Addis Ababa University

College of Natural and Computational Sciences

Dependency-based Tigrinya Grammar Checker

Mehammedbrhan Abdelkadr Gidey

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

March 2021

Addis Ababa University

College of Natural and Computational Sciences

Mehammedbrhan Abdelkadr Gidey

Advisor: *Yaregal Assabie (PhD)*

This is to certify that the thesis prepared by *Mehammedbrhan Abdelkadr Gidey*, titled: *Dependency-based Tigrinya Grammar Checker* 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: Yaregal Assabie (PhD)	_____	_____
Examiner: Dida Midekso (PhD)	_____	_____
Examiner: Minale Ashagrie (PhD)	_____	_____

Abstract

Grammar checking is the process of checking for grammatical correctness by verifying the syntax and morphology of a sentence according to the used language. For languages such As English, Arabic, Afaan Oromo, and Amharic, many efforts have been made to develop grammar checking systems. Because natural languages differ in their morphology and grammar, it's difficult to apply a grammar checker of one language to another. Although an attempt was made to develop a grammar checker for Tigrinya, the grammar checker is unable to identify the relationship between words in a sentence, parsing complex and compound sentences, and it produces possible sentence structures with syntactically correct but semantically non-sense sentences. The use of phrase-structure grammar notation for statistical and rule-based methods causes the majority of these issues because it has a complicated representation but it allows a limited level of grammar analysis.

We propose that applying dependency-based grammar checking for the Tigrinya language will have a significant role in overcoming the problems in the existing grammar checker. The system is composed of a text preprocessing module, a language dependency model, a dependency extraction module, and a grammar checking module. The text preprocessing module is in charge of cleaning input text and format conversion, and it includes the tokenizer, part of speech tagger, and morphological analyzer to do so. The dependency model of the language is a pre-trained model to be used by the text preprocessing and dependency extraction modules. The dependency extraction module parses for the root of the sentence (main verb), head-dependent pairs and their corresponding relations with the use of the dependency parser inside it. Finally, the grammar checking module contains relation extractor and agreement checker components to carry out the grammatical relation extraction and the grammatical agreement checking tasks.

A test data set of 74 grammatically correct sentences and 48 grammatically incorrect sentences was used to test the grammar checker system. The system is tested with a total of 122 sentences. On the basis of the prediction results, the system is evaluated using some evaluation metrics. The system has the best precision of 92.46%, accuracy of 92.09%, and recall of 61.21% according to the evaluation results. We filled over half of the test dataset with grammatically incorrect sentences, which caused the low recall score.

Keywords: Dependency Grammar, Dependency Parser, Grammar Checker

Dedication

This work is dedicated to my mother Seada Kahsay and my father Abdelkadr Gidey for their long-lasting love and care for me since my first day in this world.

Acknowledgments

First and foremost, I would like to praise and thank Allah, the almighty, who has granted countless blessing, knowledge, and opportunity to the writer, so that I have been finally able to accomplish the thesis.

I would like to thank my supervisor, Dr. Yaregal Assabie, for the patient guidance, encouragement and advice he has provided throughout my time as his student. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly.

I would also like to express my gratitude to my examiners Dr. Dida Midekso and Dr. Minale Ashagrie for the useful comments, remarks and engagement through the learning process of this master thesis.

I also want to thank my brothers, sisters, and all my friends who supported and motivated me to complete my thesis work.

Contents

List of Tables

List of Figures

List of Algorithms

Acronyms and Abbreviations

CONLL	Computational Natural Language Learning
NLP	Natural Language Processing
NTC	Nagaoka Tigrinya Corpus
POS	Part of Speech
SOV	Subject Object Verb
UD	Universal Dependency
UDPipe	Universal Dependency Pipeline
UPOS	Universal Part of Speech
XML	Extensible Markup Language
XPOS	Language Specific Part of Speech

Chapter 1: Introduction

1.1 Overview

Natural language is a way of communication used by humans and includes all kinds of written and spoken words and sentences [1]. Morphology of a language involves modification and formation of words with respect to number, gender, and time. According to Verena and Timo [2], natural language grammar is defined as a set of syntax and morphology of components and words of the language to form a sentence. The grammar and morphology of a natural languages differ from language to language [1].

Natural language processing (NLP) is a branch of artificial intelligence that is concerned with enabling computers to understand, interpret and manipulate human language [3]. NLP is an interdisciplinary study area including computer science and computational linguistics. Its aim is to fill the gap between human communication and computer understanding. As a result of studies on different aspects of natural languages, many tools and applications are developed for different purposes. Tools like morphological analyzer, spelling checker, a speech synthesizer, speech recognition, and machine translation are some of the natural language applications in addition to grammar checker.

Grammar checking is the process of verifying syntax and morphology of a sentence or text according to the used language, to check for its grammatical correctness [1]. The tool that performs this task is called grammar checker. People can make grammatical mistakes while writing; intentionally or with the lack of good grammatical knowledge on the language. To help users solve such grammatical problem, many grammar checker tools are developed for different languages using different approaches. Some approaches that are used to develop grammar checkers are pattern matching, syntax-based, rule-based, statistical, and hybrid approaches.

Pattern matching is a very primitive method which uses data storage where common grammar mistakes and their corresponding corrections are kept [2]. A sentence or a part of a sentence is matched against an error entry in the data storage. If a match is found, an error is detected and it can be corrected with the correction from the data storage. Pattern matching is quite effective for the patterns in the data storage. But there is no generality in the patterns. Every small difference in grammar needs a new pattern. In syntax-based approach [4] a text is completely analyzed

morphologically and syntactically. It requires a lexical database, a morphological analyzer, and a parser. The parser assigns a syntactic structure to each sentence. The text is considered incorrect if the parsing does not succeed. According to the level of the linguistic analysis, this approach can be classified as a deep syntactic analysis or a shallow syntactic analysis. The rule-based approach is a more common way in which simple rules can be used to detect errors that are easy to find. Unlike pattern matching, a single rule can be used to detect errors of the same type in different entries. In more complex sentences, the rules to detect errors become more complicated. The statistical approach assumes that the grammar of a text can be corrected by only using large amounts of text forming a statistical database which is used to detect errors [5]. In this approach, there are two different ways that can be used to achieve the goal of correcting grammar. One uses the data directly to compare it with the text which should be corrected. Another one derives a grammar from statistical information which can then be used to check and parse the text. Hybrid grammar checking approach combines any of the existing approaches so that the grammar checker can be more robust and achieve higher efficiency [6].

Dependency grammar belongs to the class of grammars that emphasize words rather than constituents. The basic concept of dependency is founded on the premise that a sentence's syntactic structure is made up of binary asymmetrical relationships between its words, and that a dependence relationship exists between a head (governor or regent) and a dependent (modifier) [7]. Heads and dependents are related immediately, there are no non-terminals (left unrelated). Among the advantages of dependency based grammar over phrase-structure representations are conciseness, intuitive appeal, and closeness to semantic representations such as predicate-argument structures [8]. Phrase structure and dependency are not opponents, they are complementary notions. Therefore, using both notions together we can overcome the problems that each notion has individually.

1.2 Motivation

Tigrinya is an Afro-Asiatic language which belongs to the Semitic branch of the language family [9]. It uses Ge’ez [10] script for writing and has a considerable number of speakers in northern Ethiopia, as well as the contiguous borders of Eritrea's southern and central regions [9]. It is also one of Eritrea's official languages [9]. Even though Tigrinya language speakers can use existing office tools such as Microsoft Office to prepare and manipulate documents, these tools do not

understand the grammar of this language. They also cannot detect any grammatical errors and suggest corrections for the errors as much as they do for English. Documents that are to be written in this language have to be checked for grammatical correctness because a text with grammatical mistakes can result in a vague or meaningless text. Manually checking a document for grammatical correctness is time-consuming, bulky, and also results in a grammatically fallible document. Automating the task of grammar checking saves time, effort, and results in a grammatically better document. This has motivated us to work on the development of Tigrinya grammar checker.

1.3 Statement of the Problem

Many efforts and attempts have been made to develop grammar checker tools for languages such as English [11], Arabic [4], Afaan Oromo [12], and Amharic [13]. As there is a difference in morphology and grammar among natural languages, it is difficult to apply grammar checker of a specific language directly on another language. The morphology and grammatical nature of the language make it difficult to adapt existing grammar checker tools of other languages.

An attempt has been made to develop a grammar checker for the Tigrigna language [6]. However, this grammar checker lacks capabilities for identifying the relation between the words in a sentence, parsing complex and compound sentences, producing possible sentence representations with syntactically correct but semantically non-sense sentences, etc. Most of these problems arise due to the use of phrase-structure grammar notation for statistical and rule-based techniques. This kind of representation does not show a word-word relationship, uses a number of nodes and edges for representation and makes the sentence representation more complex than a dependency representation. The grammar notation allows only a limited level of sentence structure analysis.

Abdella's grammar checker uses the Nagaoka Tigrinya Corpus (NTC), which is a POS tagged corpus [6]. The corpus by itself includes only a few details (meta-data) for each token of the sentences inside it. These details are token id, token, and POS tag of the token [14]. It does not contain additional information such as lemma, UPOS, morphological features and grammatical relations. They did not modify the corpus in a way that adds this additional information to the tokens. This is because phrase structure grammar does not focus on advanced sentence structure analysis and it only needs POS tags to parse for tokens and phrases that form the sentence structure. As a result, models that are trained with such kind of corpus are not able to learn more details

about each token. Since the statistical method of Abdella's grammar checker is trained using NTC, it only learns token and tag sequences [6]. It lacks the capability to identify the lemma, UPOS, morphological features and grammatical relation between tokens in a sentence. In Abdella's work, morphological features are tagged using another tool called HornMorpho, which is a morphological analyzer for Afaan Oromo, Amharic, and Tigrinya. However, HornMorpho has limited resources for Tigrinya as it analyses morphology of only Tigrinya verbs [15]. Therefore, Abdella's system still lacks morphological tagger for Tigrinya nouns and other words in other parts of speeches.

Since natural languages have unlimited possibilities of morphology and sentence structures, it is very difficult to include all possible grammatical rules for a natural language. Natural languages are dynamic in nature and evolve in time drastically. Grammatical rules are mainly affected by contexts and differ their validity among contexts. A rule for a specific context may not be true (not work) for another context. Given that, the token details we can get are not enough, the rules we can prepare cannot be fully reliable for natural language sentence analysis. Abdella's rule-based checker uses rules for incorrect sentences. These rules are based on context free grammar, which emphasizes only tokens and phrases. CFG does not analyze advanced token details such as lemma, UPOS, morphological features, and grammatical relations. The statistical method in Abdella's grammar checker checks the input text against the stored statistical data, only for token and tag sequence matches. However, sequence match does not mean always true because only sequences cannot define a sentence structure. These sequences do not show the grammatical relation between them and between tokens inside a sequence. Sequences are important only for horizontal analysis of sentence structure. Sequences only show the word order, and this can be a barrier to checking sentences of a free word order language.

Grammar checking is very important not only for human beings but also for machines. Therefore, a grammar checker is expected to yield more than the checking result messages. Apart from the results message, additional information regarding the grammatically checked sentences is far more important for machines. This information should include the lemma, POS tags, morphological features, and grammatical relation (function) of each token in a sentence. A preceding NLP application or process should also contribute to the study and development of a possible following NLP application or process. Semantic analysis is a possible following process for syntactic

analysis. When it comes to semantics, dependency grammar outperforms phrase-structure grammar, because dependency representation is closer to semantic representation. It generates a simple sentence structure representation, which uses fewer nodes and edges than a phrase structure representation. For example, in the sentence እቲ ሰብኣይ ደቂሱ (ETI SEBIAY DEQISU) which means (The man sleep); the determiner "እቲ" depends on the noun "ሰብኣይ", the noun "ሰብኣይ" depends on the verb "ደቂሱ", and the verb "ደቂሱ" depends on nothing and it is the matrix verb of the sentence. The verb is central or the main element and is served as the root of the dependency tree as shown in Figure 1.1.

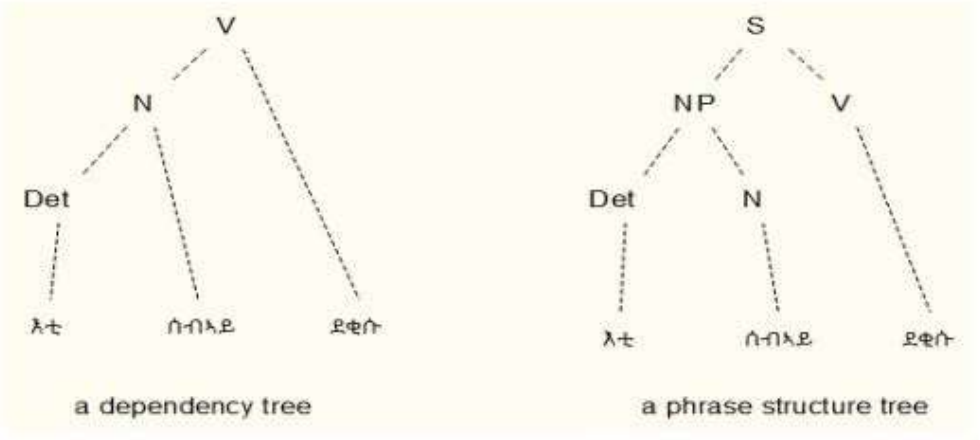

Figure 1.1- The difference between dependency and phrase structure trees.

Thus, when we see the contribution of Abdella's study and its results to semantic analysis, it is not enough as expected from a grammar checker. Because, it does not provide sufficiently detailed information about the tokens, token attributes, and the grammatical relations in the parsed sentence.

In this study we will try to solve the problems in the existing Tigrinya grammar checker using Tigrinya text corpus that has more token details, grammar checking methods with enhanced capability, and dependency grammar notation. A corpus from NTC that is in xml data format will be enhanced into morpho-syntactically annotated Tigrinya treebank and prepared in CoNLL-U data format that includes tokens and additional token details. These details should include the lemma, POS tags, morphological features, and grammatical relations of tokens in a sentence. A more capable statistical method will be used for text preprocessing and dependency extraction. Language models for the preprocessing and dependency extraction components will be trained

using the enhanced corpus. Dependency grammar notation will be used as it has simple notation and representation and it enables us to perform advanced sentence grammar analysis. Thus, we propose that applying dependency based grammar checking to the Tigrinya language will have a significant role in overcoming the problems in the existing grammar checker. Therefore, with this in mind, we aim to develop a grammar checker based on dependency grammar along with language morphology.

1.4 Objectives

General Objective

The main objective of this study is to design and develop a dependency-based grammar checker for Tigrinya language.

Specific Objectives

The specific objectives that will be carried out to achieve the general objective of the study are:

- Studying the grammatical rules and morphological structures of Tigrinya language
- Review of literature and studies on grammar checking, dependency checking, and related papers on Tigrinya
- Data collection and preprocessing
- Identifying dependencies between words and common grammatical errors in Tigrigna sentences
- Modeling the grammar of Tigrinya language
- Implementing the algorithms that can check for grammatical structure and the dependency among the words in a sentences
- Developing a prototype of the system
- Evaluate the system using well prepared test data and test dataset from related works

1.5 Methodology

Literature Review

Literature of grammar checking and other related works on the language will be reviewed in order to have a deep understanding of grammar checking, approaches for grammar and dependency checking, the grammatical rules and structures as well as the morphology of Tigrinya language, and common grammatical errors. A consultation from linguistic experts will be needed for a better and in-depth understanding of the most important grammatical issues to be considered in the development of grammar checker.

Data Collection

Data for the text corpus and test data set will be collected from various domains that use Tigrinya language and it will be preprocessed for use in the experiments we will be making in this study. Some domains we will refer to for data are Tigrinya grammar books, Tigrinya language news sites, Tigrinya wikis, Tigrinya religious books and other books in Tigrinya language. Data found especially from works of linguistic experts will be used in the testing tasks in our study.

Prototype Development

Prototype of the system that shows different approaches to grammar checking and dependency grammar will be developed and prepared for evaluation. It will be developed by integrating separately implemented components that are responsible for different tasks in the grammar checking process. The components will be integrated into a single, fully operational grammar checker system. Finally, the grammar checker will be made ready for evaluation.

Testing and Evaluation

The system will be tested based on a set of test data and evaluation will be carried out by preparing some standardized evaluation criteria of such a system that is capable of fully evaluating the overall components of the system and have a significant role in improving the overall system performance.

1.6 Scope and Limitation

The scope of this study focuses only on the design and development of dependency-based Tigrinya grammar checker that analyzes Tigrinya text for grammatical errors and probably suggests grammatically correct alternatives in case of error detection. Grammar correction and other further tasks will not be part of this work.

1.7 Application of Results

On the successful completion of this work, the system can be used to help users write and prepare a grammatically error-free document. It will also be very important in language learning. With the help of the grammar checker, computers will start to better understand Tigrinya language grammar and it will be easy for them to process and work with this language texts. Development of other natural language applications that need grammar checker tool as part of their processes can benefit out of this system; saving the time to be consumed on the development of such tools. Components that will be developed as part of the grammar checker tool will have a significant role in solving the resource limitation barrier of this language.

1.8 Organization of the Thesis

This thesis is organized in Six Chapters including the current one. Chapter Two deals with literature review section. Chapter Three deals with review of related work. Chapter Four deals with the design and development of prototype for dependency-based Tigrinya grammar checker. Chapter Five deals with experiment and evaluation of the grammar checker. Chapter Six discusses the conclusion and future works.

Chapter 2: Literature Review

2.1 Introduction

Literature of grammar checking and other related concepts will be reviewed in order to have a deep understanding of natural languages, grammar checking, approaches for grammar and dependency checking, the grammatical rules and structures as well as the morphology of Tigrinya language, and common grammatical errors. A consultation from linguistic experts will be needed for a better and in-depth understanding of the most important grammatical issues to be considered in the development of Tigrinya grammar checker.

2.2 Natural Language and Grammar

Natural language is being understood as the language used in everyday communication by human beings. English, Amharic, and Tigrinya are some examples of natural language. It includes all written and spoken words that follow a rule known as a grammar to arrange them in a meaningful way to form a sentence. A grammar is a set of rules and descriptions concerning the combinations (syntax), modifications (morphology), and classification based on function (parts of speech) of words in a language to form sentences [13]. Syntax of a language defines arrangement of words in a phrase and sentences. Morphology of a language also deals with the modification of words depending on time, person, number, and gender.

Humans can communicate with each other if they comprehend and obey the same grammatical rules in the language they use. If one makes grammatically incorrect sentences while writing or speaking without following the grammatical rules of the language, the reader or listener may not understand the writer or speaker. Therefore, the grammar rules or correct grammatical structure is among the important things for communication through natural languages [13].

2.3 Grammar Checking

The errors a writer or speaker may create which oppose the grammar rules are called grammatical errors. A system called grammar checker verifies or tests the grammatical correctness of a sentence [12]. Identifying grammatical errors in a text is one of the tasks carried out by a grammar checker system.

2.4 Approaches for Grammar Checking

There are a number of research on grammar checker conducted for many languages in different countries [4, 12, 14]. These research on grammar checker follow different approaches such as, syntax-based, rule-based, statistical, and hybrid grammar checking approaches.

2.4.1 Syntax-based Grammar Checking

Syntax-based approach uses a parser to analyze a sentence and assigns nonlinear structure of the phrase, i.e. noun phrase [11, 15]. The use of parser is to generate complex rules that help to reduce sentences to phrases, i.e. verb phrase (VP) or noun phrase (NP). Syntax-based grammar checking can find the relation between words without considering the intermediate words. Input texts are considered incorrect if the parsing is not complete. In the other hand, if the sentence is completely parsed, the sentence is assumed as a correct sentence. Then, it uses some rules to verify the correctness of the assumed corrected sentence. One example is a work titled as dependency-based rules for grammar checking with language-tool [11].

2.4.2 Statistical Grammar Checking

In Statistical approach, a large amount of POS-annotated corpus is used to train the system and the system automatically generates rules from the corpus [15]. There are two ways of implementing this approach. The first way is to directly check the token sequence of the input text with the token sequence in the corpus. This method is easy and does not need any morphological information of the text. However the main challenge of this way is correctness of a sentence depends on a single word and to include each word a huge corpus is needed. The second way uses the tag of the word to solve this problem. It is the same as the first way but the difference is it uses the tag of the word instead of the word of the sentence. One of the main advantages of this approach is language independent [4, 14]. Some examples include N-gram Based Statistical Grammar Checker for Bangla [14], language independent statistical grammar checker [2].

2.4.3 Rule-based Grammar Checking

Rule-based approach checks input text against the rules which are developed manually. The rules can easily be added, removed or modified. Rule-based systems can provide a comprehensive error message, including a statement or explanation of the grammatical rule. It is possible to check the

input sentences while they are being written. The other thing, that makes this approach so powerful, is it can cover almost all the features of a language. Some examples include a rule-based Afaan Oromo Grammar Checker [12], style and grammar checker for English [15], dependency-based rule for grammar checking [11] and Punjabi Grammar Checker [16].

Advantage of rule-based approach are [6]:

- rules can be easily added, modified or removed without any modification of the existing source code
- every rule can have a corresponding extensive explanation, helpful for the end user
- the system is easily debuggable, since its decisions can be traced to a particular rule
- the rules can be authored by the linguists, possessing limited or no programming skills
- high expressive powerful and flexibility
- it is possible to use OR and NOT logic operations (for example: “match token A or token B”; “match any token except C”)
- skip optional tokens,
- to some extent, use regular expressions

Disadvantage of this approach are:

- large amount of manual work needed to build an extensive rule set
- limitation to a specific language
- it considers the input text to be a series of tokens
- ignores the fact that real language sentences have a tree-like structure

2.4.4 Hybrid Grammar Checking

Hybrid approach combines more than one of the grammar checker approaches to achieve high efficiency and robustness. Some example include Granska Swedish grammar checker [17], COGrOO Portuguese grammar checker [18], Amharic Grammar Checker [4], etc. The Hybrid

Amharic grammar checker [13] has used the rule based approach to handle the simple sentences and the statistical approach to handle complex sentences.

2.5 Common Grammatical Errors

Tigrinya sentences follow the S-O-V (subject-object-verb) structure, unlike the English S-V-O structure. Tigrinya nouns, adjectives, and verbs are inflected for gender, number, case, definiteness, tense, person, aspect, and mood. For example, the nouns "ከተማታት /ketematat" (Cities), "ተምሃሪት /Temharit" (Student, single feminine), adjectives "ዓባይ /abay" (Big, single feminine), "ዓበይቲ /abeyti" (Big, plural), verbs "ከይዶም /kaydom" (They went, plural masculine past 2nd person), "ኣንቢበን /anbiben" (They read, plural feminine past 2nd person).

For example, the sentence "መብራህቱ በሶ በሊዓ" (MABERAHITU BASSO BALIEA) which means "MEBRAHTU ate BESSO" is grammatically incorrect because it has a subject-verb disagreement in the case of gender. The subject "መብራህቱ" is grammatically masculine and single. However, the verb "በሊዓ" shows feminine and single. Office tools such as Microsoft Office should have detected the grammatical error and suggested the user to correct it as "መብራህቱ በሶ በሊዑ" (MABERAHITU BASSO BALIEU).

Some of the different mistakes people make whenever they write sentences using Tigrinya language are discussed as follows. The majority of the grammar error categories covered in this section are also seen in other languages.

2.5.1 Subject-Verb Disagreement

In a sentence, the subject and verb must agree in terms of number, person, and gender. In all circumstances, the disagreement causes the sentence's reader to be perplexed or confused. The following sentences show this kind of grammatical error in different cases [13].

1. ንሱ ናብ ገጠር ከይዳ። /nesu nab geter kayda /He went to countryside
2. ኣነ ቁርሰይ በሊዑ። /ane qursay bali'u /I ate my breakfast
3. ሰሚራ ናብ ቤት ትምህርቲ ከይደን። /semira nab biet temehereti kaydan /Semira went to school

In the first sentence the subject "ንሱ" / "nesu" /"He" is in third person singular masculine form however the verb "ከይዳ" / "kayda" /"She went" is in third person singular feminine form, here

disagreement in gender is occurred. In the second sentence the subject “ኣነ” / ”ane” /”I” is in the first person singular masculine form however the verb “በሊዑ” / “bali’u” /”He ate” is in third person singular masculine form, here disagreement in person is occurred. In the third sentence the subject “ሰሚራ” / ”semira” /”Semira” is in the third person singular feminine form however the verb “ከይዶን” / “kaydan” /”They went” is in third person plural feminine form, here disagreement in number has occurred. When the kinds of subject-verb disagreements in each sentence are resolved the above three sentences can be corrected as follows.

1. ንሱ ናብ ገጠር ከይዱ። /nesu nab geter kaydu /He went to countryside
2. ኣነ ቁርሰይ በሊዐ። /ane qursay bali’e /I ate my breakfast
3. ሰሚራ ናብ ቤት ትምህርቲ ከይዳ። /semira nab biet temehereti kayda /Semira went to school

2.5.2 Object-Verb Disagreement

Object and verb must also agree for number, person and gender. The following sentences show the kinds of object-verb disagreement [13].

1. ክብሮም ንሓፍቱ ክዳን ገዚኡሉ። /kebrom nehaftu kedan gazi’ulu /Kibrom bought a cloth for his sister
2. መምህር ሰላም ነዞም ተምሃሮ ሓቲታቶ። /mameher selam nazom tameharo hatitato /Teacher Selam asked the students
3. ኣነ ዝገዛእኩዎ ኩኩናይ ቀይሕ እየ። /ane zigaza’ekuwo kukunay kayeh eye /The Cock I bought is red

In The first sentence the object “ንሓፍቱ” / ”nehaftu” /”for his sister” is in third person singular feminine form however the verb “ገዚኡሉ” / “gazi’ulu” /”He bought for him” is in third person singular masculine form, here disagreement in gender is occurred. In the second sentence the object “ተምሃሮ” / ”tameharo” /”Students” is in the third person plural form however the verb “ሓቲታቶ” / “hatitato” /“She asked them” is in third person singular form, here disagreement in number has occurred. In the third sentence the object “ዝገዛእኩዎ ኩኩናይ” / ”zigaza’ekuwo kukunay” /”The cock I bought” is in the third person singular form however the verb “እየ” / “eye” /”I am” is in first person singular form, here disagreement in person has occurred. When the kinds of object-verb disagreements in each sentence are resolved the above three sentences can be corrected as follows.

1. ክብሮም ንሓፍቱ ክዳን ገዚኡላ። /kebrom nehaftu kedan gazi'ula /Kibrom bought a cloth for his sister
2. መምህር ሰላም ነዞም ተምሃሮ ሓቲታቶም። /mameher selam nazom tameharo hatitatom /Teacher Selam asked the students
3. ኣነ ዝገዛእኾዎ ኩኩናይ ቀይሕ እዩ። /ane zigaza'ekuwo kukunay kayeh eyu /The Cock I bought is red

2.5.3 Incorrect Word Order

Word order has different forms in different languages. It contains the modifier and modified word order (adjective and noun or adverb and verb), noun and verb order, verb and sentence end marker order, noun and conjunction order, and others [6, 13]. For example in Tigrinya the grammar rule does not allow the verb before the noun. The English sentence structure is SVO (subject verb object), whereas the Tigrinya sentence structure is SOV (subject object verb). It can also appear as OSV in informal texts. The words of a phrase should be in the correct order depending on the natural language used [13].

2.5.4 Adjective-Noun Disagreement

Modifiers are words or phrases that are used to explain something in a statement. Adjectives that modify nouns and adverbs that modify verbs are examples of modifiers. Adjectives in Tigrinya come before the nouns they alter. In terms of number and gender, the Tigrinya Adjective should match the noun it modifies. In a phrase, an adjective is employed to paint a clear image of the word it modifies. It's just a term that adds to the meaning of the noun. Adjectives are used to modify nouns in Tigrinya. These adjectives can indicate the noun's number and gender (feminine or masculine). If a noun's number and gender are marked by an adjective, the marker should match the noun's number and gender. The phrases are improper due to the conflict between adjectives and nouns [6, 13].

Example: "ነዊሕ ሰበይቲ" / "nawih sabayti" "tall (masculine) woman" is grammatically incorrect as it has disagreement in its gender marker. I.e. masculine marker ("nawih" /"tall (masculine)") in the adjective and a feminine marker ("sabayti" /"Woman") in the noun. It can be corrected by adjusting the marker in the adjective or noun. Adjusting the marker in the adjective corrects the

disagreement in gender marker resulting in feminine marker in both the adjective and noun “ነዋሕ ሰበይቲ” /”nawah sabayti” “tall (feminine) woman”. Adjusting the marker in the noun corrects the disagreement in gender marker resulting in masculine marker in both the adjective and noun “ነዊሕ ሰብኣይ” /”nawih sab’ay” “tall (masculine) man”.

2.5.5 Adverb-Verb Disagreement

Adverbs are words that alter or describe the meaning of other words. The Tigrinya Adverb is used to modify the verb that comes after it. Adverbs in Tigrinya are rare in number and can be found as single words, as a mixture of two words appearing as a single word, or as detached words. Adverbs in Tigrinya are divided into subclasses such as adverbs of time, place, circumstance, and so on. The time adverbs are used to describe when something happened. These adverbs can be used to indicate a specific time or duration of an action. One of the most prevalent Tigrinya grammatical mistakes is the time adverb and tense disagreement. For the verb, the appropriate adverb should be used, and vice versa. [13].

2.6 N-Gram and N-Gram Model

N-grams are sentence sequences with an N number of tokens, where N is the number of tokens. A letter, word, tag, punctuation mark, or other symbol can be used as a token [2]. An n-gram model is a probabilistic model that determines the likelihood of a token appearing in the preceding N-1 [19] token sequence. The sequences can be given different names depending on the N value. Bigram, trigram, quad gram, and pentagram are the symbols for the numbers 2, 3, 4, and 5 respectively.

Given the number of token and the value of N, the maximum number of sequences for the tokens can be calculated by Equation 2.1:

$$Ns = Nt - (N - 1)$$ Equation 2.1

Where, Ns is the number of sequences, Nt is the number of tokens and N is the N value in the N-gram.

Suppose Ns is a list of sequences for a given sentence. One of the sequences in Ns has W1 and W2 in it (this means the value of N is 2 this time). The probability of the sequence i.e.

Probability of W2 given W1 is calculated as in Equation 2.2:

$P(W2|W1) = COUNT(W1\ W2)/COUNT(W1)$ Equation 2.2

Where, W2 is the second word and W1 is the first word in the sequence.

Counting the number of the three words appearing together over the number of the first and second words combined gives the probability of sequences for N = 3, trigram. If W1, W2, and W3 are the first, second, and third words in a series, the sequence's probability (which is the likelihood of W3 given W1 and W2) can be computed as follows:

$P(W3|W1W2) = COUNT(W1\ W2\ W3)/COUNT(W1\ W2)$ Equation 2.3

The generalized formula of the probability of the sequence for N = n is given by Equation 2.4:

$P(Wn|W1W2 \ldots Wn-1) = COUNT(W1\ W2\ W3 \ldots Wn\text{-}1\ Wn)/COUNT(W1\ W2\ W3 \ldots Wn\text{-}1)$ Equation 2.4

For a given N value, the probability of the entire sentence can be computed by multiplying the probability of all the sequences in the sentence for that N value as shown in Equation 2.5.

$P(S) = P(Ns1) * P(Ns2) * P(Ns3) * P(Ns4) * \ldots P(Nsn)$ Equation 2.5

Where, Ns1, Ns2, Ns3 ... Nsn the sequences in the sentence.

2.7 Corpora

A corpus is a collection of naturally occurring texts in both written and spoken language that has been organized in a systematic manner. A large and grammatically correct corpus is required to train a statistical grammar checker. Finding a large error-free corpus to train the system is the key issue with statistical approaches [6].

Nagaoka Tigrinya corpus 1.0 (NTC 1.0) is the first publicly available part-of-speech (POS) tagged corpus of Tigrinya language. The corpus is used to train the system.

A Corpus has some components [20]. These components are:-

- The text data itself, a text in natural language
- Possibly Metadata that describe the text data,

- And linguistic annotations related to the text data. This may be a POS tag, morphological analysis of words and linguistics information about the letters, word, sentences and the like.

Many languages are under-resourced especially languages in developing countries like Ethiopia [21, 22]. Tigrinya language also has a shortage of sample annotated corpus. To teach statistical grammar checkers, they need grammatically correct texts. Finding an error-free corpus to train the system is the fundamental issue with statistical approaches. The training and test data sets should be annotated as well, which means that words in the corpus should be POS tagged and morphologically evaluated. As a pre-process on the text to be examined, a rule-based grammar checker tags and analyzes it [6].

2.8 Tokenization

A token is an instance of a sequence of characters, i.e. every word, every number, every punctuation, every abbreviation, etc., in some particular document that are grouped together as a useful semantic unit for processing. One challenging part when input text is given to natural language processing system is the task of word and sentence segmentation. This is also known as tokenization [2].

Tokenization is the task of splitting text up into meaningful pieces is done using a tokenizer system. Splitting text in Tigrinya is not as complex as in other languages like English. In Tigrinya, Some symbolic characters, e.g. "፡፡ ? !" is used as sentence boundary markers [6]. In the case of a grammar checker, input texts should be split down into sentences, which should subsequently be broken down into words. In the morphological analyzer, the output words can be tagged and analyzed separately. If utilizing a statistical methodology, both the training and test data sets should pass the tokenization procedure. Tokenization of the input text to be checked is also required for the rule-based approach. In order to be tagged and analyzed in a rule-based manner, the input text should be separated into sentences and then into words. [13].

2.9 Morphological Analysis

The word morphology comes from a Greek word morph which means shape or form. Morphology is the study of a thing's form or forms, depending on which field it's applied to. Morphology is the study of the formation of words and their internal structure in linguistics [23]. The root word and

other significant parts of the word make up the morphe, which is the smallest unit of morphology [23, 24].

Morphological analysis is the process of finding the morphemes of the word and providing grammatical information for the word based on the identified morphemes. For example, the word "በሊዑ" /"bEliU" /"He ate" has the morphemes "በልዐ" /"bEliE" /"eat" and "-ኡ" /"-U" which stand as the root word and the meaningful piece of the word respectively. "-ኡ" /"-U" is a morpheme that is used to refer to a "masculine gender", the gender the word "በሊዑ" /"bEliU" is referring to. A morphological analyzer is a program or system that recognizes morphemes and offers grammatical information for the morphemes of an input word.

Many higher-level natural language systems, like machine translation, speech recognition, information retrieval, and grammar checkers, depend largely on morphological analysis. Morphological analysis should be performed as a pre-process in the grammar checking process [13].

HornMorpho is a morphological analyzer done for three languages; Amharic, Tigrigna and Oromo by Michael Gasser [25]. Finite state transducers (FST) are used to do the analysis on a finite state machine. The system accepts a word for analysis and displays the analysis result, which contains the word's POS group and grammatical structure. It designates a word category for first, second, and third person pronouns, gender (feminine, masculine), number (single, multiple), definiteness (definite, indefinite), grammar (perfective, imperfective, gerundive, jussive), and other terms. It first Romanizes the word in Amharic and Tigrigna, then performs the analysis on the Romanized term. The results will be written/displayed in the same language as the input word after the analysis [25].

2.10 Phrase-structure Grammar

Phrase-structure also called constituency grammar is one of the different grammatical notations used to represent a sentence structure. It deals with the constituents of a sentence such as noun, noun phrase, verb, verb phrase, and so on [19]. It refers to the analysis that splits sentences into its subparts or phrase [26].

The most widely used mathematical approach for describing constituent structure in natural languages is a Context Free Grammar (CFG), which consists of a collection of rules or productions

and a lexicon of words and symbols. Each rule specifies how the symbols in the language are organized and arranged. Context-free rules can be hierarchically embedded, allowing us to combine the previous rules with others that convey lexical facts, such as the ones below.

A noun phrase (NP) can be made of either a Proper Noun or a Determiner (Det) followed by a Nominal; a Nominal can be one or more Nouns, as shown in the following productions [26].

NP → Det Nominal

NP → Proper Noun

Nominal → Noun | Nominal Noun

Det→ a

Det→ the

Noun → flight

Where *NP* = Noun Phrase, *Det*=Determiner

A CFG's symbols are classified into two categories: terminal symbols and non-terminal symbols. Terminal symbols are the symbols that correlate to words in the language (e.g., "the" and "flying"); the lexicon is the set of rules that introduce these terminal symbols. Non-terminals are symbols that express clusters or generalizations of them. Each context free rule has an ordered list of one or more terminals and non-terminals to the right of the arrow (), and a single non-terminal symbol indicating some cluster or generalization to the left of the arrow. The lexical category, or part-of-speech, associated with each word is its non-terminal [26].

A context free grammar G is a quadrupled [27]:

G = (VNT, VT, P, S) where:

1. *VNT* is a set of non-terminal symbols (phrases)
2. *VT* is a set of terminal symbols (lexical items)
3. *P* is a set of production rules of the form A → x, where A is a terminal symbol and x is a sequence of terminal and non-terminal symbols.
4. *S*: a designated start symbol, $S \in VNT$

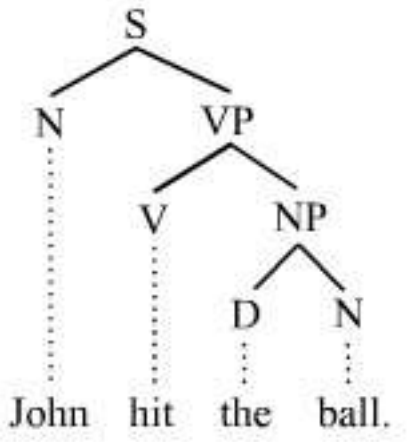

Figure 2.1- Constituency based grammar representation

2.11 Dependency Grammar

Dependency grammar is another type of grammatical notations used to represent a sentence structure. It states that the syntactic structure of a sentence consists of binary asymmetrical relations between the words of the sentence. A dependency relation holds between a head (governor or regent) and a dependent (modifier) [28]. It represents the syntactic structure of a sentence by means of dependency trees.

Dependencies are motivated by grammatical function. A word depends on another either if it is a complement or a modifier of the latter. For example, a transitive verb "love" requires two dependents, one noun with the grammatical function subject and one with the function object. To demonstrate further, consider a sentence "a man sleeps". The indefinite article "a" depends on the noun "man" which in turn depends on the verb "sleep". The verb "sleep" depends on nothing, which is the root of the sentence [7]. This shows that dependency grammar formalism is closer to natural language. In other words, dependency representation is more similar to the human understanding of a language.

The capacity to deal with morphologically rich languages and a relatively unrestricted word order are two significant advantages of dependency grammar. Another reason to employ a dependency-based method is because head-dependent connections approximate the semantic link between predicates and their arguments, making them immediately helpful in a variety of applications like co-reference resolution, question answering, and information extraction [26].

2.11.1 Dependency Relation

The arguments to grammatical relations consist of a head and a dependent. In constituency or phrase structure grammar, the head word of a constituent is the central organizing word of a larger constituent (e.g., the primary noun in a noun phrase, or verb in a verb phrase). The remaining words in the constituent are either direct or indirect dependents of their head. But, in dependency-based approaches, the head-dependent relationship is made explicit by directly linking heads to the words that are immediately dependent on them, bypassing the need for constituent structures [19].

In addition to specifying the head-dependent pairs, dependency grammars allow us to further classify the kinds of grammatical relations, or grammatical function, in terms of the role that the dependent plays with respect to its head such as subject, direct object and indirect object as shown in Figure 2.2.

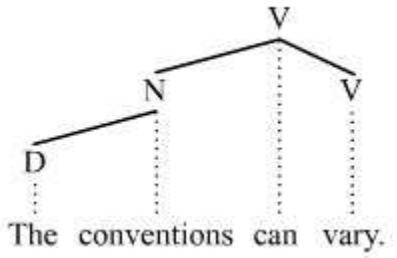

Figure 2.2- Dependency-based grammar representation

2.11.2 Dependency Tree

A dependency tree for a sentence is a directed tree whose nodes are all the words of that sentence. Each arc of a tree represents a single syntactic dependency directed from the head to its modifiers or dependent, and labelled with the specific syntactic function such as SBJ for subject of a sentence, OBJ for object of a sentence, NMOD for modifier of a noun, and so on [29].

There are two types of dependency tree representation: projective and non-projective. If the edges of a dependency tree can be traced in the plane above the words of a sentence without crossing each other, it is called projective.

In a non-projective dependency tree, the edges are crossing each other. Non-projectivity is typically needed to handle long distance dependencies and flexible word order mostly in complex sentences [26].

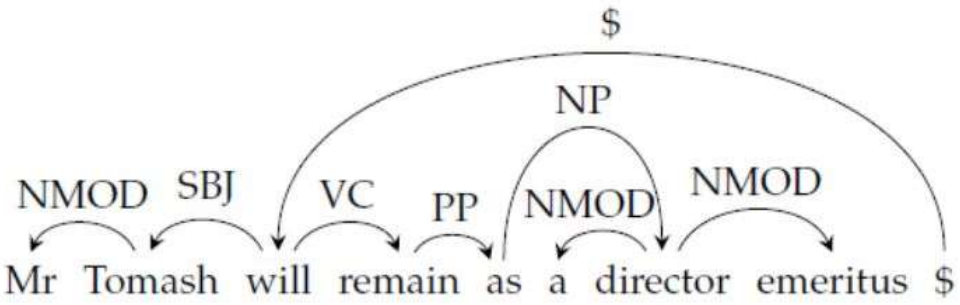

Figure 2.3- Projective dependency tree graph

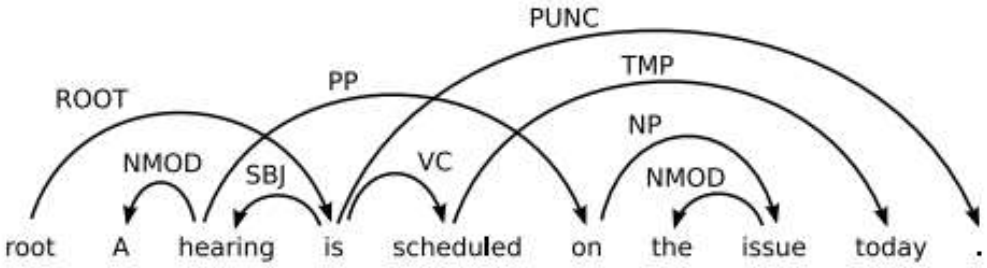

Figure 2.4- Non-projective dependency tree graph

2.11.3 Dependency Parsing

The syntactic structure of a sentence is characterized in terms of the words in the sentence and an associated set of directed binary grammatical relations between the words in dependency parsing [19]. The parsing process is an important step of sentence preprocessing for many NLP applications such as grammar checking, segmentation, classification, sequence prediction problem, etc. [30].

Relations among the words are illustrated above the sentence with directed, labeled arcs from heads to dependents as shown in Figure 2.3 and Figure 2.4. Because the labels are derived from a predefined inventory of grammatical relations, such a dependency relation is known as typed dependency structure. It also comprises a root node, which denotes the tree's root, as well as the entire structure's head [26].

The internal structure of the dependency parse consists of directed relations between lexical items in the sentence. These relationships directly encode important information that is often buried in the more complex phrase structure parses.

2.11.4 Dependency Parsing Algorithms

Syntactic parsing is the process of identifying and assigning grammatical structures to sentences in a text. Parsing is often an automated procedure that plays an important part in the development of NLP applications.

Data-driven statistical analysis from a set of instances, parsers learn to syntactically interpret sentences. This collected data, which is in the form of a treebank, is known as training data. After learning patterns in the training data (treebank), these parsers anticipate dependency trees that aid in determining the likelihood of two words being part of a head-modifier relationship.

Two main approaches dominating data-driven dependency parsing. These are graph-based and transition-based approaches.

Graph-based dependency parsing is the process of constructing a parse tree by predicting the Maximum Spanning Tree (MST) in the input sentence's digraph [29]. Each word corresponds to a vertex in this digraph, and these vertices are all connected by directed edges (arcs). At the learning or training stage, each arc in the graph is awarded a score based on frequency counts in the training data (treebank). The score of the graph is the total of all the arc scores, according to a common arc factorization assumption (weights). When choosing from among the proposed candidates, the parser must discover the highest scoring tree, that is, a subgraph that includes all vertices and just the minimum number of arcs to be connected, known as the MST. A graph-based parser is MSTParser [31].

In transition-based dependency parsing, on the other hand, the training phase entails learning the correct parser action based on the input string and parse history. Parser actions are dictated by the learned model during the parsing step. These actions are based on a shift-reduce parser, which involves putting tokens from a buffer onto a stack (shift) or removing them once they have been entirely digested (reduction) (reduce).

In transition-based parsing, the parser moves from left to right through a sentence, making decisions as to which words will make up dependency pairs with the help of a classifier. The transition-based parsing algorithms use a buffer containing the sentence tokens in linear order, a stack onto which each token is pushed as part of the processing step and an arc list that contains the proposed head-modifier relations [32, 33]. In this approach, the parser looks to see what is on the top of the stack and appearing next in the buffer. Due to the fact that it does not look beyond

the next item in the buffer nor does it undo any decisions it has already made, it is referred to as a greedy algorithm [26].

2.11.5 MaltParser

MaltParser is a freely available transition-based parsing model for research and educational purpose [32]. With MaltParser, we can induce a new parsing model as it is a language independent parsing tool. The parsing model is sometimes called shift-reduce as it reduces the problem of parsing a sentence to the problem of finding an optimal path through a transition system [34].

MaltParser supports a number of parsing and learning algorithms [32]. The parsing algorithms in MaltParser can be categorized into three families, Nivre's Algorithm, Covington's Algorithm, and Stack Algorithm [34]. The concept of configuration, which consists of a stack, an input buffer of words, or tokens, and a set of relations defining a dependency tree, is crucial in transition-based parsing.

Since MaltParser is an inductive dependency parser, it uses a learning algorithm to induce a parsing model. The learning problem of MaltParser is to induce a classifier for predicting the next transition given a feature representation in the form of training data that have dependency structure. MaltParser employs two built-in learning algorithms, LIBSVM and LIBLINEAR [35].

The learning type of LIBSVM is a support vector machine (SVM) that learns by examining hundreds or thousands of data. As a learning system, LIBSVM involves two steps: first, training a data set to obtain a module and second, using the module to predict information of a testing data set [36].

The other learning library, LIBLINEAR, utilizes various linear classifiers including SVMs [35]. LIBLINEAR is an open source library for large scale linear classification. It supports regression and linear SVM [37]. Similar to LIBSVM, LIBLINEAR follows two steps to induce a model: training and predicting.

When the performance of the two libraries is being compared, LIBLINEAR is very efficient than LIBSVM for training large-scale problems. A comparison is made between them on a corpus with more than 600,000 examples. The LIBLINEAR takes only several seconds to train a text classification problem, however, the LIBSVM would take several hours [36]. On the contrary, LIBSVM is more memory efficient than LIBLINEAR since it does not store weight vectors explicitly.

Miguel et al [34] suggests two heuristic methods that one should follow to optimize MaltParser. As shown in the figures below the methods are depicted in the form of decision trees for projective and non-projective algorithms.

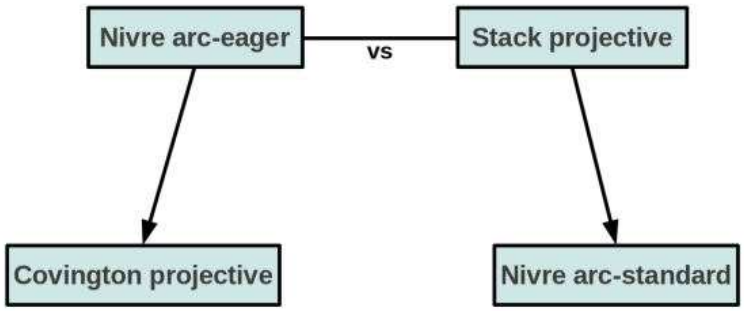

Figure 2.5- Decision tree for best projective algorithm

Decision tree for projective algorithm has four nodes representing projective algorithms. As Miguel et al [34] proposes, first we have to compare the results of Nivre arc-eager and Stack projective algorithms. If Nivre arc-eager is better than Stack projective, then Covington projective will be tested. The final output of this wing is the comparison result of Nivre arc-eager and Covington projective algorithms. On the other hand, if Stack projective outperforms Nivre arc-eager then Nivre arc-standard is going to be tested. Finally, the best algorithm of this wing will be decided after comparing Stack projective and Nivre arc-standard algorithms.

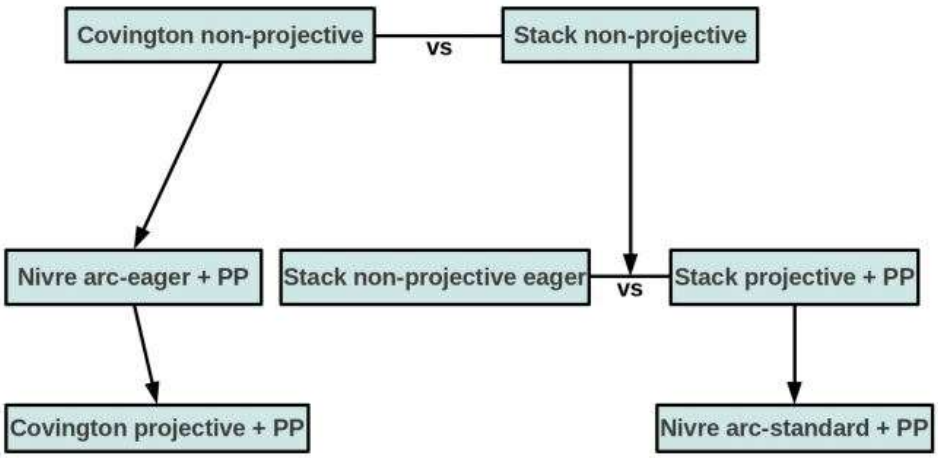

Figure 2.6- Decision tree for best non projective algorithm

Decision tree for non-projective algorithms has seven nodes that represent the algorithms. First, Covington non-projective and Stack non-projective algorithms are compared. If Covington non-projective better performs, then Nivre arc-eager followed by Covington projective are going to be tested combined with pseudo-projective algorithms to handle non-projective structure of the language [34]. The pseudo-projective algorithms include head, path or head plus path [35]. If Stack non-projective overtakes then we made a comparison between Stack non-projective eager and Stack projective combined with pseudo-projective algorithms. If Stack non-projective eager beats, we stop here and compare all the results. If Stack projective combined with pseudo-projective algorithms is better, we further test Nivre arc-standard combined with pseudo-projective algorithms.

2.11.6 Dependency Treebank

A treebank is a corpus of text tagged with syntactic metadata characterizing each sentence's grammatical structure. Dependency analysis is a type of syntactic analysis that involves extracting sets of labelled relations between pairs of words in a phrase. Treebanks are useful not just for linguistic study and corpus analysis, but also as a source of training data for statistical parsing models [38].

The information available on a treebank normally contains tokens, index of the tokens in the sentence, lemma of the tokens, part-of-speech tag, morphological data, the index of the head and the description of that attachment or dependency label [39].

Treebanks vary according to different considerations that are taken during development, including type of syntactic representation and labelling schemes [39]. When creating a treebank, there are a variety of syntactic representations or grammatical formalisms to choose from, all of which are based on different linguistic theories and formalisms. A phrase structure grammar representation, for example, defines constituents and phrases inside sentences in a hierarchical manner, whereas a dependency grammar labels relations between words inside a sentence based on their functional responsibilities.

Labelling schemes define how linguistic structures are represented and labelled in a treebank. Treebanks are often closely linked to the chosen syntactic representation or chosen formalism. They are also influenced greatly by specific linguistic phenomena of the language in question. In addition, there are various ways of assisting treebank development by leveraging from existing NLP resources such as POS taggers, morphological analyzers and existing corpora [39].

In his work, Nivre [40] discusses theoretical assumptions that lay foundation for dependency based analysis. These assumptions address issues including layer (level) of representation, nature of lexical element, nature of dependency types and criteria for identifying heads and dependents words in a sentence.

The layer of representation in dependency treebank development can be either mono-stratal or multi-stratal.

Mono-stratal representation describes the syntax dimensions of a sentence which are word order, agreement and syntactic valency [41]. Syntactic analysis requires both a grammar and a parser, the output of which is a representation of the sentence that reveals the structural dependency relationships between words of a sentence [26].

On the other hand, multi-stratal representation consists of both syntax and semantic dimensions. The semantic layer handles semantic valency [41]. That means, the semantic layer deals with analysis of meaning of words, phrases and sentences [26]. The syntactic and semantic dimensions are illustrated on top and bottom of the words of the sentence respectively.

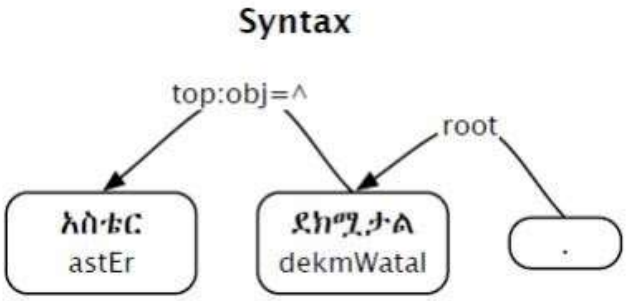

Figure 2.7- Mono-stratal representation

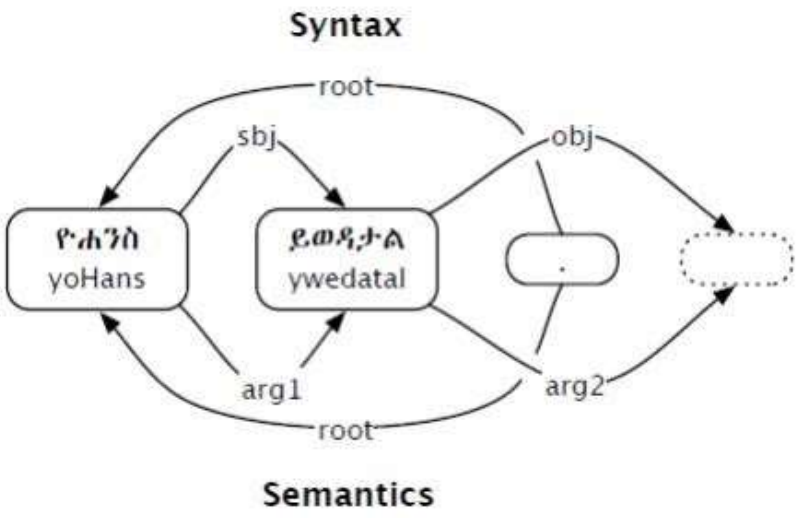

Figure 2.8- Multi-stratal representation

Once we decide the layer of syntactic representation, the next assumption is to identify the nature of lexical elements in the abstraction. Is the lexical element morpheme? Word? Or a multi-word unit? In this regard, Binyam [42] has identified three problems including consonant length or germination, how to represent compound words and how to segment content and functional words that are coupled together.

A Dependency grammar clearly requires criteria for forming dependency relations and identifying the head and the dependent in such relationships. Some criteria for recognizing a syntactic relationship between a head H and a dependent D in a construction C have been given [43] [44]:

1. H determines the syntactic category of C and can often replace C.
2. H determines the semantic category of C; D gives semantic specification.
3. H is obligatory; D may be optional.
4. H selects D and determines whether D is obligatory or optional.
5. The form of D depends on H (agreement or government).
6. The linear position of D is specified with reference to H.

2.11.7 Treebank Data Format

Parsing systems such as MaltParser [32] and MSTParser [31] are language independent systems that allow users to build parsing models using their own choice of treebank. Treebanks are a great platform for linguistic analysis because they give a rich representation of linguistic information in a language. Linguists utilize treebanks to test linguistic theories and analyze syntactic structures

in corpus linguistics. Treebanks are also crucial for the creation of many NLP applications, particularly data-driven parsers.

The dependency tree structure must be encoded in a suitable format in order to train and test a parser algorithm. Various researches employ a variety of forms, including the XML format and the column-based structure. In XML format, a sentence is made up of 3-4 lines of meta-data [32].

```
<sentence>
<word form="This" postag="DT" head="2" deprel="SBJ"/>
<word form="is" postag="VBZ" head="0" deprel="ROOT"/>
<word form="an" postag="DT" head="4" deprel="DET"/>
<word form="old" postag="JJ" head="4" deprel="NMOD"/>
<word form="story" postag="NN" head="2" deprel="PRD"/>
<word form="." postag="." head="2" deprel="P"/>
</sentence>
<sentence>
<word form="So" postag="RB" head="2" deprel="PRD"/>
<word form="is" postag="VBZ" head="0" deprel="ROOT"/>
<word form="this" postag="DT" head="2" deprel="SBJ"/>
<word form="." postag="." head="2" deprel="P"/>
</sentence>
```

Alternatively, in a column-based format each token in a sentence is on a new line, and contains POS information, dependency label and attachment information [30]. A sentence consists of one or more tokens each of which are represented on one line, and encompasses ten fields separated by a TAB. This representation is called either CoNLL-X or CoNLL-U formats. See Table 2.1.

Table 2.1- CoNLL-X data format

No	Field	Description
1	ID	Token counter, starting at 1 for each new sentence.
2	FORM	Word form or punctuation symbol.
3	LEMMA	Lemma or stem (depending on the particular treebank) of word form, or an underscore if not available.
4	CPOSTAG	Coarse-grained part-of-speech tag, where the tagset depends on the treebank.
5	POSTAG	Fine-grained part-of-speech tag, where the tagset depends on the treebank. It is identical to the CPOSTAG value if no POSTAG is available from the original treebank.
6	FEATS	Unordered set of syntactic and/or morphological features (depending on the particular treebank), or an underscore if not available. Set members are separated by a vertical bar (|).
7	HEAD	Head of the current token, which is either a value of ID, or zero ('0') if the token links to the virtual root node of the sentence. Note that depending on the original treebank annotation, there may be multiple tokens with a HEAD value of zero.
8	DEPREL	Dependency relation to the HEAD. The set of dependency relations depends on the particular treebank. The dependency relation of a token with HEAD=0 may be meaningful or simply 'ROOT' (also depending on the treebank).
9	PHEAD	Projective head of current token, which is either a value of ID or zero ('0'), or an underscore if not available. The dependency structure resulting from the PHEAD column is guaranteed to be projective (but is not available for all data sets), whereas the structure resulting from the HEAD column will be non-projective for some sentences of some languages (but is always available).
10	PDEPREL	Dependency relation to the PHEAD, or an underscore if not available

Another variation of column-based format is CoNLL-U [45] format which is a revised version of CoNLL-X format. It also has ten fields to represent a sentence. The fields DEPS and MISC replace the obsolete fields PHEAD and PDEPREL of the CoNLL-X format [45] as shown in Table 2.2.

Table 2.2- CoNLL-U data format

No	Field	Description
1	ID	Word index, integer starting at 1 for each new sentence; may be a range for multiword tokens; may be a decimal number for empty nodes.
2	FORM	Word form or punctuation symbol.
3	LEMMA	Lemma or stem of word form.
4	UPOS	Universal part-of-speech tag.
5	XPOS	Language-specific part-of-speech tag; underscore if not available.
6	FEATS	List of morphological features from the universal feature inventory or from a defined language-specific extension; underscore if not available.
7	HEAD	Head of the current word, which is either a value of ID or zero (0).
8	DEPREL	Universal dependency relation to the HEAD (root iff HEAD = 0) or a defined language-specific subtype of one.
9	DEPS	Enhanced dependency graph in the form of a list of head-deprel pairs.
10	MISC	Any other annotation.

2.12 Universal Dependency

Universal Dependency (UD) [46] is a project to create cross-linguistically consistent treebank annotation for a variety of languages, with the purpose of simplifying multilingual parser development, cross-lingual learning, and parsing research from the standpoint of language typology. The UD initiative's goal is to create a universal inventory of categories and norms that will allow for consistent annotation of identical constructs across languages while allowing for language-specific expansions as needed. UD annotation standards are revised with each new iteration in order to achieve treebanks that are more robust, successful, and balanced in terms of the tradeoff between computational tractability and linguistic correctness. At the time of this writing, the most recent UD release was version 2.0 with more than 100 treebanks and 60 languages [46].

Currently, the data format being used in Universal Dependency is CoNLL-U format [47, 48, 49]. In CoNLL-U, A sentence is broken into individual words (tokens), each of which is then analyzed based on the standards and guidelines suggested by UD.

While preparing treebanks, tokens are going to be tagged with universal POS tagsets and optionally with language specific POS tagsets. If language specific POS tags are used, the treebank-specific documentation should define a mapping from specific to universal POS tagset [45] as shown in Table 2.3.

Table 2.3- Universal POS tagset

Universal POS tags	Description
ADJ	adjective
ADP	adposition
ADV	adverb
AUX	auxiliary
CCONJ	coordinating conjunction
DET	determiner
INTJ	interjection
NOUN	noun
NUM	numeral
PART	particle
PRON	pronoun
PROPN	proper noun
PUNCT	punctuation
SCONJ	subordinating conjunction
SYM	symbol
VERB	verb
X	other

Similarly, tokens are annotated with morphological features such as gender, person, case, definiteness, etc. from universal feature inventory [45] as shown in Table 2.4.

The syntactic annotation of words describe the dependency relationship of head and dependent words. The dependency relation value should be a universal dependency relation or a language-specific subtype of such a relation defined in the language-specific documentation [45].

Table 2.4- UD Relations

Universal Features Inventory		
Inflectional Features		**Lexical Features**
Verbal	**Nominal**	
VerbForm	Gender	PronType
Mood	Animacy	NomType
Tense	Number	Poss
Aspect	Case	Reflex
Voice	Definite	
Person	Degree	
Negative		

2.13 Evaluation Metrics

Accuracy, precision, recall, and F1 measure will be used as evaluation metrics. They are used to test the performance of information retrieval, information extraction, and classification applications. They are measured based on the predictions of the induced models. Predictions of these models can be either true, which means correctly classified, or false, to mean incorrectly classified.

The result of classification is described by confusion matrix and it can only have one of four results. In confusion matrix, the row of the table represents the predicted class and the column represents the actual class. These results are true positive (TP) and true negative (TN) which means the number of positive and negative instances that are correctly classified, false positive (FP) and false negative (FN) which means the number of positive and negative instances that are incorrectly classified, respectively as shown in Table 2.5.

Table 2.5- Confusion matrix

	Actual positive class	**Actual negative class**
Predicted positive class	True positive (TP)	False negative (FN)
Predicted negative class	False positive (FP)	True negative (TN)

The selected evaluation metrics can be determined based on the confusion matrix:

A. Accuracy: measures the ratio of correct predictions over the total number of evaluated instances.

$$accuracy = \frac{TP+TN}{TP+FP+TN+FN} \qquad \text{Equation 2.1}$$

B. Precision: measures the positive patterns that are correctly predicted from the total predicted patterns in the positive class.

$$precision = \frac{TP}{TP+FP} \qquad \text{Equation 2.2}$$

C. Recall: measures the ratio of positive patterns that are correctly classified.

$$recall = \frac{TP}{TP+TN} \qquad \text{Equation 2.3}$$

D. F1: represents the harmonic mean between recall and precision values.

$$F1 = \frac{2*precision*recall}{precision+recall} \qquad \text{Equation 2.4}$$

We will also use Labeled Attachment Score (LAS) and Unlabeled Attachment Score (UAS), to evaluate dependency parser [33].

- ✓ LAS (Both Right): A token is counted as a hit if both the head and the dependency label are the same as in the gold-standard data.
- ✓ UAS (Head Right): A token is counted as a hit if the head is the same as in the gold-standard data.

Chapter 3: Related Work

3.1 Introduction

A variety of studies have been conducted on grammar checking and other related tasks. Grammar checking systems for several languages are among them. The majority of these works are aimed at developing grammar checkers for English, Arabic, Afaan Oromo, Amharic, and Tigrinya. We'll look at what approaches were utilized, what language the study was done for, who did the study, what grammar notation the authors used in the study, what and how a test data-set was employed, and what the study's performance evaluation revealed in each review of work. Grammar checking methods included syntax-based, rule-based, statistical, and hybrid approaches in the studies examined. In the studies, grammar notations such as phrase-structure or dependency grammar were also employed. The following are the sections where the most related studies will be discussed in detail.

3.2 Grammar Checker for English

Maxim Mozgovoy [11] developed dependency-based rules for grammar checking with LanguageTool. The study has attempted to overcome the problems that cannot be solved with a rule-based grammar checking using a dependency based rules for grammar checking to show immediate word-word relationships and their syntax allows writing rules that analyze word-word dependencies in a given phrase. They have shown real examples of language phenomena, where such rules are much more helpful than their built-in LanguageTool instruments.

Daniel Naber [13] developed an English language style and grammar checker that is open source. The work's outcome can be utilized as a standalone application or as part of a word processor. The style and grammar checker includes 54 grammar mistake rules with explanations and 4 python building rules. The majority of the rules are written in XML format. Rules created in Python detect rules that are difficult to write in XML. It takes an input text and assigns a part-of-speech tag to each token in order to detect problems. After the word has been tagged, each sentence is divided into chunks, such as a noun phrase or a verb phrase. The processed text is then compared to all of these previously set error rules. The text is expected to be erroneous if the rule matches. Finally, the system displays the error and its cause. Precision and recall are used to assess the suggested method. A corpus of yet unprocessed text, the Mailing List Error corpus (contains just sentences

with errors), and the British National Corpus, with all errors marked up, are required to determine these values. However, at the time, such a corpus was not publicly available. As a result, the system was evaluated using two corpora. The Mailing List Error corpus contains 224 errors that were mostly found on worldwide public mailing lists, and the majority of the letters deal with technical difficulties such as programming. The error checker properly recognizes 42 of these mistakes. The British National Corpus is a British English commercial corpus. It's a proofread text with 100 million words culled from fictional literature, technical documents, newspapers, and articles, among other sources. When the checker was run on this corpus, it found 16 mistakes out of 75,900 sentences. The majority of these problems are false alarms because the corpus has been proofread. The text was erroneously split into sentences, a word was assigned an inaccurate part-of-speech tag, or the rule is just not strict enough and activates too often, as the researcher explained.

3.3 Grammar Checker for Afaan Oromo

Debela Tesfaye [12] has developed Afaan Oromo grammar checker that accepts paragraph as input and tokenizes it into sentences, further into words. Each word is assigned a part of speech using a tagger based HMM (hidden Markov model) that uses manually tagged corpus. Certain types of affixes are removed using substitution rules of a stemming algorithm that applies only when a certain condition holds. After reducing the affixes, a number, gender, and tense agreement between the subject and verb, subject and adjective, main verb and subordinate verb is discovered. Finally, the system can provide correct sentence alternatives in case of disagreement. The system has outstanding results, but it fails to detect grammatical errors in complex and compound sentences. The system is populated with some incorrectly tagged words, it results in the system to generate false flags. Their evaluation shows that the grammar checker has achieved a precision of 88.89% and recall 80.0% performance.

3.4 Grammar Checker for Arabic

Khaled and shaalan [4] have developed Arabic Grammar Checker which is a syntax-based grammar checker for modern standard Arabic. To detect grammatically erroneous Arabic sentences, the system uses deep syntactic analysis and a feature relaxation method. This program can detect and offer corrections for a variety of common grammatical problems. Arabic Grammar Checker has basically composed of two parts: a morphological analyzer and a standard bottom-up

chart parser including a grammatical checking handler. The tool is tested and shows good results in short and simple sentences. The evaluation does not show how much capable the tool is on long and complex sentences. This grammar checker does not show dependence among the words in a sentence.

3.5 Grammar Checker for Amharic

In their study, Aynadis and Yaregal [13] constructed an Amharic grammar checker, using a rule-based method for simple sentences and a statistical method for both simple and complex phrases. Rules are manually constructed, and n-gram and probabilistic methods, as well as statistical approaches, were utilized to check for grammatical faults in Amharic sentences. Sentence patterns and occurrence probabilities are automatically retrieved from a training corpus and saved in the repository. Using the stored patterns and their probabilities, sentence probabilities are calculated and some threshold and probability of the sentence are used to determine the grammatical correctness. However, the efficiency of the system is affected by the ineffective morphological analyzer and the quality of the corpus used. Moreover, their grammar checker cannot represent the dependency among the words in a sentence.

Abraham Gebreamlak [50] have designed and developed a dependency based Amharic grammar checker. Accordingly, they proposed an Amharic grammar checker integrated with dependency parsing system. The parser is based on dependency grammar formalism through which relationship of a word and its modifier is identified. They have implemented the system prototype using Python 3.7.2 programming language, UDpipe 2.0 to obtain and evaluate tokenizing and tagging models; MaltParser 1.9.2 to induce dependency parsing models and MaltEval 1.0 to evaluate the result of parsing model. The models were trained with a dependency treebank for Amharic. Lastly, they reported the performance of the induced models and grammar checker with randomly selected sentences from dependency treebank. The tokenizer and the tagger were also evaluated with raw texts collected from newspapers. Their findings show that the tokenizer performs well with an accuracy of 100%. However, the tagger's performance was 43.11%. The dependency parser was also evaluated during development to select best algorithm which was Covington Non-projective algorithm.

3.6 Grammar Checker for Tigrinya

Abdella Nurahmed [6] has developed a hybrid grammar checker for Tigrinya language by integrating statistical and rule-based approaches. The statistical part has two phases; training and checking phases. In the training phase, unique patterns of word and tag n-grams are extracted and stored. In the checking phase, patterns of word n-grams of the input text are extracted similar to the training phase. Then, word n-grams that are not found in the database are considered as assumed errors. A threshold is used to avoid false positives. All assumed errors are weighted individually and summed up to calculate the overall error. Sentences are declared as incorrect when the overall error is above the threshold. For errors, that the syntactical approach cannot handle, a rule-based approach is used. Rules for common grammatical errors are constructed manually. When the pattern of the input text matched with the stored rules, the grammar of the text is considered incorrect. The statistical approach depends on the quality and amount of the statistical data and tag set whereas the rule-based approach was affected by the incompleteness of the rules. Further, the morphological analyzer also affected the performance of the grammar checker as a whole. The grammar checker scores the highest performance with recall values 91.95% for automatically and 96.48% for manually analyzed test data set and precision value 91.86% for manually analyzed test data set. Resulting from the capability of the approaches used, this grammar checker cannot show the word-word relationships and the sentence representation is more complex and subject to produce syntactically correct but semantically non-sense sentences.

3.7 Summary

As we can see from the above discussion on related work, there are different works on grammar checking for other languages. Since there is a difference in morphology and grammatical structure among natural languages, it is difficult to share and apply grammar checkers of a specific language for use directly in another language. We can see that from the related works there are problems that a phrase structure rule-based approach cannot address. These problems can be addressed by using dependency-based grammar checking approach. Thus, in this work we will use dependency-based approach to develop grammar checker for Tigrinya.

Chapter 4: Dependency-based Tigrinya Grammar Checker

4.1 Introduction

In this chapter, we will look at the overall design and architecture of the grammar checking system, what processes are involved in the system, what components are required and used for the processes, the flow of processes and how the components are organized, what each component requires and produces as a result of its task, how each component performs its task, and what algorithms or models are used to implement components of the system. Finally, we will see what and how test data-sets are used to evaluate the grammar checker system.

4.2 Architecture of the System

The proposed grammar checker system is composed of different components responsible for different tasks in the grammar checking process. These components are tokenizer, POS tagger, and morphological analyzer, dependency model of the language, dependency parser, relation extractor and agreement checker. These components are grouped into three modules named preprocessing module, dependency extraction module, and grammar checking module. The components tokenizer, POS tagger, and morphological analyzer are subcomponents of the preprocessing module. The dependency parser is subcomponent of the dependency extraction module and it uses the dependency model of the language as a reference. The grammar checking module includes the grammatical relation extractor and grammatical agreement checker components as shown in Figure 4.1.

The tokenizer in dependency-based Tigrinya grammar checker (DBTGC) accepts Tigrinya text as input and tokenizes the text into sentences then to words and punctuations. The tagger recognizes each token and tags it with the appropriate syntactic structure (POS tag and others). The morphological feature annotator identifies the morphological features of each tagged token (gender, person, tense, number etc.). The dependency parser identifies the root of the sentence, head-modifier (dependent) pairs of tokens and the dependency relation between each token in the sentence. The relation extractor translates the dependency relations into grammatical relations and filter out the most significant relations and the words in these relations alongside their syntactic information for agreement checking.

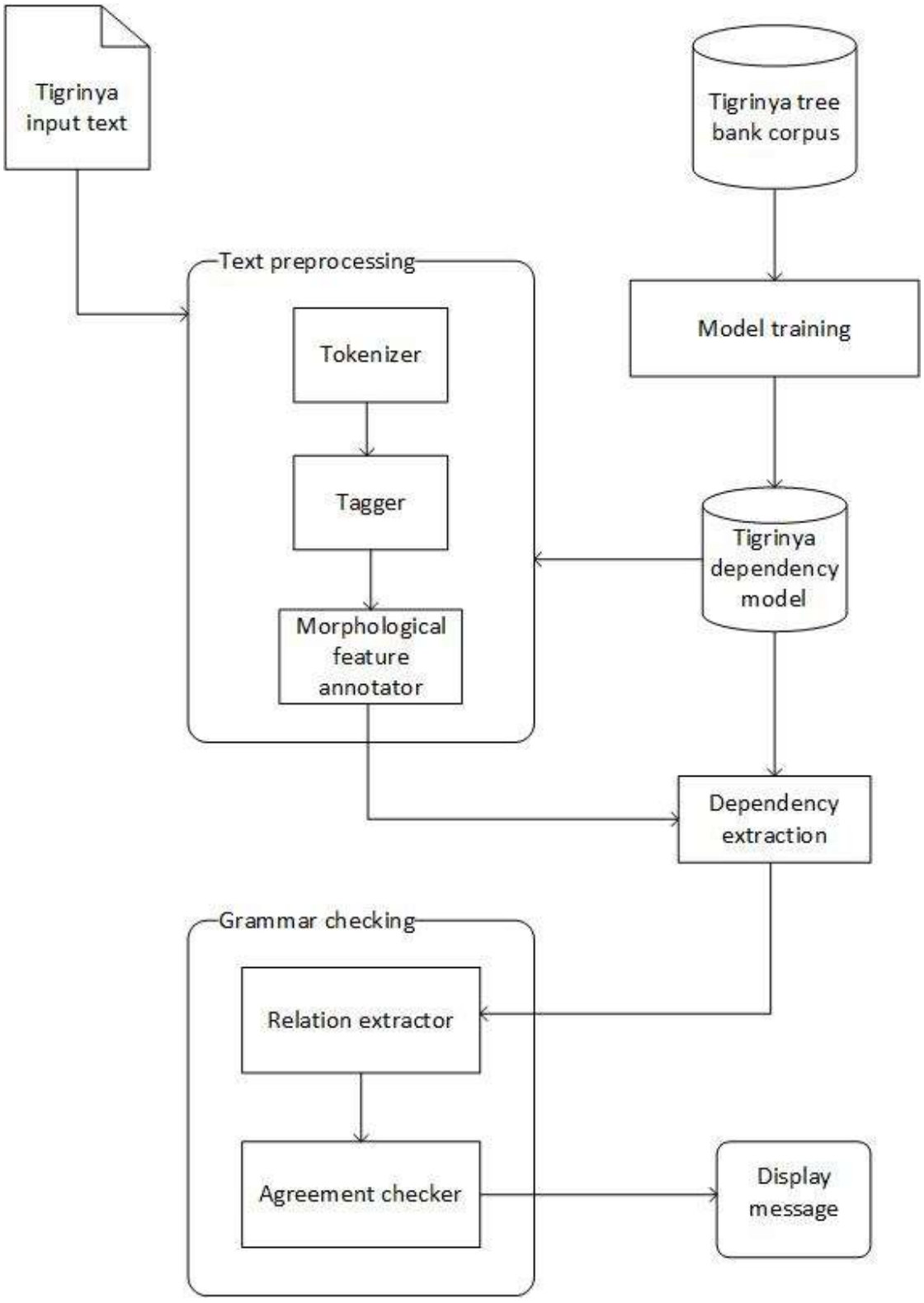

Figure 4.1- Architecture of Dependency Based Tigrinya Grammar Checker (DBTGC)

Finally the agreement checker compares the words in each relation if their syntactic information matches and presents the result. However, prior to using the DBTGC; dependency model of the language for the components in the preprocessing and dependency extraction modules must be trained and prepared for use in the DBTGC. This model is trained by using a text corpus prepared in Computational Natural Language Learning (CoNLL-U) data format.

These information are token index (ID), the word form (FORM), lemma (LEMMA), universal POS tag (UPOS), language-specific POS tag (XPOS), morphological features (FEATS), head of the token (HEAD), dependency relation (DEPREL) of the token to its head, dependencies (DEPS), and other miscellaneous information (MISC). Here, the Tigrinya corpus is taken from the work of Yemane [51], Nagaoka POS tagged Tigrinya corpus; manually converted into a dependency treebank using CoNLL-U data format.

4.3 Input Text Preprocessing

An input text to the DBTGC should pass through steps of preprocess to make it suitable for further analysis. These steps include cleaning unnecessary content, tokenizing, tagging, and morphological feature annotation. This process yields a partially filled CoNLL-U representation of the input text. Once an input text passed the preprocess steps, then it is ready for further analysis.

A. Tokenizer

This component takes an input text and splits the entire text into sentences. It then splits each sentence into tokens which means words, punctuations and other symbols. It uses sentence marker and token (word) marker symbols to split an input text into sentences and sentences into tokens respectively. These markers can vary from language to language. In our context the sentence marker symbol in Tigrinya can be a four point symbol (::) that serves as a full stop, a question mark (?), or an exclamation mark (!) and the token (word) marker can be a space character ( ), or occurrence of symbols other than sentence markers before or after a word with or without a separating space character (“, ‘, -, ፣, ፤, etc.). An example tokenization process is shown below.

“ኣሎኒ እትብልዮ ዘለኺ ቅዳሕ እዩ ። ዝበል ኮይኑ ረኺብናዮ ኣሎና ።" Example 1

“ALONI ETIBILIYO ZELEKHI KIDAX EYU /ZIBIL KOYNU REKKHIBINAYO ALENA”

“It's the copy you say you have” / “We found it to be”

The tokenizer splits the above text into sentences as follows.

1 ኣሎኒ እትብልዮ ዘለኺ ቅዳሕ እዩ ። Sentence 1

2 ዝበል ኮይኑ ረኺብናዮ ኣሎና ። Sentence 2

While splitting a sentence into tokens, a combination word of functional and content word can be encountered. In this case these combined words should be separated into their true independent forms as shown in Table 4.2.

Table 4.1- Word tokenization of an input text (sentence 1 and sentence 2)

1	ኣሎኒ
2	እትብልዮ
3	ዘለኺ
4	ቅዳሕ
5	እዩ
6	።
1	ዝበል
2	ኮይኑ
3	ረኺብናዮ
4	ኣሎና
5	።

As a result, the tokenizer component returns a set of tokens for the input text and it fills the index (ID) and word form (FORM) fields which are the first and second columns in the CoNLL-U representation of the input text respectively.

The tokenizer component was developed in a way that involves training and experiments on a number of tokenizer models. The training and experiment processes is performed using a training and testing datasets and an annotation model training toolkit package called UDPipe.

B. Part of Speech (POS) Tagger

Given a set of tokens, the POS tagger component adds the appropriate POS tags and identify the lemma to each token in the selected sentence. These POS tags include the UD POS tags (UPOS) and Tigrinya-specific tags (XPOS). The POS tagset used in the tagging process are depicted in Table 4.3.

Table 4.2- UPOS and XPOS tag-sets

#	UD POS tag	Tigrinya POS tag	Description	Example
1	ADJ	ADJ	Adjective	ሰፊሕ “wide”, ዓብዪ “big”
2	ADP	PRE	Preposition	ናብ “to”, ብ “by /through”
3	ADV	ADV	Adverb	ብጣዕሚ “very”, ምሒር “very”
4	AUX	V_AUX	Auxiliary	እዩ እያ “is”
5	CCONJ	CON	Conjunction	ወይ “or”, ን “and”
6	DET			
7	INTJ	INT	Interjection	እወ “yes”
8	NOUN	N	Noun	መኪና “car”, ስራሕ “job”
9	NUM	NUM	Numeral	2 /ክልተ “two”
10	PART			
11	PRON	PRO	Pronoun	ኣነ “I”, ንሳ “she”
12	PROPN	N_PRP	Proper noun	ናሆም “nahom”, ዓድዋ “adwa”
13	PUNCT	PUN	Punctuation	። ፣ ?
14	SCONJ			
15	SYM			
16	VERB	V		ከይዱ “he went”, ሰሪሑ “he did”
17	X			

POS tagged data of sentence 1 looks as shown in Table 4.4.

Table 4.3- POS tagged representation of sentence 1

1	ኣሎ፡ኒ	ኣሎ፡ኒ	VERB	V_AUX
2	እት፡በልዮ፡	እት፡በልዮ፡	VERB	V_REL
3	ዘለ፦ኺ	ዘለ፦ኺ	ADV	V_REL
4	ቅዳሕ	ቅዳሕ	ADJ	ADJ
5	እዩ	እዩ	AUX	V_AUX
6	።	።	PUNCT	PUN

The POS tagger component returns a set of POS tagged (XPOS and UPOS) tokens and fills the UPOS or XPOS and lemma fields (the third, fourth and fifth columns) in the CoNLL-U representation of the input text.

The tagger component was developed through a process of training and experiments on a number of tagger models using a training and testing datasets and annotation model training toolkit called UDPipe. The developed tagger model includes XPOS, POS, and lemma, and morphological feature taggers.

C. Morphological Analyzer

The morphological analyzer takes a set of POS and lemma tagged tokens. It annotates each token with available morphological features. These features can be of nominal, verbal, or other type. These features are sometimes called syntactic information. Some examples of such features are gender, number, tense, aspect etc. Here, the given tokens are annotated with values for the available attributes of morphological features. The set of these features is shown in Table 4.5.

Table 4.4- Set of morphological features

#	Category	Feature	Tag	Description
1	Nominal	Gender	Com	Common
			Masc	Masculine
			Fem	Feminine
			Neut	Neuter
		Number	Sing	Singular
			Plur	Plural
2	Verbal	Tense	Past	Past
			Pres	Present
			Fut	Future
			Imp	Imperfect
			Pqp	Pluperfect
		Person	1	1st person
			2	2nd person
			3	3rd person
		Voice	Act	Active
			Pass	Passive

These features are added into the CoNLL-U representation of each sentence in an attribute-value pairs and each pair is separated by a vertical bar (|). The attribute and its value are separated by equal sign (=).

Morphological features are represented as attribute-value pair, with an equal sign (=) separating the attribute from the value sorted alphabetically by attribute name. There are cases where an attribute can have multiple values which are separated by comma alphabetically sorted: Case=Acc, Det. In sorting, uppercase letters are considered identical to their lowercase counterparts [45] as shown in Figure 4.2.

The morphological analyzer returns a set of POS tagged and morphologically (morpho-syntactically) annotated tokens for a sentence. It fills the FEATS field (the sixth column) in the CoNLL-U representation of the input text.

The morphological analyzer model is developed together with the tagger models.

1	አሎኒ	አሎኒ	VERB	V_AUX	Gender=Com|Number=Sing|Person=1|Tense=Pres
2	እትብልዮ	እትብልዮ	VERB	V_REL	Gender=Fem|Number=Sing|Person=2|Tense=Pres
3	ዘሰኺ	አሎኒ	ADV	V_REL	Gender=Fem|Number=Sing|Person=2|Tense=Pres|Voice=A...
4	ቅዳሕ	ቅድሕ	ADJ	ADJ	Gender=Neut|Number=Sing|Person=3|Voice=Pass
5	እዩ	እዩ	VERB	V_AUX	Gender=Neut|Number=Sing|Person=3
6	።	።	PUNCT	PUN	_

Figure 4.2- Morphological feature annotation of sentence 1

4.4 Model Training

Given a training data corpus in CoNLL-U data format and appropriate training options or dependency parsing algorithm, DBTGC is trained to generate a model that can handle the tokenizing, tagging and dependency extraction tasks. Once the model is generated, it can be used in the DBTGC as a reference model for the tokenizer, tagger and dependency parser components.

4.5 Language Dependency Model

The dependency parser model is trained using a data-driven parser generator system called MaltParser. MaltParser is a language-independent system that learns from a dependency tree statistical composition. We have used a tree bank from the converted Nagaoka POS tagged corpus work of Yemane [51]. The development process involves training and experiments on a number of algorithms implemented in MaltParser, and the model was finally trained and generated using the algorithm that yields best results. This model is used to guide the dependency parser on how to extract the root, head-dependent pairs and their relations based on previously learned sentence structure and patterns.

4.6 Dependency Extraction

This component takes the partially filled CoNLL-U representation of the input text. It parses for the root of the sentence, head-dependent pairs of tokens, and the dependency relation between each token in the head-dependent pairs of the sentence.

Dependency Parser

This is a parsing tool that predicts and extracts the root, head-dependent pairs of tokens and the dependency relations between each token in the input text. It uses the dependency model of the language to refer to a previously learned sentence dependency structure to perform the parsing task. The parser accepts sentences in CoNLL-U format in which all fields are populated with the required information except HEAD and DEPREL fields. The parser then predicts HEAD and DEPREL values. As a result, the dependency parser returns a set of dependency parsed sentences as shown in Figure 4.3.

1	ኣሎኒ	ኣሎኒ	VERB	V_AUX	Gender=Com|Number=Sing|Person=1|Tense=Pres	2	2:csubj	_	_
2	እትብልዮ	እትብልዮ	VERB	V_REL	Gender=Fem|Number=Sing|Person=2|Tense=Pres	4	4:advcl	_	_
3	ዘለኺ	ኣሎኒ	ADV	V_REL	Gender=Fem|Number=Sing|Person=2|Tense=Pres|Voice=A...	2	2:advmod*	_	_
4	ቅጻሕ	ቅድሕ	ADJ	ADJ	Gender=Neut|Number=Sing|Person=3|Voice=Pass	5	5:obj	_	_
5	እዩ	እዩ	VERB	V_AUX	Gender=Neut|Number=Sing|Person=3	0	0:advcl	_	_
6	።	።	PUNCT	PUN	_	5	5:punct	_	_

Figure 4.3- Dependency extraction result of sentence 1

4.7 Grammar Checking

This module checks the sentence for any grammatical disagreements among its words based on given common grammatical agreements and dependency parsed sentence. It has two components to carry out the process. These components are relation extractor and agreement checker. The relation extractor searches for head-dependent pairs along their dependency relations in the sentence. The agreement checker in turn checks the grammatical agreement in each head-dependent pair using their corresponding morphological features and the common grammatical agreements as a reference.

A. Grammatical Relation Extractor

The text preprocessing and dependency extraction modules have been able to recognize the dependency structure of sentences. However, the information is not yet clear to evaluate if head-dependent words are in agreement. Therefore, we need to make the result suitable for agreement checking through a relationship defined by five parameters for each word extracted from the parsing result. The parameters are ID, FORM, UPOS, FEATS and DEPREL as shown in Table 4.6. The ID field specifies where the word is located in the sentence. The FORM field is the word itself. The UPOS field is the universal POS tag of the word, the FEATS field is the morphological features of the word such as gender, number, person, tens, aspect, etc. And the DEPREL field is the relationship between the head and dependent words. We need a function getHead() that returns the id of the head of the current word. So that, we can check the agreements between the paired words in terms of number, gender, and person.

Table 4.5- A sentence structure as an object

Sentence

Word	**Word**	**…**	**Word**
ID	ID		ID
FORM	FORM		FORM
UPOS	UPOS		UPOS
FEATS	FEATS		FEATS
DEPREL	DEPREL		DEPREL

Depending on the relational model depicted in Figure 4.4, a parsed sentence will definitely has more than one relational pairs that determine the size of the relational dataset during construction.

For example Sentences 1 has six pair of words (አሎኒ, እትብልዮ) (እትብልዮ, ዘለኺ) (እትብልዮ, ቅዳሕ) (ቅዳሕ, እዩ) (እዩ, ።) and (።, እዩ) and their corresponding dependency relations are csubj, nsubj, adjmod, obj, root and punct respectively.

```
# head-dependent pairs
csubj (‘ኣሎኒ’, ‘እትበልዮ’)
nsubj (‘እትበልዮ’, ‘ዘለኺ’)
adjmod (‘እትበልዮ’, ‘ቅዳሕ’)
obj (‘ቅዳሕ’, ‘እዩ’)
root (‘እዩ’, ‘።’)
punct (‘።’, ‘እዩ’)
```

Figure 4.4- Head-Dependent relationship of sentence 1

Here, we ignore the root and punct relationships which exist between root word “እዩ” and a punctuation mark “።”; and between the punctuation mark “።” and the root word “እዩ” because we do not need them for agreement checking. Therefore, the head-dependent relationships of the Sentence will only be four as shown in Figure 4.5.

```
# selected head-dependent pairs
csubj (‘ኣሎኒ’, ‘እትበልዮ’)
nsubj (‘እትበልዮ’, ‘ዘለኺ’)
adjmod (‘እትበልዮ’, ‘ቅዳሕ’)
obj (‘ቅዳሕ’, ‘እዩ’)
```

Figure 4.5- Selected head-dependent pairs

So, the head-dependent relationships of input sentences are extracted through Algorithm 4.1.

```
BEGIN
READ parse sentence from file
INIT train to sentence, heads list[], forms list[],relations list[]
FOR each value in train
  FOR each token in train
    ADD head token in heads list
    ADD form token in forms list
    ADD deprel token in relations list
    ADD feats token in morph_ feats list
SET x to the length of heads list
SET i to 0
FOR i in range of x
  SET h to the ith head minus one word in the forms list
  SET d to the ith word in the forms list
  SET rel to the ith relation in the relations list
  CHECK IF rel of i is not root or punct
    OUTPUT rel, d and h
    //which represent the dependent and head words respectively
END
```

Algorithm 4.1- Head-Dependent relation extraction algorithm

B. Grammatical Agreement Checker

This component takes a set of grammatical relations and the words in the relation with their respective syntactic information. It then extracts the common keys from the morphological features of the words in each relation and compares the words for grammatical agreement using the values for the common keys.

The common grammatical agreements in Tigrinya that we have seen in Section 2.5, are mapped in accordance with the type of grammatical relationships returned from the relation extractor. The relation extractor returned four dependency relationships including nsubj, obj, advmod and amod as shown in Figure 4.5. These relationships are mapped to the agreements identified previously as shown in Table 4.7. For example, the nsubj is subject-verb agreement.

Table 4.6- Grammatical agreements - dependency relations mapping

#	Grammatical agreement	Dependency relation	Description
1	Subject – verb	nsubj	nominal subject of verb
2	Object – verb	obj	object of verb
3	Adjective – noun	amod	adjective of noun
4	Adverb – verb	advmod	adverb of verb

In order to check agreements based on dependency relationship mapping, we need to first extract morphological features of head and dependent words and represent them in a dictionary. A dictionary is a way of representing information in the form of key-value pair data structure. Each key-value pairs are separated by a comma. The keys and values, however, are separated by a colon. All the key-value pairs are enclosed by a curly braces to make up a single dictionary. The value of a particular item is traversed through the associated key item.

{key1:value1, key2:value2, key3:value3 …keyn:valuen}

For example, the linguistic information of ‘እትብልዮ’ /ETIBILIYO/ are third person, singular number, feminine gender, active voice, and its head is ‘ዘለኺ’ /ZELEKHI/ which is also a third person, singular number, feminine gender. Therefore, their dictionary representations will be:

‘እትዉበልዮ’ /ETIBILIYO/ {Gender: Fem, Number: Sing, Person: 3, Voice: Act}

‘ዘለኺ’ /ZELEKHI/ {Gender: Fem, Number: Sing, Person: 3}

Next, we look for the common keys of the two dictionaries. Since we do not specify the same morphological feature value more than once for the same word during input sentence formatting, the keys of the dictionaries are unique. The common keys of ‘እትዉበልዮ’ /ETIBILIYO/ and ‘ዘለኺ’ /ZELEKHI/ are gender, number and person. Such common keys are extracted using Algorithm 4.2.

```
BEGIN
// include the relation extraction algorithm
INVOKE relation extraction
//Search for morphological features of dependent words
FOR i in range of list of words
  ADD the ith item in morph_feats to dep_morp_feats
//Search for the morphological features of head words
SET x to the length of heads list
SET i to 1
FOR i in range of x
  SET j to the ith head word in the head list
  ADD the (j-1)th morph_feats to head_morp_feats
//Looking for common keys
FOR i in dep_morp_feats
  IF i in head_morp_feats AND ith key of dep_morp_feats = ith key of
head_morp_feats
    ADD i to common_key
END
```

Algorithm 4.2- Common keys extraction algorithm

Figure 4.6- Result of common key extraction

There are duplicate values in the extracted common keys. But these keys are for different relations. The first three keys are for relation nsubj that occurs between the subject and the verb. The other two common keys of the object and the verb of the sentence.

These common keys will be used to compare the values of the two dictionaries. When we traverse through the two dictionaries of the paired words using the common keys, we compare the associated values of the two dictionaries. For example, the first key among the common items is gender. Then, we compare the values of gender of the two words - 'እትብልዮ' /ETIBILIYO/ and 'ዘለኺ' /ZELEKHI/. The gender comparison returns true for nsubj relation between 'እትብልዮ' /ETIBILIYO/ and 'ዘለኺ' /ZELEKHI/. It continues the comparison for person and number. However, there is no comparison for voice as there it is not a common key for the pairs. We make similar comparison for the remaining common keys and relations which is depicted in Table 4.8.

Table 4.7- Comparison of Words of Sentence 1

Relation	**Dependent**	**Head**	**Comparison keys**		
			Gender	**Person**	**Number**
csubj ('ኣሎኒ', 'እትብልዮ')	ኣሎኒ	እትብልዮ	True	False	True
nsubj ('እትብልዮ', 'ዘለኺ')	እትብልዮ	ዘለኺ	True	True	True
adjmod ('እትብልዮ', 'ቅዳሕ')	እትብልዮ	ቅዳሕ	False	NA	True
obj ('ቅዳሕ', 'እዩ')	ቅዳሕ	እዩ	False	NA	True

If there are no common keys between two words in a relation, we do not compare these words to check if their grammatical structure agree. Similarly, a common key that is not found in both words

is not used to compare the words. Finally the agreement in grammatical structure is checked using Algorithm 4.3.

```
BEGIN
// include the common key extraction algorithm
INVOKE common key extraction
//Agreement checking
FOR i in range of word list
  Assign the ith common_key to k
  IF the kth element of head_morp_feats is equal to the kth element
of dep_morp_feats70
    OUTPUT The agreement based on the relation and comparison key
//Disagreement checking
FOR i in range of word list
  Assign the ith common_key to k
  IF the kth element of head_morp_feats is equal to the kth element
of dep_morp_feats
    OUTPUT The disagreement based on the relation and comparison key
END
```

Algorithm 4.3- Agreement and disagreement checking algorithm

```
# grammar checking result message

The subject and the verb agree on: {'person': {3}, 'gender': {Fem}}

The object and verb agree on: {'person': {3}, 'number: {Sing}}
```

Figure 4.7- Agreement checking results display

Finally, it returns a message about the agreement type along with the linguistic information that the pair of words agree on. As depicted in Figure 4.11, the subject and the verb agree with respect to Gender and Person. That is both the subject and the verb are third person feminine. The object and the verb also agree on number and person as both are third person singular.

Chapter 5: Experiment and Evaluation

5.1 Introduction

So far we have seen the proposed design of dependency based Tigrinya grammar checker. In this Chapter we will see the experiments and evaluation results of the proposed design. Here, some necessary models will be trained for the modules of the DBTGC to perform their respective tasks accordingly. The models to be trained will be used for the preprocessing module (tokenizer, tagger, and morphological feature annotator) and dependency extraction module. Algorithms and functions for the grammar checking module will also be developed. Finally the DBTGC will be tested with a test data and evaluate the results according to the selected evaluation metrics. A comparison of the DBTGC and the existing Tigrinya grammar checker [6] will also be discussed.

5.2 DBTGC System Development and Model Training

Here, we will discuss how prototype of the DBTGC system and its components are developed based on the proposed design. For the development of DBTGC system prototype a number of components should be prepared beforehand. These components are; a Tigrinya text corpus, trained language model, text preprocessing module that is capable of accepting Tigrinya text input, tokenizing, and tagging, a dependency extraction module, and grammar checking module. The components inside text preprocessing and dependency extraction modules need to have a pre trained language model to refer to and such a model should be trained and prepared for use. We will see the component development and the process of training the required language model.

5.2.1 Corpus Data Preparation

This corpus is a collection of grammatically correct Tigrinya sentences in which each token in each sentence is morphologically and syntactically annotated. It is prepared using dependency tree of Tigrinya sentences in CoNLL-U format. In CoNLL-U format, a text is split into sentences and a sentence is split into tokens (words and punctuations). Sentences are separated using an empty line and tokens are separated using new lines. Then each token is annotated with 10 (ten) different information separated using a tab space as shown in Table 5.1.

Table 5.1- Example of CoNLL-U data format of a sentence

ID	FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	DEPS	MISC
\/	\/	\/	\/	\/	\/	\/	\/	\/	\/
1	ንሱ	ንሱ	PRON	PRO	Gender=Masc|Person=3|Number=Sing	2	nsubj	2:nsubj	_
2	መፃኢ	መፃኢ	VERB	V	Gender=Masc|Person=3|Number=Sing	0	root	0:root	_

A text corpus is needed to train a model for Tigrinya language. We have prepared such corpus in CoNLL-U data format that enables to have a structured text corpus with more token information. The corpus was taken from a POS only tagged corpus, NTC. This corpus was edited using a web based CoNLL-U data editor that we have developed for this purpose. The editor is developed using PHP, HTML, and JavaScript. A sample CoNLL-U formatted text is shown in Annex A.

5.2.2 Model Training for Preprocessing Module

We have used an annotation model training tool called UDPipe, to train the model. UDPipe uses a CoNLL-U formatted sentences as a training data to induce a model [52]. Tokenizer and tagger model training requires a training data in CoNLL-U format. Among the morphological fields of CoNLL-U format, the XPOS, UPOS, LEMMA and FEATS are used by UDPipe to build tokenizer and tagger models based on the available values of the selected fields in the training data.

The CoNLL-U formatted treebank corpus contains 312 sentences and 1248 tokens. Among this corpus, 80% or 250 sentence are used to train the models and the remaining 20% or 62 sentences are used to test and evaluate the induced models. In addition to the corpus, the models are evaluated by raw text collected from Eritrean magazine, “Hadas Eritrea” (online).

We have applied different options for the models we have built. For example, the tokenizer recognizes options such as epochs, batch size, learning rate and dropout. The tagger also

recognizes hyper parameters that need to be optimized; such as lemma, XPOS, morphological features, guesser rules, and guesser dictionary [52].

During hyper parameter search, the options of tokenizer have the following values:

- ✓ Epochs: its default value is 100
- ✓ Batch size: is uniformly chosen between 50 and 100. Its default value is 50.
- ✓ Learning rate: is a range between 0.0005 and 0.01. Its default value is 0.005.
- ✓ Dropout: its default value is 0.1.

Similarly, the options of the tagger have different values. The guesser rule and guesser dictionary have default values of 8 and 6 respectively [52].

We have trained and experimented on 6 (six) tokenizer models differing on the options we have used to induce the models. We have based the tokenizer model experiments on the properties shown in Table 5.2.

Table 5.2- UDPipe Hyperparameter option values for tokenizer training

Hyper parameters	**Tokenizer model 1**	**Tokenizer model 2**	**Tokenizer model 3**	**Tokenizer model 4**	**Tokenizer model 5**	**Tokenizer model 6**
epochs	50	60	70	80	90	100
batch_size	50	60	70	80	90	100
learning_rate	0.0005	0.0025	0.0045	0.0065	0.0085	0.01
dropout	0.1	0.2	0.3	0.4	0.5	0.6

We have trained and experimented on 5 (five) tagger models differing on the options we have used to induce the models. The experiment on the tagger models is based on the options shown in the Table 5.3.

Table 5.3- UDPipe Hyperparameter option values for tagger training

Hyperparameters	**Tagger model 1**	**Tagger model 2**	**Tagger model 3**	**Tagger model 4**	**Tagger model 5**
guesser_rules	8	9	10	11	12
guesser_dictionary	6	7	8	9	10

We have used UDpipe to evaluate the experiment results of the trained models. UDpipe uses precision, recall, and F1 to evaluate a tokenizer model [52]. The performance of the models is evaluated against the data prepared for testing as shown in Table 5.4.

Table 5.4- Evaluation results of tokenizer models

Tokenizer model	**Precision %**	**Recall %**	**F1 %**
Model 1	93.3	35.48	53.55
Model 2	93.3	35.48	53.55
Model 3	93.3	35.48	53.55
Model 4	93.3	35.48	53.55
Model 5	93.3	35.48	53.55
Model 6	93.3	35.48	53.55

The trained tagger models are also evaluated using UDpipe and similar test data set. UDpipe uses accuracy to evaluate induced tagger models [52]. The test results of the induced tagger models are shown in Table 5.5.

Table 5.5- Evaluation results of tagger models

Tagger model	**XPOS accuracy %**	**UPOS accuracy %**	**FEATS accuracy %**	**LEMMA accuracy %**
Model 1	80.19	79.71	84.31	97
Model 2	79.51	79.14	82.38	97
Model 3	79.77	79.5	83.16	97
Model 4	80.03	79.66	84.15	97
Model 5	78.94	78.67	82.64	97

Evaluation results of the trained tokenizer models show that all models experience the same results regardless of different hyperparameter option values. The precision, recall, and F1 measures of the models are 93.3%, 35.48%, and 53.55% respectively. This implies that we can use any of the induced tokenizer models.

Evaluation results of the trained tagger models show different results when we apply different values for the hyperparameter options. The tagger model evaluation matrix shows accuracy for XPOS, UPOS, lemma, and morphological features. Since the lemma guesser accuracy is the same for all the tagger models we did not consider the lemma accuracy to compare the tagger models. We take into consideration the accuracy for XPOS, UPOS, and morphological feature guessers to choose the best tagger model. Therefore, the tagger model with the highest accuracy is **tagger model 1** with accuracy of UPOS tag 80.19%, XPOS tag 79.71% and feature annotation 84.31%.

5.2.3 Model Training for Dependency Extraction Module

We have used MaltParser, a data-driven parser generator system to induce a dependency parser model. Since MaltParser implements a number of algorithms we need to experiment every algorithm to get which one best fits to our context which is Tigrinya sentence dependency grammar checking. In order to have an optimized parsing model we need to choose the best algorithm that MaltParser provide as discussed in Section 2.11.5.

We have used two forms of dataset to experiment on dependency parsing model. One for the training process to look for an efficient dependency parsing algorithm and the other to test the induced dependency parsing model. The testing data set is considered as a gold standard corpus against which the parsing results of each algorithms are going to be checked. Both datasets are similar in that both have equal number of tokens and similar sentences. The only difference is that the first corpus is unparsed whereas the other one is manually parsed. They are taken from the Nagaoka Tigrinya dependency Treebank [51].

We have obtained seven dependency parsing models one for each of the seven parsing algorithms implemented by MaltParser. However, the learning algorithm applied is only LIBLINEAR due to the fact that the other learning algorithm, LIBSVM, is not available on the current version of MaltParser 1.9.2.

The parsing results are tested against the corpus assigned as a gold reference. We have used MaltEval 1.0 [53] to evaluate the parser. The results of the evaluations is described in the table below.

Table 5.6- Evaluation result of parsing algorithms

#	**Algorithm**	**UAS**	**LAS**
1	nivrestandard	95.67	91.48
2	nivreeager	95.07	91.08
3	covproj	94.87	91.28
4	covnonproj	95.07	91.98
5	stackproj	95.47	91.38
6	stackeager	95.77	91.78
7	stacklazy	95.77	91.48

As we can see the evaluation results in the previous table the parsing model induced by Covington non-projective algorithm along with LIBLINEAR learning algorithm has a maximum LAS of 91.98%. Therefore, this parsing model is selected as a parsing model and trained with the remaining 80% of the dataset reserved for final training.

5.2.4 Text Preprocessing and Dependency Extraction Module Development

These two modules are developed using UDPipe model implementation in the Python programming language and have a model with functions to read, tokenize, tag, and parse an input text. However, this implementation requires a trained model for the intended language in order for the functions inside to use it as a reference model to perform their intended tasks. The code snippets for these functions are depicted in Annex B and Annex C.

5.2.5 Grammar Checking Module Development

In the design phase, we have seen that an agreement of a pair of words in a sentence, that have dependency relationship, are going to be checked with respect to their morphological features. For this purpose we have developed a prototype for the grammar checking module using Python programing language. The prototype has two functions; namely Relation Extractor and Agreement Checker. They perform relation extraction and agreement checking tasks respectively. The main purpose of the prototype is to demonstrate and evaluate the grammar checking module. The code snippets for this module are shown in Annex D and Annex E.

For agreement checking, we have used 16 randomly selected grammatically correct sentences of the Nagaoka Tigrinya treebank. These sentences are intentionally taken as test cases to check whether the grammar checking module correctly predicts their grammatical agreement or not.

We have tested the performance of the grammar checking module, by providing the test sentences to the grammar checking module one by one. The number of agreements returned are then counted and compared against the actual available agreements that should be identified. The grammar checking module governs major types of agreements such as subj-verb, obj-verb, adj-type and adv-verb agreements. This would help us to figure out the accuracy of the grammar checking module. The accuracy is, therefore, computed by dividing the identified count by the actual count. The result obtained from the evaluation of the grammar checking module is depicted in the table below.

Table 5.7- Evaluation result of grammar checking module

Grammatical agreement	**Identified count**	**Actual count**	**Accuracy (%)**
Subject – verb	9	16	56.25
Object – verb	9	12	75
Adverb – verb	4	8	50
Adjective – noun	7	12	58.33
Total	29	48	60.41

The evaluation result shows us, the grammar checking module has identified 56.25% of subject-verb agreement, 75% of object-verb agreement, 50% of adverb-verb agreement, and 58.33% of adjective-noun agreement.

Finally, a single language model for the components inside text preprocessing and dependency extraction module was trained in a way that applies the training settings and algorithms with the best results to produce a single all-inclusive language model. This model includes all the required information for the tokenizer, tagger (POS and morphological features), and dependency extraction components. The separately developed components of the DBTGC system were also integrated into a single fully functional grammar checker application that uses a statistically trained language model as a reference for the system components that require it and the system made ready for evaluation.

5.3 Testing and Evaluation on DBTGC

Now that the DBTGC system implementation is complete, we can test and evaluate the system using an input test dataset. The input text used to test the DBTGC system is prepared in a way that encompasses grammatically correct and incorrect sentences and sentences with tokens that are included and not included in the corpus. We have used 74 grammatically correct sentences, 48 grammatically incorrect sentences, 98 sentences whose tokens are included in the corpus, and 24 sentences with tokens that are not included in the corpus. A total of 122 sentences are used to test the DBTGC system.

Table 5.8 - Count of agreements and disagreements

Grammar (agreement) checking evaluation			
Actual count of agreements	**Identified count of agreements**	**Actual count of disagreements**	**Identified count of disagreements**
221	203	140	127

Table 5.9 - Evaluation results of DBTGC system

DBTGC evaluation results (%)		
Precision (%)	**Accuracy (%)**	**Recall (%)**
92.46	92.09	61.21

We have made a comparison between the DBTGC and Abdella's grammar checker based on some comparison criteria that we found very important to tell which one is better. The comparison criteria are the amount of detail in the text corpus used, the training process, capability of preprocessing methods, the grammatical notation, parsing result details, and contribution to possible following applications or processes.

Abdella's grammar checker scores the highest performance with recall values 91.95% for automatically and 96.48% for manually analyzed test dataset and precision value 91.86% for manually analyzed test dataset. That is an average recall of 94.22% and precision value of 91.86%. Abdella's grammar checker uses a corpus that has only a few token details. These details are the token itself and its language specific POS tag. The corpus lacks the meaningful and most important

details lemma, morphological features, and grammatical relation. Although most POS types of words need morphological annotation, the morphological annotator used was capable only for Tigrinya verbs. The grammatical notation used phrase structure grammar, generates complex grammar representation however it has limited capability in that it allows only a limited level of grammatical structure analysis. The parsing results do not provide information that describe the sentence structure in detail.

The DBTGC system has scored highest performance of precision value 92.46%, recall value 61.21% and accuracy value of 92.09%. The reason for lower score of recall is that we have filled almost half of the test dataset with grammatically incorrect sentences. In the DBTGC, we were able to prepare a Tigrinya corpus with more token details than the NTC. The corpus contains tokens that are tagged with XPOS, UPOS, lemma, morphological features, head, and the dependency relation. The corpus allowed us to train a model that can learn more details and structures with a corpus of less size. The language model that we have trained and developed for use in the text preprocessing and dependency extraction purposes enable them to predict and identify more token details than Abdella's. With the use of dependency grammar notation, we were able to analyze sentence grammar structure better in that we can get the lemma, POS tags, morphological features, and grammatical relations of tokens inside a sentence. It was also able to provide results with more detailed information about the parsed sentence.

5.4 Discussion

The preprocessing module has performed well except that it tags only tokens that exist in the dictionary of the trained language model. It was able to tokenize an input text appropriately and annotates the tokens with details lemma, XPOS, UPOS tag, and morphological features. Although it tags some tokens incorrectly; such a gap was created due to the existence of tokens that are tagged incorrectly in the corpus that is used to train the language model. This problem can be solved by preparing a carefully tagged corpus by the specific linguistic experts.

The dependency extraction module was able to predict the root (main verb) of the sentence, head of each token and the dependency relations. However, it cannot predict these information for tokens with no information in the language dependency model. Moreover, the dependency grammar notation enabled us to use less training data, learn more details and structures, and then

parse many sentences. It was also able to parse most of the compound and complex sentences in the input text.

The grammar checking module had also performed well. It identifies more subject-verb agreements than the others. The grammar checking module sometimes returns nothing as if there are no agreements between head and dependent words. This is because if one of the two words does not have morphological features in the input CoNLL-U text, the grammar checking module considers them as if they disagree. The grammar checking module also has a limitation if tokens of the input sentence are not tagged properly and completely. It works fine for sentences with completely and carefully tagged tokens.

The performance of the DBTGC system is mainly affected by the size, content variety, and quality of the text corpus. Since there are limitations to altering the training algorithms, the third party training tools also have considerable effect. The relation extraction algorithm was developed manually and a rule-based agreement checking method was used. The performance can be improved using large and better quality text corpus prepared by specific linguistic experts. An automated relation extraction and statistically trained automatic agreement checking methods can also help improve the performance of the DBTGC.

Chapter 6: Conclusion and Future Works

6.1 Conclusion

The main objective of this work was to design and develop a dependency-based Tigrinya grammar checker. It has incorporated syntactic parsing of dependency grammar and pre-processing module to convert input sentences to CoNLL-U format.

The grammar checker's modular components are preprocessing module, dependency extraction module and grammar checking module. The language model used by the preprocessing and dependency extraction modules is induced by third party NLP tools which require input sentences in CoNLL-U format to induce the corresponding model. We have used Nagaoka Tigrinya dependency treebank that we have prepared to train the language model. However, the size and quality of the treebank was limited. This affects the performance of the preprocessor and the dependency extraction modules. Therefore, the performance of the grammar checker depends on the size, content variety, and quality of the tree bank and performance of third party tools used to generate the language models. Generally, the system evaluation shows that, it was able to yield a highest precision of 92.46% and highest accuracy of 92.09% performance.

6.2 Contribution

The dependency based Tigrinya grammar checker provides clues that lay foundation for other dependency based research work. The main contributions of this research work are discussed as follows:

We have prepared a dependency treebank for Tigrinya language through a process that involves cleaning and tagging token attributes of text taken from Nagaoka POS tagged Tigrinya corpus. Text preprocessing components tokenizer and tagger that make use of a pre trained dependency model of Tigrinya language are also implemented. We have trained a dependency model for Tigrinya language using the dependency treebank prepared earlier and a third party annotation model training tools. We have also implemented a grammar checking component that carries out grammatical relation extraction and grammatical agreement checking processes.

6.3 Future Works

This research can be improved and upgraded by enhancing the performance of each component of the system. This can be achieved through utilizing large and quality treebank to train an induced model, rigorously experimenting the language models applied in this research work to obtain optimum model, adding other grammatical agreements of the language, and incorporating statistical approaches of grammar checker. Specifically, the future work arising from this study are:

Large and High Quality Treebank: The availability of larger and high quality treebanks will improve linguistic models. As treebanks grow in size and quality, they will become more useful in literary and historical studies, where the linguistic structure of texts will vary and become investigable in a more precise way.

Language Independent Tools: universal dependency framework employs different tools for consistency annotation, format conversions and dependency parsing. In this study, we have used MaltParser and UDpipe to build dependency parser as well as tagger models. However, the latter modules performance was poor and need to be investigated by other tools to come up with enhanced model.

Statistical Approach: Currently, the grammar checker is implemented with rule-based approach in which morphological features of head-dependent words are compared. Thus, statistical approach improves the performance of the grammar checker.

References

[1] L. Bauer, The Linguistics Student's Handbook, Edinburgh: Edinburgh University Press, 2007.

[2] V. H. a. T. Reuter, "LISGrammarChecker: Language Independent Statistical Grammar Checking," Hochschule Darmstadt & Reykjavík University, Departments of Computer Science, 2009.

[3] J. K, H. G and R. S, Natural Language Processing: The PLNLP Approach, Springer US, 2013.

[4] K. F and Shaalan, "Arabic GramCheck: a grammar checker for Arabic," *SOFTWARE PRACTICE AND EXPERIENCE,* 2005.

[5] N. S, B. R.P and B. B. Pawar, "GRAMMAR CHECKERS FOR NATURAL LANGUAGES: A Review," in *3rd International Conference on Signal, Image Processing and Embedded Systems*, 2017.

[6] A. Nurahmed, "Hybrid Grammar Checker for Tigrinya," Debre Berhan University, 2017.

[7] R. Debusmann, "Introduction to Dependency Grammar," 2000.

[8] G.-J. M. Kruijff, Formal and Computational aspects of Dependency Grammar - Historical development of DG, Saarbrucken Germany: Computational linguistics University of the Saarland, 2002.

[9] "Ethnologue: Languages of the World," *SIL International,* vol. 17th, 2013.

[10] Ayele Bekerie, Ethiopic an African Writing System: Its History and Principles, Asmara, Eritrea: Red Sea Press, 1997.

[11] M. Mozgovoy, "Dependency-Based Rules for Grammar Checking with LanguageTool," in *Federated Conference on Computer Science and Information Systems - FedCSIS* , Szczecin, Poland, 2011.

[12] D. Tesfaye, "A rule-based Afan Oromo Grammar Checker," *International Journal of Advanced Computer Science and Applications,* vol. 2, p. 8, 2011.

[13] Y. A and A. T, "Development of Amharic Grammar Checker Using Morphological Features of Words and N-Gram Based Probabilistic Methods," *In Proceedings of the The 13th International Conference on Parsing,* pp. 106-112, 2013.

[14] M. J. Alam, N. UzZaman and M. Khan, "N-gram based Statistical Grammar Checker for Bangla and English," *Centre for Research on Bangla Language Processing (CRBLP),* 2006.

[15] D. Naber, "A Rule-Based Style And Grammar Checker," *Diplomarbeit,* no. Technische Fakultät Bielefeld, 2003.

[16] M. Singh, G. Singh and S. Sharma, "A Punjabi Grammar Checker," *2nd international conference of computational linguistics: Demonstration paper. Punjabi University,* pp. 149-132, 2008.

[17] R. K. O. C. C. Domeij, "Granska - an efficient hybrid system for Swedish grammar checking," *Proceedings of NODALIDA,* pp. 49-56, 1999.

[18] L. S. C. E. D. d. M. J. Kinoshita, "CoGrOO: a Brazilian-Portuguese Grammar Checker based on the CETENFOLHA Corpus," *LREC,* 2006.

[19] H. J. Jurafsky Daniel, "An introduction to natural language processing, computational linguistics and speech recognition," *Speech and Language Processing,* 2007.

[20] T. Semere, "Probabilistic Tigrigna-Amharic Cross Language Information Retrieval (Clir)," *School Of Information Science, Addis Ababa University, Addis Ababa, Ethiopia,* 2013.

[21] M. Yifiru, S. Abate and L. Besacier, "Part-of-Speech Tagging for Under-Resourced and Morphologically Rich Languages – The Case of Amharic," *Conference on Human Language Technology for Development, Alexandria, Egypt,* 2011.

[22] B. Gamback, F. Olsson and A. L. A. Alemu, "Methods for Amharic Part-of-Speech Tagging," *Proceedings of the EACL 2009 Workshop on Language Technologies for African Languages – AfLaT 2009, Athens. Greece, 31 March 2009,* pp. 104-111, 2009.

[23] K. F. Mark Aronoff, What is Morphology?, vol. 18, Wiley-Blackwell, 2010.

[24] M. Getachew, "Automatic Part of Speech Tagging for Amharic Language: An Experiment Using Stochastic Hidden Markov (HMM) Approach," Addis Ababa University, Addis Ababa , 2001.

[25] M. Gasser, "HornMorpho: a system for morphological processing of Amharic, Oromo, and Tigrinya," *Conference on Human Language Technology for Development,* 2011.

[26] D. J. &. J. H. Martin, Speech and Language Processing, Prentice-Hall, 2008.

[27] A. Fokkens, *Dependency Grammars - Syntactic Theory,* Saarland, Germany: Department of Computational Linguistics , 2009.

[28] T. Lucien, Elements of Structural Syntax: Translated by. Timothy Osborne and Sylvain Kahane, John Benjamins, 2015.

[29] C. F. a. S. L. Alexander Clark, The Handbook of Computational Linguistics and Natural Language Processing, Wiley-Blackwell, 2010.

[30] E. M. Sabine Buchholz, "CoNLL-X shared task on Multilingual Dependency Parsing," in *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, New York, 2006.

[31] R. P. F. R. K. a. H. J. McDonald, "Non-projective dependency parsing using spanning tree algorithms," *Proceedings of Human Language Technology Conference on Empirical Methods in Natural Language Processing, Vancouver, British Columbia, Canada,* 2005.

[32] J. H. J. N. Joakim Nivre, "MaltParser: A Data-Driven Parser-Generator for Dependency Parsing,," *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC), Genoa-Italy,* 2006.

[33] J. N. Johan Hall, "MaltParser User Guide," August 2018. [Online]. Available: http://www.maltparser.org/userguide.html#inout.

[34] M. B. a. J. Nivre, "MaltOptimizer: A System for MaltParser Optimization," in *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC)*, Istanbul. Turkey, 2012.

[35] J. &. H. J. Nivre, "A Quick Guide to MaltParser Optimization," 2010.

[36] C.-C. C. a. C.-J. Lin, LIBSVM: A Library for Support Vector Machines, Taipei, Taiwan: Department of Computer Science National Taiwan University, 2013.

[37] K.-W. C. C.-J. H. X.-R. W. C.-J. L. Rong-En Fan, "LIBLINEAR: A Library for Large Linear Classification," *Journal of Machine Learning Research,* vol. 9, pp. 1871-1874, 2008.

[38] J. N. E. b. A. L. a. M. Kytö, Corpus linguistics: An international handbook, vol. 1, Walter de Gruyter, 2008, pp. 225-241.

[39] T. L. a. (. J. Foster, Irish Dependency Treebanking and Parsing, Dublin, Germany: School of Computing Dublin City University, Department of Computing Macquarie University, 2016.

[40] J. Nivre, Inductive Dependency Parsing, Springer Netherlands, 2006.

[41] M. Gasser, A Dependency Grammar for Amharic, Bloomington, Indiana USA: School of Informatics and Computing Indiana University, 2010.

[42] Y. M. B. Y. M. Binyam Ephrem Seyoum, "Morpho-syntactically Annotated Amharic Treebank, in Corpus Linguistics," in *Corpus Linguistics Fest 2016*, Bloomington, 2016.

[43] A. M. Zwicky, "Heads," *Journal of Linguistics,* vol. 21, pp. 1-29, 1985.

[44] R. A. Hudson, English Word Grammar, Blackwell, 1990.

[45] "universaldependencies.org," [Online]. Available: http://universaldependencies.org/format.html. [Accessed 24 07 2020].

[46] "Universal Dependency v2.0," 2019. [Online]. Available: http://universaldependencies.org/introduction.html. [Accessed 24 07 2020].

[47] Y. M. B. Y. M. Binyam Ephrem Seyoum, "Universal Dependencies for Amharic," in *LREC proceedings*, Tokyo, Japan, 2018.

[48] "CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies," [Online]. Available: http://universaldependencies.org/conll17/. [Accessed 10 june 2020].

[49] "CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies," [Online]. Available: http://universaldependencies.org/conll18/. [Accessed june 2020].

[50] Y. A. and A. G. , "Dependency-based Amharic Grammar Checker," Addis Ababa University, Addis Ababa , 2018.

[51] K. K. Y. A. M. Yemane, "Nagaoka Tigrinya Corpus: Design and Development of Part-of-speech Tagged Corpus," *International Journal of Computer Applications,* 2016.

[52] Institute of Formal and Applied Linguistics, , "UDPipe 2," Charles University, Czech Republic, Faculty of Mathematics and Physics, [Online]. Available: https://ufal.mff.cuni.cz/udpipe/2. [Accessed 23 03 2020].

[53] J. Nilsson, "User Guide for MaltEval 1.0 (beta)," 2012. [Online]. Available: https://cl.lingfil.uu.se/~nivre/docs/MaltEvalUserGuide.pdf. [Accessed 12 07 2020].

[54] M. Gasser, "Expanding the Lexicon for a Resource-Poor Language Using a Morphological Analyzer and a Web Crawler," *Seventh International Conference on Language Resources and Evaluation,* 2010.

[55] M. B. a. J. Nivre, "MaltOptimizer: A System for MaltParser Optimization," in *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC)*, Istanbul, Turkey, 2012.

[56] R. Batstone, Grammar: A Scheme for Teacher Education, Oxford University Press, 1994.

[57] M. M. W. a. M. Meshesha, "Experimenting Statistical Machine Translation for Ethiopic Semitic Languages: The Case of Amharic-Tigrigna," in *Information and Communication Technology for Development for Africa*, Springer, Cham, 2018, pp. 140-149.

[58] Mohamed Zakaria Kurdi, Natural Language Processing and Computational Linguistics: speech, morphology, and syntax, vol. Volume 1, Wiley-ISTE, 2016.

[59] M. K. Ralph Debusmann, "Dependency Grammar: Classification and Exploration," in *Resource-Adaptive Cognitive Processes*, Berlin, Heidelberg, Springer, 2007, pp. 365-388.

Annexes

Annex A: Sample Training Data in CoNLL-U Format

```
1 ድምጻ ድምጻ NOUN  N Gender=Fem|Number=Sing|Person=3 5 5:nsubj _ _
2 ናብ  ናብ  ADP PRE _ 3 3:appos _ _
3 ጭዳረ ጭዳረ NOUN  N _ 5 5:csubj _ _
4 ገጹ  ገጹ  NOUN  N Gender=Neut|Number=Sing|Person=3  3 3:amod  _ _
5 ተቐየረ  ተቐየረ  VERB  V_PRF Gender=Neut|Number=Sing|Person=3|Tense=Past|Voice=Pass  0 0:root  _ _
6 ። ። PUNCT PUN _ 5 5:punct _ _

1 ኑብሳ ኑብሳ NOUN  N Gender=Fem|Number=Sing|Person=3 3 3:nsubj _ _
2 እውን እውን CCONJ CON _ 1 1:conj  _ _
3 ከንቀጥቅጥ  ከንቀጥቅጥ  VERB  V_IMF Gender=Com|Number=Sing|Person=3|Voice=Pass  4 4:advmod* _ _
4 ተራእየ  ተራእየ  VERB  V_AUX Gender=Neut|Number=Sing|Person=3  0 0:root  _ _
5 ። ። PUNCT PUN _ 4 4:punct _ _

1 ሲናይ ሲናይ PROPN N_PRP _ 3 3:nsubj _ _
2 ርእሱ ርእሱ NOUN  N Gender=Masc|Number=Sing|Person=3  3 3:csubj _ _
3 ቀበረ ቀበረ VERB  V_PRF Gender=Masc|Number=Sing|Person=3|Voice=Act  0 0:root  _ _
4 ። ። PUNCT PUN _ 3 3:punct _ _
```

Annex B: Sample Tokenizer Code Snippet from UDPipe Python Library

```python
import ufal.udpipe
# ufal.udpipe.Model etc. are SWIG-magic and cannot be detected by pylint
# pylint: disable=no-member

class Model:
    def __init__(self, path):
        """Load given model."""
        self.model = ufal.udpipe.Model.load(path)
        if not self.model:
            raise Exception("Cannot load UDPipe model from file '%s'" % path)

    def tokenize(self, text):
        """Tokenize the text and return list of ufal.udpipe.Sentence-s."""
        tokenizer = self.model.newTokenizer(self.model.DEFAULT)
        if not tokenizer:
            raise Exception("The model does not have a tokenizer")
        return self._read(text, tokenizer)
```

Annex C: Sample Tagger and Parser Code Snippet from UDPipe Python Library

```
def tag(self, sentence):
    """Tag the given ufal.udpipe.Sentence (inplace)."""
    self.model.tag(sentence, self.model.DEFAULT)

def parse(self, sentence):
    """Parse the given ufal.udpipe.Sentence (inplace)."""
    self.model.parse(sentence, self.model.DEFAULT)

def write(self, sentences, out_format):
    """Write given ufal.udpipe.Sentence-s in the required format (conllu|ho

    output_format = ufal.udpipe.OutputFormat.newOutputFormat(out_format)
    output = ''
    for sentence in sentences:
        output += output_format.writeSentence(sentence)
    output += output_format.finishDocument()

    return output
```

Annex D: Sample Relation Extraction Code Snippet in Python

```
    relations = []
    print("# relations (head-dependent pairs)")
    displayTextFile = open('_displayText.txt', mode='a', encoding='utf-8')
    displayTextFile.write("# relations (head-dependent pairs) \n")
    for i in range(x):
        head = heads_list[i]
        if head not in ["_", 0]:
            headIndex = int(head)
            h = forms_list[headIndex-1]
            d = forms_list[i]
            rel = relations_list[i]
            if rel not in ["root", "punct"]:
                # collect pairs of relations into a dict
                relDict = {
                    "rel": rel,
                    "dep": str(d),
                    "head": str(h),
                }
                relations.append(relDict)
                # display relations in "rel ('dependent', 'head')" form
                relText = relDict["rel"]+" ('"+relDict["dep"]+"', '"+relDict["head"]+"') \n"
                # print (relText)
                displayTextFile.write(str(relText))
    displayTextFile.close()
    return relations
```

Annex E: Sample Common Key Extraction Code Snippet in Python

```
common_keys = []
print("# common keys \n")
displayTextFile = open('_displayText.txt', mode='a', encoding='utf-8')
displayTextFile.write("# common keys \n")
# search for common keys
y = len(dep_morph_feats)
i = 1
for i in range(y):
    key = dep_morph_feats[i-1]
    dep_feats = dep_morph_feats[i-1]
    head_feats = head_morph_feats[i-1]
    if key != "_":
        if (key in head_morph_feats) and (dep_feats == head_feats):
            # collect common keys in dict
            common_keys.append(key)
            # # display common keys
            print(key+"\n")
            displayTextFile.write(key+"\n")
displayTextFile.close()
return common_keys
```

Annex F: Sample Agreement Checker Code Snippet in Python

```
gcMessages = []
print("# grammar checking result message \n")
displayTextFile = open('_displayText.txt', mode='a', encoding='utf-8')
displayTextFile.write("# grammar checking result message \n")
# checking for agreement
numOfCommKeys = len(common_keys)
for i in range(numOfCommKeys):
    depForm = forms_list[i]
    headForm = forms_list[int(heads_list[i])-1]
    depComKeys = common_keys[i]
    print(depComKeys)
    if len(depComKeys) >= 1:
        print(depComKeys)
        for comKey in depComKeys:
            dep_key_value = dep_feats_dict[i][comKey]
            head_key_value = head_feats_dict[i][comKey]
            if dep_key_value == head_key_value:
                gcMsg = "the "+depForm+" and the "+headForm+" agree on: "+comKey+"\n"
                gcMessages.append(gcMsg)
                print(gcMsg)
                displayTextFile.write(gcMsg)
            else:
                gcMsg = "the "+depForm+" and the "+headForm+" disagree on: "+comKey+"\n"
                gcMessages.append(gcMsg)
                print(gcMsg)
                displayTextFile.write(gcMsg)
displayTextFile.close()
return gcMessages
```

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 duly acknowledged.

Declared by:

Name: ____Mehammedbrhan Abdelkadr_____

Signature: _____

Date: _____

Confirmed by advisor:

Name: ____Yaregal Assabie (PhD) _____

Signature: _____

Date: _____