NP(DET ^ti)(N bord))
VP((NP(ADJ ^ato)(NP(N yaynxet))(VP(N tmali)(V moytom))))

After correcting the error the parser parsed the sentence exactly as

S(NP(N Halafi)(NP(DET ^ti)(N bord))
VP((NP(ADJ ^ato)(NP(N yaynxet))(VP(ADV tmali)(V moytom))))

The corrected tag in the sentence was to tag the word ‘tmali’(yesterday) as a tag set ADV instead of N, which intern resulted in replacing the rule VP→ $N\ V$ by the rule VP→ $ADV\ V$.

An experiment is also held on the second test sets containing 20 complex sentences and the result was found to be hopeful, which was 90%. This promising result was achieved because of the existence of all the words on the lexicon with associated probability, and the occurrence of good PCFG rules that can parse the sentence. 18 of them were correctly parsed as compared to the handy parsed sentence. A summary for the various experiments conducted is illustrated in tabular form in Table 5.1. Another set out of the two sets which

contains 10 complex sentences was collected and was given to the parser and the parser attained a good result of 80%. In this case some of the words in the third test set were not in the grammar lexicon because the lexicon doesn't contain all words of the language. Thus, the parser cannot parse a string in which the morphemes are not contained in the grammar lexicon.

Table 5. 1 Experimental Summary

Sets	Sentence Type	
	Simple Sentences	Complex Sentences
Test set_1	95 %	91 %
Test set_2	94 %	90 %
Test set_3	85 %	80 %

5.6 Solution and Suggestion to the Recognized Problems

Several problems were encountered in the experimental findings most of them were due to mistagged words in the lexicon, the absence of the words in the lexicon and the lack of PCFG rules that can parse the sentence. To come up with the absence of the words in the lexicon there must have been a fully prepared lexicon that could contain all the words of the language. But it is very difficult to have a lexicon that can contain all words all words of a language. The other problem which is mistagged words was held manually since we did not integrate any part of speech tagger (POST) that can correctly tag each and every word in the sample corpus. Thus, an efficient POST for Tigrigna language is recommended to improve the performance of the parser.

The other problem is the problem of misparsing which occurs due to missing rules, when a parser is presented with new sentence, deficiency in the syntactic rules or lexicon. In such cases the correct analysis cannot be assigned. To deal with this problem a grammar was developed iteratively and PCFG was done again and again, additional PCFG rules were added, and the miscarried sentences were re- analyzed.

Chapter 6: Conclusion and Recommendation

6.1 Conclusion

The purpose of this thesis is to design and develop an automatic sentence parser for simple and complex Tigrigna sentences using bottom up PCFG chart parsing. The study began with brief discussion on Tigrigna language behavior, a discussion on Tigrigna language, Tigrigna grammar (i.e., which includes sub sections Tigrigna word class, Tigrigna phrases and Tigrigna sentence), the different grammar formalisms proposed so far and finally the tools for parsing, approaches to parsing and parsing strategies are discussed. Accordingly, an approach and strategy is selected based on the review which is appropriate to the language. After detail review on the major approaches it was concluded that the bottom up PCFG chart parser is selected to develop the Tigrigna parser. The reason for the selection of this approach was clearly stated.

The design of the parser needs corpus collection. Thus, we collected sample corpora both simple and complex Tigrigna sentences from different sources by using technique and the corpus was preprocessed. An architectural design of the parser is proposed with the commonly used components. Each component of the system architecture is discussed and the interaction between other components is shown. The components of the architecture include components like corpus preprocessing, lexicon generation, morphological analysis and tokenization. The different algorithms and pseudo codes adapted in the study are shown. Manual tagging and parsing process was held by two language experts from the Tigrigna language department. The grammar rule induction, the probability assignment to the rules, preparation of the lexicon and the selection of the most probable parse among the various parses for a sentence was done on both types of the selected sample corpus. The PCFG chart parsing algorithm is adapted for the purpose of the study. The algorithm was employed to return all possible parse structure for a given sentence with their associated probability of each possible parse. The Viterbi parse algorithm is adapted for the selection of most probable parse among the possible parses for a given sentence.

We conducted three experiments. The first test is from the training set and the second test is done on test sets from the sample corpora which were selected for experimental purposes.

The third sets are apart from the two sets which are selected outside the sample corpus. Cross validation is a model evaluation method used in this study. Random permutations cross validation is the type of cross validation we selected. This method provides finer control on the proportion of samples on each pair of train and test. Performance evaluation technique used for the purpose of this study is the fraction of the constituents that match between the system's parse and human (linguist) parse. This technique is employed for the purpose of simplicity. The experiment was iteratively done on the three set for both simple and complex sentences by detecting and correcting the errors on each sets. The result achieved based on the first set of sample sentences was good, which was 95.5% and approximately 94% on the second set. The result achieved on the third set was 85%. Besides experiments was done on the other type of sample sentences, the complex sentences. The reason is the grammar rule induction and probability calculation for these sentences is taken differently. Based on the experiments, result has been estimated and experimental results have been done repeatedly. Accordingly, on the first test set the result achieved was 91%. Another experiment was done on the rest 20 test sets from the sample corpus of the complex sentences. The result achieved was good, was 90% because of the existence of many the words on the lexicon with associated probability and the occurrence of full PCFG rules that can parse the sentence. The result found on the third set containing 10 complex sentences out of the sample corpora was also promising, had an accuracy of 80%. From the test results of the conducted experiments in this research, it can be concluded that, the method we selected showed a good result.

Due to lack of large Annotated corpora, lack of willing or responsible language experts to do the manual work, time constraint, financial constraints the grammar rule induction and the PCFG extraction was constrained to small corpus. Thus, we had to do many of the work by hand in the given limited time. Having these difficulties, we have developed a sentence parser as an initial work in the area of probabilistic chart parsing for Tigrigna language.

6.2 Contribution of the Thesis

This research work contributes the following.

- Since this is the first work on Tigrigna parsing, we contributed the architecture for Tigrigna parsing

- In this research work, we have showed as there are available techniques, approaches and algorithms that can be used to develop Tigrigna parsing
- In this research work, a tag set for Tigrigna apart is proposed for the different word classes.
- A lexicon for different inflections and derivations of a word is prepared using the lexicon generation.

6.3 Recommendation

Even though the parser developed in this study has showed some encouraging results, it needs additional effort to improve its accuracy so that it can handle any sentence in the language.

Currently there are many hot topics in the area of NLP that can be done for different Ethiopian languages. Even though Tigrigna language is morphologically complex, has no sufficient references and there are no many research studies done on it previously and unavailability of an annotated corpus large enough to experiment statistical systems. Many works in the language is expected for future researchers in the language. Thus, to help these NLP researchers in Tigrigna language, it will be vital if a standard corpus for Tigrigna language is prepared.

Thus, as a continuation of this study we recommend the following future works.

- Integrate an efficient Tigrigna automatic part of speech tagger (POST) in order to improve its performance.
- Enhance it to be a full-fledged parser that accepts any sentence in the language.
- Conduct an experiment using a large set of corpora.
- Induce a large set of grammar which is a representative of the language.
- Advance the developed parser by preparing a Lexicon that contains all words of the language
- The grammar rules, probabilities of the rules, probability of the words in the corpus are static values. We suggest to make them dynamic may be by adding the words into the lexicon re-calculate the probabilities and add new grammar rules if needed during the parsing process

References

[1] W. Shuly, *Natural Language Processing*. University of Haifa Press, Israel, 2009.

[2] Materials from Uppsala University, "Introduction to Parsing", Materials, 2015, Retrievedfrom:http://demo.clab.cs.cmu.edu/fa201511711/images/1/17/IntroToParsing.pdf,last accessed on, June 5, 2016.

[3] J. Allen, *Natural Language Understanding*, the Benjamin/Cummings Publishing Company Inc, San Francisco.1995.

[4] A. Tayal, M. Raghuwanshi and L. Malik, "Syntax parsing: implementation using Grammar rules for English language", Electronic Systems, Signal Processing and Computing Technologies (ICESC), 2014 International Conference, IEEE Computer Society Washington, DC, USA , Jan 2014.

[5] Tutorial Point Simply Easy Learning "AI - Natural Language Processing", Tutorial, retrieved from https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_natural_language_processing.htm, Last accessed on Nov 10, 2017.

[6] Ethiopian Central Statistical Agency, "Population *and Housing Census Report-Tigray Region*", retrieved from http://www.csa.gov.et/index.php/census-report/complete-report/census-2007, last accessed on June 1, 2016.

[7] Yonas Fisseha, **"***Development of Stemming Algorithm for Tigrigna Text***",** Unpublished Master Thesis, Department of Information Science**,** Addis Ababa University, Addis Ababa, 2011.

[8] Teklay Gebregzabiher**, "***Part of Speech Tagger for Tigrigna Language***",** Unpublished Master Thesis, Department of Computer Science, Addis Ababa University, Addis Ababa, 2010

[9] Gebrehiwot Assefa, "*A Two Step Approach for Tigrigna Text Categorization"*, unpublished Master Thesis, Department of Information Science, Addis Ababa University, Addis Ababa, *2011.*

[10] Hafte Abera, *"Hidden Markov Model Based Tigrigna Speech Recognition"*, Unpublished Master Thesis, Information Science, Addis Ababa University, Addis Ababa, *2009.*

[11] A. Al-Taani, M. Msallam and S. Wedian, “A Top down Parser for Analyzing Arabic Sentences”, *The international Arab journal of information technology*, Vol.9 , No.2, March 2012

[12] B. Bataineh and E. Bataineh, “An Efficient Recursive Transition Network Parser for Arabic Language”, *Proceedings of the World Congress on Engineering*, London, 2009.

[13] Abiyot Bayou, *"Design and Development of Word Parser for Amharic Language"*, Unpublished Master’s Thesis, Addis Ababa University, Addis Ababa, 2000

[14] Atelach Alemu, *"Automatic Sentence Parsing for Amharic Text an Experiment using Probabilistic Context Free Grammar"*, Unpublished Master Thesis, School of Information Studies for Africa, Addis Ababa University, Addis Ababa. 2002

[15] Daniel Gochel, *"An integrated approach to automatic complex sentence parsing for Amharic text”*, Unpublished Master Thesis, School of Information Studies for Africa, Addis Ababa University, Addis Ababa, 2003.

[16] Diriba Megersa, *"Automatic Sentence Parser for Oromo Language Using Supervised Learning Technique"*, Unpublished Master Thesis, School of Information Studies for Africa, Addis Ababa University , Addis Ababa,2002

[17] Institute for Computational Linguistics, “National Research Council of Italy”, retrieved from http://www.ilc.cnr.it/EAGLES96/rep2/node41.html, last accessed on August 15, 2017.

[18] Ominglot the Online Encyclopedia of Writing Systems and Languages, “Tigrigna (ትግርኛ)”, retrieved from http://www.omniglot.com/writing/Tigrigna .htm, last accessed on June 1, 2016.

[19] M. Gasser. “Semitic Morphological Analysis and Generation Using finite State Transducers with Feature Structures”, *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, Athens, Greece — March 30 - April 03, 2009

[20] Daniel Teklu. Tigrigna modern grammar (*ዘበናዊ ሰዋሰው ቋንቋ ትግርኛ*), Mega printing enterprise, Addis Ababa. 2000.

[21] Kassa Gebrehiwet, Tigrigna grammar (*ሰዋሰው ትግርኛ*), Mega printing enterprise, Addis Ababa. 2004.

[22] Baye Ymam. Amharic grammar (*የአማርኛ ሰዋሰው*). EMPDA. Addis Ababa.1987

[23] J. Mason. *Tigrigna Grammar*. Red Sea Press Inc.1996

[24] A. Smith**, "**Novel Estimation Methods for Unsupervised Discovery of Latent Structure in Natural Language Text", A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy, Baltimore, Maryland October, 2006

[25] T. Chen and C. Tseng and C. Chen, "Automatic Learning of Context-Free Grammar", in proceedings of ROCLING, Taiwan, 2006.

[26] K. Johnson, "Introduction to Transformational Grammar", University of Massachusetts at Amherst, 2007

[27] W. Woods. "Transition Network Grammars of natural Language Analysis". Communication of the ACM, Vol.13, No.10, pp.500-591,1970

[28] B. Neveln, and B. Alps, "Unification-Based Grammar", Proofcheck.org, 2011

[29] J. Nivre, "Parsing with PCFGs", Semanticscholar.org, 2013.

[30] A. Vijayan, "Context Sensitive Grammar", Department of Computer Science and Automation 2011

[31] A. Joshi. "Parsing Techniques*", in Proceedings of the 1987 Workshop on Theoretical Issues in Natural Language Processing*, Cambridge University press, Cambridge, 1998.

[32] T. Fujisaki, F. Jelinek, J. Cocke, E. Black, and T. Nishino. A probabilistic parsing method for sentence disambiguation, Kluwer Academic Publishers, Boston. 1991

[33] A. Jones and M. Eisner. "A probabilistic parser and its applications*"*. In *AAAI Workshop on Statistically-Based NLP Techniques*, San Jose, CA. 1992

[34] H. Ney. "Stochastic grammars and pattern recognition*", in Proceedings of the NATO Advanced Study Institute*, Cetraro, Italy, 1992.

[35] K. Lari and J. Young. "Applications of stochastic context-free grammars using the Inside-Outside algorithm*". Journal of Computer Speech and Language*,Vol.4, No., pp.35-56, 1991

[36] Y. Yao, K. Lua. "A Probabilistic Context-Free Grammar Parser for Chines*", journal of Computer Processing of Oriental Languages*, Vol. 11, No. 4, pp. 393-407, 1998.

[37] Tutorials on PCFGs "Probabilistic Context frees Grammars (PCFGs*)"*.Lecture Notes, 2009,retrived from: https://courses.cs.washington.edu/courses/cse590a/09wi/pcfg.pdf, last accessed on Sept. 2016

[38] V. Alfred. *Compilers principles, techniques and tools*, Pearson Publishing Company, London, 1986.

[39] S. Bird, E. Klein & E. Loper , "Chart Parsing and Probabilistic Parsing", in Introduction to Natural Language Processing (DRAFT), 2008.

[40] Department of Linguistics of Indiana University, "PCFGs", notes, 2015, retrieved from http://cl.indiana.edu/~md7/15/645/slides/11-pcfgs/11b-pcfg1-2x3.pdf, last accessed on Nov 7, 2017.

[41] Mesfin Getachew. "*Automatic Part of Speech Tagging for Amharic Language: An Experiment Using Stochastic Hidden Markov (HMM) Approach"*, Unpublished Master Thesis, School of Information Studies for Africa, Addis Ababa university, Addis Ababa, 2001.

[42] M. O'Donnell, Sentence Analysis and Generation -- a Systemic Perspective, Ph.D. Dissertation, Linguistics Department, University of Sydney, 1994

[43] Institute for Computational Linguistics <<A. zampolli>> National Research Council of Italy, retrieved from http://www.ilc.cnr.it/EAGLES96/rep2/node41.html, last accessed on August 18, 2017.

[44] H. Liang and K. Sagae, "Dynamic programming for linear-time incremental parsing." In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 1077-1086. Association for Computational Linguistics, 2010

[45] A. *Clark, C. Fox, and S. Lappin* "The Handbook of Computational Linguistics and Natural Language Processing", A John Wiley & Sons, Ltd. Publication, 2010.

[46] Mesfin Getachew. "*Automatic Part of Speech Tagging for Amharic Language: An Experiment Using Stochastic Hidden Markov (HMM) Approach"*, Unpublished Master Thesis, School of Information Studies for Africa, Addis Ababa university, Addis Ababa, 2001

[47] B. Eric. "Transformation-Based Error-Driven Parsing". *Spoken Language Systems Group Laboratory for Computer Science,* Vol.21, No.4, 1993

[48] B. Eric and M. Pop, "Unsupervised Learning of Disambiguation Rules for part of Speech Tagging", *Proceedings of the third workshop on very large,* Vol.30, No., 1995

[49] E. Charniak "Statistical Parsing with a Context-free Grammar and Word Statistics", *journal of AAAI/IAAI,* Vol.2005, No.18, pp.598-603, July 1997

[50] P. Marcus, B. Santorini and M. Marcinkiewicz, "Building a large annotated corpus of English: the Penn Treebank", *Journal of Association for Computational Linguistics - Special issue on using large corpora,* Vol.19, No.2,pp.313-330, 1993

[51] M. Gasser, "a System for Morphological Processing of Amharic, Oromo, and Tigrigna ", *In Conference on Human Language Technology,* Indiana University, Alexandria, 2011

[52] S. Bird, E. Klein, and E. Loper., N*atural language processing with python*, O'Reilly Media Inc., California. 2009.

[53] Natural Language Toolkit, "Viterbi Probabilistic Parser", python document, retrieved from: http://www.python.org, last accessed on May 15, 2016.

[54] B. Steven, "NLTK: the natural language toolkit**",** *Proceedings of the COLING/ACL on Interactive presentation sessions Association for Computational Linguistics,* Sydney, Australia — July 17 - 18, 2006

[55] UMN School of Statistics, "*Cross Validation for Selecting a Model*", Retrieved from http://users.stat.umn.edu/papers/ACV_v30, last accessed on July 20, 2017.

Appendices

Appendix A. Ethiopic/Ge'ez Script

Name	IPA	e	u	i	a	ie	ɨ/-	o
pa	p	ፐ	ፑ	ፒ	ፓ	ፔ	ፕ	ፖ
ba	b	በ	ቡ	ቢ	ባ	ቤ	ብ	ቦ
pha	p'	ጰ	ጱ	ጲ	ጳ	ጴ	ጵ	ጶ
ma	m	መ	ሙ	ሚ	ማ	ሜ	ም	ሞ
fa	f	ፈ	ፉ	ፊ	ፋ	ፌ	ፍ	ፎ
va	v	ቨ	ቩ	ቪ	ቫ	ቬ	ቭ	ቮ
wa	w	ወ	ዉ	ዊ	ዋ	ዌ	ው	ዎ
ta	t	ተ	ቱ	ቲ	ታ	ቴ	ት	ቶ
da	d	ደ	ዱ	ዲ	ዳ	ዴ	ድ	ዶ
tha	tʼ	ጠ	ጡ	ጢ	ጣ	ጤ	ጥ	ጦ
tsa	ts'	ጸ	ጹ	ጺ	ጻ	ጼ	ጽ	ጾ
na	n	ነ	ኑ	ኒ	ና	ኔ	ን	ኖ
sa	s	ሰ	ሱ	ሲ	ሳ	ሴ	ስ	ሶ
za	z	ዘ	ዙ	ዚ	ዛ	ዜ	ዝ	ዞ
ra	r	ረ	ሩ	ሪ	ራ	ሬ	ር	ሮ
la	l	ለ	ሉ	ሊ	ላ	ሌ	ል	ሎ
ca	ʧ	ቸ	ቹ	ቺ	ቻ	ቼ	ች	ቾ
ja	ʤ	ጀ	ጁ	ጂ	ጃ	ጄ	ጅ	ጆ
cha	ʧ	ጨ	ጩ	ጪ	ጫ	ጬ	ጭ	ጮ

Name	IPA	e	u	i	a	ie	ɨ/-	o
nya	ɲ	ኘ	ኙ	ኚ	ኛ	ኜ	ኝ	ኞ
sha	ʃ	ሸ	ሹ	ሺ	ሻ	ሼ	ሽ	ሾ
zha	ʒ	ዠ	ዡ	ዢ	ዣ	ዤ	ዥ	ዦ
ya	j	የ	ዩ	ዪ	ያ	ዬ	ይ	ዮ
ka	k	ከ	ኩ	ኪ	ካ	ኬ	ክ	ኮ
kwa	kw	ኰ		ኲ	ኳ	ኴ	ኵ	
ga	g	ገ	ጉ	ጊ	ጋ	ጌ	ግ	ጎ
gwa	gw	ጐ		ጒ	ጓ	ጔ		
qa	k'	ቀ	ቁ	ቂ	ቃ	ቄ	ቅ	ቆ
qwa	kw'	ቈ		ቊ	ቋ	ቌ	ቍ	
kxa	x	ኸ	ኹ	ኺ	ኻ	ኼ	ኽ	ኾ
kxwa	xw	ዀ		ዂ	ዃ	ዄ	ዅ	
qha	ʁ'	ቐ	ቑ	ቒ	ቓ	ቔ	ቕ	ቖ
qhwa	ʁw	ቘ		ቚ	ቛ	ቜ	ቝ	
hha	ħ	ሐ	ሑ	ሒ	ሓ	ሔ	ሕ	ሖ
ʾä	ʕ	ዐ	ዑ	ዒ	ዓ	ዔ	ዕ	ዖ
ʾä	ʔ	አ	ኡ	ኢ	ኣ	ኤ	እ	ኦ
ha	h	ሀ	ሁ	ሂ	ሃ	ሄ	ህ	ሆ

Appendix B. List of Punctuation Marks

Tigrigna	Punctuation marks	English
።	Full stop	.
፣	Comma	,
፤	Colon	:
፥	Preface-colon	:
፦	Semi colon	;
፧	Question mark	?

Appendix C. The Dictionary used for Transliterating Tigrigna to Latin

First order		Second order		Third order		Fourth order		Fifth order		Sixth order		Seventh order	
ሀ	he	ሁ	Hu	ሂ	hi	ሃ	ha	ሄ	hE	ህ	h	ሆ	ho
ለ	le	ሉ	Lu	ሊ	li	ላ	La	ሌ	lE	ል	l	ሎ	lo
ሐ	He	ሑ	Hu	ሒ	Hi	ሓ	Ha	ሔ	HE	ሕ	H	ሖ	Ho
መ	me	ሙ	Mu	ሚ	mi	ማ	ma	ሜ	mE	ም	m	ሞ	mo
ሠ	^se	ሡ	^su	ሢ	^si	ሣ	^sa	ሤ	^sE	ሥ	^s	ሦ	^so
ረ	re	ሩ	Ru	ሪ	ri	ራ	Ra	ሬ	rE	ር	r	ሮ	ro
ሰ	se	ሱ	Su	ሲ	si	ሳ	Sa	ሴ	sE	ስ	s	ሶ	so
ሸ	xe	ሹ	Xu	ሺ	xi	ሻ	Xa	ሼ	xE	ሽ	x	ሾ	xo
ቀ	qe	ቁ	Qu	ቂ	qi	ቃ	Qa	ቄ	qE	ቅ	q	ቆ	qo
ቐ	Qe	ቑ	Qu	ቒ	Qi	ቓ	Qa	ቔ	QE	ቕ	Q	ቖ	Qo
በ	be	ቡ	Bu	ቢ	bi	ባ	Ba	ቤ	bE	ብ	b	ቦ	bo
ቨ	ve	ቩ	Vu	ቪ	vi	ቫ	Va	ቬ	vE	ቭ	v	ቮ	vo
ተ	te	ቱ	Tu	ቲ	ti	ታ	Ta	ቴ	tE	ት	t	ቶ	to
ቸ	ce	ቹ	Cu	ቺ	ci	ቻ	Ca	ቼ	cE	ች	c	ቾ	co
ኀ	Ke	ኁ	Ku	ኂ	Ki	ኃ	Ka	ኄ	KE	ኅ	K	ኆ	Ko
ነ	^he	ኑ	^hu	ኒ	^hi	ና	^ha	ኔ	^hE	ን	^h	ኖ	^ho
ኘ	ne	ኙ	Nu	ኚ	ni	ኛ	Na	ኜ	nE	ኝ	n	ኞ	no
ኘ	Ne	ኙ	Nu	ኚ	Ni	ኛ	Na	ኜ	NE	ኝ	N	ኞ	No
አ	^e	ኡ	^u	ኢ	^i	ኣ	^a	ኤ	^E	እ	^	ኦ	^o
ከ	Ke	ኩ	Ku	ኪ	ki	ካ	Ka	ኬ	kE	ክ	k	ኮ	ko
ወ	We	ዉ	Wu	ዊ	wi	ዋ	Wa	ዌ	wE	ው	w	ዎ	wo
ዐ	'e	ዑ	'u	ዒ	'i	ዓ	'a	ዔ	'E	ዕ	'	ዖ	'o
ዘ	Ze	ዙ	Zu	ዚ	zi	ዛ	Za	ዜ	zE	ዝ	z	ዞ	zo
ዠ	Ze	ዡ	Zu	ዢ	Zi	ዣ	Za	ዤ	ZE	ዥ	Z	ዦ	Zo
የ	Ye	ዩ	Yu	ዪ	yi	ያ	Ya	ዬ	yE	ይ	y	ዮ	yo
ደ	De	ዱ	Du	ዲ	di	ዳ	Da	ዴ	dE	ድ	d	ዶ	do
ጀ	je	ጁ	ju	ጂ	ji	ጃ	Ja	ጄ	jE	ጅ	j	ጆ	jo
ገ	Ge	ጉ	Gu	ጊ	gi	ጋ	Ga	ጌ	gE	ግ	g	ጎ	go
ጠ	Te	ጡ	Tu	ጢ	Ti	ጣ	Ta	ጤ	TE	ጥ	T	ጦ	To
ፈ	Ce	ፉ	Cu	ፊ	Ci	ፋ	Ca	ፌ	CE	ፍ	C	ፎ	Co
ጰ	Pe	ጱ	Pu	ጲ	Pi	ጳ	Pa	ጴ	PE	ጵ	P	ጶ	Po
ፀ	^Se	ፁ	^Su	ፂ	^Si	ፃ	^Sa	ፄ	^SE	ፅ	^S	ፆ	^So
ጸ	Se	ጹ	Su	ጺ	Si	ጻ	Sa	ጼ	SE	ጽ	S	ጾ	So
ፈ	Fe	ፉ	Fu	ፊ	fi	ፋ	Fa	ፌ	fE	ፍ	f	ፎ	fo
ፐ	Pe	ፑ	Pu	ፒ	pi	ፓ	Pa	ፔ	pE	ፕ	p	ፖ	po
ሏ	hWa	ሟ	mWa	ሧ	^sWa	ሷ	sWa	ቧ	xWa	ቈ	qWe	ቍ	qWi
ኳ	qWa	ቋ	qWE	ቍ	qW	ቈ	Qwa	ቌ	QWE	ቍ	QW	ቧ	vWa
ጇ	cWa	ጎ	^hWi	ጓ	^hWa	ጔ	^hWE	ጎ	^hW	ሯ	TWa	ፗ	pWa
ኩ	kWi	ኳ	kWa	ኴ	kWE	ኵ	kW	ኹ	Kwi	ኻ	KWa	ኼ	KWE
ዅ	KW	ዟ	zWa	ዷ	dWa	ጇ	jWa	ጐ	gWe	ጒ	gWi	ጓ	gWa
ጔ	gWE	ጒ	gW	ኈ	HWa	ሯ	rWa	ቈ	Qwe	ቧ	bWa	ፘ	tWa
ጎ	ʰWa	ኗ	nWa	ኟ	NWa	ኰ	kWe	ኽ	Kwe	ኺ	ZWa	ፏ	CWa
ፏ	PWa	ፗ	Swa	፯	fWa	።	.	፣	,	፤	;	፥	:

Appendix D. Transliterated Sample Corpus

Sample Training set for Simple Tigrigna Sentences

wetaderat nab Tornet keydom .

lemlem nberhe nsraH ^Sewi`ato .

Hayelom n^xetey ^njera beli`u .

berhe gobez memhr ^yu .

yaynxet n^xtey geza ^aleto .

sebat leyti ^ab`arat ydqsu .

kaHsay nlemlem genzeb hibwa .

meles ^ab geza deqisu .

^taQol`a bTa`mi ^SbQti ^ya .

beleTe nab CQa wediQu .

Sample Training set for Complex Tigrigna Sentences

^ab wereda welqayt ^Sryet tmhrti tegeli^Su

sdeteNatat berma ^ab bangladx nTmEt ^aqali`om

cayna ntaywan kemHanti gz^at tQo^Sra ^ala

meraKebti Hafax gWesgWas mure^Suna tmali akaydom

^ab ^adis^abeba yuniversti din myyT kgber ^yu

wetaderawi sr`at berma bzf^Smo Cqonan teKesisu

^akabi ^ti bord profEser merga tmali moytom

gazETeNa ^Seb^Sab ^ab zeQrbelu ^wan myyT gEru

^zom sraHti lm`at tezazimom nglgalot beQi`om

^ab kll ^amHara ferenji lm`at ^akaydom

prEzdant gram bzuHat nayseraH `dlat ^alewu ^ilom

Sample Test set for Simple Tigrigna Sentences

^astEr n^xtey `arat gezi^a

nay Hawey `amet ^atyu

kaHsay n^xtey `tro gezi^u

babur tmali sraH tejemiru

^ti memhr me`ar hibuni

^ti memhr Ha^Sir ^yu

^tom xmagle seb^ay wediQom

^tom xmagle nberhe weQi`wom

kaHsay nab ^aksum keydu

Qol`u lomi ^ab geza ^alewu

Sample Test Set for Complex Tigrigna Sentences

^ab cayna meraKebti Hafax ^Sryet ^akaydom

wetaderawi prEzdant ^Ertra bzf^Smo Cqonan teKesisu

^tom Harestot l`li 16.8milyon qrxi blQaH tereKibom

^ayte kaHsay ms weyzero gram teraKibom

^abzi `amet ^ab wereda welqayt ^Sryet tmhrti ^akaydom

^ab wereda meQele sraHti lm`at tezazimom nglgalot beQi`om

prEzdant ^Obama ^ab wereda `adwa tereKibom

^ayte kidane ^abzi kremti lm`at tewafirom

me`alti ^Sryet ^ab mela^ `alem teKebiru

kentiba wereda wuQro za`ba ^aQribu

Appendix E. Special Tags Identified in the Study

No	Tag	Explanation
1	N	All pronouns and Inflected nouns to number, gender and case remain the same to the word class noun(N) e.g. ከባቢ/ “place”, ከባቢታት/ “places”, ንሱ /“he”
2	NV	Verbal nouns in Tigrigna like ምብላዕ /“eating”, ምስታይ/“drinking”
3	NPREF	Noun formed by prefixing the prefix “በዓል” to nouns e.g. በዓል-ገዛ/“the owner of the house”
4	NPREP	A nouns with a preposition e.g. ብእግሪ/”on foot”, ብኣየር/by plane
5	NC	A noun suffixed with a conjunction e.g. ማይን ፀባን/ water and milk , ማይከ/what about water
6	AUX	Auxiliary verbs and all their other forms. e.g. ኣሎ/”he ,it present”, ነይሩ/”he, it was”, እዩ/”it is” እያ/”she is” እዮም/”they are”
7	VCOMP	Compound verbs e.g. ፀራጊ-ጫማ/
8	CARDPREP	A cardinal with preposition e.g. ንሓደ/for one, ብፍርቂ/by half ንርብዒ/for quarter
	CARDN	A cardinal number followed by a name e.g. 1ሚልየን//1million
9	VPREP	Any verb headed by a preposition e.g. ተኸይዱ/if he goes ስለዝኸደ/since he goes
10	VCONJ	Any verb suffixed or prefixed by a conjunction. e.g. ይከድእሞ/he goes and, ክሳብዝኸድ/until he goes, ምስከደ/when he goes
11	COMP	Words that are formed by combining two bound morphemes or free morphemes e.g. ፀሊምሰሌዳ/blackboard, ሰራሒጫማ/shoemaker
12	ADJCONJ	An adjective together with a conjunction. E.g. ሓፂርን ቀጢን/short and thin
13	ADJNUM	Numeral that function as an adjective. E.g. ክልተኪሎ/two kilo ሓደማንካ/one spoon
14	ADJPN	A preposition together with a noun that function as an adjective. E.g. ናይቻይና ጫማ/ shoe made in china, ናይማይ ብርጭቆ/a glass for water

15	ADJPREP	The adjective is headed by a preposition. e.g. ብሕማቅ/in a bad way, ብፅቡቕ/in a good way
16	NCONJ	Nouns together with a conjunction. E.g. ሰብኣይን ሰበይትን/husband and wife
17	ADVCONJ	An adverb suffixed by a conjunction .e.g. ሕዚእዉን/even now, ትማሊእዉን/even yesterday
18	CC	Combining conjunction. e.g. ነገርግና/however, ምንምእንኳ/even though
19	VREL	Relative clause. e.g. ኮፍዝበሉ/those who stand-up ዝተወደአ/the one that is finished ዝተሰረቆ/one who is stolen
20	UNR	Unrecognized word whose tag is not set
21	PUNC	?,:,::,፣፤!

Appendix F. Tagged Corpus

Sample Tagged Corpus for Simple Tigrigna Sentences

wetaderat/N nab/PREP Tornet/N keydom/V ./PUNC

lemlem/N nberhe/NPrep nsraH/NPREP ^Sewi`ato/V ./PUNC

Hayelom/N n^xetey/ADJ ^njera/N beli`u/V ./PUNC

berhe/N gobez/ADJ memhr/N ^yu/AUXV ./PUNC

yaynxet/N n^xtey/ADJ geza/N ^aleto/V ./PUNC

sebat/N leyti/N ^ab`arat/NPREP ydqsu/V ./PUNC

kaHsay/N nlemlem/NPREP genzeb/N hibwa/V ./PUNC

kaHsay/N netiseb^ay/NPREP misTir/N negirwo/V ./PUNC

meles/N ^ab/PREP geza/N deqisu/V ./PUNC

^taQol`a/N bTa`mi/ADV ^SbQti/ADJ ^ya/AUX ./PUNC

Sample Tagged Corpus for Complex Tigrigna Sentence

^ab/PREP wereda/N welqayt/N ^Sryet/N tmhrti/N tegeli^Su/V ./PUNC

sdeteNatat/N berma/N ^ab/PREP bangladx/N nTmEt/NPREP ^aqali`om/V ./PUNC

cayna/N ntaywan/NPREP kemHanti/CARDPREP gz^at/N tQo^Sra/VPREP ^ala/V ./PUNC

meraKebti/ADJ Hafax/N gWesgWas/N mure^Suna/N tmali/ADV ^akaydom/V ./PUNC

^ab/PREP ^adis^abeba/N yuniversti/N din/ADJ myyT/N kgber/VPREP ^yu/AUX ./PUNC

wetaderawi/ADJ sr`at/N berma/N bzf^Smo/VPREP Cqonan/N teKesisu/V ./PUNC

^akabi/N ^ti/DET bord/N profEser/ADJ merga/N tmali/ADV moytom/V ./PUNC

gazETeNa/N ^Seb^Sab/N ^ab/PREP zeQrbelu/VPREP ^wan/ADV myyT/N gEru/V ./PUNC

^zom/DET sraHti/N lm`at/N tezazimom/V nglgalot/NPREP beQi`om/v ./PUNC

^abzi/PREP `amet/N ferenji/N ^ab/PREP kll/N ^amHara/N lm`at/N ^akaydom/V ./PUNC

Appendix G. Probabilistic Context Free Grammar (PCFG) Extracted

PCFG Extracted for Sample Tigrigna Sentences

RULE	COUNT	TOTAL	PROBABILITY
S → NP VP	96	110	0.873
S → PP VP	14	110	0.127
NP → ADJ N	22	128	0.172
NP → N ADV	3	128	0.023
NP → N N	2	128	0.015
NP → CARD N	1	128	0.008
NP → N V	13	128	0.102
NP → ADV V	1	128	0.008
NP → ADV N	1	128	0.008
NP → N	66	128	0.515
NP → DET N	12	128	0.094
NP → DET ADJ	7	128	0.055
PP → PREP N	32	36	0.889
PP → N PREP	2	36	0.055
PP → NPREP PREP	1	36	0.028
PP → PREP ADJ	1	36	0.028
VP → PP V	22	118	0.187
VP → PP AUX	1	118	0.008

PCFG Extracted for Tigrigna Complex Sentences

RULE	COUNT	TOTAL	PROBABILITY
S → NP VP	69	100	0.69
S → PP VP	31	100	0.31
PP → PREP N	45	51	0.882
PP → CARDPREP N	1	51	0.020
PP → PREP ADJ	4	51	0.078
PP → PREP VREL	1	51	0.020
VP → NP V	7	196	0.036
VP → ADV V	2	196	0.010
VP → V AUX	5	196	0.026
VP → V V	9	196	0.046
VP → N V	57	196	0.291
VP → PP VP	12	196	0.061
VP → NP AUX	1	196	0.005
VP → VP NP	1	196	0.005
VP → N VP	2	196	0.010
VP → NP VP	79	196	0.403
VP → NPREP V	9	196	0.046
VP → N VREL	1	196	0.005
VP → VPREP V	8	196	0.041
VP → VPREP VP	1	196	0.005

Appendix H. Parsed Output Sentences

Sample Parsed from Simple Tigrigna Sentence

S (NP (N wetaderat)

VP((PP (PREP nab) (N Tornet))(V keydom)))

S (NP (N lemlem)

VP((NPrep nberhe)(VP(NPREP nsraH)(V ^Sewi`ato))))

S (NP (N Hayelom)

VP((NP (ADJ n^xetey) (N ^njera))(V beli`u)))

S (NP (N berhe)

VP((NP(ADJ gobez) (N memhr))(AUX ^yu)))

S (NP (N yaynxet)

VP ((NP (ADJ n^xtey)(N geza))(V ^aleto)))

S (NP (N sebat)

VP((N leyti)(VP (NPREP ^ab`arat)(V ydqsu))))

S (NP (N kaHsay)

VP((NPREP nlemlem)(NP(N genzeb)(V hibwa))))

S (NP (N kaHsay)

VP((NPREP netiseb^ay)(NP(N misTir)(V negirwo))))

S (NP (N meles)

VP((PP(PREP ^ab)(N geza))(V deqisu)))

S (NP (DET ^ta) (N Qol`a)

VP((ADJP(ADV bTa`mi)(ADJ ^SbQti))(AUX ^ya)))

Sample Parsed from Complex Tigrigna Sentences

S (PP(PREP ^ab)(N wereda)

VP ((NP(N welqayt)(N ^Sryet)(VP(N tmhrti)(V tegeli^Su))))

S (NP(N sdeteNatat)(N berma)

VP ((PP(PREP ^ab)(N bangladx))VP(NPREP nTmEt)(V ^aqali`om))))

S (NP(N cayna)(NPREP ntaywan))

VP ((PP(CARDPREP kemHanti)(N gz^at)) (VP(VPREP tQo^Sra)) (V ^ala)))

S (NP(ADJ meraKebti)(NP(N Hafax)))

VP((NP(NP(N gWesgWas)(N mure^Suna))(ADV tmali))(V ^akaydom)))

S (NP(PP(PREP ^ab)(N ^adis^abeba) (N yuniversti))

VP((NP(ADJ din)(NP(N myyT)(VPREP kgber)))(AUX ^yu)))

S (NP(ADJ wetaderawi)(NP(N sr`at)(N berma))

VP((VPREP bzf^Smo)(VP(N Cqonan)(V teKesisu))))

S (NP (N ^akabi)(NP(DET ^ti) (N bord))

VP ((NP(ADJ profEser)(NP(N merga))(VP(ADV tmali)(V moytom)))

S (NP (N gazETeNa)(N ^Seb^Sab)

VP ((NP(PREP ^ab)(NP(VPREP zeQrbelu)(ADV ^wan))(VP(N myyT)(V gEru)))))

S (NP(DET ^zom)(N sraHti)

VP((VP(N lm`at)(V tezazimom))(NP(NPREP nglgalot)(V beQi`om))))

S(NP(PREP ^ab)(NP(N kll))

VP((NP(N ^amHara)(N ferenji))(VP(N lm`at) (V ^akaydom))))

Appendix I. Chart Rules

Fundamental Rule

If the chart contains the edges

[A→ α.Bβ, (i, j)]

[B→ γ., (j, k)]

Then add the new edge

[A→ αB.β ,(i, k)]

Where α , β and γ are sequences of terminals or non-terminals

Bottom up Initialization Rule

For every word w_i add the edge

$[w_i \rightarrow \bullet, (i, i+1)]$

Bottom Up Predict Rule

If the chart contains the complete edge

$[A \rightarrow \alpha \bullet, (i, j)]$

and the grammar contains the production

$B \rightarrow A \ \beta$

Then add the self-loop edge

$[B \rightarrow . \ A \ \beta, (I, i)]$

Appendix J. Sample Codes

Normalization Processes on the Original Corpus

```
import text
import csv, codecs
def remove_ml_tags(in_text):
                """"Removes all HTML/XML-like tags from the input text.
                Inputs: s --> string of text
                Outputs: text string without the tags”””
                # convert in_text to a mutable object (e.g. list)
                s_list = list(in_text)
                i,j = 0,0
                while i < len(s_list):
                        if s_list[i] == '<':
                                while s_list[i] != '>':
                                        s_list.pop(i)
                                s_list.pop(i)
                        else:
                                        i=i+1
                join_char=''
                return join_char.join(s_list)
if __name__ == '__main__':
```

Transliteration Process

```
import re
import string, codecs
print("In Progress Please Wait A Moment.....")
dic={ 'ט' : 'he' , 'ט־' : 'hu'...}
ls=['ִ','ֵ','*?','ְ']
i=0
fh=open("F:/N.txt","r+", encoding="utf-8")
h=open("F:/N4.txt","w+", encoding="utf-8")
fr=fh.readlines()
for word in fr:
   for ch in word:
      if ch in dic.keys():
         c=ch.replace(ch,dic[ch])
         h.write(c)
      if ch==" ":
         h.write(" ")
      if ch in ls:
          print(ch)
print("Process Complete")
print("Thank You.....")
h.close()
fh.close()
```

Fundamental Rule

```
class ProbabilisticFundamentalRule(ChartRule):
   NUM_EDGES=2
   def apply(self, chart, grammar, left_edge, right_edge):
      # Make sure the rule is applicable.
      if not (left_edge.end() == right_edge.start() and
```

```
            left_edge.nextsym() == right_edge.lhs() and
            left_edge.is_incomplete() and right_edge.is_complete()):
        return

    # Construct the new edge.
    p = left_edge.prob() * right_edge.prob()
    new_edge = ProbabilisticTreeEdge(p,
                span=(left_edge.start(), right_edge.end()),
                lhs=left_edge.lhs(), rhs=left_edge.rhs(),
                dot=left_edge.dot()+1)

    # Add it to the chart, with appropriate child pointers.
    changed_chart = False
    for cpl1 in chart.child_pointer_lists(left_edge):
        if chart.insert(new_edge, cpl1+(right_edge,)):
            changed_chart = True

    # If we changed the chart, then generate the edge.
    if changed_chart: yield new_edge
```

Bottom up Initialization Rules

```
class ProbabilisticBottomUpInitRule(AbstractChartRule):
    NUM_EDGES=0
    def apply(self, chart, grammar):
        for index in range(chart.num_leaves()):
            new_edge = ProbabilisticLeafEdge(chart.leaf(index), index)
            if chart.insert(new_edge, ()):
                yield new_edge
```

Bottom up Predict Rule

```
class ProbabilisticBottomUpPredictRule(AbstractChartRule):
    NUM_EDGES=1
    def apply(self, chart, grammar, edge):
        if edge.is_incomplete(): return
        for prod in grammar.productions():
            if edge.lhs() == prod.rhs()[0]:
                new_edge = ProbabilisticTreeEdge.from_production(prod, edge.start(), prod.prob())
                if chart.insert(new_edge, ()):
                    yield new_edge
```

The Parser

```
class BottomUpProbabilisticChartParser(ParserI):
def __init__(self, grammar, beam_size=0, trace=0):
if not isinstance(grammar, PCFG):
            raise ValueError("The grammar must be probabilistic PCFG")
        self._grammar = grammar
        self.beam_size = beam_size
        self._trace = trace
    def grammar(self):
        return self._grammar
    def trace(self, trace=2):
       self._trace = trace
def parse(self, tokens):
      self._grammar.check_coverage(tokens)
```

```
chart = Chart(list(tokens))
grammar = self._grammar
# Chart parser rules.
bu_init = ProbabilisticBottomUpInitRule()
bu = ProbabilisticBottomUpPredictRule()
fr = SingleEdgeProbabilisticFundamentalRule()
queue = []
# Initialize the chart.
for edge in bu_init.apply(chart, grammar):
   if self._trace > 1:
      print('  %-50s [%s]' % (chart.pretty_format_edge(edge,width=2),
                              edge.prob()))
   queue.append(edge)
while len(queue) > 0:
   # Re-sort the queue.
   self.sort_queue(queue, chart)
   # Prune the queue to the correct size if a beam was defined
   if self.beam_size:
      self._prune(queue, chart)
   # Get the best edge.
   edge = queue.pop()
   if self._trace > 0:
      print('  %-50s [%s]' % (chart.pretty_format_edge(edge,width=2),
                              edge.prob()))
   # Apply BU & FR to it.
   queue.extend(bu.apply(chart, grammar, edge))
   queue.extend(fr.apply(chart, grammar, edge))
# Get a list of complete parses.
parses = list(chart.parses(grammar.start(), ProbabilisticTree))
# Assign probabilities to the trees.
prod_probs = {}
for prod in grammar.productions():
   prod_probs[prod.lhs(), prod.rhs()] = prod.prob()
for parse in parses:
   self._setprob(parse, prod_probs)
```

Signed Declaration Sheet

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been properly acknowledged.

Declared by:

Name: _____
Signature: _____
Date: _____

Confirmed by advisor:

Name: _____
Signature: _____
Date: _____