ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

FACULTY OF HUMANITIES

DEPARTMENT OF COMPUTATIONAL LINGUISTICS

DEVELOPMENT OF MORPHOLOGICAL ANALYZER FOR AMHARIC COMPOUND WORDS

A thesis submitted to the School of Graduate Studies of Addis Ababa University in partial fulfillment of the requirements for the Degree of Master of Science in Computational Linguistics

BY

YOHANNES ABEBE

January 2013

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

FACULTY OF HUMANITIES

DEVELOPMENT OF MORPHOLOGICAL ANALYZER FOR AMHARIC COMPOUND WORDS

By

YOHANNES ABEBE

Signature of the Board of Examiners for Approval

Name	Title	Signature	Date
_____	Chair Person	_____	_____
Baye Yimam	Advisor		5-4-13
Tsegaye Legesse	Examiner	Dulelu B.	05/04/13
Wondwossen Mulugeta	Advisor		05/04/2013
Martha Yifiru	Examiner		18/04/2013

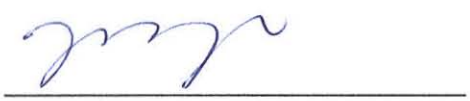
Declaration

This thesis is my original work and has not been submitted as a partial requirement for a degree in any university

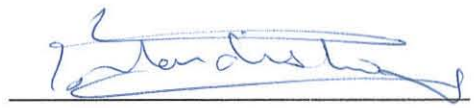
Yohannes Abebe Mekonnen

January 2013

The thesis has been submitted for examination with our approval as university advisors.



Professor Baye Yimam



Ato Wondwossen Mulugeta

Acknowledgment

For the successful completion of this thesis, first I would like to thank GOD. Then, I would like to thank and express my deepest appreciation to my advisors Prof. Baye Yimam and Ato Wondwossen Mulugeta for their meticulous supervision, support, and constructive advice. I thank you also for your concern and readiness to help me whenever I needed.

I would also like to thank Daniel Yacob, Ge'ez Frontier Foundation, for making Amharic dictionaries available digitally. His work saves lots of my time.

My special thanks go to my wife Haregewoin Gedamu and my daughter Edom Yohannes for their love, patience, and being strength to me.

I particularly thank my brother Dr. Mesafint Abebe. You helped me in all directions. I am proud being your brother and you are my hero. You do make a difference. If the world had more people like you, it would be a better place.

I am grateful to my brothers Dani and Teme for their brotherhood and financial support.

I am grateful to the rest of my family: Alemye, Emaye, Gasheye, Haileye, Eyobe, Ze, Azi, Nebi, Rahel, Menbi and Serke for their support and understanding.

My final thanks go to my classmates and those who stood by my side.

Yohannes Abebe

January 2013

Abstract

The purpose of this study is to develop morphological analyzer for Amharic compound words. A number of researchers attempted to develop morphological analyzer for Amharic since 2000 (Abiyot, 2000; Tesfaye, 2002; Saba and Gibbon, 2005; Gasser, 2011) and their analyzers provide a very good performance. However, as far as the researcher has noted, nothing is reported on their findings and results about analysis of compound words. For this reason, the researcher decided to develop morphological analyzer for Amharic compound words using rule-based approach on the basis of two-level morphology.

A morphological analyzer is a computer program that takes a word or string of charcters as input and delivers an analysis as output. Amharic, in addition to simple words, uses compound words such as አየር መንገዶች ayyär-mängädočč 'airlines', ምልክ ምልካም mälk-ä-mälkam 'beautiful', ልጃገረድ ləj-a-gäräd 'virgin', etc. The developed anlyzer can recognize and deliver the given compound word with its word class, each constituents of the compound word with their POS and grammatical functions of the attached suffixes.

The study covers all compound categories in Amharic (i.e. compound nouns, adjectives, verbs, and adverbs) with their grammatical and syntactical information. The grammatical features included in this work are number, gender, person, case, and definiteness.

In identifying and analyzing compound nouns, adjectives, and adverbs, the system performs well and the sample used to the development and test set can be considered as representative of Amharic compounds. However regarding compound verbs, it covered only the main verbs, verb to 'say' and to 'do', and some of their variations, not all.

In this study, algorithms that can identify and analyze Amharic compound words are developed from scratch. The performance of the system is evaluated using the training and test sets. The system accuracy on the test set is 98.67% and its precision and recall are 100% and 98.5%, respectively.

Contents Pages

List of Tables

List of Figures

Abbreviations and Symbols Used

''	What is inside is the equivalent meaning in English
E.C	Ethiopian calendar
CL	Computational Linguistics
Sg	Singular
Pl	Plural
Fem	Feminine
Masc	Masculine
DEF	Definite
ACC	Accusative
1Sg	First person singular
2Sg	Second person singular
3Sg	Third person singular
1Pl	First person plural
2Pl	Second person plural
3Pl	Third person plural
2FS	Second person feminine singular
3FS-ACC	Third person feminine singular accusative
3MS	Third person masculine singular
3MSDEF	Third person masculine singular definite
3PlDEF	Third person plural definite
3FS	Third person feminine singular
N	Noun

NLP	Natural Language Processing
V	Verb
Adj	Adjective
POS	Parts of speech
SERA	System for Ethiopic Representation in ASCII

CHAPTER ONE

Introduction

1.1. Background

Linguistics is the scientific study of human language. It has an important impact on different fields such as mathematics, computer science, sociology, language teaching, cognitive psychology, and others. One of the most fundamental units of linguistic structure is a word. A word is associated with different kinds of information: phonetic/phonological, lexical structure, syntactic, semantic, and pragmatic information (Akmajian et al, 2001). **Phonetic/Phonological information** is related with the pronunciation and pattern of sounds of the word. For example, knowing the word ሰው säw 'human being' is knowing the sounds: 's', 'ä', 'w' and …

In addition, the word ሰው säw 'human being' cannot be broken down into any meaningful parts unlike the word ሰዎች säw-očč 'human beings', which can be broken into two: the word ሰው säw and an additional element, ኦች očč, and this information is **lexical structure information**. Understanding how the word fits into the overall structure of sentences in which it can be used is **Syntactic information**. The meaning of the word gives **semantic information** (knowing the word implies that understand that it has a certain meaning). **Pragmatic information** is related with knowing how to use the word in the context of discourse or conversation.

The subfield of linguistics that studies the internal structure of words and relationships between words is **Morphology**. Words have internal structure, which is rule-governed (Fromkin et al, 2009). One of the rules of Amharic, for instance, is to be categorized as nouns; words have to accept the suffix ኦች očč (Baye, 1987 E.C).

There are two categories of words, **simple** and **complex**. Simple words such as ሰው säw 'human being' cannot be decomposed into other smaller meaningful units whereas complex words such as ሰዎች säw-očč 'human beings' can be decomposed into smaller meaningful units, ሰው säw and

ኦች očč. Complex words contain several distinct units of meaning. The smallest meaningful unit in the grammar of a language is called **Morpheme**. The Amharic word ቤቶች betočč 'houses', for instance, has two morphemes, ቤት bet 'house' and ኦች očč.

There are three major types of morphological processes: **inflection**, **derivation**, and **compounding**. Inflection deals with the inflected forms of words. It is a change in the form of a word, usually by adding a suffix, in order to indicate a change in its grammatical function. Derivation, on the other hand, is the process that new words are formed from existing words by affixation. In Amharic, for example, from the word ሰው säw 'human being', the word ሰውነት säwənnät 'body or being a human' is formed. Compounding is also the process of making new words from two or more words. ትምህርት ቤት təmhərt-bet 'school' is an Amharic word that is formed by combining the two words, ትምህርት təmhərt 'education' and ቤት bet 'house'.

1.1.1. Computational Morphology

Computational linguistics (CL), which is a subfield of linguistics and computer science, is concerned with the interactions of human language and computers. It includes speech and text analysis, automatic translation of text and speech from one language into another, the use of human languages for communication between computers and people, and the modeling and testing of linguistic theories (Fromkin et al, 2009). One of the major concerns in the field of CL is computational morphology.

Computational morphology is the processing of word and word forms using computers in both their written and spoken form (Trost, 2003). Here in this study, it is focused on processing of words in written forms. Computational morphology has a wide range of practical applications. It contributes not only for low-level applications but also for many speech and language processing systems. Some applications of computational morphology are hyphenation, grapheme-to-phoneme conversion, spelling correction, and stemmer.

The most basic task in computational morphology is **morphological analysis**, which is the process of taking a string of characters as input and delivers an analysis as output. Morphological analysis is the segmentation of words into their component morphemes. For example, if the input

is the Amharic word ቤቶቹ betočču 'the houses', the output could be ቤት+ኦች+ኡ bet+očč+u or bet+NounPlDEF.

Morphological analysis can be performed using automatic morphological analyzer. **Morphological analyzer** is a system that is used to produce all possible analyses for a given word – what lexeme, prefix, and suffix the word includes and for each of these, provide its part of speech and the list of its inflections (Gasser, 2011).

A number of researchers attempted to develop morphological analyzer for Amharic since 2000 (Abiyot, 2000; Tesfaye, 2002; Saba and Gibbon, 2005; Gasser, 2011). However, as far as the researcher has noted, nothing is reported on their findings and results about analysis of compound words. Thus, in this study, it has developed morphological analyzer for Amharic compound words using rule-based approach on the basis of two-level morphology.

1.2. Statement of the Problem

Amharic is the official language of Ethiopia, with about 30 million speakers (Gamback and Asker, 2010). It is the second most widely spoken Semitic language in the world. Amharic is one of the "under-resourced" languages that have very few computational linguistic tools (Gamback and Asker, 2010).

One of the basic computational linguistic resources for Amharic is morphological analyzer (Gamback and Asker, 2010). As mentioned above, several researchers have attempted to develop the morphological analyzer for Amharic. The most recent and complete analyzer is called HornMorpho, which was developed by Gasser[1] (2011). HornMorpho is a Python program that analyzes Amharic, Oromo, and Tigrinya words. The analyzer is freely available.

HornMorpho has limitations that are indicated by Gasser in HornMorpho user's guide[2]. The limitations are related with the general usability, morphological, and lexical issues. In addition to these limitations, HornMorpho lacks efficient handling of compounding morphology. It has been tested in this study by the researcher on 100 compound words (written as conjunct words) and it

[1] http://www.cs.indiana.edu/~gasser/Research/software.html
[2] http://www.cs.indiana.edu/~gasser/Research/projects.html

couldn't analyze any of the words (the analyzer failed 73% to analyze, 2% (መሶበወርቅ and ቀልበቢስ) analyzed the part of speech and the stem but not as compounding stem, and 25% used its guesser analyzer and gave the wrong analysis).

Regarding to the other mentioned analyzers, as far as the researcher has noted, nothing is reported on their findings and results about the morphological analysis of Amharic compound words, although it is mentioned in Saba and Gibbon are reported that some of the analyzer is complete Amharic morphological analyzer (Saba and Gibbon, 2005; Gasser, 2011). Due to this, it was important doing a research on the development of morphological analyzer for Amharic compound words. Hence, this research focuses on the development of morphological analyzer for Amharic compound words.

1.3. Objectives

The main objective of this study is to develop a morphological analyzer for Amharic compound words. To perform this general objective, the study has attempted addressing the following specific objectives:

- Study the morphological property of compound words in Amharic and to identify properties useful for the development of a morphological analyzer;
- Assessing the various techniques (or approaches) suggested for the development of morphological analyzer;
- Develop and test a prototype of morphological analyzer for Amharic compound words;
- Evaluate the performance of the developed system;
- Set future work and recommendation.

Finally, the research answers the following questions:

- How to identify compound words automatically in Amharic text and use them for the analysis? (Automatic identification and extraction of compound words in Amharic text).
- How to analyze Amharic compounds?
- How to handle orthographic variations of compounds in Amharic text?

1.4. Methodology

This thesis followed the rule based approach on the basis of two-level morphology to design and develop the system and it uses the following methods to achieve the above mentioned objectives of the study.

1.4.1. Literature Review

Journal articles, research reports, books, manuals and other documents, which were necessary for this thesis, have been reviewed to investigate:

- What morphological analysis is especially compound word analysis;
- Different methods of developing morphological analyzer;
- Features of Amharic morphology particularly compounding;
- Features of the programming language used in this work, Python3.2;

1.4.2. Lexicon Preparation

The data sources, for this work, were wordlists that are derived from dictionaries, which are available digitally[3]. The dictionaries are አዲስ የአማርኛ መዝገበ ቃላት Addis Yamarinya Mezgebe Kalat of Desta Tekle Wold, ከሳቴ ብርሃን ተሰማ የአማርኛ መዝገበ ቃላት Kasatie Birhan Tessema Yamarinya Mezgebe Kalat of Tessema Habte Mikael, and አማርኛ እንግሊዝኛ መዝገበ ቃላት Amharic English Dictionary of Amsalu Aklilu.

Lexicons are collected mainly from Amsalu's dictionary and built in the system as a knowledge base. This dictionary is chosen because it is available in an advanced stage of development. First, preprocessing tasks were performed using Python programming language and then lexicons are extracted automatically and stored in three text files independently, i.e. nouns, adjectives, and verbal nouns. In addition, list of prefixes and suffixes are prepared manually and saved in two separate files. Finally, the collected lexicons and affixes are transliterated based on SERA[4] system.

[3] available digitally at http://nlp.amharic.org/resources/lexical/word-lists/
[4] System for Ethiopic Representation in ASCII

1.4.3. Algorithms Development and Implementation

Algorithms are designed from scratch for identification and extraction of compound words from Amharic texts and doing the morphological analysis of compounds. Developments of the algorithms are based on:

- Criteria of compound words (which will be discussed in chapter 3);
- The rules of Amharic compounds identified by Baye (2000 E.C);
- The computational perspective of the researcher that is noticed on this thesis.

The system is implemented using Python 3.2 programming language on Linux environment. Python is chosen because Python (Bird et al, 2009):

- is a simple but powerful programming language;
- has excellent functionality for processing linguistic data;
- is portable, it runs everywhere;
- is open source with unrestrictive license.

1.4.4. Testing Techniques

Based on the criteria of compounds (which will be discussed in chapter 3) and discussions with the language experts, sample compound words are collected. This sample corpus is collected from Baye (1987 E.C; 2000 E.C) and Getahun (1990 E.C), from Amharic dictionary of Desta (1970 E.C) and Tessema (1959 E.C) and from wordlist of Amharic single names. It is prepared based on Judgmental Sampling technique, which is a non-statistical sampling method. On this sampling method samples are selected on the basis of the researcher's knowledge about the language, and the nature of the research aims (Atelach, 2002).

The sample corpus is partitioned based on percentage, 70% for development set and 30% for test set. During the development time, the system is tested many times using development set until the performance of the analyzer was acceptable. Then, the developed system is tested using the test set and the performance of the analyzer is evaluated by the language expert. Accuracy of the

system is presented using statistical techniques after summarizing the test result and reported with discussions.

1.5. Contributions

The following are some of the contributions of the developed system:

- Complements the existing systems and helps the development of full-fledged morphological analyzer for Amharic. As discussed previously (in section 1.2), most systems developed for morphology analysis of Amharic are focused on inflectional and derivational morphology and they lack efficient handling of compounding. However, a system to be a full-fledged morphological analyzer beside inflectional and derivational has to consider compounding.
- Other local languages can easily adapt to develop their own morphological analyzer since the system is developed on the basis of two-level morphology.
- It is significant for applications such as decomposition of non-lexicalized compounds for spelling correction, information extraction/retrieval and word-level text alignment between languages. Thus, researchers/developers who are working in such applications are benefited.
- This system helps the teaching-learning process in computational linguistics field (and other related fields) and used as a tool.

1.6. Scope and Limitation

There are two ways of pluralizing adjectives in Amharic – suffixing the plural marker ኦች očč and by reduplicating the penult consonant and adding the vowel 'a' as a connector. The scope of the thesis is limited to the first one, suffixing the plural marker ኦች očč.

Regarding to compound verbs, though Amharic has compound verbs to a limited extent, the thesis covers the most common one.

1.7. Organization of the Thesis

The thesis is organized in six chapters. In chapter one, the thesis introduced what linguistics, computational linguistics and computational morphology are about. Statement of the problem, the objective, the methodology and significance of the research are also presented. Chapter two discusses morphological analysis and approaches to morphological analysis. Compounding in Amharic is discussed in chapter 3. The next two chapters discuss the design of lexicon, algorithm, and experiment and implementation, results, and evaluation, respectively. Finally, the conclusions and recommendations are presented in the last chapter.

CHAPTER TWO

Morphological Analysis

2.1. Introduction

This chapter addresses morphological analysis. It includes morphology, approaches to morphological analysis and morphological analysis for Amharic. Most of the examples are drawn from Amharic.

Morphological analysis is about the internal structure of word forms and relations (Roark and Sproat, 2007). It is the process of taking a string of characters as input and delivers an analysis as output.

2.2. Morphology

The term morphology is originated from aged Greek – morphe, which means the study of shape or form. Morphology is generally defined as the study of form or pattern, i.e. the shape and arrangement of parts of an **object** and how these conform to create a whole (Bajaj, 2008). The object in question can be physical objects (such as an organism, geography or ecology), social objects (such as an organization) or mental objects (such as linguistic forms, concepts or system of ideas). This study focuses on one of mental objects, i.e. linguistic forms.

Morphology, from a linguistics perspective, is the study of words and how they are put together (Lieber, 2010). Some words are composed by putting together smaller elements to form larger words with more complex meanings (Akmajian et al, 2001). These types of words can be decomposed into smaller meaningful units. In contrast, there are words that cannot be decomposed, called free morphemes.

Knowledge of morphology includes knowledge of individual morphemes, i.e. their pronunciation, meaning, and knowledge of the rules that combine morphemes to form complex

words (Fromkin et al, 2009). Knowledge of morphological rules helps to understand words that have never been encountered before and to create new words.

2.2.1. Words and Morphemes

As mentioned in chapter one, a morpheme is a minimal meaningful unit in a word. Types of words formed by composing smaller elements are **complex words**. Words that can't be decomposed into smaller meaningful units, on the other hand, are **simple words**. For example, the Amharic word ላም lam 'cow' is a simple word whereas ላሞች lamočč 'cows' is a complex one since it is composed of two meaningful minimal units: ላም lam, which is a noun, and the plural marker ኦች očč.

A word may be a result of one or more morphemes (Fromkin et al, 2009). The above Amharic word ላም lam consists of only one morpheme whereas the other word ላሞች lam + očč contains two morphemes. Likewise, ላሞቹ lam + očč + u 'his/the cows' has three and ላሞቹን lam + očč + u + n[5] 'his/the cows' has four morphemes. Morphemes like ላም lam, ሰው säw 'human being', and ወፍ wäf 'bird' are **free**. A free morpheme can stand alone as a word in a phrase, such as the word ቤት bet 'house' in ይህ የዮሐንስ ቤት ነው 'This is Yohannes's house'. Conversely, morphemes like ኦች očč and ኡ u give meaning and can occur only if attached to some other morphemes like ላም lam and ሰው säw. Such morphemes are **bound** because they can't stand by themselves (Akmajian et al, 2001).

Bound morphemes can be **affixes**, such as prefixes and suffixes. A **prefix** is attached to a base initially. የ yä in የላም yälam 'cow's', lä in ለላም lälam 'to cow', and kä in ከላም kälam 'from cow' are some of the prefixes of Amharic. On the other hand, a **suffix** is attached to a base. The Amharic bound morphemes ኦች očč and ኡ u can be examples of suffixes. Some Amharic suffixes, like ነት nnät as ሰውነት säw + nnät 'body or being a human', are used to derive a noun from another noun (Baye, 1987 E.C).

A morpheme that indicates grammatical features is called **inflectional morphemes** whereas a morpheme that is used to derive a new word when it is added to a word or word stem is

[5] 'u' refers definiteness and 'n' is the accusative case marker

derivational morpheme. The above mentioned morphemes, ኦች očč and ኡ u, are examples of inflectional morphemes of Amharic and ነት nnät, on the other hand, is a derivational morpheme.

2.2.2. Roots and Stems

The ideas of roots and stems are very important for morphological analysis. In English, for instance, the root of a word is the stem minus its word formation morphemes such as "work" in worker and "manage" in manager are roots (Booij, 2009). A stem is formed when a root morpheme is combined with an affix and by adding other affixes to the stem; a more complex stem can be formed (Fromkin et al, 2009). For instance, adding a suffix *atic* to the root *system* will form the stem *systematic* and also the stem *unsystematic* is formed by adding the prefix *un* to the stem *systematic*.

In Semitic languages such as Amharic the roots of verbs and most nouns are consonants, particularly three consonants. Related words can be derived by varying the pattern of vowels and syllables (Fromkin et al, 2009). For example, the root ጽሕፍ *s'hf* 'write' is used to form words like ጸሐፊ *s'ähafi* 'writer'and መጽሐፍ mäs'əhaf 'book' by infixing vowels through the process of derivational morphology.

Amharic nouns are either "primitive", i.e. they have no connection with verbal roots or nominal bases such as ሰው säw 'human being', or they are derived from verbal roots as ልብስ ləbs 'cloth' is derived from the root ል–ብስ l_bs and nominal bases as ዝምድና zəmdənna 'relationship' is derived from the noun ዘመድ zämäd 'relative' (Leslau, 1969). Table 2.1 and 2.2 present nouns derived from verbal root and stem, respectively (Baye, 2000 E.C; Getahun, 1990 E.C).

ስር Root	**ስም noun**
ል–ብስ l–bs	ልእብስ ləbs ልብስ 'cloth'
ም–ርት m–rt	ምእርት mərt ምርት 'product'
ል–ምድ l–md	ልእምድ ləmd ልምድ 'experience'

Table 2.1 Derived nouns from verbal roots

አምድ Stem	**ስም noun**
ቅልም qəlm–	ቅልም–አሽ qəlm-oš ቅልሞሽ 'paint'
ግርድ gərd–	ግርድ–አሽ gərd-oš ግርዶሽ 'darkness/ shadow'
ግብር gəbr–	ግብር–እና gəbr-ənna ግብርና 'farming'
ጥብቅ Təbq–	ጥብቅ–እና Təbq-ənna ጥብቅና 'to stand as an attorney'

Table 2.2 Derived nouns from stem

2.2.3. Inflectional Morphology

Words can also be classified as **content words** and **function words**. Content words express concepts while function words show functions or grammatical roles. Content words, also called **open-class** words, belong to major parts-of-speech or classes: Nouns, Verbs, Adjectives, and Adverbs. Some examples of content words in Amharic are shown below:

Content word	POS
ዛፍ zaf 'tree'	Noun
ፈረስ färäs 'horse'	Noun
አለፈ alläfä 'pass'	Verb
ጥሩ t'əru 'good'	Adjective
ገና gäna 'yet'	Adverb

Table 2.3 Examples of content words in Amharic

On the other hand, function words, also called **closed-class** words, indicate grammatical relations and have little or no semantic content. ከ kä 'from' and ወደ wädä 'on the way to' are some of the function words in Amharic.

Amharic content words have inflection whereas function words except few pronouns do not have inflection (Baye, 2000 E.C). ዛፎቻችን zafočča̅čn (zaf-oč-ača̅čn) 'our trees', for example, is inflected with two inflectional morphemes: ኦች očč, plural marker and ኣችን ača̅čn, person marker (1Pl). Adding inflectional morphemes to a word or word stem do not change the POS of the

word or the stem. As an example, when the plural marker ኦች očč is suffixed to the noun ድመት dəmät 'cat', the plural noun ድመቶች dəmätočč 'cats' is formed without any change in the POS.

2.2.4. Derivational Morphology

Derivational morphology is about word formation. It involves attaching derivational affixes to roots or stems. The result is a new word belonging to a different part of speech. Some of the rules that are used for deriving Amharic nouns, according to Baye (1987 E.C), are the following

Noun root + the derivational morpheme ነት nnät **→ derived nouns**

ስም 'Noun'	ውልድ ስም 'derived nouns'
ነጻ näs'a 'free'	ነጻነት näs'a + nnät 'freedom'
ሙሉ mulu 'full'	ሙሉነት mulu + nnät 'fullness'
ሰው səw 'human'	ሰው + ነት səw + nnät 'body or being a human '

Adjective + the derivational morpheme ነት *nnät* **→ derived nouns**

ቅጽል 'Adjective'	ውልድ ስም 'derived nouns'
ባዶ bädo 'empty'	ባዶነት bädo + nnät 'meaninglessness'
ትልቅ təlləq 'big'	ትልቅነት təlləq + nnät 'bigness'

Figure 2.1 Examples of derivatives in Amharic

One of the properties of derivational morphemes is that it leads semantic change. The meaning of the derived word is to some extent opaque (Plag, 2002). For instance, the Amharic derivational morpheme ኧት ät changes the meaning of አውቅ əwəq 'famous' to አውቀት əwəq + ät 'knowledge'. Both አውቅ əwəq and አውቀት əwəqät are nouns however they differ in meaning.

2.2.5. Compounding

Compounding is a morphological process of forming compound words. A word that is formed from two or more lexemes, include roots, stems, and free words that can make up compounds, is

a compound word (Lieber & Štekauer, 2010). In Amharic by combining two words or word stems as the word ትምህርት təmhərt 'education' and the word ቤት bet 'house', a new word - ትምህርት ቤት təmhərt bet 'school' is formed (see details in chapter 3).

There are three types of compound words written as **closed**, **hyphenated**, and **open**. Closed, or **solid,** compounds are written as single words with no separation. Compounds with hyphen are also possible. Open compounds, on the other hand, are written as sequences of words separated by spaces (Akmajian et al, 2001). For example, ቤተሰብ bet-ä-säb "family" is a closed compound, ሥነ-ልሳን sin-ä-ləsan "linguistics" is hyphenated, and ዳቦ ቤት dabo bet "bakery" is open compound.

2.3. Approaches to Morphological Analysis

There are several language dependent and independent approaches, which can be classified generally into machine learning (corpus-based), rule-based and algorithmic-based, used for developing morphological analyzer (Antony et al, 2012). The most commonly used approaches are the following:

2.3.1. Machine Learning Approach

Machine learning (or corpus-based) is programming computers to optimize a performance criterion using example data or past experience (Alpaydın, 2010). It is concerned with the design of algorithms that learn from examples given (Kumar M et al, 2010). A model is defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future or descriptive to gain knowledge from data, or both (Alpaydın, 2010).

In machine learning, model construction and usage are the two-step process. In this approach, a large sized well generated corpus is required for training and test dataset. Machine learning algorithm is used to train the corpus and collects the statistical information and other necessary features from the corpus. From the collected information, the model will be created. The model is represented as classification rules, decision trees, or mathematical formulae. The test dataset is

used to evaluate the model. It gives good results when the training set and testing data are similar (Shaalan, 2010).

There are mainly two types of machine learning. These are supervised and unsupervised machine learning algorithms. In supervised machine learning, examples of the available inputs with the required output are given to the machine. The task in supervised learning is to learn the mapping from the input to the output whose correct values are provided by a supervisor (Alpaydın, 2010). The algorithm is expected to produce generalized and classified rules from the given input.

Compare to the supervised learning, there is no supervisor in unsupervised learning, there is only input data. The aim is to find the regularities in the input. There is a structure to the input space such that certain patterns occur more often than others, and we want to see what generally happens and what does not (Alpaydın, 2010).

Machine learning approaches do not require hand coded morphological rules; rather they need corpora with linguistic information (Kumar M et al, 2010). The corpora can be used for training the machine and testing purposes. The necessary knowledge or morphological rules are automatically extracted from the corpora, i.e. training the machine and testing data. The performance of the system will depends on the feature and size of the corpus. The disadvantage is that corpus creation is a time consuming process. This approach is suitable for languages having well organized corpus (Antony et al, 2012).

2.3.2. Rule-Based Approach

Rule-based, also called knowledge-based, approach is based on a theory laid down by language experts (Kazakov and Manandhar, 2001). This approach is based on linguistic knowledge and it has been used in developing many NLP systems successfully (Shaalan, 2010). Experts of a specific language produce the rules of the language. The approach is more applicable to languages that have well defined sets of rules (Selvam and Natarajan, 2009).

Many scholars are motivated following the rule-based approach due to lack of available resources (Shaalan, 2010). According to Shaalan (2010), a rule-based approach has the following characteristics.

- It has a strict sense of well-formedness in mind,
- It imposes linguistic constraints to satisfy well-formedness,
- It allows the use of heuristics, and
- It relies on hand-constructed rules that are to be acquired from language specialists rather than automatically trained from data.

Rule-based approach has advantages in that it is easy to incorporate domain knowledge into the linguistic knowledge which provides highly accurate results. Furthermore, the linguistic knowledge acquired for one natural language processing system may be reused to build knowledge required for a similar task in another system (Shaalan, 2010). This approach is advantageous in that even though there may not be enough resource in the language, the approach can be used (Selvam and Natarajan, 2009).

2.3.3. Two-Level Morphology

This approach is developed by Kimmo Koskenniemi, a Finnish computer scientist, in 1983 and the approach is a general computational model for word-form recognition and generation (Antony et al, 2012). The development of two-level morphology was one of the major breakthroughs in the field of morphological parsing. Its advantage is that the model does not depend on a rule compiler, composition or any other finite-state algorithm. The "two-level" morphological approach consists of two levels called **lexical** and **surface form** and a word is represented as a direct, letter-for-letter correspondence between these forms (Antony et al, 2012). The Two-level morphology approach is based on the following three ideas:

- Rules are symbol-to-symbol constraints that are applied in parallel, not sequentially like rewrite rules;
- The constraints can refer to the lexical context, to the surface context, or to both contexts at the same time;
- Lexical lookup and morphological analysis are performed in tandem.

2.3.4. Finite State Approach

Finite state machine or automation FSA (or finite automation) uses regular expressions and is used to accept or reject a string in a given language (Antony et al, 2012). In general, an FSA is used to study the behavior of a system composing of state, transitions and actions. When FSA start working, it will be in the initial stage and if the automation is in any one of final state it accept its input and stops working.

Finite State Transducers (FST) is a modified version of FSA by accepting the principles of a two level morphology (Antony et al, 2012). A finite state transducer essentially is a finite state automaton that works on two (or more) tapes. The most common way to think about transducers is as a kind of "translating machine" which works by reading from one tape and writing onto the other. FST's can be used for both analysis and generation and it act as two level morphology. By combining the lexicon, orthographic rules and spelling variations in the FST, it can be built a morphological analyzer and generator at once.

2.3.5. Hybrid Approach

A hybrid approach, as the name indicates, is a combination of different approaches. The two machine learning approaches: a combination of supervised and unsupervised or the rule-based and the machine learning approaches can be used.

From the above mentioned approaches, this thesis followed the rule based approach on the basis of two-level morphology. The reason why this thesis selects rule based approach is besides its mentioned advantages, it is assumed that the criteria and rules of compounding in Amharic that are presented explicitly by experts of the language are good enough and can be used to design and develop a system.

2.4. Morphological Analysis in Amharic

Several researchers have been attempted to design and develop Amharic morphological analyzer, since Abiyot (2000) who is the pioneer for the development of Amharic morphological system.

Abiyot (2000) tried to develop a morphological analysis system for Amharic verbs and their derivations. The paper describes the morphology of Amharic verbs and attempts to extract features that enable to implement automatic Amharic word parser. The morphological data described and analyzed in this study is taken from study outputs in the area of Amharic word morphology. The prototype system, developed by Abiyot, uses different dictionaries (for suffix, prefix, and stem) and hand encoded rules (represented as algorithms) to provide that information. The experimental result of the system based on 200 verbs and 200 nouns shows a 94% accuracy level in morphological parsing for both parts of speech.

Unsupervised morphology learning was the other attempts by Tesfaye (2002). He uses a 5,236 words corpus to learn the morphology of Amharic using Linguistica. Linguistica is a freely available morphology learning program and requires a large corpus ranging from 5,000 to 1,000,000 words (Goldsmith, 2000). Tesfaye developed a stem internal morphological parser (called ASMA – Amharic Stems Morphological Analyzer) to handle the root-pattern Non-concatinative morphology of Amharic that Linguistica couldn't. To test the performance, the output of the systems (500 words and 255 stems from the output of Linguistica and ASMA, respectively) has been examined by two linguists and 94% of the stems and 87% of the words have been parsed successfully by ASMA and Linguistica, respectively.

Using the well–known finite state toolset, i.e. Xerox Finite State Tools, Saba and Gibbon (2005) developed a finite state based computational morphology system for Amharic. Their system was developed to be a complete analyzer of Amharic words of all categories. The analyzer display the root, pattern and feature tags indicating part of speech, person, number, gender, mood, tense, etc. The system achieved 88–94% recall and 54–94% precision when tested on biblical text (1620 words). Saba (2007) indicated that "the transducer for verbs was so complex to develop, particularly a deeper analysis of the constraints was lacking, which then resulted in over generation. The incomplete lexicon of nouns and adjectives has also resulted in not recognizing certain words".

Using a finite state approach, a tool called HornMorpho is developed by Gasser (2011). HornMorpho, which is a set of Python programs, is a freely available system for morphological processing of Amharic, Oromo, and Tigrinya. The analyzer was evaluated using 200 Amharic

words that are selected randomly from the Amharic wordlist[6]. According to the evaluation result, the analyzer made 2 errors on the Amharic verbs (99% accuracy) and 9 errors on the Amharic nouns and adjectives (95.5% accuracy). HornMorpho is a work in progress.

Wondwossen and Gasser (2012) have used a supervised machine learning approach to morphological analysis of Amharic verbs. In their report, they show the use of Inductive Logic Programming (ILP) to fast-track the process of learning morphological rules of complex languages like Amharic with a relatively small number of examples. They prepared the training data used to learn the morphological rules manually. The training resulted in 108 stem extraction and 19 root template extraction rules from the examples provided. After combining the various rules generated, the program has been tested using a test set containing 1,784 Amharic verbs. They achieved an accuracy of 86.99%.

Concluding the chapter, first it is discussed what morphology is and its major divisions: inflectional, derivational and compounding. Approaches to morphological analysis were discussed in second. The approach of this thesis was also mentioned in this section. Finally, the development of morphological analyzer for Amharic was discussed. The next chapter deals with compounding in Amharic.

[6] http://www.cs.ru.nl/~biniam/geez/crawl.php

CHAPTER THREE

Compounding in Amharic

3.1. Introduction

Compounding is one of the word formation processes in Amharic (Baye, 2000 E.C; Getahun, 1990 E.C). In the previous chapter, it has already been mentioned that compounding is a process of forming a new word by combining existing words. These words may belong to different lexical categories, i.e. nouns, adjectives, and verbs.

This chapter focuses on issues related to Amharic compounding. These include headedness, meaning and category.

3.2. Compound Basics

Most Amharic compounds are combination of two words. The following table presents examples of two word compounds.

1st Constituent	2nd Constituent	Compound
ሀገር hagär 'country'	ወዳድ wäddad 'lover'	ሀገር ወዳድ hagär wäddad 'lover of its own country'
ወደር wädär 'compare'	የለሽ yälläš 'no one'	ወደርየለሽ wädäryälläš 'incredible'
አልጋ alga 'bed'	ወራሽ wäraš 'claim'	አልጋ ወራሽ alga wäraš 'Prince '
ልብ ləbb 'heart'	ወለድ wäläd 'birth'	ልብ-ወለድ ləbb wäläd 'fiction'
ቃል qal 'word'	አቀባይ aqäbbay 'giver'	ቃል አቀባይ qal aqäbbay 'spokesperson'

Table 3.1 Some Amharic compounds

As it is observed from the above examples, orthographic form of Amharic compound can be in one of the three mentioned (Chapter 2, Sub section 2.2.5) forms: the closed form with no space in between, the open form with space between, or the hyphenated form with a hyphen in

between. There are also some compounds that combine more than two words. These include the following:

ሥነ-ሥርዓት አክባሪ	sən-ä-sərəat akbari	'obedient'
ሕገ መንግስት አርቃቂ	həgg-ä-məngəst arqqaqi	'constitution's architect'
ንግድ ሥራ ትምህርት ቤት	nəgd səra təmhərt bet	'commercial college/school'

ሥነ-ሥርዓት አክባሪ sən-ä-sərəat akbari and ሕገ መንግስት አርቃቂ həgg-ä-məngəst arqqaqi are compounds whose 1st member is a compound: ሥነ-ሥርዓት sən-ä-sərəat 'discipline' and ሕገ መንግስት həgg-ä-məngəst 'constitution' respectively. In contrast, ንግድ ሥራ ትምህርት ቤት nəgd səra təmhərt bet is a compound of two compounds: ንግድ ሥራ nəgd səra 'commercial' and ትምህርት ቤት təmhərt bet 'school'.

Suppose X, Y and Z are variables that each represents a word or word stem. A compound can be expressed schematically as follows:

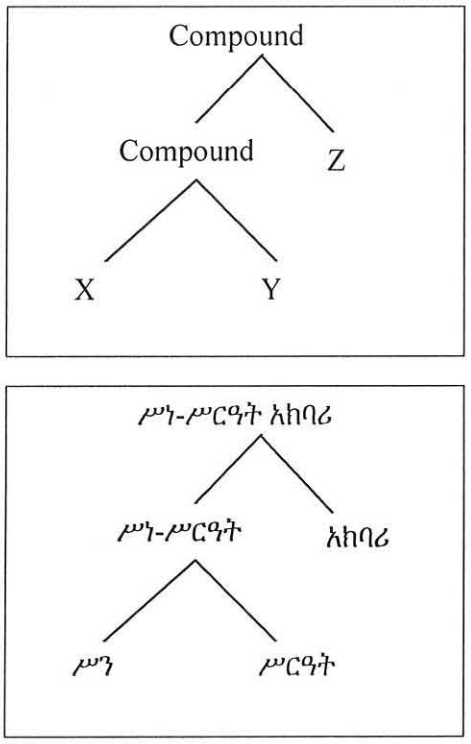


Figure 3.1 Example of Amharic compound scheme

3.2.1. Compounding Criteria in Amharic

How can compounds be lexical words and not phrases? In other words, it needs justification for a compound to be a lexical word. According to Baye (1987 E.C; 2000 E.C) and Getahun (1990 E.C), Amharic compound words, like compounds in other languages, should satisfy the following criteria fully or partially to be compound. A compound:

- has its own meaning that differs from its constituent words;
- can be a base for the derivation of other words;
- can take inflections;
- belongs to one of the major word classes or parts of speech;
- does not allow any morpheme between its constituents

There are compounds such as: ሊቀ መንበር liq-ä-mänbär 'Chair-person', አየር ወለድ ayyär wälläd 'Airborne', and ፋና ወጊ fana wägi 'vanguard (or the first in doing something)' that can fulfill the above criteria. Let's take ፋና ወጊ fana wägi and demonstrate the above criteria:

The first criterion says that a compound has its own meaning that differs from its constituent words as the meaning of the compound ፋና ወጊ fana wägi 'vanguard' is different from the meaning of its constituents: ፋና fana 'light' and ወጊ wägi 'piercer'.

The second criterion is about derivation of other words based on the compound. The compound ፋና ወጊ fana wägi is a base in forming the new word ፋና ወጊነት fana wäginnät 'to be vanguard' by adding ነት nnät to the end of the compound.

The compound word ፋና ወጊ fana wägi also fulfills the third criterion in that it allows the plural marker ኦች očč as ፋና ወጊዎች fana wägiočč and the definite marker ው w for masculine as ፋና ወጊው fana wägiw and ዋ wa for feminine as ፋና ወጊዋ fana wägiwa.

ፋና ወጊ fana wägi belongs to noun word class, which is the fourth criterion, and it doesn't accept any word or morpheme within it and by this it fulfills the fifth criterion. It is not acceptable or meaningless if the above derivational morpheme ነት nnät or the definite marker ው w and ዋ wa,

inserted between the two constituents of the compound i.e. *ፋናነት ወጊ fananät wägi, or *ፋናው ወጊ fanaw wägi or *ፋናዋ ወጊ fanawa wägi, respectively.

3.2.2. Amharic System of Compounding

The other point in Amharic compounding is that the language uses its system or Ge'ez[7] compounding system. When the Ge'ez system of compounding is used, the vowel ኧ ä appears on the first constituent as ቤት-ኧ-ክርስቲያን bet-ä-kərəstiyan (ቤተክርስቲያን) 'Church'. In contrast, in the Amharic system of compounding, a compound can exist with or without a connector. With a connector such as ልጃ ገረድ ləj-a-gäräd 'virgin' (feminine) is a combination of ልጅ ləj 'boy/girl' and ገረድ gäräd 'maid' with a connector ኣ a or without a connector such as ዳቦ ቤት dabo-bet 'bakery', which is a combination of ዳቦ dabo 'bread' and ቤት bet 'house'. The following compounds demonstrate this phenomenon.

አፍ-ኧ-ጉባኤ (አፈ-ጉባኤ)	af-ä-gubaə	'spokes person'
ዐይን-ኧ-ስውር (ዐይነ-ስውር)	ayn-ä-səwər	'blind'
ሕግ-ኧ-መንግስት (ሕገ-መንግስት)	həg-ä-mängəst	'constitution'
ልጅ-ኣ-ገረድ (ልጃገረድ)	ləj-a-gäräd	'virgin'
ቡና-ቤት	buna-bet	'bar'

Figure 3.2 Example of Amharic compounds with and without connector

Adding inflectional and derivational morphemes in a compound follows the same method as simple word, i.e. morphemes are attached to the right-end of a word.

[7] Ge'ez is an ancient language of Ethiopia (Semitic classical). It is mainly used now in the Ethiopian Orthodox Tewahdo Church (EOTC) and there are lots of books written in it. Source

Examples of adding the plural marker ኦች očč to compounds:

Singular	Plural
ትምህርት ቤት təmhərt bet 'School'	ትምህርት ቤቶች təmhərt betočč 'Schools'
ቤተክርስቲያን bet-ä-kərəstiyan 'Church'	ቤተ ክርስቲያኖች bet-ä-kərəstiyanočč 'Churches'
ሊቀመንበር liq-ä-mänbär 'Chairman'	ሊቀመንበሮች liq-ä-mänbäročč 'Chairmen'

Table 3.2 Examples of adding plural marker to compounds

Examples of adding the derivational morpheme ነት nnät to compounds:

ሊቀመንበር liq-ä-mänbär 'Chairman'	ሊቀመንበርነት liq-ä-mänbärnät 'being a chairman'
ቃል አቀባይ qal-aqäbay 'spokes person'	ቃል አቀባይነት qal-aqäbay 'being a spokes person'

Table 3.3 Examples of adding derivational morpheme to compounds

Though this is the process of adding plural marker to a compound in Amharic compounding system, it is also common to use the Ge'ez compounding system for Amharic. As shown in the table below, the system of Ge'ez makes the first constituent of the compound plural.

Plural in Amharic system	Plural in Ge'ez system
ሊቀ መንበሮች liq-ä-mänbäročč 'Chairmen'	ሊቃነ መንበር liqan-ä-mänbär 'Chairmen'
ቤተ ክርስቲያኖች bet-ä-kərəstiyanočč 'Churches'	አብያተ ክርስቲያን abiyatä-kərstiyan 'Churches'

Table 3.4 Examples of plurals in Amharic and Ge'ez system of compounding

3.2.3. Connectors in Compounding

A connector in compounding is used to connect constituents of the compound. Compounds in Amharic may have a connector (or linker), particularly compounds formed by following Ge'ez system of compounding, as mentioned in the above sub section, have a connector ኧ ä. The connector of a compound in Amharic may exist either attached to the first constituent (left-hand member of the compound) such as ኧ ä in ቤተክርስቲያን bet-ä-kərəstiyan 'Church' or independently in the middle of the two constituents such as በ bä in ቀስ-በ-ቀስ qäs-bä-qäs 'gradually'. The following table shows connectors of Amharic compounds:

Connector	Compound	1st constituent	2nd constituent
ኧ ä	መልከመልካም mälk-ä-mälkam 'beautiful'	መልክ 'face'	መልካም 'nice'
በ bä	እግርበግር əgər-bä-əgər 'in short distance'	እግር 'leg'	እግር 'leg'
ለ lä	ሰውለሰው säw-lä-säw 'human for human'	ሰው 'human'	ሰው 'human'
አ a	ልጃገረድ ləj-a-gäräd 'virgin'	ልጅ 'child'	ገረድ 'maid'

Table 3.5 Connectors of Amharic compounds

3.2.4. Compounding Levels

There are three levels of compounding in Amharic: fused, tight, and loose (Baye, 2000 E.C). Compounds such as ቀኛዝማይ (ቀኛዝማች qäññazmač 'a general officer of the right front') and ግራዝማይ (ግራዝማች gra-azmač 'a general officer of the left front') are fused, i.e. they are transformed to single word instead of being compound. According to Baye (2000 E.C), in the case of fused compound, it is not asked which the head[8] of a compound is. Most compounds that has a connector ኧ ä or -ኦ o such as ቤት-ኧ-መንግስት (ቤተ መንግስት) bet-ä-mängəst 'Palace' and ውል-ኦ-ገብ (ውሎ ገብ) wəl-o-gäb 'one day trip' are tight compounds (Baye, 2000 E.C). Compounds that have no connector such as ዳቦ ቤት dabo bet 'bakery' are loose.

3.3. Headedness and Meaning

The broad meaning of most Amharic compounds can be determined by seeing one of its constituents, which is the head. However, there are compounds where meaning cannot be determined from the head word.

3.3.1. Headedness

In compounding of two words, one of the constituents is a **head**, which is mostly the rightmost element of the compound, and the other is its **modifier** or non-head. The modifier of a compound modifies the head. A head is the one that can replace the whole compound without losing the general meaning and that has the same part of speech with the compound (Baye, 2000 E.C).

[8] Headedness is discussed in section 3.3

For example, the noun phrase ሕገ መንግስት ጣሽ həgg-ä-mängəst Taš 'constitution breaker' contains the compound ሕገ-መንግስት həgg-ä-mängəst 'constitution', which is a kind of ሕግ həgg 'law. Because the compound ሕገ-መንግስት həgg-ä-mängəst 'constitution' is a kind of ሕግ həgg 'law, it can be replaced by ሕግ həgg without losing the general meaning of the phrase, it is acceptable to say ሕግ ጣሽ həgg Taš 'law breaker'. When the compound is replaced by መንግስት mängəst in the phrase as መንግስት ጣሽ mängəst Taš, though it is correct, the general meaning is not the same as the phrase. The general meaning of the phrase is about breaking the law.

Although, in this specific example, both of the constituents have the same part of speech with the compound, i.e. noun, the compound belongs to the same class as the head. Because, the word ሕግ həgg can replace the compound and it has the same class as the compound, it is the head of the compound ሕገ-መንግስት həgg-ä-mängəst, not መንግስት mängəst.

The position of the head, in Amharic compounds, is not always to the right of its modifier. Some Amharic compounds that follow the Ge'ez compounding system are left-headed. Ge'ez compounding system, as mentioned above, uses the sound ኧ ä as a connector of the two words. The compound ቤተ መንግስት bet-ä-mängəst 'Palace', for instance, is a left-headed compound where ቤት bet 'house' is the head and the word መንግስት mängəst 'government' is the modifier. The meaning of the compound ቤተ መንግስት bet-ä-mängəst is 'house for the government or palace'.

In Amharic, even though it is said that compounds that follow the Ge'ez system are left-headed, there are compounds that follow the same system but they are right-headed (Baye, 2000 E.C). For example, ኧ ä connects the word ልብ ləbb 'heart', which is a noun and ትንሽ tənəšš 'small/narrow', which is an adjective and produce the compound ልብ-ኧ-ትንሽ (ልበ ትንሽ) ləbb-ä-tənəšš 'narrow-minded'. The head is not ልብ ləbb, it is ትንሽ tənəšš since the meaning of the compound is about narrowness as ትንሽ tənəšš and the compound is an adjective so as ትንሽ tənəšš. However, the connector ኧ ä is attached to the modifier (or non-head) ልበ ləbbä, not to the head.

Other compounds that follow the Amharic system of compounding are right-headed as ቤት bet 'house' in ሐኪም ቤት hakim bet 'hospital' (ሐኪም hakim 'physician'; ቤት bet 'house') and ዶሮ doro 'chicken' in አውራ ዶሮ awəra doro 'Cock' (አውራ awəra 'head/chief'; ዶሮ doro 'chicken'). አውራ ዶሮ awəra doro 'Cock' is kind of ዶሮ doro 'chicken' and it can be replaced by ዶሮ doro 'chicken'. For

instance, አውራ ዶሮው ጮኸ awəra dorow Čoĥä 'the cock is blared' can be replaced by ዶሮው ጮኸ dorow Čoĥä 'the chicken is blared' without losing the general meaning of the sentence. However, if it is አውራ ጮኸ awəra Čoĥä 'chief blared', it will have different meaning from the sentence. Therefore, the head is ዶሮ doro 'chicken', not አውራ awəra.

The other thing about Amharic compound head is, whether the head is to the right or not, inflectional affixes are attached to the right of the compound. The plural marker ኦች očč is attached to the compound ሐኪም ቤት hakim bet 'hospital', which is head final, as ሐኪም ቤቶች hakim betočč 'hospitals' and to the compound ቤተ መንግስት bet-ä-mängəst 'Palace/house of government', which is head initial, as ቤተ መንግስቶች bet-ä-mängəstočč 'Palaces'.

3.3.2. Meaning

There is a semantic relation between ቤት bet 'house' and the compound ትምህርት ቤት təmhərt bet 'education house or School', ዳቦ ቤት dabo bet 'bakery', and ቡና ቤት buna bet 'Bar', i.e. ቤት bet is the head of each of the specified compounds. In other words, these specified compound words are hyponym of the word ቤት bet. They have the IS-A[9] relation, i.e. the compound words: ትምህርት ቤት təmhərt bet, ዳቦ ቤት dabo bet, and ቡና ቤት buna bet **is a** special type of ቤት bet. Most compounds of Amharic have this kind of semantic relation, i.e. the meaning of a compound word is a special type of the meaning of the head and these kinds of compounds are called **endocentric** compounds (Plag, 2002).

Example:

ዳቦ ቤት dabo bet 'bakery' is kind of ቤት bet 'house'

ሰብአዊ መብት säbawi-mäbt 'human-right' is kind of መብት mäbt 'right'

መጋቢ መንገድ mägabi-mängäd 'detour' is kind of መንገድ mängäd 'road'

[9] IS-A is kind of relation that shows a class X **is a** subclass of a class Y or class Y **is a** generalization of X.

3.4. Compound Categories

Compound words in Amharic can be produced from combinations of major word classes. The lexical category is determined by the head constituents, for example, if the head is noun, the compound is a noun.

3.4.1. Compound Nouns

Amharic compound nouns can be produced by combining nouns and other major lexical categories. Baye (2000 E.C) differentiates the rules of compound nouns into five. These rules are:

Rule 1: **ስም 'Noun' + አጣማሪ 'connector' + ስም 'Noun' → Compound noun:**

ቤት-ኧ-ክርስቲያን	bet-ä-kərəstiyan	'church'
ቤት-ኧ-ሰብ	bet-ä-säb	'family'
አፍ-ኧ-ጉባኤ	af-ä-gubae	'spokesperson'

Rule 2: **ስም 'Noun' + ስም 'Noun' → Compound noun:**

አየር መንገድ	ayyär-mängäd	'airline'
ቡና ቤት	bunna-bet	'bar'
ምግብ ቤት	məgəb-bət	'restaurant'

Rule 3: **ስም 'Noun' + ግስ- 'Verb' → Compound noun:**

ወለድ አገድ	wälläd-aggäd	'trust/trustee'
ሰው ሰራሽ	säw-säraš	'man-made'
ወዝ አደር	wäz-addär	'laborer'

Rule 4: **ግስ- 'Verb' + ግስ- 'Verb' → Compound noun:**

ወርሮ በላ (ወሮበላ)	wäro-bäla	'gangster'
ወድዶ ገብ- (ወዶገብ)	wädo-gäb	'volunteer'

Rule 5: **ግስ- 'Verb' + አ o + ስም 'Noun'** → **Compound noun:**

ፈጥን-አ ደራሽ (ፈጥኖ ደራሽ)	fäTn-o-däraš	'fast arrival'
ሰርት-አ አዳሪ (ሰርቶ አዳሪ)	särt-o-addari	'hard worker'

The hyphen shows that the derived form is a compound. Though there is inconsistency of use, any of the three possibilities can be used. Single word as ቡናቤት bunabet, two separate words as ቡና ቤት buna bet, or hyphenated as ሥነ-ሥርዓት.

As it is observed from the above examples, some compounds show the morpheme ኧ ä, which is a linker. This is common in Noun-Noun or Noun-Adjective compounds.

3.4.2. Compound Adjectives

Compound adjectives are formed from the combination of a noun and an adjective with the linker vowel ኧ ä. Examples are the following:

ስም 'Noun' + ቅጽል 'Adjective':

እግር əgər 'leg' + ቀላል qälal 'easy' = እግረ-ቀላል əgər-ä-qälal 'fast'

መልክ mälk 'face' + መልካም mälkam 'fine' = መልከ-መልካም mälk-ä-mälkam 'beautiful/handsome'

Some compound adjectives are formed from the combination of ስም 'Noun' and negative particles such as:

ልብ ləb 'heart' + ቢስ bis 'not have' = ልበ-ቢስ ləb-ä-bis 'forgetful'

አቅም aqəm 'capacity' + ቢስ bis 'not have' = አቅመ-ቢስ aqəm-ä-bis 'weak'

The head of these kinds of compounds cannot be identified using the criterion; a head is the one that can replace the whole compound without losing the general meaning, because both of the

constituents cannot replace the whole compound without losing the general meaning[10]. Instead, the head can be identified by comparing the meaning of the compound with its constituents. For example, the meaning of the compound ልበ-ቢስ ləb-ä-bis 'forgetful' is related to the meaning of one of its constituent, ቢስ bis 'not have', and thus the head of the compound is ቢስ bis.

3.4.3. Compound Verbs

Amharic has compound verbs to a limited extent. According to Baye (2000 E.C), the limitation is due to the following:

- The first constituents is always an ideophone such as ዝም zəm and
- The second constituent is the verb to 'say' and to 'do'.

Some of compound verbs are the following:

ዝም zəm 'silence' + አለ alä 'he said' = ዝም አለ zəm alä 'he kept quiet'

ብድግ bədəg 'stand-up' + አለ alä 'he said' = ብድግ አለ bədəg alä 'he stood up'

ጸጥ s'ä'ī 'quiet' + አደረገ adärägä 'he did' = ጸጥ አደረገ s'äī adärägä 'he did silent'

Words like ዝም zəm and ብድግ bədəg cannot inflect. However, the compound as a whole shows inflection. The inflection is attached to the head አለ alä and አደረገ adärägä:

ዝም አለ zəmm alä 'he kept quiet'

ዝም አደረገ zəmm adärägä 'he did silent'

ዝም አለች zəmm aläč 'she kept quiet'

ዝም አደረገች zəmm adärägäč 'she did silent'

ዝም አሉ zəmm alu 'they kept quiet'

ዝም አደረጉ zəmm adärägu 'they did silent'

[10] See Baye (2000 E.C, p. 223 and 224) for details.

3.4.4. Compound Adverbs

Adverbs in Amharic can be formed by reduplication and compounding. Compound adverbs formed by combining words or word stems, which have no explicit class, with a connector. A compound adverb, like other compounds mentioned before, does not allow inserting any morpheme kind between its constituents (Baye, 2000 E.C). These compounds have the following pattern:

$C = c1 + con + c2$; where C is a compound, *con* is one of the connector, either በ bä or ለ lä, and *c1* is the same word as *c2*.

According to Baye (1987 E.C; 2000 E.C), identifying the head of compound adverbs is very hard, however following the patterns of other compounds, it can be concluding that compound adverbs are right-headed.

3.5. Inflectional Affixes of Amharic Compound

Amharic nouns and adjectives are inflected by adding suffixes to indicate gender, number, definiteness, and case (Baye, 1987 E.C; Getahun, 1990 E.C; Baye, 2000 E.C). There are also prepositions that are attached to compounds.

3.5.1. Compound Suffixes

Semantic and syntactic information of compounds are inherited from their head (Plag, 2002). For instance, if the head is feminine, the compound will be feminine. Head in Amharic occurs on the right-hand side of the compound though there are compounds formed by following the Ge'ez system of compounding in which head appears to the left (see Section 3.3). Examples of inflectional suffixes of Amharic are:

1) ምግብ ቤቶች	4) ሊቀመንበራቸው
məgəb-bet-occ	liq-ä-mänbär-accäw
food-house-Pl	genius-connector-seat-3Pl
'restaurants'	'their chairman'
2) ቤተሰብሽ	5) ነፍሰ ገዳይዋን
bet-ä-säb-š	näfs-ä-gäday-wa-n
house-connector-human-2FS	soul-connector-murderer-3FS-ACC
'your family'	'murderer'
3) ልበ ቅኖች	6) ዝም አለ
ləb-ä-qən-occ	zəm-al-ä
heart-connector-kind-Pl	silent-stem for 'say'-3MS
'kind-hearted'	'he keeps quiet'

Figure 3.3 Examples of suffixes showing grammatical features of compounds

The plural marker አች occ is suffixed to compound nouns such as in (1) and adjectives such as in (3). Adjectives also have another way to indicate their plural number, by reduplicating the penult consonant and adding the vowel 'a' as a connector. Take the compound ልበ ትልቅ ləb-ä-tələq 'broad-minded' as an example:

Singular	Plural	
	Method 1	Method 2
ልበ ትልቅ ləb-ä-tələq	ልበ ትልቆች ləb-ä-tələq-occ	ልበ ትላልቅ ləb-ä-tələlaq

Table 3.6 Examples of methods of pluralizing adjectives

3.5.2. Compound Prepositions

Amharic preposition that indicate, for instance, place and time may attach to the beginning of a compound. Prepositions may not have meaning by themselves but they may show functions when they are attached to nouns (Baye, 2000 E.C). Prepositions of Amharic are the following:

Preposition	Example
የ yä 'of'	የቤተ ክርስቲያን yä-betäkərəstiyan
ከ kä 'from'	ከምግብ ቤት kä-məgəb bet
በ bä 'by'	በብረት ምጣድ bä-bərät məTad
ለ lä 'to/for'	ለልበ ቢስ lä-ləbbäbis
ስለ slä 'instead of'	ስለአየር መንገድ sələ-ayyär mängäd
ወደ wädä 'toward'	ወደአየር መንገድ wädä-ayyär mängäd
ከነ känä 'together with'	ከነቤተሰባችን kännä-betäsäb-ačč ən
እስከ əskä 'until'	እስከአየር ኃይል əskä-ayyär hayl
እንደ əndä 'as/like'	እንደአየር ወለድ əndä-ayyär wäläd

Table 3.7 Prepositions of Amharic

Attaching preposition to a compound forms a prepositional phrase such as ከነቤተሰቡ känä-betäsäb-u 'together with his/the family' as shown in the figure below.

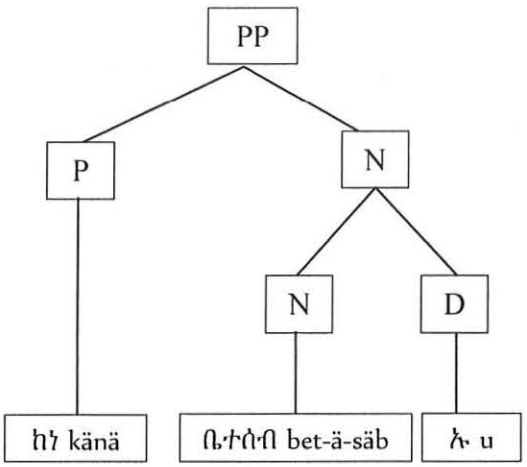


Figure 3.4 Examples of prepositional phrase

In this example, the preposition is ከነ känä 'together with', ቤተሰብ bet-ä-säb 'family' is the compound and -ኡ -u is a suffix which indicates the 3rd person masculine singular definite (3MSDEF).

CHAPTER FOUR

Design and Experiment

4.1. Introduction

This chapter describes the design of lexicon of Amharic, algorithms used in the development of Amharic compound analyzer and the rules of the analyzer. The first section describes the design of the lexicon required by the analyzer. The next section discusses the design of algorithms that are necessary for the identification and analysis of Amharic compounds. Finally, rules that are used in this work are discussed.

In chapter 2 of this thesis, different approaches to morphology analysis are discussed. In this research, the system for the analysis of Amharic compounds uses the rule based approach with lexicon of simple words and word stems maintained in the system.

4.2. Lexicon

Lexicon is a mental dictionary that lists the words of the language (Fromkin et al, 2009). It is one of the components of morphological analyzer. Enhancement level of the lexicon is directly related with the quality of the analyzer. There are two aspects that contribute to this enhancement level, the number of lexical entries contained in the lexicon and the richness in linguistic information contained by the lexical entries (Köprü and Miller, 2009).

In this research, simple words and word stems[11], which cover the major lexical categories of the language, with their linguistic features are maintained (saved) in the system. Major lexical categories include nouns, adjectives, verbs and their derivatives. These categories are selected because Amharic compounds are formed from combinations of items from these lexical categories. With this the required quality is reassured.

[11] Simple word is free morpheme (see section 2.2.1) where as word stem is a form that bound morpheme is attached to it (see Baye's (1987 E.C, p. 327) book).

Lexicon of the language can be maintained either manually or automatically using corpuses. This research used both methods. Most of the lexicons of the language are acquired automatically from Amharic dictionary of Amsalu (1979 E.C). This dictionary is available[12] digitally by Daniel Yaqob under the Creative Commons Attribution-Noncommercial 3.0 United States License.

4.2.1. Preprocessing

Source of the lexicon, a dictionary, contains all the lexical categories of the language besides the major ones: nouns, adjectives, verbs, adverbs. It contains simple words, word stems, and compound words. The format of the dictionary looks like the following.

ሀብት	+	n.	
+	ሀብተ ሰባራ		adj.
+	ሀብተ ስንኩል		adj.
+	ምጣኔ-	n.	
+	ሀብታም	adj.	
+	ሀብታምነት		n.
ሀይ ሀይ አለ		+	v.t.
ለምጽ	+	n.	
+	ለምጻም	adj./n.	
ፍሬ	+	n.	
+	-ቢስ	adj.	
+	ፍሬ ቢስ	adj.	

Figure 4.1 Format of the dictionary

In the dictionary, the word appears in the first order such as ሀብት häbt 'resource' and ፍሬ fəre 'fruit' is the base whereas words that appear in the second order are relatives of the bases. The other point is, when hyphen appears before and after the word, it shows that the word attaches the base word before and after it. As shown in Figure 4.1, for instance, in the second order there is a word ምጣኔ-, the hyphen, appears after it, shows the base word, ሀብት, can be attached after it

[12] at <http://nlp.amharic.org/resources/lexical/word-lists/dictionaries/

The analyzer requires only simple words and word stems because they are constituents of compound words and the system searches for constituents of a given word not the compound itself. Therefore, compound nouns such as ኅብረተሰብ həbrät-ä-säb and adjectives such as ፍሬ ቢስ fəre bis and phrasal nouns such as መሸቀጫ ገንዘብ mäšäqäCa gänzäb are not taken in the lookup files. Results of the algorithm after preprocessing of the text in Figure 4.1 are shown below.

ሀብት
ምጣኔ
ሀብታም
ሀብታምነት
ባለሀብት
ለምጽ
ለምጻም
ፍሬ
ቢስ

Figure 4.8 Results of the algorithm after preprocessing of the text in Figure 4.1

Each constituent word of a compound and phrase independently exists in the dictionary. So, constituent words of the stripped compound and phrase are included in the lexicon. For instance, the source dictionary contains both the compound ፍሬ ቢስ fəre-bis and its constituents' ፍሬ fəre and ቢስ bis, however the algorithm ignores the compound ፍሬ ቢስ fəre-bis and includes the constituents, ፍሬ fəre and ቢስ bis, in the list.

There are compounds such as ሴታሴት set-a-set 'effeminate' that are not separated by space. Because they are not separated by space, the algorithm takes such compounds as simple words and includes in the list. These compounds are removed manually, because, as it is mentioned above, the analyzer requires only simple words and word stems.

4.2.3. Transliteration

Transliteration is writing or describing words or letters using letters of a different alphabet or language (Wehmeier et al, 2005). The Amharic character (graphic representation) cannot display the consonant and the vowel separately. For instance, the letter ለ lä consists of the consonant ል

(l) and the vowel ኧ (ä). In the case of compounds such as ቤተሰብ bet-ä-säb, it is not possible to extract the two words ቤት bet and ሰብ säb using the Amharic orthography. Extracting the connector ኧ ä is a must to identify constituents of the compound and it is possible when the compound is transliterated.

In this study, the transliteration is based on the conventions of the SERA system, which is used for Ethiopic representation in ASCII[13] (Yitna and Yacob, 1996). The SERA system is selected because it is easily available and it is also commonly used in linguistic research for local languages of Ethiopia (Gasser, 2011; Gambäck and Asker, 2010). However, some modifications are made by the researcher.

There are letters that have the same pronunciation, such as ሀ, ሐ, and ኀ, and the SERA system represents each independently. Representing these letters independently is important for meanings of words. However, in this study, meanings of the words are not important. Moreover, representing these characters with one letter decreases number of characters in the lookup table (or memory space) and increases searching speed in the time of transliteration. Thus, letters, which have the same pronunciation, are represented by one letter. For example, ሀ, ሐ, and ኀ are represented by letter (fidel) hä. Besides, when the analyzer generates the output (the analysis), the input word appears as it is, in Ge'ez script. Only the analysis part is presented in the transliterated form.

Some of the letters the SERA system uses for representation are replaced by IPA equivalents, which have Unicode representation, to make the letters more readable when the system produces the output to the users. For instance, the vowel **e** that is used for the first order of Amharic alphabet (fidel) in SERA is replaced by **ä**. Because of such modifications, the transliterated data is represented in Unicode. The following example demonstrates the modification, the transliteration and the final output.

[13] ASCII stands for American Standard Code for Information Interchange.

Input word:	ቤተክርስቲያን
Transliteration with SERA:	bE te k r s ti ya n
Transliteration with the modified:	be tä k r s ti ya n
The final output:	be tä kə r sə ti ya n

Figure 4.3 Transliteration example

4.2.3.1. Transliteration Algorithm for Input Text

In this research, an algorithm is designed for the purpose of transliteration. In the compound identifier and analyzer, the input text may be from file or keyboard and it is in Ge'ez script. Then the algorithm reads an Amharic letter from the input text and searches for a match in the list of character table. When the algorithm finds a match, it will replace the letter by the representing character. This process continues until the last letter is replaced. The following is the algorithm:

```
Input: text
Output: transliterated list of words
Accept text
1       char_dic ← empty dictionary
2       chs ← list of characters in character table
3       for i ← 0 to length(chs)-1 do
4           char_dic[chs[i]] ← chs[i+1]
5       for ch in text do
6           if char_dic.__contains__(ch)
7               text ← text.replace(ch, char_dic[ch])
8       words ← tokenized text
9       return words
```

Figure 4.4 Algorithm for transliteration

The above algorithm firs it put each character from character table to empty dictionary[14] as a key and value mode, then it reads the first character of the input text and when it finds a match in the dictionary (by comparing the character with the keys of the dictionary), the algorithm replaces

[14] Since the system is developed using Python programming, it uses dictionary, one of the Python's data type.

the character with the value of the dictionary. Some of the characters in the character table list look like the following:

Key	Value
ሀ	hä
ለ	lä
ሁ	hu
ሉ	lu
ሂ	hi
ሊ	li

Figure 4.5 Sample characters in character table

Let's take ለልበ መሉወ as the input text and show the processes of the algorithm. First, the algorithm takes the letter ለ and searches for a match in the list of character table. It will get a match for ለ and replaces it by the representing character, **lä**. It will continue searching and replacing all until it reaches the end of the text. The result of the algorithm for the given input text is shown below:

ለ	ል	በ	መ	ሉ	ወ
lä	lə	bä	mu	lu	w

Figure 4.6 Sample transliteration

After the lexicon is extracted and manually checked, each item of lexicon is transliterated using SERA system with some modifications (Section 4.3) and saved separately as text files. The source dictionary contains 16,231 words and word stems. Out of these words, a total of 6, 219 words and word stems have been extracted and maintained in the lexicon. The composition of the words in the source dictionary and in the lexicon is presented in the following table.

Total number of nouns in the source dictionary	6,888
Extracted number of nouns after the first preprocessing	6,298
Number of nouns after transliteration and last preprocessing	4,991
Total number of adjectives in the source dictionary	1,832
Extracted number of adjectives after the first preprocessing	1,705
Number of adjectives after transliteration and last preprocessing	1,228

Table 4.1 Statistics of lexical items included in the research

Moreover, verbal nouns such as አደር adär and verbs such as ሰርቶ särto are collected manually and saved as a text file in a separate file.

4.3. Affixes

In addition to words and word stems, affixes particularly prefixes and suffixes of Amharic that are attached to compound nouns and adjectives are maintained in the system. This section describes the design of affixes. The following prepositional phrases may show some examples of prefixes and suffixes that might be found in a word:

1) ከቤተመንግስቱ
   kä-bet-ä-mängəst-u
   from-house-connetor-gevernment-DEF/3MS
   'from the/his palace'
2) ለትምህርት ቤቴ
   lä-təmhərt-bet-e
   for-education-house-1Sg
   'for my school'

Figure 4.9 Examples of affixes of Amharic

In the above example, ቤተመንግስት bet-ä-mängəst 'palace' and ትምህርት ቤት təmhərt-bet 'school' are compound nouns, ከ kä 'from' and ለ lä 'for' are prefixes (prepositions) and ኡ u, marker for definiteness and 3rd person masculine singular, and ኤ e (possessive suffix pronoun), marker for 1st person singular, are suffixes, in (1) and (2) respectively. As discussed in chapter 3, some compounds of Amharic uses a connecter such as -ä in (1) to combine the two constituents in the process of forming compounds.

4.3.1. Prefixes

As discussed in chapter 3, when a preposition is prefixed in compound nouns or adjectives, prepositional phrases will be formed. To identify and extract compounds from prepositional phrases, recognizing and separating prepositions is the first. For this purpose, Amharic prepositions (see chapter 3) are collected manually and maintained in the system as text lookup file in concatenated and non-concatenated form.

4.3.2. Suffixes

Amharic nouns and adjectives, as discussed in chapter 3, are inflected by adding suffixes to indicate gender, number, definiteness, and case. In this research, suffixes that indicate gender, number, definiteness, and case are collected in two forms, concatenated and non-concatenated, and saved in text files independently.

In concluding to this section, there are eight text files used as knowledge base to the analyzer. These files are:

Type of files	Number of files	Remark
Lexicon of Nouns	1	
Lexicon of Adjectives	1	
Lexicon of Verbal nouns and verbs	1	
Prefix list	1	
Suffix list	2	one for holding type of suffix with their grammatical meaning and the other for holding suffixes list
Character list	1	for transliteration
Punctuations list	1	for preprocessing

Table 4.2 Summary of files used

4.4. Algorithms Development for the Analyzer

Recognizing compounds is the first thing to do in the analysis of compounds. In recognizing Amharic compound, knowing the input word has at least two constituents and satisfies the criteria of compounding is a must. The system has to be capable of recognizing affixes, prepositions and suffixes, of a compound.

The next thing after recognizing the compound and its affixes (if any) is determining the morphosyntactic properties of a compound. It involves presenting the stem, the compound and each of the constituents with their POS and interpreting affixes. To do these, recognizing and analyzing of Amharic compounds, the following algorithms are developed from scratch. Each of the algorithms developed is implemented as functions of the system.

4.4.1. Tokenizer

Tokenization is the process of breaking up the string into words. It is applied, in this research, if the input is a raw text. In the case of wordlist, after preprocessing, the wordlist is tokenized and

each line of words is taken for the next process. On the other hand, when the input is a raw text, because compounds can exist in open form, the tokenization must include not only tokens that exist freely but also compounds that exist in open forms (see Chapter 3, Section 3.2). Take the following raw text as input:

'እግረ ቀጭን እንደ ሰሳ ፣ ልበ ሙሉ እንደ አንበሳ' አያለ" ጧት ማታ ራሱን የሚያሞካሸ ፣ እኔን ሲሻው "አቅመ ቢስ!" ሲሻው "ቀልበ ቢስ!"
əgrä qäCCn əndä säsa ləbä mulu əndä anbsa əyalä Təwat mata rasun yämiyamokaš ənen sišaw aqmä bis sišaw qälbä bis

Figure 4.10 Examples of raw text

The raw text contains the following five compounds and all of them are in open form (i.e. separated by space):

እግረ ቀጭን	əgrä qäCn	'the one who has thin legs'
ልበ ሙሉ	ləbä mulu	'unafraid'
ጧት ማታ	Təwat mata	'daily'
አቅመ ቢስ	aqmä bis	'weak'
ቀልበ ቢስ	qälbä bis	'unmindful'

Figure 4.11 Examples of compounds in open form extracted from raw text in Figure 4.10

So, by following the word level tokenization, these kinds of open form compounds will be missed. This is because, word level tokenization takes each freely existing word, not two words that are separated by space though they are constituents of a compound word. To include these kinds of compound words to the wordlist, which are given to compound identifier, the following algorithm is designed:

```
Input: the preprocessed raw text
Output: wordlist
Accept: preprocessed raw text
1       words ← list of words in raw text
2       c ← list of possible second constituents of Amharic compounds
3       l ← empty list
4       for i ← 0 to length[words] do
5           add words[i] to l
6           bigram ← words[i] + words[i+1]
7           if words[i] ends with 'ä'
8               add bigram to l
9           else if words[i+1] in c and words[i] has no suffix
10              add bigram to l
```

Figure 4.12 Algorithm to tokenize a raw text

Suppose the input is the raw text in Figure 4.10, the algorithm takes the transliterated text and does the following:

Process #1: *tokenize the raw text in word level, 20 words;*

Process #2: *It has been prepared the list of possible second constituents of Amharic compounds and the list is assigned to c;*

Process #3: *prepare an empty list, l;*

Process #5: *add each word to l*

Process #7: *if word W ends with ä, take W and the word next to W in concatenation and add to l, process #8;*

Process #9: *if word W is one of the possible second constituents of Amharic compound (first, suffixes, if any, are removed from the word W because the list, c, contains simple words and word stems) and the word, previous to W, has no suffix*[15]*, then take W and the word previous to W in concatenation and add to l, process #10;*

Finally, the wordlist will have 27 words, the five compounds in Figure 4.11 and the 20 tokens, process #1, and the two words , እንደ ሰሳ əndä säsa and እንደ አንበሳ əndä anbsa, because the word እንደ əndä ends with ኧ ä. If only the wordlist that are identified by following the word level

[15] See compounding criteria in Chapter 3, Sub section 3.2.1

tokenization (unigram) is supplied to the system, the system identifies nothing as a compound because the raw text contains no compound in closed form. However, the wordlist prepared by this algorithm contains not only simple words but also bigrams, which are nominated to be compounds. This wordlist is supplied to the system for further processes and the system identifies compounds in Figure 4.11.

4.4.2. Prefix Extractor

The system uses the following algorithm to extract prefixes (prepositions) from a given input word by searching for the equivalent prefix in the concatenated prefixes list.

```
Input: the transliterated input word
Output: prefix
Accept word
1       prefixes ← list of concatenated prefixes
2       pr ← ''
3       for i ← 0 to length[word] do
4           pr ← word – i number of character from the last
5           If pr in prefixes
6               return pr
7               break
8       return pr
```

Figure 4.13 Algorithm to extract prefixes from a compound word

To demonstrate how the algorithm works, take the word አስካቤተሰቤ əskänä-bet-ä-säb-e 'along with my family' is a given input; then the prefix extractor starts search in the concatenated prefixes list to get an equivalent prefix by taking the input word as a whole (since i starts from 0, there is no character to be sliced at the start). If an equivalent prefix is not found, then it will take the rest of the word as input after removing the left most character and continue searching until it finds one or no character is leftover. If there is no equivalent prefix found, the algorithm will return an empty string. Otherwise, it will return the extracted prefix and stop searching. In this specific example, the prefix አስካ əskänä 'along with' is found at the eighth iteration. The iteration looks like the following:

Iteration	Pr	Is pr a preposition?	Remove	Left	Status
Start	əskänäbetäsäbe	No	e	əskänäbetäsäb	Continue
1	əskänäbetäsäb	No	be	əskänäbetäsä	Continue
2	əskänäbetäsä	No	äbe	əskänäbetäs	Continue
3	əskänäbetäs	No	säbe	əskänäbetä	Continue
4	əskänäbetä	No	äsäbe	əskänäbet	Continue
5	əskänäbet	No	täsäbe	əskänäbe	Continue
6	əskänäbe	No	etäsäbe	əskänäb	Continue
7	əskänäb	No	betäsäbe	əskänä	Continue
8	əskänä	Yes	-	-	Break

Table 4.3 Prefixes extractor algorithm demo

4.4.3. Suffix Extractor

The process starts by taking the stem, which is bare after extraction of the prefix (if any) and the first constituent from the input word and then continues searching for equivalent suffixes in the concatenated suffixes list by taking the whole stem as a suffix. If equivalent suffix is not found, then the algorithm takes the rest of the stem after striping the first letter that is the initial one of the stem since it needs to find the suffix and continues searching until it finds one or no letter is leftover. If there is no equivalent suffix found, the algorithm will produce an empty string. Otherwise, it will return the extracted suffix and stop searching.

```
Input: stem
Output: suffix
Accept stem
1       suffixes ← list of concatenated suffixes
2       sfx ← ''
3       for i ← 0 to length[stem] do
4           s ← stem minus i number of characters from the start
5           if s in suffixes
6                sfx ← s
7                break
8       return sfx
```

Figure 4.14 Algorithm to identify suffixes

Suppose the input word is the previous one, አስካቤተሰቤ əskänä-bet-ä-säb-e 'along with my family'. The stem that is accepted by the suffix extractor will be ችሰቤ ä-säb-e since the prefix, አስካ əskänä and the first constituent, ቤት bet is extacted. Then, the process looks like the following.

Iteration	sfx	s	Is s in concatenated suffixes list?	If no, remove initial	If yes, assign s to sfx	Left	Status
Start	''	äsäbe	No	ä	-	säbe	Continue
1		säbe	No	s	-	äbe	Continue
2		äbe	No	ä	-	be	Continue
3		be	No	b	-	e	Continue
4		e	Yes	-	e	-	Break and return sfx, e

Table 4.4 Suffixes extractor algorithm demo

At the 4th iteration (cycle), the algorithm finds a suffix match, -ኤ -e in the concatenated suffixes list, assigns it to sfx and it stops after returning the extracted suffix. However, if there is no match to s in the concatenated suffixes list, there is no assignment of s to sfx. Thus, sfx remains empty string and the algorithm returns this empty string.

4.4.4. Extractor of the First Constituent of the Compound

As it is observed in Baye's (1987 E.C; 2000 E.C) and Getahun's (1990 E.C) books, the first constituent of the compound in Amharic is noun, verbal noun or verb. Thus, this algorithm takes the first character of the word from the left and searches the list of nouns or verbal nouns from the lexicon. If it finds one, it will return the constituent and its POS or else it will continue searching by adding a character from the left until no character is left. If there is no match found as a first constituent, and then the algorithm assumes a prefix to be there. Thus, the algorithm will check for a prefix and if there is one, it will continue searching for the first constituent after stripping the prefix. If no prefix is found or no first constituent is found though the prefix is found and stripped, the algorithm will decide that the first constituent is not included in the lexicon and it will return an empty string.

The POS of the first constituent is determined by the lexicon file for which the match for a constituent is found. For instance, if a match for the first constituent is found in the noun lexicon file, then the POS of the constituent will be noun.

```
Input: the transliterated input word
Output: the first constituent and its POS
Accept word
1       nouns ← list of noun lexicons
2       verbs ← list of verbal noun and verb lexicons
3       c1, pos ← ''
4       for i ← 0 to length[word] do
5               x ← word minus i number of characters from final
6               if x in nouns or verbs list
7                       c1 ← x
8                       pos ← list name
9                       break
10      if c1 is null
11              call prefix extractor to check if there is prefix
12              if any prefix
13                      word ← word – prefix and go to 5
14      return c1, pos
```

Figure 4.15 Algorithm to extract the first constituent

Suppose the input word is ዳቦ ቤቱ dabo-bet-u 'the/his bakery'. The algorithm takes the input word as it is as input, and searches in the lexicon of nouns and verbal nouns. If there is a match, it will return the match as the first constituent and its POS, otherwise it removes the final character, which will be - **u**, and continues searching for a match. This will continue until the extractor finds a match from the lexicon. In this case, the extractor gets a match, ዳቦ dabo 'bread', in the fourth iteration as shown below.

Iteration	X	Is x in nouns or verbs list?	Remove	Left	Status
Start	dabobetu	No	u	dabobet	Continue
1	dabobet	No	t	dabobe	Continue
2	dabobe	No	e	dabob	Continue
3	dabob	No	b	dabo	Continue
4	dabo	Yes	-	dabo	Break and return x

Table 4.5 First constituent extractor algorithm demo

At this time, the algorithm stops searching after reporting the match, dabo, and its POS, noun. However, what if there is a prefix attached to the input word such as ከዳቦ ቤቱ kä-dabo-bet-u 'from the/his bakery'? Well, the algorithm removes the prefix and continues searching again, process number 11. When the preposition, ከ kä 'from', is removed, the compound, ዳቦ ቤቱ dabo-bet-u, is left and then the above process (Table 4.5) continues and the match, dabo, is returned.

4.4.5. Extractor of the Second Constituent of the Compound

This algorithm accepts stem which are left after stripping the prefix (if any) and the first constituent from the given word. It will take the stem as it is and searches for a match, which will be taken as second constituent of the compound, in the list of nouns, adjectives, verbal nouns. If the algorithm finds a match, it will return the match and its POS. Otherwise, it will search again after stripping the last letter from the right assuming that the last letter may be a suffix since Amharic compounds add their suffixes at the right-hand side. The process will continue until a

match is found or no character is left. Finally, if there is no match found in the lexicon, the algorithm returns an empty string.

```
Input: stem
Output: the second constituent and its POS
Accept stem
1       nouns ← list of maintained nouns
2       adjs ← list of maintained adjectives
3       verbs ← list of maintained verbal nouns and verbs
4       c2 ← ''
5       pos ← ''
6       y ← stem
7       for i ← 0 to length[y]
8               if y in nouns or adjs or verbs list
9                       c2 ← y
10                      pos ← list name
11                      break
12              else
13                      y ← stem - i number of characters of the stem from final
14      return c2, pos
```

Figure 4.16 Algorithm to extract the second constituent

Let's take again the input word ዳቦ ቤቱ dabo-bet-u 'the/his bakery'. The algorithm starts by taking the stem, ቤቱ bet-u, because ዳቦ dabo 'bread' was extracted as a first constituent. Then it takes the stem as it is and starts searching to get a match for the stem in the lexicon. However, it cannot find any match since a suffix is attached to the stem. Then the process continues searching for a match after removing the last letter, **u**, and gets a match, ቤት bet 'house' as shown below.

Iteration	x	Is x in nouns or adjectives or verbs list?	Remove	Left	Status
Start	betu	No	u	bet	Continue
1	bet	No	-	-	Break and return x

Table 4.6 Second constituent extractor algorithm demo

Then the algorithm stops processing after returning the match and the POS.

4.4.6. Extractor of Derivational Morphemes

There are derived nouns such as ቤተሰብነት bet-ä-säb-nnät 'be a family' and adjectives such as ሥነ-መለኮታዊ sən-ä-mäläkot-awi 'theological' that are derived from compound nouns ቤተሰብ bet-ä-säb 'family' and ሥነ-መለኮት sən-ä-mäläkot 'theology', respectively. Derivational morphemes of Amharic (see chapter 2) such as -ነት nnät and -ኣዊ -awi are suffixed to the compound as shown below.

Compound	derivational morpheme	Result
ቤተሰብ bet-ä-säb	-ነት -nnät	ቤተሰብነት bet-ä-säb-nnät
ሥነ-መለኮት sən-ä-mäläkot	-ኣዊ -awi	ሥነ-መለኮታዊ sən-ä-mäläkot-awi
ቤተሰብ bet-ä-säb	-ኣዊነት -awi-nnät	ቤተሰባዊነት bet-ä-säb-awi-nnät

Table 4.7 Examples of derivational morphemes

To extract such morphemes, the following algorithm is designed:

```
Input: stem
Output: derivational morpheme
Accept stem
1       dms ← list of concatenated derivational morphemes
2       dm ← ''
3       for i ← 0 to length[dm]
4               x ← stem - i number of characters from final
5               if x in dms
6                       dm ← x
7                       break
8       return dm
```

Figure 4.17 Algorithm to extract derivational morphemes

Let us take the compound ቤተሰባዊነት bet-ä-säb-awi-nnät and demonstrate how the system works. The first constituent, ቤት bet 'house', the connector, ኧ ä, and the second constituent, ሰብ säb 'human being', are identified and extracted by the system and the stem ኣዊነት awinnät is left. The algorithm in Figure 4.14 accepts the stem ኣዊነት awinnät and searches for a match in a concatenated derivational morphemes list by taking the stem as it is. If the algorithm finds a match, it will return the match and stops; otherwise it will strip the final letter and continue searching. In this specific example, it finds a match, ኣዊነት awinnät.

4.4.7. Inflectional Suffixes

For words that have suffix, this algorithm can interpret the suffix type and its grammatical meaning. The algorithm accepts the suffix that is identified and extracted by the suffix extractor and it checks in the non-concatenated suffixes list. Based on the suffix type, the algorithm prints the suffix type and its grammatical meaning. The following is the algorithm:

```
Input: suffix
Output: print suffix type and grammatical feature
Accept concatenated suffix returned from suffix extractor
1       sfxs ← non-concatenated suffixes list
2       z ← ''
3       if suffix in sfxs
4           indx ← suffix index
5           z ← sfxs[indx + 1]
6           print suffix type and grammatical features of z
7           break
8       else
9           for i ← 0 to length[suffix] do
10              last ← the i number of letters of the suffix from final
11              frst ← the first two letter of the suffix
12              frst1 ← the first three letter of the suffix
13              if frst = 'oč' or frst1 = 'woč' or frst = 'at' or frst = 'it'
14                  ll ← suffix minus frst
15                  if length(suffix) = 2 or frst1 = 'woč'
16                      print suffix type and grammatical features of frst
17                      break
18                  else if ll in sfxs
19                      indx ← index of ll
20                      z ← sfxs[indx + 1]
21                      print suffix type and grammatical features of z
22                      break
23                  else if last = 'n'
24                      mid ← suffix - the first two and the final letter
25                      if mid in sfxs
26                          indx ← index of mid
27                          z ← sfxs[indx + 1]
28                          print suffix type and grammatical features of z
29                          break
30                      else
31                          print('Grammar: accusative')
32                          break
```

Figure 4.18 Algorithm to interpret inflectional morphemes

To demonstrate this algorithm, let us take the following words:

Compounds	Gloss	Extracted suffixes
አፈ-ጉባዔዎቻችንን af-ä-gubae-woč-ačn-n	mouth-connector-meeting-Pl-1Pl-ACC	wočačnn
ለፍቶአዳሪዎቹ läfto-adari-woč-u	hardly-spend the night-Pl-3MS/DEF	woču
ቤተክርስቲያኒቷ bet-ä-kərstiyan-it-wa	house-connector-christian-3FS-DEF	itwa
ቤተመንግስትን bet-ä-mängəst-n	house-connector-government-ACC	n

Table 4.8 Examples of grammatical features[16]

From the first word, the suffix **wočačnn** is extracted by suffix extractor and the above algorithm, first takes the suffix as it is and searches for a match in the non-concatenated suffixes list, process number 3. However, since the input suffix is not a single suffix, the algorithm cannot find a match. Then it continues searching for a match by taking the first two and three and the last letters from the concatenated suffixes, process number 9 to 12. At this point, the algorithm gets a match for **woč**, the first three letters, and continues searching using the rest of the characters, **ačnn**. Since **ačnn** is also in concatenated form, the algorithm will not find a match in the non-concatenated suffixes list. Then process number 23 is used and the algorithm finds a match for the last letter, **n**. Finally, using processes number 24 to 29, it finishes searching and prints the result.

For the rest of the suffixes in the table, the algorithm uses the following processes:

For the suffixes – **woču**:

In process number 3, no match for **woču**;

In process number 13, it finds a match for **woč**;

In process number 18, it finds a match for **u**;

In process number 21 and 22, it prints the result and stops

For the suffixes – **itwa**:

In process number 3, no match for **itwa**;

[16] Note: about the connector, ä, see chapter 3

In process number 13, it finds a match for **it**;

In process number 18, it finds a match for **wa**;

In process number 21 and 22, it prints the result and stops;

For the suffixes – **n**:

In process number 3, it finds a match and stops;

In process number 4-7, it prints the result;

4.5. Rules of the System

Human language at all levels is rule-governed. Every known language has systematic rules governing pronunciation, word formation, and grammatical construction (Akmajian et al, 2001). Rules of compounding in Amharic are discussed in the previous chapter (Section 3.5).

The analysis of Amharic compound involves presenting the stem, the compound and each of the constituents with their POS and interpreting affixes. The system, Amharic compound identifier and analyzer, uses the following rules, designed in this work, to produce the analysis.

Suppose:

- *I* is the preprocessed input word
- ***clen*** is the sum of the length of identified parts of a compound;
- ***wlen*** is the length of the input word;
- ***c1*** is the first constituent;
- *c is a connector;*
- ***c2*** is the second constituent;

4.5.1. Rules for Identifying Prefix

The initial letter of some compounds in Amharic is the same as some of the prepositions such as ከ kä 'from' in ከብት አርቢ käbt arbi 'farmer (livestock)', ለ lä 'to/for' in ለዛ ቢስ läza bis 'unlovely' and በ bä 'by' in በትረ መንግሥት bätr-ä-mängəst 'head of the government'. When the system analyzes a compound of these kinds, it extracts the initial of the compound as a prefix though it

is not the real prefix. To handle this and other issues related with prefix, the following rules are formulated:

Rule 1: *If the prefix is the same as the initial of c1 and the second letter of I,*
Accept the extracted prefix;

Rule 2: *If the prefix is the same as the initial of c1 but not the same as the second letter of I,*
Reject the extracted prefix;

Example 1: Suppose the input word is ክክርሠምድር kä-kärs-ä-mədr 'from bottom of earth' and it has the prefix ክ kä 'from' which is the same as the initial letter of *c1* and the second character of *I*. Thus, the system uses rule 1 and takes the prefix as the real prefix.

Example 2: Suppose the input word is ክርሠ ምድር kärs-ä-mədr 'bottom of earth' and the ክ kä 'from' taken as a prefix, but it is not. The second letter of *I*, ር = r, is not the same as the prefix and also if the system searches for the first constituent by taking the stem that is found by removing the extracted prefix from the input word, it finds nothing since there is no word or word stem that begins with ርሠ = rs that is included in the lexicon. Even if it finds one, it is not the correct *c1* and it may lead to the wrong analysis or the input word will not recognize as a compound. As far as the researcher knows, there is no valid word that starts with ርሠ rs in Amharic. So, the system uses rule 2 and rejects the extracted prefix.

4.5.2. Rules for Checking Connector

A compound in Amharic may use a connector or linker to connect its constituents (see Chapter 3, Sub section 3.2.3) and it is extracted by the system after the extraction *c1*. However, in some compounds, because the initial of *c2* is the same as one of the connectors, it is taken wrongly as *c*. To protect this and other problems that are related with the connector, the following rules are formulated.

Rule 3: *If c is the same as the initial of c2 and wlen is not equal to clen,*
Reject the connector;

Rule 4: *If c is* ***bä*** *or* ***lä*** *and c2 is empty,*
Add 'ə' to the stem minus c and search for c2

If c2 is there and same as c1,

Accept the extracted c

Else

Reject the extracted c;

Rule 5: *If c is **bä** or **lä** and the first constituent is not the same as the second constituent,*

Reject the connector;

Example 3: Suppose the input word is ቀኝአዝማች qäN-azmac 'a general officer of the right wing'. The system uses the same stem in extracting the connector and in searching *c2,* i.e. አዝማች azmac, and it extracts:

c1 = ቀኝ qäN 'right side'

c = 'a'

c2 = አዝማች azmac 'the one who commands the army'

Because the initial *c2* is the same as one of the connector in Amharic, ኣ a, the system extracts ኣ a as a connector. When the system uses rule 3, it finds the initial of *c2* is wrongly taken as a connector.

$$[wlen = 8] \quad \neq \quad [clen = 9]$$

The length of the input word, which is equal to 8 characters, is **not equal to** the length of the sum of extracted parts of the compound (i.e. the first constituent, 3 characters, the connector, 1 character, and the second constituent, 5 characters, which is equal to 9).

Even if the system uses the stem after extracting the connector, i.e. ዝማች zmac, and search for *c2*, it finds nothing or it finds other word or word stem because the initial is taken. Thus, the system uses rule 3 and rejects the extracted connector.

Rule 4 and 5 uses to control compound adverbs (see Chapter 3, Sub section 3.5.4) because these compounds have the following pattern:

c1 + *c* + *c2*; where c is either ***bä or lä*** and c1 is the same as c2

So, when the system identifies a connector ***bä or lä***, it will check whether *c2* is the same as *c1*, if not it rejects *c*.

Example 4: Suppose the input word is ኣድላይ əj-lä-j 'hand to hand' and the system identifies *c1*, ኣይ əj 'hand', and *c*, ለ lä 'to/for', but not *c2*, since there is no word or word stem of ለይ läj in the lexicon of the system. So, the system using rule 4, it will add *'ə'* to the stem minus *c*, ይ j, and searches for *c2* and then it finds ኣይ əj, which is the same as *c1*. Thus, the connector ለ lä is accepted.

Example 5: Suppose the input word is ቡና-ለቃሚ buna-läqami 'coffee collector' and the stem ለቃሚ läqami 'collector' is left when c1, ቡና buna 'coffee', is extracted. This stem is used when the system searches for *c2*; simultaneously the connector is extracted from this stem. The extracted connector, ለ lä, is not the real connector because if two words or word stems are connected by ***bä or lä*** to form a compound, then they have to be the same. Thus, the connector will be rejected.

Because the system uses the same stem, for extracting both the connector and the second constituent, and uses rule 5, it minimizes producing of wrong analysis as ቡና-ለ-ቃሚ buna-lä-qami:

c1 = ቡና buna 'coffee'
c = ለ lä
c2 = ቃሚ qami 'addicted'

The system uses the same stem in extracting the connector and in searching *c2* but it will check whether the initial of *c2* is taken wrongly as a connector or wrong word is extracted by not removing the connector from the stem. In both ways, extracting *c2* with the stem that includes the connector or excludes it, the system may extract wrong word. However, it rechecks to avoid wrong extraction.

4.5.3. Rules for Checking Suffix

In relation to extracting the second constituent of the compound, the difficulty may be with wrongly taken connector, as discussed above, and with wrongly extracted suffixes. The following rules are formulated, to solve the latter:

Rule 6: *If the extracted suffix is the same as c2 final and the letter second to final of I,*

Accept the extracted suffix

Else

Reject the extracted suffix

Rule 7: *If the initial of extracted suffix and final of c2 have a match,*

If the unmatched letter(s) is a valid suffix,

Accept the unmatched suffix

Else

Accept the extracted suffix

Example 6: In the input word is ወንደላጤ wänd-ä-laTe 'bachelor', the system takes the final letter, ኤ e, wrongly as suffix. In Amharic, there is a suffix ኤ e that indicates first person singular (1Sg). Using rule 6, the extracted suffix, ኤ e, is the same as the final letter of second constituent but not the same as the letter second to final of the input word, ጥ T. Therefore, the system rejects the extracted suffix.

Example 7: Suppose the input word is ቀኘአዝማቹ qäN-azmac-u and it has a suffix ኡ u, which represents third person masculine singular (3MS) or the definite article 'the'. Extracting suffixes start by taking the stem and it searches for the longest suffix. In this example, the stem is አዝማቹ azmacu and when the system searches for suffixes in the concatenated suffixes list, it will find **acu**, which represents second person plural (2Pl). Then the system compares the final letters of the second constituent with the initial letters of the extracted concatenated suffix as shown below.

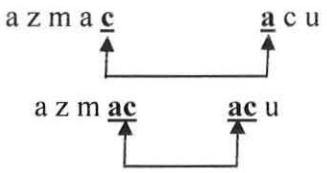


Figure 5.6 Compare second constituent and suffixes

The system finds a match in the second attempt, **ac** with **ac**, and takes the rest of the suffix, **u**, and searches for a match in the concatenated suffixes list. The system finds a match and accepts ኡ **u** as correct suffix.

4.5.4. Rules for Checking the Second Constituent

In Amharic compounds, if the first constituent ends with the same letter as that of the initial letter of the second constituent, it is possible to use only one of the two letters. In this case, since the system uses letters for searching constituents, it could not find the second constituent. For example, the compound ግራዝማች grazmač (gr**a**-**a**zmač) 'a general officer of the left front' is used '**a**' for both constituents: ግራ gra 'left' and አዝማች azmač 'the commander of the left front'. The following rule is designed to solve this problem:

Rule 8: *If c1 ends with 'a' and c2 is not found,*
Add 'a ' at the initial of c2 and search for c2

The first constituent is already identified as ending in **-a**, the system will search using this rule by inserting '**a**' at the initial position of the second constituent, i.e. **a+zmač**. Then it finds the second constituent, **azmač**.

4.5.5. Rules for Checking before Display the Analysis

The system, when it finishes the process, display the input word with the analysis, if the input word identified as a compound, or the input word with a statement declaring that the input word either it is simple word or it cannot be analyzed. To produce the analysis as a compound, the system uses the following rules:

Rule 9: *If c1 and c2 identified with their POS and clen is equal to wlen, then produce the analysis*;

Rule 10: *If c1 ends with 'a' and c2 starts with 'a', then wlen = clen – 1*;

The system can wrongly identify two constituents from the input word with their POS; however before displaying the result as a compound, it will check each letters of the input word is included in the analysis. To do the checking, the system uses the equality of the length of the input word and the sum of the length of all identified parts of the input word. If they are equal, it assumes the input word is a compound; otherwise it will go to rule 10, which is designed following rule 8.

Suppose the input word is ግራዝማች grazmač (see Rule 8) and to display the final result, it uses rule 10, the length of a given compound is one less than its constituents as grazmač (= 7) is less than the sum of its parts: length of *c1* + length of c2 = 8

Concluding the chapter, first it is discussed the maintained lexicon, which is stored in different text files that are used to show the POS of the lexicon. Prepositions and suffixes are also collected and stored. Different algorithms and rules designed for this research were presented and discussed with examples and dcmonstrations. The next chapter presents the implementation, the results and evaluation of the experiment and the test.

CHAPTER FIVE

Implementation, Experimental Results and Evaluation

5.1. Introduction

This chapter describes the implementation of Amharic compound analyzer. The implementation is based on the Amharic compound rules and computational rules, which are identified by the researcher. The algorithms developed for this thesis are implemented using Python. The system accepts a word, a raw text or list of words as input using the Amharic orthography and produces analyzed output, in the transliterated form, if the word is a compound. Otherwise, the system returns the word as it is with a statement declaring that the input word either it is simple word or it cannot be analyzed.

Next to discussion of implementation, experimental results are described with examples of the results. Finally, evaluation of the system is discussed.

5.2. Implementation

The system is implemented by building a Python module. To analyze compound words, checking whether the input word consists of more than one word or not, is the beginning. In other words, first it will check whether the input word contains more than one word (constituent) or not. If the input word does not contain more than one constituent, the system treats the input word as single word. If the input word has more than one word (constituent), the next step is checking the word, whether it is a compound or not, because there are words that have two components that are not compounds such as ክስክስ kəs-kəs 'related with brake'. These and other forms were considered in the implementation process.

The module implements the rules and algorithms discussed in the preceding chapters. There are seven functions or methods in the module, each handles a specific problem.

5.2.1. How the System Works

The following diagram shows the process flow of the system

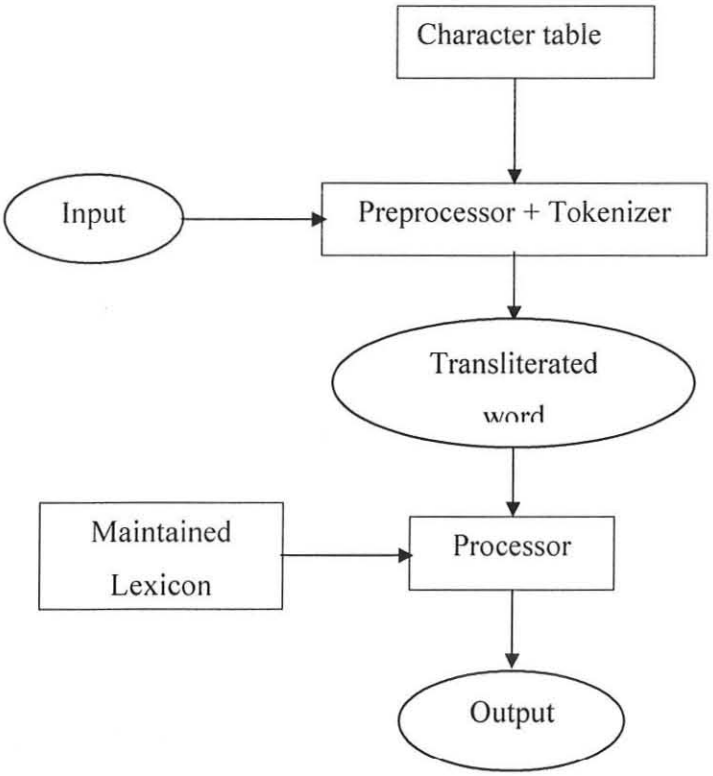


Figure 5.1 the process flow of the system

The system accepts inputs from the keyboard or file in the Ge'ez script. The input can be a word, list of words, or raw texts (i.e. text written in Amharic letters and can contain punctuations, English letters, numbers, etc). The system identifies raw texts based on the number of words in a line, if the input has more than two words in a line, then the system accepts as raw text. Otherwise, the system takes the input as a word though it contains words separated by space because it assumes the two words are constituents of a compound in open form, and then it starts the process.

Step 1: Preprocess

The same preprocess is applied for the input as it is applied in maintaining the lexicon of the system. The input may contain inappropriate features such as punctuations, numbers and English letters. The preprocessor removes these features and also performs the transliteration. The following text is prepared as an example of raw text input and includes numbers, 1 and 2, punctuations, "('፣!)", and English letters, words, to show the capability of the preprocessor. The preprocessor removes these characters and transliterate the Amharic letters as shown below:

- 'I' *represents the input text and* 'R' *represents the results of the preprocessing*

I: 1) 'እግረ ቀጭን እንደ ሰሳ' ፣ 2) 'ልበ ሙሉ እንደ አንበሳ' are proverbs; "አቅመ ቢስ!" and "ቀልበ ቢስ!" are compounds

R: əgrä qäCn əndä säsa ləbä mulu əndä anbsa aqmä bis qälbä bis

Step 2: Tokenize

When the input is a raw text or wordlist (as mentioned above the system accepts the input as raw text when the number of words in a line is more than two), tokenization is applied (see Chapter 4, Sub section 4.4.1). In the case of raw text, the tokenization uses the tokenizer algorithm otherwise it tokenizes based on line, each line of the wordlist accepts as an input word whether it has words separated by space or not. Suppose the input text is the above raw text, step 1, the tokenizer takes the transliterated text and tokenizes, first in word level and then based on the rules set in the tokenizer algorithm. So the wordlist, prepared by the tokenizer, includes not only simple words but also words such as əgrä qäCn.

Step 3: Analyze

In this step, the system takes a word or each word from the wordlist, which is the output of step 2, and recognizes each part of a word and decides whether the word is a compound or not based on the recognition result. If the input word contains two words or word stems and fulfills the criteria to be a compound (discussed in chapter 3; section 3.2), then it will be accepted as a compound and analyzed; otherwise it will not be accepted.

Step 4: Display the analysis

The system will display the analysis if the input word is a compound, it will check whether the input word has constituent words with their POS, and if there is no character left.

5.2.2. Examples of the System Output

As discussed in chapter 3 (Section 3.5), Amharic has compound nouns, adjectives, verbs, and adverbs. Each of them can be formed by combining different lexical categories. In this sub section, compounds of each type are presented as an example and show the results of the system.

Example 1: suppose the system accepts ቤተክርስቲያኗ bet-ä-kərstiyan-wa 'The Church', a compound noun that has a connector, compound nouns rule 1[17], and the system produces:

Word: ቤተክርስቲያኗ
Compound: bet-ä-kərstiyan POS*: noun*
First element: bet POS*: noun*
Connector: ä
Second element: kərstiyan POS*: noun*
Possessor: 3, Sing, Fem
Function: definite

Example 2: suppose the system accepts ትምህርት ቤቶቻችን təmhərt-bet-oč-ačn, a compound noun without a connector, compound nouns rule 2, and the system produces:

Word: ትምህርት ቤቶቻችን
Compound: təmhərt-bet POS*: noun*
First element: təmhərt POS*: noun*
Second element: bet POS*: noun*
Possessor: 1, Pl
Function: definite

[17] See compounding rules in Chapter 3, Section 3.5

Example 3: suppose the system accepts ወዝ አደሩን wäz-addär-u-n, a compound noun without a connector, compound nouns rule 3, and the system produces:

Word: ወዝ አደሩን
Compound: wäz-addär POS*: noun*
First element: wäz POS*: noun*
Second element: addär POS*: verb*
Possessor: 3, Sing, Masc
Function: definite, accusative

Example 4: suppose the system accepts ወዶገቦች wädo-gäb-očč, a compound noun without a connector, compound nouns rule 4, and the system produces:

Word: ወዶገቦች
Compound: wädo-gäb POS*: noun*
First element: wädo POS*: verb*
Second element: gäb POS*: verbal noun*
Function: plural

Example 5: suppose the system accepts ለፍቶ አዳሪነቱን läfto-adari-nnät-u-n, a compound noun without a connector, compound nouns rule 5, and the system produces:

Word: ለፍቶ አዳሪነት
Stem: läfto-adari-nnät POS*: noun*
Compound: läfto-adari POS*: noun*
First element: läfto POS*: verb*
Second element: adari POS*: noun*
Possessor: 3, Sing, Masc
Function: definite, accusative

Example 6: suppose the system accepts መልከ መልካሞቹ mälk-ä-mälkam-očč-u, a compound adjective with a connector and the system produces:

Word: መልክ መልካሞቹ

Compound: mälk-ä-mälkam POS*: adjective*

First element: mälk POS*: noun*

Connector: ä

Second element: mälkam POS*: adjective*

Possessor: 3, Sing, Masc

Function: plural, definite

5.3. Results and Evaluation

The effectiveness of the developed system is measured by testing it using test set. The system is effective when it:

- Recognizes compound words in Amharic text;
- Separates constituents of a given Amharic compound and assign the correct POS to each of the constituents;
- Interpret inflectional morphemes, which show grammatical features of a given compound, correctly;

A corpus that contains 500 words is prepared based on the criteria of compounding in Amharic discussed in chapter 3. Of these words, 100 compounds are collected manually from books of Baye (2000 E.C) and Getahun (1990 E.C) and the rest of the corpus, the 400 words, is selected automatically from the Amharic dictionaries of Desta (1970 E.C) and Tessema (1959 E.C) and from wordlists[18] of Amharic single names.

The corpus includes compounds of nouns, adjectives, verbs and adverbs with all inflectional affixes of number, gender, case, and definiteness. It also includes simple words that are selected randomly from the wordlist and the two dictionaries. The selection of words, though it is automatic, uses judgmental sampling method to assure the representativeness of compound word types.

[18] These dictionaries and wordlists are available at http://nlp.amharic.org/resources/lexical/word-lists/.

The corpus is partitioned into the development set and test set on the basis of percentage of 70% and 30%, respectively.

5.3.1. The Development Test

A development test set is a test set that is being used during the cycle of system development and improvement (Resnik and Lin, 2010). Using the development set, the system is tested several times until the performance of the system was acceptable. Some of the corrections made when the development set is carried out are presented as follows.

- The system was not differentiating simple words and compounds. Though the system produces analysis for compounds correctly, it also displays analysis for simple words. This was corrected by developing a rule, which says

  Unless getting two constituents of the given word, don't produce analysis;

- When the second constituent of the compound is a derived noun (derived from other nouns by suffixing a derivational morpheme such as -ነት -nnät (chapter 2) or a derived adjectives like those derived from nouns by adding -ኣዊ -awi, the system only identifies the word or stem before its derivation. Because there is no derivational morpheme list in the maintained lexicon. To solve this problem, an algorithm is developed (Chapter 4, Sub section 4.4.6).
- Over generation is another problem identified during the development of the system. The following words, for instance, are accepted in the language as simple words but the system has identified them as compounds.

መልካሳ mälkasa 'geographic name'	mälk 'face' + asa 'fish'
ሰብስቤ säbsəbe 'name of a person'	säb 'human being' + səb 'fat'
አይጠገብ ayəTägäb 'lovely'	ayəT 'rat' + ä + gäb 'related to entry'
አጥላባቸው aTlabačäw 'name of a person'	aT 'related to loss' + lab 'facial' + 3PlDEF
ወንድማማች wändəmamač 'brothers'	wändəm 'brother' + amač 'son-in-law'
ደናግል dänagəl 'virgins'	dän 'forest' + a + gəl 'personal'
ፈርካሳ färkasa 'breakable'	fär 'in order/line' + kasa 'claim/name of a person'

Table 5.1 Some of Amharic simple words identified as compounds

The over generation was solved by adding these simple words to the built-in lexicon. Thus, the system identify each of them as a simple word not as a compound.

These and other errors that were identified during the development are corrected by designing rules (see Chapter, Section 4.5) and the final result is the following:

	Unanalyzed	Analyzed
Simple words	37	7
Compound words	8	298

Table 5.2 Results of the analyzer on the development

The accuracy of the analyzer is measured on the basis of correct predictions. Simple words are supposed to be unanalyzed and compound words are supposed to be analyzed.

$$\text{Accuracy} = \frac{\text{Unanalyzed simple words} + \text{Correctly analyzed compound words}}{\text{Total number of words}}$$

$$\text{Accuracy} = \frac{37 + 298}{350} = 95.7\%$$

The true positive rate (or recall) of the analyzer is the proportion of compound words that are correctly analyzed.

$$\text{Recall} = \frac{\text{Analyzed compound words}}{\text{Unanalyzed compound words} + \text{Analyzed compound words}}$$

$$\text{Recall} = \frac{298}{8 + 298} = 97.38\%$$

Precision of the analyzer is the proportion of analyzed compound words over the total analyzed words.

$$\text{Precision} = \frac{\text{Analyzed compound words}}{\text{Analyzed simple words} + \text{Analyzed compound words}}$$

$$\text{Precision} = \frac{298}{7 + 298} = 97.7\%$$

```
yohannes@yohannes-Satellite-A100:~/Morphology/thesis_development$ python3
Menu:
        1. for keyboard input
        2. for file input
 2

        1. wordlist
        2. raw-text
1
Please type your source and destination file names separated by comma:
dev_test.txt,rdev.txt
Total number of words:  350
Successfully analyzed number of words:  305
Unanalyzed number of word(s):  45
```

Figure 5.2 Screen shot of the analyzer on the development

5.3.2. Error Analysis

Two kinds of errors are found, simple words analyzed as compound words (false positive) and unanalyzed compound words (false negative). False positive errors are occurred because of overganeration and it can be solved, as discussed in the above sub section (sub section 5.3.1), by adding the analyzed simple words to the maintainied lexicon. Thus, the system identify each of them as a constituent not as a compound.

In the case of false negative errors, some of the compound words are unanalyzed because of missing entries and lack of rules. It can be solved in the same way as done for false positive and by adding and modifying rules. For instance, one of the unanalyzed compound words is እግረቀጫጭን əgər-ä-qäCCaCCn 'those who have thin legs'. The system can identify the first constituent እግር əgər and the connector ኧ ä, but not the second constituent, because the rest of the word ቀጫጭን qäCCaCCn 'thins' is in the plural form. The word ቀጭን qäCCn 'thin' is stored in the maintained lexicon of adjectives.

As discussed in chapter 3, adjectives in Amharic follow two ways to show them plural number: one is like nouns, adding the morpheme **ኦች očč** and the other is reduplicating the penult consonant and adding the vowel '**a**' as a connector. However, the sytem has no knowledge about the other way (the latter one) of pluralizing adjectives. As mentioned above, adding the word with the plural marker in the maintained lexicon of the system can help the system to analyze the compound እግረቀጫጭን əgər-ä-qäCCaCCn.

5.3.3. The Test Set

The test set, as described in the previous section, contains 150 words selected randomly from the designed corpus. Of these, 137 words are compounds and the remaining 13 words are simple words. The following table summarizes the result of the test.

	Unanalyzed	Analyzed
Simple words	13	0
Compound words	2	135

Table 5.3 Results of the analyzer on the test set

The accuracy of the analyzer is measured on the basis of correct predictions. Simple words are supposed to be unanalyzed and compound words are supposed to be analyzed.

$$\text{Accuracy} = \frac{\text{Unanalyzed simple words} + \text{Analyzed compound words}}{\text{Total number of words}}$$

$$\text{Accuracy} = \frac{13 + 135}{150} = 98.67\%$$

The true positive rate (or recall) of the analyzer is the proportion of compound words that are correctly analyzed.

$$\text{Recall} = \frac{\text{Analyzed compound words}}{\text{Unanalyzed compound words} + \text{Analyzed compound words}}$$

$$\text{Recall} = \frac{135}{2 + 135} = 98.5\%$$

Precision of the analyzer is the proportion of analyzed compound words over the total analyzed words.

$$\text{Precision} = \frac{\text{Analyzed compound words}}{\text{Analyzed simple words} + \text{Analyzed compound words}}$$

$$\text{Precision} = \frac{135}{0 + 135} = 100\%$$

```
yohannes@yohannes-Satellite-A100:~/Morphology/thesis_development$ pythor
Menu:
        1. for keyboard input
        2. for file input
 2

        1. wordlist
        2. raw-text
1
Please type your source and destination file names separated by comma:
test_set.txt,rtest.txt
Total number of words:  150
Successfully analyzed number of words:  135
Unanalyzed number of word(s):  15
```

Figure 5.3 Screen shot of the analyzer on the test set

As shown from the table, the accuracy of the system for the test set is 100%. To achieve such accuracy, as indicated above, the system was tested repeatedly and corrected whenever errors occurred.

5.3.4. Raw Text Test

As discussed in this chapter (Section 5.2.1), the system can accept a raw text as input. In this test, it is expected to test the performance of the system in relation to recognizing and analyzing compounds. The following raw text, which is prepared as an exercise to identify compound nouns, adjectives, and verbs, is taken from Baye's (2002 E.C) book for testing the system.

"ምነው ልጄ ፣ አንቺን የመሰለ ቆንጆ ይስቃል ፣ ይጫወታል እንጂ ፣ እንዴት ሆደ ባሻ ይመስል ዝም ብሎ መንገድ ላይ ያለቅሳል?" ሲሉ ይጠይቋታል። እሷም ፣ "አባቴ የማለቅሰው ዝም ብየ ሳይሆን ፣ 'እግረ ቀጭን እንደ ሰሳ ፣ ልበ ሙሉ እንደ አንበሳ' እያለ" ጧት ማታ ራሱን የሚያሞካሽ ፣ እኔን ሲሻው "አቅመ ቢስ!" ሲሻው "ቀልበ ቢስ!" እያለ የሚያናንቅና ተስፋ የሚያስቆርጥ ባል አጋጥሞኝ ሆድ ቢብሰኝ ነው" ትላቸዋለች።

Figure 5.4 Raw text used as test

The system recognizes 8 compounds out of 9, which were identified by an expert of the language, from the raw text (Figure 5.4). From the recognized 8 compounds, the system can analyze 6 compounds. The following table shows the result of the test:

	Compounds	POS	Identified	Analyzed
1	ሆደ ባሻ hod-ä-bašša	Noun/Adjective	✓	✓
2	ዝም ብሎ zəm-bəlo	Adverb	✓	×
3	ዝም ብየ zəm-bəyyä	Noun	✓	×
4	እግረ ቀጭን əgər-ä-qäCCn	Noun/Adjective	✓	✓
5	ልበ ሙሉ ləb-ä-mulu	Noun/Adjective	✓	✓
6	ጧት ማታ Təwat-mata	Adjective	✓	✓
7	አቅመ ቢስ aqəm-ä-bis	Adjective	✓	✓
8	ቀልበ ቢስ qälb-ä-bis	Adjective	✓	✓
9	ሆድ ቢብሰኝ hod-bibsäN	Adjective	×	

Table 5.4 Results of the identifier and analyzer of compounds on the raw text (training)

The compound ሆድ ቢብሰኝ hod-bibsäN is not identified because the second constituent of the compound, ቢብስ bibs, is not exist in the maintained lexicon of the system. In the case of ዝም ብሎ zəm-bəlo and ዝም ብየ zəm-bəyä, the system identified as compounds by looking up the maintained look up list of possible second constituents of a compound. However, the system could not analyze and give any information about grammatical features for the constituents of the compound because no information was maintained in the lexicon of the system related to these compounds.

The next chapter is the final and it will discuss the conclusions and recommendations of the research.

CHAPTER SIX

Conclusions and Recommendations

6.1. Conclusions

This thesis has presented the system for morphology analysis of Amharic compounds. Its primary goal was to design and develop a system that identifies and analyze Amharic compound words automatically. In achieving this goal, a system is developed using rule-based approach on the basis of two-level morphology.

Although a number of researchers attempted to develop morphological analyzer for Amharic, nothing is done regarding compound words or compounding. The system, developed in this work, can identify and extract Amharic compounds in all the specified three forms – closed as ቀኛዝማች qäññazmač, open as ሕገ መንግሥት həgä mängəst and hyphenated as ሥነ-ሥርዓት sən-ä-sərəat - from a raw text (or wordlist or dictionary) and can do morphological analysis for compounds of Amharic.

Particularly, the study answered questions on identifying compound words automatically and analyzing them. In identifying compounds from raw text, the system applies the following criteria and takes words that fulfill these criteria for further processing:

- Words end with the vowel **ä**, except few functional words;
- A word, which has a match in the predefined list of possible second constituent words, with the previous word;

The study covers all compound categories in Amharic (i.e. compound nouns, adjectives, verbs, and adverbs) with their grammatical and syntactical information. The grammatical features included in this work are number, gender, person, case, and definiteness. When the system recognizes a compound word, it will give syntactic information for each constituents and grammatical information for the compound as a word.

In identifying and analyzing compound nouns, adjectives, and adverbs, the system performs well and the sample used to the development and test set can be considered as representative of Amharic compounds. However regarding compound verbs, it covered only the main verbs, verb to 'say' and to 'do', and some of their variations, not all.

Some words that are accepted as simple words in the language were taken as compounds by the system (over generation). In this study, this over generation problem was controlled by adding these kinds of words in the maintained lexicon of the system.

At this level, the system can be used as a tool for identifying compounds from corpus and for morphology analysis of compound words of Amharic.

6.2. Recommendations

The system developed in this study is recognized and analyzed only compound words. Further study is required to integrate the developed system with other Amharic morphological analyzer tool, particularly with HornMorpho, can give a very good morphological analyzer for Amharic.

The developed system is more focused on compound nouns, adjectives, and adverbs than compound verbs. It only identifies and analyzes the basics of compound verbs. Adding more verbs of 'to say' and 'to do' in the maintained lexicon and designing more rules regarding to compound verbs will increase the performance of the developed system.

Some words that are accepted as simple words in the language were taken as compounds by the system (over generation). In this study, this over generation problem was controlled by adding these kinds of words in the maintained lexicon of the system. However, it needs further study, to identify whether these words are simple or not.

References

Abiyot Bayou. 2000. *Developing automatic word parser for Amharic verbs and their derivation.* Master's thesis, Addis Ababa University.

Atelach Alemu. 2002. *Automatic Sentence Parsing for Amharic Text: An Experiment Using Probabilistic Context Free Grammars*, Master's Thesis. Addis Ababa University.

Akmajian, A., Demers, R.A, Farmer, A.K., and Harnish, R.M.. 2001. *Linguistics: An Introduction To Language and Communication*, 5th Edition. The MIT Press, Cambridge, Massachusetts, London, England.

Antony et al. 2012. *Computational Morphology and Natural Language Parsing for Indian Languages: A Literature Survey.* International Journal of Computer Science & Engineering Technology (IJCSET), Vol. 3 No. 4

Alpaydın, E. 2010. *Introduction to Machine Learning.* Second Edition. The MIT Press Cambridge, Massachusetts London, England.

Baye Yimam, 1995. የአማርኛ ሰዋስው yäamarəñña säwäsəw; E.M.P.D.A, Addis Ababa.

Baye Yimam, 2008. የአማርኛ ሰዋስው yäamarəñña säwäsəw; E.M.P.D.A, Addis Ababa.

Booij, G. 2009. *Morphological Analysis.* University of Leiden. http://www.leidenuniv.nl/letteren/booijge/pdf/Morphological%20analysis.pdf

Dressler, W.U. 2006. *The Representation and Processing of Compound Words: Compound Types.* Oxford University Press Inc., New York, United States.

Fromkin, V., Rodman, R. and Hyams, N. 2009. *An Introduction to Language*, Ninth Edition. Wadsworth. Boston, USA

Gasser, M. (2011). *HornMorpho: a system for morphological processing of Amharic, Oromo, and Tigrinya.* Conference on Human Language Technology for Development, Alexandria, Egypt.

Gambäck, B. and Asker, L. 2010. *Experiences with Developing Language Processing Tools and Corpora for Amharic.*

Getahun Amare. 1997. Zämänawi yamarNa Säwasäw bäqälal aqäraräb. Commercial Printing Press, Addis Ababa.

Goldsmith, J. 2001a. *Unsupervised Learning of the Morphology of a Natural Language.* Computational Linguistics also available at http://humanities.uchicago.edu/faculty/goldsmith.Lingustica2000/Paper/Paper.html

Kavčič, A.. 2004. Grammar Seminar Paper, *Compounding.* University of Ljubljana

Kazakov, D. and Manandhar, S. 2001. *Unsupervised Learning of Word Segmentation Rules with Genetic Algorithms and Inductive Logic Programming.* Kluwer Academic Publishers. Manufactured in The Netherlands.

Kumar, A.M, Dhanalakshmi V, Soman K.P, and Rajendran S. 2010. *A Sequence Labeling Approach to Morphological Analyzer for Tamil Language.* International Journal on Computer Science and Engineering, Vol. 02, No. 06.

Köprü, S. and Miller, J. 2009. *A Unification Based Approach to the Morphological Analysis and Generation of Arabic.* AppTek, Inc.

Leslau, W. 1969. *An Amharic Reference Grammar.* University of California, Los Angles.

Lieber, R, 2010. *Introducing Morphology.* Cambridge University Press, Cambridge, United Kingdom.

Lieber, R. and Štekauer, P. 2011. *The Oxford Handbook of Compounding.* Oxford University Press.

Lieber, R. and Štekauer, P. 2010. *Hand Book of Compounding*

Marsi, E., Bosch, A. and Soudi, A. 2005. *Memory-based morphological analysis generation and part-of-speech tagging of Arabic.*

Plag, I. 2002. *Word-formation in English*. Cambridge University Press

Roark, B. and Sproat, R. 2007. *Computational Approaches to Morphology and Syntax*. Oxford University Press Inc., New York

Resnik, P. and Lin, J. 2010. *The Handbook of Computational Linguistics and Natural Language Processing*.

Saba Amsalu and Gibbon, D. 2005b. *Finite state morphology of Amharic*. In Proceedings of International Conference on Recent Advances in Natural Language Processing. Borovets, Bulgaria.

Saba Amsalu. 2007. *Bilingual Word and Chunk Alignment*: A Hybrid System for Amharic and English . PhD thesis, Universität Bielefeld.

Selvam, M. and Natarajan, A.M. 2009. *Improvement of Rule Based Morphological Analysis and POS Tagging in Tamil Language via Projection and Induction Techniques*. International Journal of Computers, Issue 4, Volume 3, 2009

Shaalan, K. 2010. *Rule-based Approach in Arabic Natural Language Processing*. International Journal on Information and Communication Technologies, Vol. 3, No. 3.

Tesfaye Bayu, 2002. *Automatic morphological analyzer for Amharic: An experiment employing unsupervised learning and auto segmental analysis approaches*. Master's thesis, Addis Ababa University.

Trost, H. 2003. *Morphology*, Oxford Handbook of Computational Linguistics

Wehmeier, S., McIntosh, C., Turnbull, J., & Ashby, M. 2005. *Oxford advanced learner's dictionary of current English*. Oxford: Oxford University Press.

Wondwossen Mulugeta and Gasser, M. 2012. *Learning Morphological Rules for Amharic Verbs Using Inductive Logic Programming*. Language Technology for Normalization of Less-Resourced Languages.

Yitna Firdyiwek and Yaqob, D. 1997. *The System for Ethiopic Representation in ASCII.* Also available at <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.3191>

Appendices

Appendix 1. List of character table

ሀ	hä	ገ	gä	ዉ	wu	ቺ	či
ሀ	hä	ጠ	Tä	ዑ	u	ኺ	hi
ለ	lä	ጨ	Cä	ዙ	zu	ኒ	ni
ሐ	hä	ጰ	Pä	ዠ	Zu	ኺ	Ni
መ	mä	ጸ	S`ä	ዩ	yu	ኢ	i
ሠ	sä	ፀ	S`ä	ዱ	du	ኪ	ki
ረ	rä	ፈ	fä	ጁ	ju	ኺ	hi
ሰ	sä	ፐ	pä	ጉ	gu	ዊ	wi
ሸ	šä	ሁ	hu	ጡ	Tu	ዒ	i
ቀ	qä	ሉ	lu	ጩ	Cu	ዚ	zi
በ	bä	ሑ	hu	ጱ	Pu	ዠ	Zi
ቨ	vä	ሙ	mu	ጹ	S`u	ዪ	yi
ተ	tä	ሡ	su	ፁ	S`u	ዲ	di
ቸ	čä	ሩ	ru	ፉ	fu	ጂ	ji
ኀ	hä	ሱ	su	ፑ	pu	ጊ	gi
ነ	nä	ሹ	šu	ሂ	hi	ጢ	Ti
ኘ	Nä	ቁ	qu	ሊ	li	ጪ	Ci
አ	a	ቡ	bu	ሒ	hi	ጲ	Pi
ከ	kä	ቩ	vu	ሚ	mi	ጺ	S`i
ኸ	hä	ቱ	tu	ሢ	si	ፂ	S`i
ወ	wä	ቹ	ču	ሪ	ri	ፊ	fi
ዐ	a	ኁ	hu	ሲ	si	ፒ	pi
ዘ	zä	ኑ	nu	ሺ	ši	ሃ	ha
ዠ	Zä	ኙ	Nu	ቂ	qi	ላ	la
የ	yä	ኡ	u	ቢ	bi	ሓ	ha
ደ	dä	ኩ	ku	ቪ	vi	ማ	ma
ጀ	jä	ኹ	hu	ቲ	ti	ሣ	sa

ራ	ra	ሄ	he	ጰ	Pe	ድ	d
ሳ	sa	ሌ	le	ጼ	S`e	ጅ	j
ሻ	ša	ሐ	he	ፄ	S`e	ግ	g
ቃ	qa	መ	me	ፌ	fe	ጥ	T
ባ	ba	ሜ	se	ፔ	pe	ጬ	C
ቫ	va	ሬ	re	ህ	h	ጵ	P
ታ	ta	ሴ	se	ል	l	ጽ	S`
ቻ	ča	ሼ	še	ሕ	h	ፅ	S`
ኃ	ha	ቄ	qe	ም	m	ፍ	f
ና	na	ቤ	be	ሥ	s	ፕ	p
ኛ	Na	ቬ	ve	ር	r	ሆ	ho
አ	a	ቴ	te	ስ	s	ሎ	lo
ካ	ka	ቼ	če	ሽ	š	ሐ	ho
ኻ	ha	ኄ	he	ቅ	q	ሞ	mo
ዋ	wa	ኔ	ne	ብ	b	ሦ	so
ዓ	a	ኜ	Ne	ቭ	v	ሮ	ro
ዛ	za	ኤ	e	ት	t	ሶ	so
ዣ	Za	ኬ	ke	ች	č	ሾ	šo
ያ	ya	ኼ	he	ኀ	h	ቆ	qo
ዳ	da	ዌ	we	ን	n	ቦ	bo
ጃ	ja	ዔ	e	ኝ	N	ቮ	vo
ጋ	ga	ዜ	ze	አ	ə	ቶ	to
ጣ	Ta	ዤ	Ze	ክ	k	ቾ	čo
ጫ	Ca	ዬ	ye	ኽ	h	ኖ	ho
ጳ	Pa	ይ	de	ዉ	w	ኖ	no
ጻ	S`a	ጄ	je	ዕ	ə	ኞ	No
ፃ	S`a	ገ	ge	ዝ	z	ኦ	o
ፋ	fa	ጤ	Te	ዥ	Z	ኮ	ko
ፓ	pa	ፏ	Ce	ይ	y	ኾ	ho

ዎ	wo	ኗ	lwa	ኰ	kwe	ጐ	gWu
ዖ	o	ኋ	hwa	ዟ	zwa	ቀኵ	qWi
ዘ	zo	ሟ	mwa	ዧ	Zwa	ኈኵ	hWi
ዧ	Zo	ሧ	swa	ደ	dwa	ኩ	kWi
የ	yo	ሯ	rwa	ጀ	jwa	ጒ	gWi
ዶ	do	ሷ	swa	ገ	gwe	ቋ	qWa
ጆ	jo	ሿ	šwa	ጧ	Twa	ኋ	hWa
ጎ	go	ቈ	qwe	ጫ	Cwa	ካ	kWa
ጠ	To	ቧ	bwa	ጰ	Pwa	ጓ	gWa
ጨ	Co	ቫ	vwa	ጸ	S`wa	ቈ	qWe
ጶ	Po	ቷ	twa	ፏ	fwa	ኈ	hWe
ጸ	S`o	ቿ	čwa	ፐ	pwa	ኬ	kWe
ፆ	S`o	ኈ	hwe	ቀኵ	qWu	ጌ	gWe
ፎ	fo	ኗ	nwa	ኈኵ	hWu	ኧ	ä
ፖ	po	ኟ	Nwa	ኩ	kWu		

Appendix 2. List of words for development test

ቤተሰባዊነት	ልብወለዳችን	ስለሊቀመንበሮች	ሽለሙቅ	በአውቆአበድ
መልካሳ	መላቢስ	ስለልበደንዳናው	ቀልበቢሱ	በአየርኃይል
ሐሞተቢሱ	መልከመልካም	ስለመንግስተሰማይ	ቀልበቢስነት	በአየርወለዱ
ሐሞተቢስ	መልከጥፉ	ስለምድረበዳነት	ቀኛአዝማቾች	ደናግል
ሐሞተቢሶች	መሶበወርቁ	ስለምድረበዳው	ቀኛዝማች	በአየርወለድነት
ሐረገወይን	መሶበወርቄ	ስለሰርቶአደር	ቀኝአዝማች	በአይነስውርነት
አህያፈጅ	መሶበወርቅ	ስለሰርቶአዳሪ	ቀኝአዝማቾቹ	በአገረገዢነት
ሀብተምድር	መሸታቤት	ስለቀልበቢሱ	ቆርጦቀጥል	በአገረገዢው
ሐዲስኪዳኑ	መንግስተሰማያት	ስለቀኛዝማቹ	አይጠገብ	በእናትሀገራችን
ሐዲስኪዳን	መዓትአውሪ	ስለቀኛዝማች	ሰብስቤ	በእናትሀገር
ሀፍረተሥጋው	መዓትአውሪው	ስለበትረመንግስት	በሀኪምቤት	በእጀጠባብ
ሀፍረተሥጋውን	መኪናነጂ	ስለቤተመዘክር	በህገመንግስቱ	በወሮበሎች
ህገመንግስት	መጋቢመንገዱ	ስለቤተሰብ	በህገመንግስታችን	በወዝአደሩ
ሕገወጡ	መጽሐፍቅዱሱ	ስለቤተሰቦቻችን	በህገመንግስት	በየሀኪምቤት
ሕገወጦች	መጽሐፍቅዱስ	ስለአስርአለቃው	በሕግአውጪነት	በየሊቀመንበሮች
ሕግአስከባሪ	ምድረበዳ	ስለአውቆአበድ	በልበሙሉነት	በየሰርቶአደር
ሕግአስከባሪው	ምግብቤቱ	ስለአየርኃይላችን	በልብወለድ	በየቀኛዝማች
ሕግአውጪ	ምግብቤታችን	ስለአየርኃይል	በመልከመልካም	በየቡናቤቱ
ሕግአውጪው	ምግብቤትዋ	ስለአይነስውር	በመልከመልካምነት	በየቤተመንግስቱ
ሕግአውጪዎች	ምግብቢቶች	ስለእናትሀገር	በመልከጥፉነት	በግራአዝማችነት
ሆደሰፊ	ሞፈርዘመት	ስለእጀጠባብ	በመሶበወርቅ	ቡናቤቱ
ሆደሰፊዋ	ሰርቶአደሩ	ስለእጀጠባብዋ	በመንግስተሰማይ	ቡናቤቱን
ሆደባሻ	ሰርቶአደር	ስለእግረመልካም	በሰርቶአደሮች	ቡናቤት
ሆደባሻው	ሰርቶአደሮች	ስለእግረመንገድ	በሰርቶአዳሪው	ቤተመቅደስ
ለሰርቶአደሩ	ሰርቶአዳሪዎቻቸው	ስለእግረቀላሉ	በሰውሰራሹ	ቤተመንግስቱ
ለሰርነገቡ	ሰርቶአዳሪዎቻችን	ስለወዶዘማች	በሰውሰራሽ	ቤተመንግስት
ለሰርነገብ	ሰርነገቡ	ስለዘመነመንግስቱ	በሥራፈትነት	ቤተመዘክሩ
ላብአደሩ	ሰርነገብ	አጥላባቸው	በቀኛዝማች	ቤተመዘክር
ላብአደር	ሰርነገቦች	ሥራአጥ	በበትረመንግስቱ	ቤተሰቡ
ላብአደሮች	ሰብለወንጌል	ሥራፈት	በበትረመንግስት	ቤተሰብ
ልበቢስነት	ሰውሰራሽነት	ሥነሥርዓታቸው	በቤተመዘክር	ቤተክርስቲያኑ
ልበሙሉነት	ስለሀኪምቤት	ሥነሥርዓት	በቤተሰብ	ቤተክርስቲያን
ልበሙሉዋ	ስለህገመንግስቱ	ሥነሥርዓትዋ	በቤተክርስቲያን	ቤተክርስቲያኖች
ልበቅን	ስለሕገወጥ	ሥነሥርዓቶች	በብረትድስት	ቤትጠባቂ
ልበደንዳናው	ስለሕግአስከባሪ	ሹልአፍ	በትረመንግስት	ብረትምጣድ
ልብስሰፊ	ስለሆደባሻው	ሹልአፎች	በአስርአለቃ	ብረትድስቱ

ብረትድስት
ትምህርትቤቱ
ትምህርትቤት
አለምአቀፉ
አለምአቀፋዊ
አለምአቀፋዊነት
አለምአቀፋዊው
አቅመቢስ
አውቆአበድ
አውቆአበዶች
አየርኃይል
አየርመንገዱ
አየርመንገዳችን
አየርመንገድ
አየርወለዱ
አየርወለዳችን
አይነመልካሙ
አይነመልካም
አይነስውሩ
አይነስውር
አገረገዢ
አገረገዢነት
አገረገዢው
እንጀራጋጋሪ
እንጀራአናቱ
እንጀራአናታችን
እንጀራአናት
እንጀራአናትዋ
እጀረጅሙ
እጀጠባቡ
እጀጠባብ
እጀጠባብ
እጀጠባብዋ
እጅአዙር
እግረመልካሙ
እግረመንገዱን
እግረመንገድ
እግረቀላል
እግረቀጭን
እግረቀጭኖች
ከሐሞተቢስ
ከህኪምቤት
ከህገመንግስታችው
ከሊቀመንበራችን
ከሊቀመንበር
ከሊቀመንበሮች
ከልበሙሉ
ከልበቅን
ከልበደንዳና
ከምድረበዳ
ከሰርቶአዳሪዎች
ከሥራፈት
ከሥራፈቶች
ከቀልበቢሱ
ከቀልበቢስ
ከቀኖዝማች
ከበትረመንግስት
ከቤተመንግስት
ከቤተሰባችን
ከቤተሰብ
ከቤተክርስቲያን
ከትምህርትቤት
ከአውቆአበድ
ከአየርወለዱ
ከእናትሀገር
ከእግረመልካሙ
ከክፍለሀገር
ከወዝአደሮቻችን
ከየሕገወጡ
ከየሕግአስከባሪዎች
ፈርካሳ
ከየሊቀመንበሩ
ከየሊቀመንበሮች
ከየቀኛዝማች
ከየቤተሰቡ
ከግራአዝማቹ
ከግራአዝማችነት
ከግራዝማቾች
ከብረመንግስት
ከብረሰማይ
ከፍለሀገር
ከፍለሀገሮች
ወልዶገደል
ወሮበላነት
ወሮበላው
ወዝአደሩ
ወዶገቡ
ወዶዘማቹ
ወዶዘማች
ወጥቤቶቻቸው
ወፍዘራሹ
ወፍዘራሽነት
ውሃወለድ
ውሃገብ
ዘመነመንግስቱ
ዘመነመንግስታችን
ዘመነመንግስት
የሊቀመንበሩ
የልበሙሉ
የልበቅን
ወንድማማች
የቀልበቢስ
የበትረመንግስት
የቡናቤቱ
የብረትድስቱ
የብረትድስት
የአቅመቢስ
የክፍለሀገሩ
የክፍለሀገር
የወጥቤቱ
ገምጦወጥ
ጉዋዙብዙ
ግራአዝማችነት
ግራአዝማቾቻችን
ግብረኃይሉ
ግብረኃይል
ጠጅቤት
ፈጥኖደራሽ
እጅለጅ
ጸጥአላችሁ
ዝምአልን
ጸጥአደረጉ
ዝምአደረገ
ጉዝጉዝ
ሀረግሬሳ
ዝምብሎ
ስለጠባብነት
እግረቀጫጭን
ገደቢስ
ገደቢሶች
ቀጥአለ
ቀጥአደረገች
ቆፈን
ጥርጊያ
ቅርፊት
አህመድ
ቅርንጭፎች
ሰደርያ
ዳመነ
ጎጣጎጥ
ብቸሬ
መበለት
ወንበዴ
አስክልካይ
ጥምቀት
ወልደያ
ታህታይ
አድማስ
አብራር
ጨቡዴ
ምስጢራዊ
ደባልቄ
ብርሌ
መጀመሪያ
ማመልከቻ
ማካበት
ማዘጋጃ
ማእከል
ሰኔ
ሰደድ
ሹካ
ሽንጥ
ቁስአካልነት
ቃሪያ
በለስ
ባሌ
ባንዲራ
ቤት
ታየች
አቡን

Appendix 3. List of words for test set

የበትረመንግስታችን	አየርወለድ	እጀሰባራ	ግራአዝማች	ሕገወጥነት
ስለመልከጥፉዎች	አገረገገርዋ	ሥራአጡ	ወጥቤታችን	የመሶበወርቅ
የልብወለድ	ልበደንዳና	ሥነሥርዓቱ	ባህርጠለቅ	በየበትረመንግስቱ
ከብረትድስቶቹ	ሥነሥርዓትአቸው	ከመልከጥፉው	መላቢሶች	ቆርጦቀጥሎች
ስለሊቀመንበር	ዘመነመንግስቶች	ጦርሜዳ	ሀኪምቤት	ግራአዝማቹ
ሹልአፉ	በወሮበላ	ስለመልከጥፉው	ልብወለዳቸው	በከፍለሀገር
ቀኝአዝማቾች	ወዝአደሮች	በሊቀመንበሮቻችን	ስለግብረኃይሉ	ሰርቶአዳሪው
ሀኪምቤቱ	ወጥቤቶች	ከወዝአደሮች	እጀረጅም	አየርወለዶች
ቀልበቢስ	አውቆአበድዋ	በሊቀመንበር	ቡናቤታችን	አለምአቀፍ
ወዝአደር	ቀኝአዝማቾቻችን	በቤተመንግስቱ	ስለሀኪምቤታችን	መንግስተሰማይ
ግራአዝማቾቹ	ልበቢሱ	ቆርጦቀጥሉ	ወዶኀብ	ልበሙሉዎች
በወዶዘማችነት	ልብወለዱ	በበትረመንግስታችን	በየትምህርትቤቱ	ስለቤተሰቡ
መንፈቀሌሊት	ከየትምህርትቤት	በእናትሀገሩ	በወዶዘማች	በሰርቶአደሩ
ወሮበላ	ልብወለድ	ስለአየርኃይሉ	ቀኝአዝማቹ	ልበሙሉ
ከግራዝማቹ	ከቤተክርስቲያናችን	በሕግአስከባሪነት	የወጥቤት	አውቆአበዱ
ልበቅኖች	ወጥቤቱ	ሆደባሻነት	ከሰርቶአደር	ቅሬታ
ልበቢስ	የቡናቤት	መላቢሱ	ሊቀመንበሩ	አለባቸው
በዘመነመንግስቱ	በሰርጎገብ	የበትረመንግስቱ	እግረመልካም	ገደቢሱ
ስለኢይነስውሯ	በግብረኃይል	ሰውሰራሹ	ሰርቶአዳሪ	ጐርፉ
ከአየርወለድ	በትምህርትቤቱ	በመልከመልካሞች	ሕገወጥ	ይሁዲ
ለሰርቶአደር	በምድረበዳ	በትረመንግስቶች	ከቡናቤቱ	ደረሰኝ
ሥነሥርዓቶቻችን	ሊቀመንበር	ወጥቤታቸው	ስለቤተክርስቲያኒትዋ	እቁበ
በኢይነመልካምነት	ልበቅንነት	አየርመንገዳቸው	ከሰርጎገብ	ሞንዳላ
ስለቆርጦቀጥል	ሀፍረተሥጋ	ትምህርትቤታችን	በሰርቶአዳሪነት	ገደቢስነት
አቅመቢሶች	ወጥቤት	አስርአለቃ	እግረቀላሉ	ደም
ቀኝአዝማችነት	ከአስርአለቃው	በትረመንግስቱ	በኢይነመልካሙ	ብልሀተኛ
ወፍዘራሽ	ስለወዶዘማቾች	ሰውሰራሽ	በእግረቀላል	እንድሪስ
መልከጥፉው	የቀኖዝማች	ቡናቤቶች	አቅመቢሱ	ሽግግር
መጋቢመንገድ	ጦርሜዳነት	የሊቀመንበር	ስለህገመንግስት	ትሁኔ
እናትሀገር	ምድረበዳው	ምግብቤት	ስለቤተክርስቲያን	ዘለሰኛ

Appendix 4. Python codes

```
#!usr/bin/python

import re

def prefix_stemmer(wrd):          #(word,noun,verb):
      pr = ''
      for i in range(len(wrd)):
            pr = wrd[:len(wrd)-i]
            if pr in prfxs:
                  break
            else:
                  pr = ''
      return pr

def suffix_stemmer(stm):
      sfx = ''
      for i in range(len(stm)):
            last = stm[i:]
            if last in csfxs:
                  sfx = last
                  break
      return sfx

def firstElement(wrd):
      pos = ''
      c1 = ''
      for i in range(len(wrd)):
            x = wrd[:len(wrd) - i]
            if x in noun:
                  c1 = x
                  pos = 'noun'
                  break
            elif x in verb:
```

Appendix 4. Python codes

```
#!usr/bin/python

import re

def prefix_stemmer(wrd):          #(word,noun,verb):
      pr = ''
      for i in range(len(wrd)):
            pr = wrd[:len(wrd)-i]
            if pr in prfxs:
                  break
            else:
                  pr = ''
      return pr

def suffix_stemmer(stm):
      sfx = ''
      for i in range(len(stm)):
            last = stm[i:]
            if last in csfxs:
                  sfx = last
                  break
      return sfx

def firstElement(wrd):
      pos = ''
      c1 = ''
      for i in range(len(wrd)):
            x = wrd[:len(wrd) - i]
            if x in noun:
                  c1 = x
                  pos = 'noun'
                  break
            elif x in verb:
```

```
                c1 = x
                pos = 'verb'
                break
        if len(c1) < 1:
            prfx = prefix_stemmer(wrd)
            if len(prfx) > 0:
                wrd = wrd[len(prfx):]
                x = ''
                for j in range(len(wrd)):
                    x = wrd[:len(wrd) - j]
                    if x in noun:
                        c1 = x
                        pos = 'noun'
                    elif x in verb:
                        c1 = x
                        pos = 'verb'
            else:
                c1 = ''
        return c1, pos


def secondElement(stm_remain):
    c2 = ''
    pos = ''
    dm = ''
    x = ''
    y = stm_remain
    for i in range(len(y)):
        if y in noun:
            c2 = y
            pos = 'noun'
            break
        elif y in verb:
            c2 = y
            pos = 'verb'
```

```
                    break
               elif y in adj:
                    c2 = y
                    pos = 'adjective'
                    break
               else:
                    dm = stm_remain[-i-1:]
                    y = stm_remain[:-i-1]
          x = dm[:6]
          dm = dm[:3]
          if len(c2) > 0:
               if dm == 'nät':
                    c2 = c2 + dm
                    #pos = 'noun'
               elif x == 'awinät':
                    c2 = c2 + x
                    #pos = 'noun'
               elif dm == 'awi':
                    c2 = c2 + dm
                    #pos = 'adjective'
          return c2, pos

def inflection(sfx):
          fh = open('suffixes_tr2.txt', 'r')
          sfxs = fh.read().split('\t')
          fh.close()

          z = ''
          if sfx in sfxs:
               indx = sfxs.index(sfx)
               z = sfxs[indx + 1]
               nf.write('Possesser: '), nf.write(z), nf.write('\n')
               nf.write('Grammar: '), nf.write('definite\n')
          else:
               for i in range(len(sfx)):
```

```
                last = sfx[-1:]
                frst = sfx[:2]
                frst1 = sfx[:3]
                if frst == 'oč' or frst1 == 'woč' or frst1 == 'at':
                      l1 = sfx[2:]
                      if len(sfx) == 2 or frst1 == 'woč':
                            nf.write('Grammar: '), nf.write('plural\n')
                            break
                      elif last == 'n':
                            mid = sfx[2:-1]
                            if mid in sfxs:
                                  indx = sfxs.index(mid)
                                  z = sfxs[indx + 1]
                                  nf.write('Possesser:  '),  nf.write(z),
nf.write('\n')
                                  nf.write('Grammar:                  '),
nf.write('plural '), nf.write('definite '), nf.write('accusative\n')
                                  break
                            else:
                                  if l1 == 'ačn':
                                        indx = sfxs.index(l1)
                                        z = sfxs[indx + 1]
                                        nf.write('Possesser:          '),
nf.write(z), nf.write('\n')
                                        nf.write('Grammar:            '),
nf.write('plural '), nf.write('definite\n')
                                        break
                            elif l1 in sfxs:
                                  indx = sfxs.index(l1)
                                  z = sfxs[indx + 1]
                                  nf.write('Possesser:     '),     nf.write(z),
nf.write('\n')
                                  nf.write('Grammar: '), nf.write('plural '),
nf.write('definite\n')
                                  break
```

```
                    elif last == 'u':
                        nf.write('Grammar: '), nf.write('plural '),
nf.write('definite\n')
                        break
                elif frst == 'it':
                    last = sfx[-1:]
                    last2 = sfx[-2:]
                    if last == 'u':
                        indx = sfxs.index(last)
                        z = sfxs[indx + 1]
                        nf.write('Possesser:    '),    nf.write(z),
nf.write('\n')
                        nf.write('Grammar:                    '),
nf.write('definite\n')
                        break
                    elif last2 in sfxs:
                        indx = sfxs.index(last2)
                        z = sfxs[indx + 1]
                        nf.write('Possesser:    '),    nf.write(z),
nf.write('\n')
                        nf.write('Grammar:                    '),
nf.write('definite\n')
                        break
                elif last == 'n':
                    mid = sfx[:-1]
                    last = sfx[2:-1]
                    last2 = sfx[2:-1]
                    if mid == 'u':
                        indx = sfxs.index(mid)
                        z = sfxs[indx + 1]
                        nf.write('Possesser:    '),    nf.write(z),
nf.write('\n')
                        nf.write('Grammar:  '),  nf.write('definite
'), nf.write('accusative\n')
                        break
```

```
                    elif mid in sfxs:
                        indx = sfxs.index(mid)
                        z = sfxs[indx + 1]
                        nf.write('Possesser:     '),     nf.write(z),
nf.write('\n')
                        nf.write('Grammar:   '),   nf.write('definite
'), nf.write('accusative\n')
                        break
                    elif frst == 'it' and last == 'u':
                        indx = sfxs.index(last)
                        z = sfxs[indx + 1]
                        nf.write('Possesser:     '),     nf.write(z),
nf.write('\n')
                        nf.write('Grammar:   '),   nf.write('definite
'), nf.write('accusative\n')
                        break
                    elif frst == 'it' and last2 in sfxs:
                        indx = sfxs.index(last2)
                        z = sfxs[indx + 1]
                        nf.write('Possesser:     '),     nf.write(z),
nf.write('\n')
                        nf.write('Grammar:   '),   nf.write('definite
'), nf.write('accusative\n')
                        break
                    else:
                        nf.write('Grammar:                          '),
nf.write('accusative\n')
                        break

def inflection2(sfx):
    fh = open('suffixes_tr2.txt', 'r')
    sfxs = fh.read().split('\t')
    fh.close()

    z = ''
```

```
    if sfx in sfxs:
        indx = sfxs.index(sfx)
        z = sfxs[indx + 1]
        print('Possesser: ', z)
        print('Grammar: definite')
    else:
        for i in range(len(sfx)):
            last = sfx[-1:]
            frst = sfx[:2]
            frst1 = sfx[:3]
            if frst == 'oč' or frst1 == 'woč' or frst1 == 'at':
                l1 = sfx[2:]
                if len(sfx) == 2 or frst1 == 'woč':
                    print('Grammar: plural')
                    break
                elif last == 'n':
                    mid = sfx[2:-1]
                    if mid in sfxs:
                        indx = sfxs.index(mid)
                        z = sfxs[indx + 1]
                        print('Possesser: ', z)
                        print('Grammar:    plural    definite
accusative')
                        break
                    else:
                        if l1 == 'ačn':
                            indx = sfxs.index(l1)
                            z = sfxs[indx + 1]
                            print('Possesser: ', z)
                            print('Grammar:          plural
definite')
                            break
                elif l1 in sfxs:
                    indx = sfxs.index(l1)
                    z = sfxs[indx + 1]
```

```
                print('Possesser: ', z)
                print('Grammar: plural definite')
                break
        elif last == 'u':
                print('Grammar: plural definite')
                break
elif frst == 'it':
        last = sfx[-1:]
        last2 = sfx[-2:]
        if last == 'u':
                indx = sfxs.index(last)
                z = sfxs[indx + 1]
                print('Possesser: ', z)
                print('Grammar: definite')
                break
        elif last2 in sfxs:
                indx = sfxs.index(last2)
                z = sfxs[indx + 1]
                print('Possesser: ', z)
                print('Grammar: definite')
                break
elif last == 'n':
        mid = sfx[:-1]
        last = sfx[2:-1]
        last2 = sfx[2:-1]
        if mid == 'u':
                indx = sfxs.index(mid)
                z = sfxs[indx + 1]
                print('Possesser: ', z)
                print('Grammar: definite accusative')
                break
        elif mid in sfxs:
                indx = sfxs.index(mid)
                z = sfxs[indx + 1]
                print('Possesser: ', z)
```

```
                    print('Grammar: definite accusative')
                    break
               elif frst == 'it' and last == 'u':
                    indx = sfxs.index(last)
                    z = sfxs[indx + 1]
                    print('Possesser: ', z)
                    print('Grammar: definite accusative')
                    break
               elif frst == 'it' and last2 in sfxs:
                    indx = sfxs.index(last2)
                    z = sfxs[indx + 1]
                    print('Possesser: ', z)
                    print('Grammar: definite accusative')
                    break
               else:
                    print('Grammar: accusative')
                    break

def inpt(wrd):

     sp_ch = open('sp_char1.txt', 'r')
     sp_char = sp_ch.read().split('\n')
     sp_ch.close()

     temp = ''

     for ch in wrd:
          ch = re.sub('[a-zA-z]', '', ch)
          ch = re.sub('[0-9]', '', ch)
          if ch in sp_char:
               wrd = wrd.replace(ch, '')
          elif ch == '-':
               wrd = wrd.replace(ch, '')
          elif ch == ' ':
               wrd = wrd.replace(ch, '')
```

```
            elif ch == '\t':
                  wrd = wrd.replace(ch, '')

      temp = wrd

      fh = open('char_table_modified.txt', 'r')

      char_dic = {}
      text = fh.read().split()
      fh.close()

      for i in range(len(text)-1):
            char_dic[text[i]] = text[i+1]

      for ch in wrd:
            if char_dic.__contains__(ch):
                  wrd = wrd.replace(ch, char_dic[ch])

      uw = analyzer2(temp, wrd)

def inptwrdlst(sname):

      fh = open(sname, 'r')
      chs = fh.read()
      fh.close()

      sp_ch = open('sp_char1.txt', 'r')
      sp_char = sp_ch.read().split('\n')
      sp_ch.close()

      temp = []

      for ch in chs:
            ch = re.sub('[a-zA-z]', '', ch)
            ch = re.sub('[0-9]', '', ch)
```

```
            if ch in sp_char:
                  chs = chs.replace(ch, '')
            elif ch == '-':
                  chs = chs.replace(ch, '')
            elif ch == ' ':
                  chs = chs.replace(ch, '')
            elif ch == '\t':
                  chs = chs.replace(ch, '')

      temp = chs.split('\n')

      fh = open('char_table_modified.txt', 'r')

      char_dic = {}
      text = fh.read().split()
      fh.close()

      for i in range(len(text)-1):
            char_dic[text[i]] = text[i+1]

      for ch in chs:
            if char_dic.__contains__(ch):
                  chs = chs.replace(ch, char_dic[ch])
      words = chs.split('\n')

      uw = ''

      for i in range(len(words)):
            uw = analyzer(temp[i],words[i])


def inptrawtxt(file_name):
      fh = open(file_name, 'r')
      chs = fh.read()
```

```
sp_ch = open('sp_char.txt', 'r')
sp_char = sp_ch.read().split('\n')
sp_ch.close()

for ch in chs:
      ch = re.sub('[a-z]', '', ch)
      ch = re.sub('[A-Z]', '', ch)
      ch = re.sub('[0-9]', '', ch)
      ch = re.sub('([)*(])', '', ch)
      if ch in sp_char:
            chs = chs.replace(ch, '')
      elif ch == '-':
            chs = chs.replace(ch, '')

l = []
wordsAm = chs.split()
temp = []

fh = open('char_table_modified.txt', 'r')

char_dic = {}
text = fh.read().split()
fh.close()


for i in range(len(text)-1):
      char_dic[text[i]] = text[i+1]

for ch in chs:
      if char_dic.__contains__(ch):
            chs = chs.replace(ch, char_dic[ch])

wordsTr = chs.split()
fh.close()
```

```
bi_gram = ''
bi_gramAm = ''
l = []
c = ''
s1 = ''
s2 = ''
w = ''
amh = ''

for i in range(len(wordsTr)-1):
      l.append(wordsTr[i])
      temp.append(wordsAm[i])

      if wordsTr[i].endswith('ä'):
            bi_gram = wordsTr[i] + wordsTr[i+1]
            l.append(bi_gram)
            bi_gramAm = wordsAm[i] + wordsAm[i+1]
            temp.append(bi_gramAm)
      else:
            bi_gram = wordsTr[i] + ' ' + wordsTr[i+1]
            bi_gramAm = wordsAm[i] + ' ' + wordsAm[i+1]
            w = bi_gram.split()
            amh = bi_gramAm.split()

            s1 = suffix_stemmer(w[0])
            s2 = suffix_stemmer(w[1])
            c = w[1][:-len(s2)]
            if w[1] in const2 and len(s1) == 0:
                  l.append(w[0]+w[1])
                  temp.append(amh[0]+amh[1])
            elif c in const2 and len(s1) == 0:
                  l.append(w[0]+w[1])
                  temp.append(amh[0]+amh[1])

words = l
```

```
    uw = ''

    for i in range(len(words)):
        uw = analyzer(temp[i],words[i])


def transliterator_modifier(wrd):
    w = ''

    if wrd[:1] in consonants and wrd[1:2] in consonants:        # and
wrd[2:3] in consonants and wrd[3:4] in consonants:
        w = wrd[:1] + 'ə' + wrd[1:2]     #   +   wrd[2:3]   +   'ə'   +
wrd[3:4] + wrd[4:]
        if wrd[2:3] in consonants and wrd[3:4] in consonants and
len(wrd) >= 4:
            w = w + wrd[2:3] + 'ə' + wrd[3:4] + wrd[4:]
        elif wrd[2:3] in consonants and len(wrd) > 3:
            w = w + wrd[2:]
        elif    wrd[2:3]    in    consonants    and    wrd[-2:]    in
['rt','nd','dr','br'] and len(wrd) == 3:
            w = w + wrd[2:]
        elif wrd[2:3] in consonants and len(wrd) == 3:
            w = w + 'ə' + wrd[2:]
        else:
            w = w + wrd[2:]
    else:
        w = wrd
    return w


def displayer(prfx,c10,c11,connector,c20,c21,sfx,am):

    nf.write('Word: '), nf.write(am), nf.write('\n')

    dm = c20[-6:]
```

```
        a = c20[-3:]
        if dm == 'awinät':
                temp = c20[:-6]
                c10 = transliterator_modifier(c10)
                c20 = transliterator_modifier(c20)
                nf.write('Stem:   '),   nf.write(c10),   nf.write(connector),
nf.write(c20), nf.write('\tPOS: noun'), nf.write('\n')
                c20 = temp
        elif a == 'nät':
                temp = c20[:-3]
                c10 = transliterator_modifier(c10)
                c20 = transliterator_modifier(c20)
                nf.write('Stem:   '),   nf.write(c10),   nf.write(connector),
nf.write(c20), nf.write('\tPOS: noun'), nf.write('\n')
                c20 = temp
        elif a == 'awi':
                temp = c20[:-3]
                c10 = transliterator_modifier(c10)
                c20 = transliterator_modifier(c20)
                nf.write('Stem:   '),   nf.write(c10),   nf.write(connector),
nf.write(c20), nf.write('\tPOS: adjective'), nf.write('\n')
                c20 = temp

        c10 = transliterator_modifier(c10)
        c20 = transliterator_modifier(c20)

        if connector in ['bä', 'lä']:
                nf.write('Compound: '), nf.write(c10), nf.write(connector),
nf.write(c20), nf.write('\tPOS: adverb'), nf.write('\n')
        elif c21 == 'noun' and c20 in adj:
                nf.write('Compound: '), nf.write(c10), nf.write(connector),
nf.write(c20), nf.write('\tPOS: noun/adjective'), nf.write('\n')
        elif c21 == 'verb':
                nf.write('Compound: '), nf.write(c10), nf.write(connector),
nf.write(c20), nf.write('\tPOS: noun'), nf.write('\n')
```

```
        else:
            nf.write('Compound: '), nf.write(c10), nf.write(connector),
nf.write(c20), nf.write('\tPOS: '), nf.write(c21), nf.write('\n')

        if len(prfx) > 0:
            nf.write('Prfx: '), nf.write(prfx), nf.write('\n')
        nf.write('first   element:   '),   nf.write(c10),   nf.write('\t'),
nf.write('POS: '), nf.write(c11), nf.write('\n')
        if len(connector) > 0:
            nf.write('connector:        '),        nf.write(connector),
nf.write('\n')
        if c21 == 'noun' and c20 in adj:
            nf.write('second      element:      '),      nf.write(c20),
nf.write('\t'),        nf.write('POS:        '),        nf.write(c21),
nf.write('/adjective'), nf.write('\n')
        else:
            nf.write('second      element:      '),      nf.write(c20),
nf.write('\t'), nf.write('POS: '), nf.write(c21), nf.write('\n')
        if len(sfx) > 0:
            inflection(sfx)
        nf.write('-------------------------------------------------
\n')

def analyzer(am,w):

        prfx = ''
        connector = ''
        sfx = ''
        sfx = suffix_stemmer(w)
        s = w[:-len(sfx)]
        al = s[-2:]
        ad = s[-6:]
        r = ''      #return

        if len(sfx) > 0 and al == 'al':
```

```
        c1 = firstElement(w[:-2-len(sfx)])
        c2 = 'al'
        if len(c1[0]) > 0 and c1[1] == 'verb':
              displayer(prfx,c1[0],c1[1],connector,c2,'verb',sfx,am)
              r = 'no'
        else:
              r = 'yes'
  elif len(sfx) > 0 and ad == 'adäräg':
        c1 = firstElement(w[:-6-len(sfx)])
        c2 = 'adäräg'
        if len(c1[0]) > 0 and c1[1] == 'verb':
              displayer(prfx,c1[0],c1[1],connector,c2,'verb',sfx,am)
              r = 'no'
        else:
              r = 'yes'
  elif len(sfx) == 0:
        al = w[-3:]
        ad = w[-7:]
        if len(sfx) == 0 and al == 'alä':
              c1 = firstElement(w[:-3])
              c2 = 'alä'
              if len(c1[0]) > 0 and c1[1] == 'verb':

  displayer(prfx,c1[0],c1[1],connector,c2,'verb',sfx,am)
                    r = 'no'
              else:
                    r = 'yes'
        elif len(sfx) == 0 and ad == 'adärägä':
              c1 = firstElement(w[:-7])
              c2 = 'adärägä'
              if len(c1[0]) > 0 and c1[1] == 'verb':

  displayer(prfx,c1[0],c1[1],connector,c2,'verb',sfx,am)
                    r = 'no'
              else:
```

```
                r = 'yes'

    connector = ''
    pdbl = ''
    sdbl = ''
    prfx = prefix_stemmer(w)

    c1 = firstElement(w)

    if c1[0][:len(prfx)] == prfx:
        pdbl = w[len(prfx):len(prfx)*2]
        if pdbl == prfx:
            c1 = c1
        elif len(c1[0]) == 2: #əjläj
            prfx = ''
        else:
            c1 = firstElement(w[len(prfx):])
            if len(c1[0]) > 0:
                prfx = prfx
                c1 = c1
            else:
                prfx = ''
    else:
        c1 = firstElement(w[len(prfx):])

    if len(c1[0]) == 0:
        c1 = secondElement(w[len(prfx):])

    stm = w[len(c1[0])+len(prfx):]

    for c in range(1, 3):
        if stm[:c] in ['ä', 'a', 'bä', 'lä']:
            connector = stm[:c]
            break
```

```
    c2 = secondElement(stm)
    sfx = suffix_stemmer(stm)
    c2_len = 0
    c2_len = stm[:-len(sfx)]

    tem = ''

    if c1[0].endswith('a') and len(stm[:-len(sfx)]) >= 2:
        if len(c2[0]) == 0 or c2[1] == 'verb':
            stm = 'a' + stm
            c2 = secondElement(stm)

    #displayer(prfx,c1[0],c1[1],connector,c2[0],c2[1],sfx,am)
    else:
        stm = w[len(c1[0])+len(prfx):]
        c2 = secondElement(stm)

    if connector == 'bä' or connector == 'lä':
        #print(c2[0])
        c2 = secondElement(stm[len(connector):])
        if len(c2[0]) == 0:
            stm_mod = 'ə' + stm[len(connector):]
            c2 = secondElement(stm_mod)
            if len(c2[0]) == 0:
                c2 = secondElement(stm)
                connector = ''
        elif c1[0] != c2[0]:
            connector = ''
    elif connector == 'ä' and c1[0] == 'bal' and c2[0] == '':
    #baläda <- baläəda
        stm_mod = 'ə' + stm[len(connector):]
        c2 = secondElement(stm_mod)
        #print(c2[0])
    elif c2[0][:len(connector)] == connector:        #in ['ä', 'a',
'bä', 'lä']:
```

```
            c2 = secondElement(stm[len(connector):])
            total_len = len(prfx) + len(c1[0]) + len(connector) +
len(c2[0]) + len(sfx)
            if len(w) == total_len:
                  connector = connector
            else:
                  connector = ''
                  c2 = secondElement(stm)
      else:
            stm = stm[len(connector):]
            c2 = secondElement(stm)

      if len(sfx) > 0:
            if c2[0][-len(sfx):] == sfx:          #ወንደላጤ     wänd-ä-laTe
not wänd-ä-laT-e
                  sdbl = w[-len(sfx)*2:-len(sfx)]
                  if sdbl == sfx:
                        sfx = sfx
                  else:
                        sfx = ''
            else:
                  for i in range(len(sfx)):
                        if c2[0][-i:] == sfx[:i]:       #ቀኛዝማቹ      qäN-
azmac-u not qäN-azmac-acu
                              sfx = sfx[i:]
                              break
      if len(sfx) > 0:
            if sfx not in csfxs:
                  sfx = suffix_stemmer(stm)
                  c2 = secondElement(stm[:-len(sfx)])

      c_len = len(prfx) + len(c1[0]) + len(connector) + len(c2[0]) +
len(sfx)          #sum of the constituents length
```

```
    if len(c1[0]) > 0 and len(c1[1]) > 0 and len(c2[0]) > 0 and
len(c2[1]) > 0 and len(w) == c_len and al != 'alä':
        displayer(prfx,c1[0],c1[1],connector,c2[0],c2[1],sfx,am)
    elif c1[0].endswith('a') and c2[0].startswith('a') and len(w) ==
c_len - 1 and al != 'alä':
        displayer(prfx,c1[0],c1[1],connector,c2[0],c2[1],sfx,am)
    elif len(c1[0]) > 0 and len(c1[1]) > 0 and len(c2[0]) > 0 and
len(c2[1]) > 0 and connector in ['ä','bä', 'lä'] and len(w) == c_len -
1 and al != 'alä':
        displayer(prfx,c1[0],c1[1],connector,c2[0],c2[1],sfx,am)
    else:
        if r != 'no':
            return ''

def displayer2(prfx,c10,c11,connector,c20,c21,sfx,am):

    print('Word: ',am)

    dm = c20[-6:]
    a = c20[-3:]
    if dm == 'awinät':
        temp = c20[:-6]
        c10 = transliterator_modifier(c10)
        c20 = transliterator_modifier(c20)
        print('Stem: ', c10, connector, c20, '\tPOS: noun')
        c20 = temp
    elif a == 'nät':
        temp = c20[:-3]
        c10 = transliterator_modifier(c10)
        c20 = transliterator_modifier(c20)
        print('Stem: ', c10, connector, c20, '\tPOS: noun')
        c20 = temp
    elif a == 'awi':
        temp = c20[:-3]
        c10 = transliterator_modifier(c10)
```

```
            c20 = transliterator_modifier(c20)
            print('Stem: ', c10, connector, c20, '\tPOS: adjective')

            c20 = temp

      c10 = transliterator_modifier(c10)
      c20 = transliterator_modifier(c20)

      if connector in ['bä', 'lä']:
            print('Compound: ',c10, connector,c20,'\tPOS: adverb')
      elif c21 == 'noun' and c20 in adj:
            print('Compound:                    ',c10,connector,c20,'\tPOS:
noun/adjective')
      elif c21 == 'verb':
            print('Compound: ', c10, connector,c20,'\tPOS: noun')
      else:
            print('Compound: ', c10,connector, c20,'\tPOS: ',c21)

      if len(prfx) > 0:
            print('Prfx: ', prfx)
      print('first element: ', c10, '\t', 'POS: ', c11)
      if len(connector) > 0:
            print('connector: ', connector)
      if c21 == 'noun' and c20 in adj:
            print('second   element:   ',   c20,   '\t',   'POS:   ',   c21,
'/adjective')
      else:
            print('second element: ', c20, '\t', 'POS: ', c21)
      if len(sfx) > 0:
            inflection2(sfx)

def analyzer2(am,w):

      prfx = ''
      connector = ''
```

```
sfx = ''
sfx = suffix_stemmer(w)
s = w[:-len(sfx)]
al = s[-2:]
ad = s[-6:]
r = ''

if len(sfx) > 0 and al == 'al':
      c1 = firstElement(w[:-2-len(sfx)])
      c2 = 'al'
      if len(c1[0]) > 0 and c1[1] == 'verb':
            displayer(prfx,c1[0],c1[1],connector,c2,'verb',sfx,am)
            r = 'no'
      else:
            r = 'yes'
elif len(sfx) > 0 and ad == 'adäräg':
      c1 = firstElement(w[:-6-len(sfx)])
      c2 = 'adäräg'
      if len(c1[0]) > 0 and c1[1] == 'verb':
            displayer(prfx,c1[0],c1[1],connector,c2,'verb',sfx,am)
            r = 'no'
      else:
            r = 'yes'
elif len(sfx) == 0:
      al = w[-3:]
      ad = w[-7:]
      if len(sfx) == 0 and al == 'alä':
            c1 = firstElement(w[:-3])
            c2 = 'alä'
            if len(c1[0]) > 0 and c1[1] == 'verb':

displayer(prfx,c1[0],c1[1],connector,c2,'verb',sfx,am)
                  r = 'no'
            else:
                  r = 'yes'
```

```
        elif len(sfx) == 0 and ad == 'adärägä':
                c1 = firstElement(w[:-7])
                c2 = 'adärägä'
                if len(c1[0]) > 0 and c1[1] == 'verb':

displayer(prfx,c1[0],c1[1],connector,c2,'verb',sfx,am)
                        r = 'no'
                else:
                        r = 'yes'

connector = ''
pdbl = ''
sdbl = ''
prfx = prefix_stemmer(w)

c1 = firstElement(w)

if c1[0][:len(prfx)] == prfx:
        pdbl = w[len(prfx):len(prfx)*2]
        if pdbl == prfx:
                c1 = c1
        elif len(c1[0]) == 2: #əjläj
                prfx = ''
        else:
                c1 = firstElement(w[len(prfx):])
                if len(c1[0]) > 0:
                        prfx = prfx
                        c1 = c1
                else:
                        prfx = ''
else:
        c1 = firstElement(w[len(prfx):])

if len(c1[0]) == 0:
        c1 = secondElement(w[len(prfx):])
```

```
stm = w[len(c1[0])+len(prfx):]

for c in range(1, 3):
      if stm[:c] in ['ä', 'a', 'bä', 'lä']:
            connector = stm[:c]
            break

c2 = secondElement(stm)
sfx = suffix_stemmer(stm)
c2_len = 0
c2_len = stm[:-len(sfx)]

tem = ''

if c1[0].endswith('a') and len(stm[:-len(sfx)]) >= 2:
      if len(c2[0]) == 0 or c2[1] == 'verb':
            stm = 'a' + stm
            c2 = secondElement(stm)

#displayer(prfx,c1[0],c1[1],connector,c2[0],c2[1],sfx,am)
else:
      stm = w[len(c1[0])+len(prfx):]
      c2 = secondElement(stm)

if connector == 'bä' or connector == 'lä':
      c2 = secondElement(stm[len(connector):])
      if len(c2[0]) == 0:
            stm_mod = 'ə' + stm[len(connector):]
            c2 = secondElement(stm_mod)
            if len(c2[0]) == 0:
                  c2 = secondElement(stm)
                  connector = ''
      elif c1[0] != c2[0]:
            connector = ''
```