ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
COLLEGE OF NATURAL SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

DESIGN AND IMPLEMENTATION OF AFAAN OROMO SPELL CHECKER

By
Gaddisa Olani Ganfure

A THESIS SUBMITTED TO
THE SCHOOL OF GRADUATE STUDIES OF THE ADDIS ABABA UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTERS OF SCIENCE IN COMPUTER SCIENCE

June 2013

ADDIS ABABA, ETHIOPIA

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
COLLEGE OF NATURAL SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

DESIGN AND IMPLEMENTATION OF AFAAN OROMO SPELL CHECKER

By: **Gaddisa Olani Ganfure**

Signature of the Board of Examiners for Approval

	Name	**Signature**
1.	Dr.Dida Midekso Advisor	
2.	Dr.Dejene Ejigu Examiner	
3.	_____ Chairman, Examining Board	_____

Dedication

To my father:

Dad, you make me strong and motivated person.

"*Ilma abbaan daakaa beeku, mucaa galaanni hin nyaatu!*"

To my mother:

Mom, you were dedicated to change my life, may God bless you!

Acknowledgements

I would like to sincerely thank all those people who have made this thesis possible. First of all, special thanks are due to my advisor **Dr. Dida Midekso**, who monitored my work and took effort in reading and providing me with valuable comments starting from title selection to this end. Thank you very much!

Beside my advisor, I would like to thank **Tariku, Bikila and Marqos** from linguistic department for their willingness to give me comments at all times without complaint on linguistic aspects and for providing me linguistic materials and in evaluating the system. I would also like to thank **Galana Kumara** and **Adugna Hailu** from *Kallacha Oromiyaa news papers* for their cooperation in providing me the softcopy of their news articles.

I equally wish to extend my appreciation and thanks to my friends **Dr. Dinaol Belina** and **Leta Lenca** for their moral support and encouragement. And last, but absolutely not least, I thank God for giving me the wisdom and the strength I need to discharge my duty.

Gaddisa Olani!

Table of Contents

List of Tables

Page

List of Figures

Page

List of Acronyms

AOSC – Afaan Oromo spell checker

NLP – Natural Language Processing

OCR – Optical Character Recognition

POS –Parts of speech

LED – Levenshtein Edit Distance

MST – Median Split Tree

FSA – Finite State Automata

DP – Dynamic Programming

Abstract

Developing language applications or localization of software is a resource intensive task that requires the active participation of stakeholders with various backgrounds (i.e. from linguistic and computational perspectives). With a constant increase in the amounts of electronic information and the diversity of languages which are used to produce them, these challenges get compounded. Various researches in the fields of computational linguistics and computer science have been carried out while still many more are on their way to alleviate such problems. Spell checker is one potential candidate to this. Use of computers for document preparation is one of those many tasks undertaken by different organizations. Introducing texts to word processing tools may result in spelling errors. Hence, text processing application software has spell checkers. Integrating spell checker into word processors reduces the amount of time and energy spent to find and correct the misspelled word. However, these tools are not available for Afaan Oromo language, Lowland East Cushitic sub-family of the Afro-asiatic super-phylum language family spoken in Ethiopia. In this thesis, we describe the design and implementation of Afaan Oromo spell checker.

Morphology based (i.e. dictionary look-up with morphological rules) computational model was employed to design and develop Afaan Oromo Spell Checker ***(AOSC)****. Algorithms that take the morphological properties of Afaan Oromo into consideration are developed from scratch and applied, as there are no previous such attempts. The proposed system was evaluated using two datasets of different size. The experiment result shows that the lexicon size and rules in the knowledge base play a vital role to recognize the valid input word, flag the invalid word and generate correct suggestion for the misspelled word.*

In general, the algorithms and techniques used in this study obtained good performance when compared to the other resource-rich languages like English. The result obtained encourages the undertaking of further research in the area, especially with the aim of developing a full-fledged Afaan Oromo spell checker.

Keywords*: Spell checker, Error detection, Error correction, Morphology, Natural Language, Natural language processing, Afaan Oromo*

Chapter One: Introduction

1.1. Background

Natural Language is any of the languages naturally used by humans, i.e. not an artificial or man-made language such as a programming language [1]. Natural language processing (NLP) is a any attempts to use computers to process natural language. NLP is a subfield of artificial intelligence and linguistics. It studies the problems of automated generation and understanding of natural human languages. The major fields of interest in NLPs are speech recognition, speech synthesis, machine translation, text categorization, natural language understanding, spell checker and the like.

A spell checker is an application program that flags words in a document that are not spelled correctly and suggests possible alternatives. A spell checker is a tool that enables us to check the spellings of the words in a text file, validates them i.e. checks whether they are rightly or wrongly spelled and in case the spell checker has doubts about the spelling of the word, suggests possible alternatives. Spell checkers are the basic tools needed for word processing and document preparation. Spell checker may be stand-alone capable of operating on a block of text, or as part of a larger application, such as a word processor, email client, electronic dictionary, or search engine [2].

The spell checking process can generally be divided into three steps: detecting errors, finding correction and ranking correction. 'Error detection' is to verify the validity of a word in the language while 'Error correction' is to suggest corrections for the misspelled word. Ranking is the ordering of suggested corrections in decreasing order of their likelihood for being actual intended word. Spelling error correction can be of two types: interactive, and automatic. In interactive correction, the spell checker can suggest more than one correction for each error and the user has to select one for replacement (e.g. the spell checker in Microsoft Office Word). In automatic correction, the spell checker has to decide on the one best correction and the error is automatically replaced with it (e.g. the spell checker in Google search engine). Automatic error correction is the requirement for those speech processing and NLP related systems where human intervention is not possible [10].

Before errors can be corrected they have to be identified and classified. A proper classification is important in order to know which kind of errors occur. Once this is known it can help to identify methods which are able to correct them. According to Kukich [8], errors are divided into two classes: non-word (or isolated word) and real-word (context sensitive) errors. A non-word error occurs if the original word is not contained in the dictionary (e.g. typing ***hte*** instead of ***the***). A real-word error occurs if the original word is correctly spelled but incorrectly used (e.g. typing ***peace of paper*** instead of ***piece of paper)***.

Problem of automatic spell checking is not new in the areas of Information Retrieval and Language processing. The research started as early as 1960s [7]. Many different techniques for detection and correction of spelling errors were proposed over the last 53 years: dictionary-lookup, n-gram analysis, Soundex, Noisy Channel equation and the like. Some of these techniques exploit general spelling errors trends while others use the phonetics of misspelled word to find likely correct words. Several researches have been done for the languages like English, Arabic, Chinese and few researches have been done for Amharic language, but none for Afaan Oromo.

Afaan Oromo (when translated it means *Oromo Language*) is one of the major African languages that is widely spoken and used in most parts of Ethiopia and some parts of other neighbor countries like Kenya and Somalia [61]. Currently, it is an official language of Oromia regional state. It is used by Oromo people, who are the largest ethnic group in Ethiopia, which amounts to 34.5% of the total population [9].

1.2. Motivation

One of the inevitable activities of any government or private office worker is to edit documents he or she or someone else has written. Spelling and typing errors are abundant in human generated electronic text. Computers have considerably minimized this activity since they automatically detect and correct spelling as well as grammatical mistakes. Thanks to this, office workers not only save considerable amount of time and money but they have also started producing relatively better documents. A spell checker has a vast application, such as Internet search improvement [3, 4], correction of errors caused by Optical Character Recognition (OCR) [5, 6], tools for text editors, and preprocessors for NLP applications. These facts show that spell

checker is the basic and relevant component for the development of most of NLP related applications. With the aforementioned facts and a view of having electronic spell checker for the language, we are motivated to do research on developing Afaan Oromo spell checker (**AOSC**).

1.3. Statement of the problem

Spell checker was developed for languages like Maltese [11, 12], English [13, 14, 15], Chinese [16], Japanese [17, 18], Arabic [19], and Thai [20]. As far as the knowledge of the researcher goes, Afaan Oromo does not have a spell checker system so far. A number of commercial and non-commercial spell checkers and correctors, such as the ones embedded in Microsoft Office Word, Unix SPELL, and other variants, have been extensively studied. However, their techniques of spell checking are still limited in their scope to a few languages. For example, Microsoft Office Word detects all Afaan Oromo words as spelling error and it tries to give a false suggestion based on the English dictionary. Since anyone who needs to process Afaan Oromo using Microsoft Word uses English as the default language, every word that is written is underlined with red color, which makes it difficult to differentiate words which are wrongly spelled (as shown in Figure 1.1).

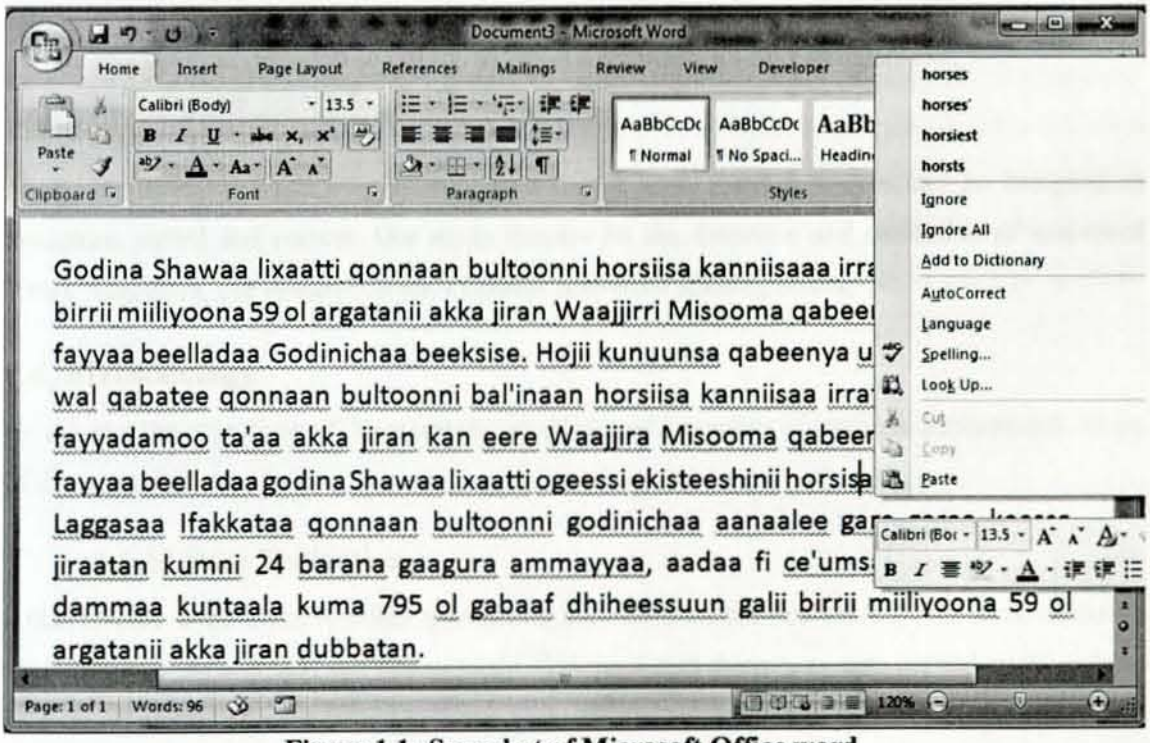


Figure 1.1: Snapshot of Microsoft Office word

Since Afaan Oromo language is used in offices, schools and other media, huge amount of electronic data is produced each day. Thus, the development of spell checker applications for this language is required for easing document preparation task.

1.4. Objectives

1.4.1. General Objective

The general objective of this research work is to develop a spell checker for Afaan Oromo.

1.4.2. Specific Objectives

The specific objectives of the research work will be to:

- Study and analyze the language specific behaviors of Afaan Oromo,
- Study the specific trends of spelling errors in Afaan Oromo text document,
- Identify and adopt one of the appropriate approaches to develop a spell checker,
- Select and customize a spell checker algorithm for use with Afaan Oromo language,
- Design Afaan Oromo spell checker,
- Develop a prototype to evaluate the performance of the designed spell checker and the developed algorithms,
- Evaluate the performance of the developed system with different datasets.

1.5. Scope of the Study

There are different types of spelling errors that a spell checker system can be designed to recognize, detect and correct. Our study focuses on the detection and correction of non-word errors. Therefore, our research doesn't address real-word spelling errors.

1.6. Methodology

To achieve the objectives of the research, we employed a number of methods (techniques). Some of them are discussed below.

1.6.1. Data Collection

Afaan Oromo does not have publicly available annotated corpus text for any NLP task including spell checker. As a result, we collected an electronic text from two state owned news papers: "*Bariisaa*" and "*Kallacha Oromiyaa*" and from the web. Words collected from these articles are

listed down alphabetically and then their possible inflection and derivation were studied. An affix analysis was employed to reduce the size of the dictionary. The goal of affix analysis is to reduce an inflected and derivated word down to its root word and affix. Finally, corpus of size more than 4, 000 was developed. Accordingly, we prepared a test corpus of size around 13, 811 words. Since the classification of root word is needed for the morphological generation part of our spell checker, we did additional work to manually classify the collected root words into different parts of speech (POS).

1.6.2. Literature Review

Different literatures that are considered to be relevant for the research were reviewed and adopted for our work. Almost all of these are recent research areas and are extensively researched particularly in languages such as English, Indian Languages, and others. How the Hunspell spell checker [79] works will be studied to understand the way in which spell checking can be performed. Afaan Oromo word formation rules, spelling error pattern and related works were reviewed to properly design and develop the desired algorithm. In addition, a discussion was made with Afaan Oromo Linguistic experts regarding the linguistic nature of the language like word formation rules, and the grammatical structure.

1.6.3. Development Environment and Tools

Windows operating system is selected as a development environment due to its popularity. Accordingly, Microsoft Visual C# programming language was selected.

1.6.4. Prototype Development

A prototype will be developed in order to measure the performance of the developed system.

1.6.5. Conducting Experiments

After the prototype will be developed, it will be tested and evaluated with different test datasets. The performances obtained throughout the conducted experiments will be given in three evaluation metrics: Lexical recall, Error recall and Precision.

1.7. Application of Results

As outlined in Section 1.2, spell checker is useful in many areas of NLP. The beneficiaries of the results of this study include, anyone who wants to process Afaan Oromo document and

researchers involved (or want to be involved) in increasing computers capability of processing Afaan Oromo Language. The study could also be used:

- As a component for higher forms of NLP, such as machine translation, grammar checker, information storage and retrieval of textual data, automatic speech recognition, OCR, etc.
- For learning and teaching Afaan Oromo

1.8. Organization of the Thesis

The rest of the thesis is organized as follows. Chapter 2 discusses literature review on different issues in developing a spell checker. In this chapter, types of spelling errors, issues related with spell checker, and the different techniques and approaches to develop spell checker are discussed. Chapter 3 is devoted to discuss related works done for different languages. Chapter 4 specifies the morphological properties of Afaan Oromo. Many language specific issues such as the writing system, word formation process like inflections, derivations and compounding have been extensively presented. Chapter 5 discusses design requirements and architectural design of **AOSC**. This chapter also details root word classification issues, algorithms and implementation of rules. Finally, the implementation of the prototype of **AOSC** is discussed. Chapter 6 is devoted to the evaluation of the system. Chapter 7 concludes the thesis by outlining the benefits obtained from the research work and limitations of the system. It also shows some research directions and recommendations that can be accomplished in developing a full-fledged **AOSC**.

Chapter Two: Literature Review

2.1. Introduction

A spell checker is a tool that enables us to check the spellings of the words in a text file, validates them i.e. checks whether they are rightly or wrongly spelled and in case the spell checker has doubts about the spelling of the word, suggests possible alternatives. The two core functionalities provided by a spell checker are: spelling error detection and spelling error correction. 'Error Detection' is to verify the validity of a word in the language while 'Error Correction' is to suggest corrections for the misspelled word.

This chapter presents a review of the concepts that are important for obtaining basic understanding of the ideas on the research area. The review aims at forwarding fundamental points for the design and implementation of a spell checker. It includes review of spell checking techniques. It also reviews and discusses the different types of spelling errors, issues related with spell checker, and the like.

2.2. Types of Spelling Errors

Until recently, most of the spelling correction techniques were designed on the basis of spelling errors trends (also called error patterns) and many studies were performed to analyze the types and the trends of spelling errors. Most famous among them are the studies performed by Damerau [7] and Peterson [21]. The majority of the early research done in the area of spell checking was based on these studies. According to these studies spelling errors are generally divided into two types, typographic errors and cognitive errors.

2.2.1. Typographic errors

Typographic errors occur when writer knows the correct spelling of the word but mistypes the word by mistake. These errors are mostly related to the keyboard and therefore do not follow linguistic criteria. A study by Damerau [7] shows that 80% of the typographic errors fall into one of the following four categories:

1. Single letter insertion, e.g. typing **anama** for **nama**
2. Single letter deletion, e.g. typing **nma** for **nama**
3. Single letter substitution, e.g. typing **noma** for **nama**

4. Transposition of two adjacent letters, e.g. typing **anma** for **nama**

The results of a study by Peterson [21] are shown in Table 2.1, which confirms the above categories. The data sources were Webster's Dictionary (WD) and Government Printing Office (GPO) documents retyped by college students.

Table 2.1: Comparison of Typographic errors

	GPO	WD
Transposition	4 (2.6%)	47 (13.1%)
Insertion	29 (18.7%)	73 (20.3%)
Deletion	49 (31.6%)	124 (34.4%)
Substitution	62 (40.0%)	97 (26.9%)
Total	144 (92.9%)	341 (94.7%)

The rows in Table 2.1 correspond to the four basic types of errors; the columns correspond to the two sources of data. For each data source, the number and percentage of each type of errors is given. The last row contains total number and percentage of single errors. The most common of these typographic errors is the substitution error (as shown in 4th row of Table 2.1). Substitution error is mainly caused by replacement of a letter by some other letter whose key on the keyboard is adjacent to the originally intended letter's key. A study by Kukich [8] showed that 58% of the errors involved adjacent typewriter keys. Insertion errors can occur due to a double pressing of a key or by accidentally hitting two adjacent keys while trying to hit one of them. According to Damerau [7] deletion and transposition errors usually occur when the eyes move faster than the hand.

2.2.2. Cognitive errors

Cognitive errors (also called orthographic errors) occur when a writer does not know or has forgotten the correct spelling of a word. It is assumed that in the case of cognitive errors, the pronunciation of misspelled word is the same or similar to the pronunciation of intended correct word. For instance, typing "**midagduu**" as "**midhagduu**" 'beautiful'.

From the above discussion on types of spelling mistakes, it is evident that there is a reasonable likelihood of both typographic and cognitive errors to occur in a given document.

2.3. Implementation aspect of a Spell Checker

A spell checker is comprised of mainly two modules, a lexicon (or dictionary) and a bunch of algorithms or techniques that use this lexicon for spell checking. These techniques generally provide three types of functionalities:

1. Lexicon lookup, i.e. error detection
2. Finding approximate matches in the lexicon, i.e. Error correction
3. Ranking of corrections.

Some spell checkers use same technique to provide all three functionalities in one step, while others use different techniques for each functionality. Lexicon lookup techniques are generally dependent on the structure of the lexicon, while the design of storage structure is sometimes governed by the choice made for error correction technique. Therefore, techniques for the above mentioned functionalities are highly dependent on each other and on the structure of the lexicon.

2.4. Issues related with Spell Checker

There are several issues to be addressed in the design of spell checker systems. The first issue concerns the error patterns generated by different text generating media such as typewriter and computer keyboard, typesetting and machine printing, OCR system, speech recognizer output, and of course, handwriting. Usually, the error pattern issue of each media concerns the relative abundance of insertion, deletion, substitution and transposition error, run-on and split word error etc. The knowledge about error pattern is necessary to model an efficient spell checker. Another important issue is the computerized dictionary which concerns the size of dictionary, the problem of inflection and creative morphology, the dictionary file structure, dictionary partitioning, word access techniques and so on.

2.5. Isolated-Word error Detection

A string of characters separated by spaces or punctuation marks may be called a candidate word. A candidate word is a valid word if it has a meaning, else it is a non-word. The two main techniques that have been explored for non-word errors detection are n-gram analysis and dictionary lookup.

2.5.1. Dictionary Lookup Method

A dictionary lookup method for non-word errors detection initially appears to be a simple matter of looking up each word in a word-list or lexicon and flagging those that are not found in the list as potential misspellings [36]. However, this is not as straightforward as it seems at first. The two main challenges of this approach are: a lexicon containing all correct words could be extremely large, resulting in a need of more space and inefficiency searching time, and for morphologically complex languages, it is practically impossible to list all correct words. Hence, instead of storing the word as it is in the lexicon, some sort of rules can be applied to reduce a given word into its root word [37]. This can be done by storing only root words in the lexicon including prefix and suffix information. Then, rules can be applied on the root words of the lexicon by using prefix and suffix information to generate derived words. To use a dictionary lookup technique, we need to be careful on the lexicon size and usage of efficient lookup algorithm. The most significant dictionary lookup techniques are hashing, binary search trees, and finite-state automata.

Hashing

As discussed in [38, 39, 40] hashing is a well-known and efficient lookup strategy. The basic idea of hashing relies on some efficient computation carried out an input string to detect where a matching entry can be found. More specifically, hashing is a technique used for searching an input string in a pre-compiled hash table via a key or a hash address associated with the word, and retrieving the word stored at that particular address. If collisions occurred during the construction of the table, a hash function tends to produce identical or similar hash addresses; the lookup process will be slower as a result.

In the spell checker context, if the word stored at the hash address is the same as the input string, there is a match. However, if the input word and the retrieved word are not the same or the word stored at the hash address is null, the input word is indicated as a misspelling. The random-access nature of hash tables eliminates the large number of comparisons required for lookups. Thus, it is a faster searching technique compared to sequential search or even tree-based search, especially for searching in a large data representation. The drawback of a hash table is that without a good function, it would require a big hash table in order to avoid collisions. Hash tables provide O(1)

constant lookup time on average. In the worst-case scenario, hashing requires the time complexity of O(m), where **m** is the number of words in a dictionary or corpus.

Consider a hash table **B [0, n)** (i.e. a bit array of n elements ranging from 0 to n-1) for spelling checker that has **n** bits which are initially set to **0**, and a word list **W [0, r)** that contains a set of **r** word. Let $w_j \in W, j=0...(r-1)$ be the **j**th word in the word list. Assume that $n>r$. Each word $\mathbf{w_j}$ is hashed by **k** different hash functions $\mathbf{h_i}$, where i=1…k. For each word $\mathbf{w_j}$ and each hash function $\mathbf{h_p}$ (where p=1…k), the bit $\mathbf{B[h_p(w_j)]}$ is set to **1**. Figure 2.1 describes how the bit vector is initialized, and Figure 2.2 shows how bit vector is checked to see if a given word is correctly spelled using a hash function. The first loop in Figure 2.1 initializes the hash table **B** to zero. The next loop enters hashed information about each word list into the table. Essentially, it uses the **k** hash functions to compute k integers to serve as k indices into table B. At each index of table B, the table's value is set to 1.

```
Function SpellHash (B, W, n)
 Start
        B: =0;
          For w ∈ W
             Do
               For i ∈ [1, k]
                 B[hi(w)]=1
           Return
End
```

Figure 2.1: Algorithm to initialize a bit vector B

```
Function SpellLookup (B, w)
Start
      For i∈[1, k]
        Do
         If B[hi(w)]= 0
             Return false
         If B[hi(w)]=1 break
             Return true
End
```

Figure 2.2: Look up Algorithm

As depicted in Figure 2.2, to check if a word or an affix-stripped word **x** is in **W**, we need to check whether all $\mathbf{B[h_p(x)]}$ are set to 1 for p=1…k. If all $\mathbf{h_p(x)}$ bits are set to 1, we assume that x is in **W** and the algorithm returns true. If not, x is reported as a misspelling and false is returned.

Binary search trees

Binary search trees, particularly median split trees (MSTs), have been used for dictionary lookup and subsequently, for spell checking [41]. Binary search is useful for checking if a particular string exists in a large set of strings, i.e. the chosen dictionary. A median split tree is a modified frequency-ordered binary search tree. The main goal of a median split tree is access to high-frequency words faster than to low-frequency words without sacrificing efficiency of operations, such as word lookup, insert, and delete. MSTs are essentially binary search trees where each node of a split tree contains a node value and a split value [41]. A node value is a maximally frequent key values in that sub tree. A split value partitions the remaining keys between the left and right sub trees in a lexical order, which is to say that it is the lexical median of a node's descendants. Hence, the tree is perfectly balanced.

MSTs are data structures for storing dictionaries, lexicons or corpora with highly skewed distributions. This is to say that, they are particularly useful for efficiently finding words when the underlying frequency distribution of the occurrence of these words is highly skewed. A MSTs is constructed from a set of keys, linear order of the keys according to their frequencies of occurrence in the input while as the lexical ordering is a strict linear order on the keys' representation. Thus, the cost of lookup is determined by both the frequency distribution and the lexical ordering of the keys. The build time is O(nlogn), where **n** is the number of strings. MSTs require log(n) searches to locate the looked up word within a set of n strings. It is also significantly more efficient compared to the lookup time of a linear search technique on a large data representation, although it is generally slower compared to the lookup time of hashing [41].

Finite State Automata

Finite State Automata (FSA) as presented in [43, 44], have been used as a basis for string-matching or dictionary-lookup algorithms that locate elements of a dictionary within an input text. FSA is used to represent a language which is defined to be a set of strings, each string being a sequence of symbols from some alphabet. Consider an FSA that is required to represent any

string in the language **U*(w)**, where **U** is a finite alphabet of symbols, and **w** is some word consisting of **i** symbols in ***U*** *(i.e. |w|=i)*. The language is thus any string of symbols from **U** that has **w** as a suffix. Such an FSA, FSA (w) = (Q, q_0, A, T) can be defined as follows:

- **Q** is the set of all states corresponding to the prefix of **w**. Thus, Q can be represented as {∈, w [0], w [0…1], …, w[0..c-2], w}
- $\mathbf{q_0} = \in$ is the start state
- In general, **A** is the set of accepting states where $A \subseteq Q$. In this particular case, A= {w}
- **T** is the transition function which maps a state and an input symbol to another state.
  Let $q \in Q$, where q is a prefix of w and let $c \in U$, where c is a character in the alphabet. Then $(q, c, qc) \in T$ if and only if **qc** is also a prefix of **w**. Otherwise, $(q, c, s) \in T$, where **s** is the longest suffix of *qc* that is also a prefix of qc that is also a prefix of **w**.

Such an FSA would only terminate in the final state if the last **i** symbols of the input of **n** symbols ($n \geq i$) constituted the word **w**. The state transition function can be visually represented by the well-known state transition diagrams. If the transition function is stored as a table, then the search time is O(n).

2.5.2. N-gram analysis

N-grams are n-letter sub sequences of words or strings where n usually is one, two or three. One letter n-grams are referred to as unigrams or monograms; two letter n-grams are referred to as bi-grams and three letter n- grams as trigrams. An example of a bi-gram analysis of the word ***mana*** 'house' would give the 2-gram set {**ma, an, na**}. N-gram analysis uses a large corpus of text from the desired language by generating a character n-gram from the list [26]. Using this technique, strings that contain unusual n-grams can be identified as possible spelling errors.

According to Kukich [8], n-gram analysis technique is very useful for detecting errors occurred in machine-generated texts such as texts generated by OCR. Its main advantage is that it works without a lexicon. However, for human generated errors, most spell checkers rely on dictionary lookup for error detection, and some applications use a hybrid of these two methods.

2.6. Spelling Error Correction

Spelling error correction can be of two types: Isolated-Word error correction, and Context-based error correction. In Isolated-Word error correction, a misspelled word is analyzed in isolation without giving any consideration to its context; therefore the corrections are based only on the misspelled word itself. In Context-based error correction the context of error is also taken into consideration for suggesting and ranking corrections. The second approach is particularly useful for correcting real-word errors. Early work in the area of spell checking was more focused on isolated-word error correction, but with the passage of time, the number of such applications increased where auto-correction was a requirement, for example in applications like text to speech synthesis systems, and Machine Translation systems. In the next subsections non-word spelling error correction techniques will be discussed.

2.6.1. Isolated-Word Error Correction

In the last 53 years, many different techniques have been invented for isolated word error correction. A good many of these techniques have been successfully implemented and are being used in different applications, but none of them has yet been succeeded in achieving near 100% accuracy without human involvement. Most of the earlier works used rule based, that means error patterns were used for finding corrections. In the recent works, probabilistic models are used for error correction which instead of subjectively making use of error patterns, use the erroneous data to automatically train the model according to the error patterns present in training data [23, 24]. Generally, Isolated Word Error Correction techniques can be divided into the following subcategories:

1. Edit distance techniques
2. N-Grams based techniques
3. Phonetics based techniques
4. Noisy Channel model
5. Rule based

2.6.1.1. Edit Distance Techniques

According to Kukich [8], the most studied spelling correction algorithms are the ones calculating edit distances between the misspelled word and words in the dictionary. Edit distance is the number of simple edit operations required to transform one string to another. The different

operations allowed are substitution of a letter, deletion of a letter, insertion of a letter, and transposition of two adjacent letters. The less the edit distance between two strings is, the more similar are the strings to each other.

a. Damerau's Single Errors Technique

As discussed in Section 2.2.1, research by Damerau [7] showed that 80% of spelling errors belong to one of the four classes of single errors: single letter insertion, single letter deletion, single letter substitution, and transposition of two adjacent letters. On the basis of this observation, he proposed a single error spelling correction technique. He generated all those words for a misspelling from which the misspelled word could be derived by applying any of the four error operations mentioned above. This means that if the length of a misspelled word is **n** and the number of alphabets in the language is **m** then there will be n possible deletions, (n-1) transpositions, (m-1)n substitutions and m(n+1) insertions. Thus a total of (2m+1) n+m-1 words will be generated. For English and Afaan Oromo this number becomes 53n+25.

Once the words are generated they are tested against dictionary for being correct words in the language. In order to increase the efficiency of this correctness test, an optimization was proposed and implemented by the designers of a Swedish spell checker STAVA [25]. Prior to dictionary look up, they tested the words using n-grams tables. These tables contain all those character sequences of length **N** that are allowed in the language. If a word contains a sequence that is not allowed in the language, the word is rejected without consulting the dictionary. In this technique the candidate words cannot be ranked as most or least likely corrections. The suggestions are all on the same level.

b. Levenshtein Distance

The Levenshtein distance technique, like Damerau's single spelling error technique, measures the distance between two character strings in terms of insertion, deletion, substitution and transposition, but it is comparatively more generic because it allows multiple errors. These algorithms allow multiple errors and use the Damerau-Levenshtein metric to calculate the similarity between two words **x** and **y**. This way, all entries in the dictionary can be ranked after their proximity to the query word. The metric is, according to Pfeifer et al. [26], given by:

$\mathbf{d}(o, o) = 0$ // the distance between the same character

$$\mathbf{d}(i, j) = \mathbf{min}[d(i-1,j) +1, \mathbf{d}(i, j-1) +1, \mathbf{d}(i-1,j-1) + \mathbf{c}(x_i, y_j), d(i-2, j-2) + \mathbf{c}(x_i, y_{j-1}) + 1] \quad \textbf{(2.1)}$$

The function **d** gives the minimum edit distance between the two words **x** and **y**, or in other words, the minimum number of edit operations required to transform **x** to **y**. If the length of **x** is **|x|**, and the length of **y** is **|y|**, then the minimum edit distance between **x** and **y** can be expressed as **d (|x|, |y|)**. $\mathbf{X_i}$ denotes the $\mathbf{i^{th}}$ letter of the word **X**. The function **c** is defined as:

$$\mathbf{c}(x_i, y_j) = 0 \text{ if } x_i = y_j \quad \text{or} \quad \mathbf{c}(x_i, y_j) = 1 \text{ if } x_i \neq y_j \quad \textbf{(2.2)}$$

The first expression in the **min** function in Equation 2.1 regards insertion, the second omission, the third substitution, and the last transposition. The Damerau-Levenshtein metric is most commonly calculated using dynamic programming (DP). Wagner and Fischer's algorithm [27] is one of the best known, allowing the edit operations insertion, substitution, and omission. It was later extended by Lawrence and Wagner [28] to also include transposition. The time complexity for calculating the edit distance between two words **x** and **y** with their DP approach is **O**(|x||y|). A zero-indexed matrix with dimensions (|x|+1) × (|y|+1) is used. The matrix is filled according to Equation 2.1 either one column or one row at a time. The matrix element at (|x|, |y|) gives the minimum edit distance between the two words x and y. Table 2.2 shows an example dynamic programming matrix for calculating the edit distance between the two names **Filips** and **Phillips**. The distance is in the lower right hand corner of the matrix, i.e. 3.

Table 2.2: Edit distance between Filips and Phillips using DP

	€	P	H	I	l	L	i	p	S
€	0	1	2	3	4	5	6	7	8
F	1	1	2	3	4	5	6	7	8
I	2	2	2	2	3	4	5	6	7
L	3	3	3	3	2	3	4	5	6
L	4	4	4	3	3	3	3	4	5
P	5	4	5	4	4	4	4	3	4
S	6	5	5	5	5	5	5	4	3

In general, the whole dictionary has to be processed in order to find the possible matches to a word, which makes matching rather slow.

2.6.1.2. N-gram similarity measure

It was based on the principle that the more n-grams they share (i.e. the candidate word and the erroneous word) the more similar they are. In a method given by Pfeifer [26] a similarity coefficient δ can be calculated by dividing the number of common n-grams of the two strings by the total number of n-grams in the two strings. Bigrams are most commonly used, along with leading and tailing blank. This addition of blanks result in assignment of extra importance to start and end of the word, as the starting and ending letter will individually form bigrams.

N-gram techniques do not show good performance on short words. For example, when using trigrams, words of length three will share no trigram with themselves just after introduction of a single error. To overcome this drawback, n-grams of different lengths are used for words of different lengths. For very short words of length three or less, unigrams are used. A study done by Freund et al. [33] shows that n-gram similarity measure works best for insertion and deletion errors, well for substitution errors, but very poor for transposition errors.

2.6.1.3. Phonetics Based Techniques

These techniques work based on the phonetics of the misspelled string. The target is to find such a word in a dictionary that is phonetically closest to the misspelling.

a. Soundex

Soundex was the first phonetic string-matching algorithm developed by Odell and Russell [34]. It was originally developed to match names. The idea was to assign common codes to similar sounding names. The length of the code is four and it is of the form *letter, digit,digit and digit*. First letter of the code is same as the first letter of the word. For each subsequent consonant of the word, a digit is concatenated at the end of the code. All vowels and duplicate letters are ignored. The letters *h*, *w* and *y* are also ignored. If the code reaches the maximum length, extra characters are ignored. If the length of the code is less than 4, zeros are concatenated at the end. Digits assigned to the different letters for English are shown in Table 2.3.

Table 2.3: Soundex codes

1	b, f, p, v
2	c, g, j, k, q, s, x, z
3	d, t
4	L
5	m, n
6	R

For example, Soundex code for **Chala** is C400 and for **Chaltu** is C430. This Soundex algorithm was actually designed for names; therefore its performance for spell-checking is not very good. A study by Stanier et al.[29] shows that even for name searching one fourth of the relevant words go undetected and only one third of the detected words are actually relevant. The large size of group 2 in Table 2.3 might be a reason for this poor performance. Soundex was refined for use in spelling correction. The major change was the breakdown of group 2 in Table 2.3 into smaller groups. Refined Soundex code is shown in Table 2.4.

Table 2.4: Refined Soundex code

1	b, p
2	f, v
3	c, k, s
4	g, j
5	q, x, z
6	d, t
7	L
8	m, n
9	R

The Soundex algorithm had many problems. First, it gives no importance to the sounds produced by letter groups, for example, sounds produced by ***ch*** and ***sh*** in English. Second, it retains first letter of the word. If an error occurs at word-initially the Soundex will not be able to correct it.

Different studies show that chances of an error at first position of word are very small. Peterson [35] tried to solve these problems by designing many variants of Soundex using different length codes, multiple codes for same word and overlapping groups of alphabet. Those letters, which could produce more than one sound, for example **c** and **g**, were included in more than one group. As a result multiple codes can be generated for same word. This increased the accuracy of Soundex.

b. Phonix

Phonix is a Soundex variant. Like Soundex, it assigns codes to letters but prior to code generation, string is standardized by applying some letter group's substitutions. This process is also called N-gram substitution [22]. About 160 letter group transformations are used. For instance, the sequence ***tch*** is mapped to ***ch***, and ***ph*** is mapped to ***f***. In some implementations, different transformation rules are written for some letter group in different positions [30]. Table 2.5 shows phonix codes.

Table 2.5: Phonix codes

1	b, p
2	c, g, j, k, q
3	d, t
4	L
5	m, n
6	R
7	f, v
8	s, x, z

c. Editex

Editex is a combination of edit distance technique and letter grouping property of Soundex and Phonix [31]. Editex forms overlapping groups of letters. To compute edit distance for Editex, **0** is added to edit distance if the corresponding letters of two strings are similar; 1 is added if they belong to the same 'Editex-group' and 2 otherwise. This approach assigns smaller edit distance value to phonetics based substitution edits as compared to any arbitrary substitution. Letter groups of Editex are shown in Table 2.6.

Table 2.6: Editex Letter's group

1	b, p
2	c, k, q
3	d, t
4	l, r
5	m, n
6	g, j
7	f, p, v
8	s, x, z
9	c, s, z

2.6.1.4. Noisy Channel Model

All the techniques discussed above rely on some sort of distance or similarity measure to find closest match. A drawback of using such measures is that certain important factors which affect the error patterns are ignored. Since spelling error trends depend on many different factors and these similarity measures make use of at most one (or sometimes none) of them. These measures become biased or insufficient. For example, phonetics based techniques give no weight to keyboard adjacencies. According to the Soundex code, "**find**" cannot be a candidate correction for "**gind**", because they have different codes, although there is a significant chance that **f** is substituted by **g** due to keyboard adjacency.

Another more generic and perhaps more appropriate way of doing this is to construct a probabilistic error model which can be trained for different languages and domains thus automatically setting its parameters. This is called 'Noisy Channel Model' of error correction [32]. The idea behind this name comes from considering the phenomenon of making spelling mistakes as the process of sending text through a noisy communication channel, which introduces errors in the text. Our task is to find the most probable transmitted word (correct dictionary word) for a received erroneous string. If the behavior of the noisy channel is properly modeled, a correct guess can be made of actual intended word by decoding the error pattern. The model assigns a probability to each correct dictionary word for being a possible correction of the

misspelling. The word with highest probability is considered the closest match (or the actual intended word). The general model of Noisy channel equation is shown in Figure 2.3.

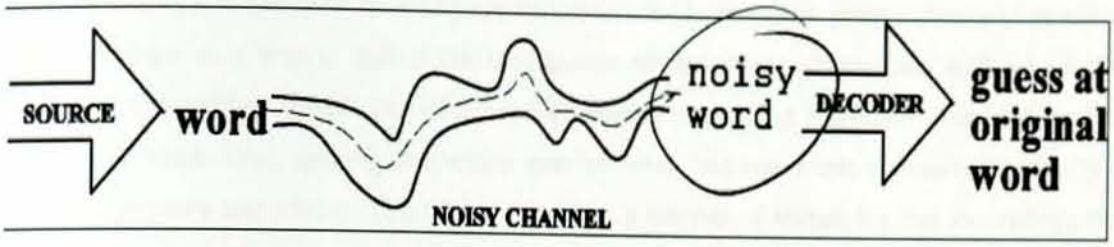


Figure 2.3: Noisy Channel Model

This way of identifying the actual form from an observed or surface form is called Bayesian classification. As stated by Jurafsky et al. [1] the problem is formally stated as follows: Let ***D*** be a dictionary and w_i be any word in *D*, for a misspelled string $s \in D$, our task is to find such $w \in D$ for which P(w|s) is maximum. Hence,

$$w = \text{argmax}_{wi} P(w_i|s) \quad (2.3)$$

Applying Bayes' rule we can rewrite this probability expression as:

$$P(w|s) = (P(s|w)P(w))/P(s) \quad (2.4)$$

Since we are maximizing **P(w|s)** over all dictionary words for our single observation string **s**, we can ignore **P(s),** as it will remain constant during single correction process. Hence, to maximize **P(w|s)** we have to maximize only **P(s|w)P(w)**. The first of these terms **P(s|w)** is the probability of typing string **s** when word **w** was intended. This probability is called *a posteriori* probability or channel model. **P(w)** is the probability that a writer will type **w** from among all the dictionary words. It is called source model or language model. According to Kernighan [32], this technique does not deal with multiple errors.

2.6.1.5. Rule-Based Techniques

Rule-based techniques attempt to represent knowledge of common spelling error patterns in the form of various rules for how to transform a misspelled word into a valid one [59]. It does not require a stored lexicon for correcting spelling errors.

Chapter Three: Related Work

In writing text, it is desirable to produce a document that is free from grammatical and spelling errors, perhaps as a way to reflect the intelligence of the author or the care with which the document was written. Indeed, one of the earliest papers on spelling correction was published in 1962 [45]. Since then, spelling correction systems have become more pervasive, typically a feature of modern text editors. This Chapter provides a number of researches that incorporate the techniques described in Chapter 2 for finding spelling errors and suggesting corrections.

3.1. Spell Checker for Indian languages

There are many spell checkers developed for Indian languages using different techniques. This section provides brief discussion of some available spell checkers for Indian languages.

3.1.1. Spell Checker for Punjabi

Punjabi is the mother tongue of more than 110 million people of Pakistan (66 million), India (44 million) and many millions in America, Canada and Europe [46]. Punjabi is written in Gurmukhi script. Design and implementation of spell checker for Gurmukhi-SUDHAAR was developed by Rupinderdeep Kaur and Parteek Bhatia at Thapar University, Patiala [46]. There are two different applications in the system. The first application creates the dictionary, which is executed once to create the dictionary. The second application is designed for spell checking of Punjabi text where user gives the input Punjabi text and the system detects the errors by looking up for that word in the dictionary and provides the suggestions for those errors. Then the user can select the suggestion from the list and make changes accordingly.

They created a corpus of around one million words, which contains unique Punjabi correct words, which will act as the dictionary for the spell checker. A dictionary look up technique is used for error detection, which checks each word of input text for its presence in the dictionary. If the word is in the dictionary, then it is a correct word otherwise it is put into the list of error words. In order to provide correct suggestions for errors in any Punjabi text, they performed error pattern analysis. First, they identify commonly encountered Punjabi typing and spelling errors like insertion error, deletion error, substitution error, transposition error, run-on error, and split word error. Then they used minimum edit distance technique for generating and ranking the

suggestions. Finally, they have tested the system by first giving the input from a well known Punjabi newspaper where their system has given zero errors because that text was already tested for its correctness. Then they intentionally made hundred errors in the same text and when that text is given as an input, it showed approximately 80 errors. It showed the correct word in suggestion list for approximately 70 words out of 80 that shows the accuracy of 85% for error correction and 80% for error detection.

3.1.2. Malayalam Spell Checker

It is a sub system that can be executed with Microsoft word as a macro or as the Malayalam editor, developed by Tiruvananth Puram, to check the spelling of words in a Malayalam text file [47]. While running as a macro in a word, it functions as an offline spell checker in the sense that one can use this software with the previously typed text file only. Both offline and online checking are possible when it is integrated with the text editor. It generates suggestion for wrongly spelled words. The system is based on a dictionary look up approach. In order to reduce the size of dictionary they only store the root word. Morphological analyzer developed by other researchers was integrated into the new system. This module splits the input words into root word, suffixes, post positions etc. The error detection module checks the validity of each word using the rule database, and finally it checks the dictionary to find whether the root word is present in the dictionary. If anything goes wrong in the checking it is detected as an error and the error word is reprocessed to get 3 up to 4 valid words which are displayed as a suggestion.

3.1.3. Spell Checker for Kannada

In their work Murthy et al. [37] designed and implemented a non-word Kannada spell checker using a morphological analyzer and dictionary lookup method. In order to build spell checker for Kannada they identified that, one would require a powerful dictionary for reference in the word error detection and suggestions prediction phases. Creating a dictionary having all the words of the language is a laborious task considering the large variation of affix combinations in Kannada. To handle this problem, they used a morphological analyzer and a dictionary of only the root words. Further, they divided the dictionary into twenty seven sub-dictionaries each having related words and mapping to a particular paradigm table. The division or grouping is done based on a set of rules.

A paradigm table is a set of suffixes that creates inflections for all the words of a sub-dictionary of root words. There are also twenty seven paradigm tables having one-to-one correspondence with sub-dictionaries of the roots. Initially, the input word is searched in the root trie, if the input word does not exist in the dictionary as a root word, then the algorithm will employ a morphological analyzer to strip all the suffixes and get the root word. The root word is searched in the selected dictionaries once again. If the search is unsuccessful then the error is reported. They used Levenshtein edit distance to generate suggestions for all types of non-word errors. To improve the searching efficiency they used a trie data structure. A *trie* is a data structure that stores the information about the contents of each node in the path from the root to the node, rather than the node itself. The tool has been tested extensively for various test cases. The tool is found to provide correct results for both single word and also for a block of text. Test cases were designed to cover all types of errors.

3.2. Spell Checker for Arabic

Arabic is considered as one of the oldest languages in the world, and it is ranked fifth in the world [48]. Unlike English and other languages, Arabic text is oriented right to left without the use of capital letters. In addition, Arabic differs from other languages syntactically, morphologically, and semantically which makes it one of the most difficult languages for written and spoken language processing [49]. Spell checkers for Arabic languages have been developed by a number of researchers at different times using different techniques. Among such efforts, it is worth considering the works of Hasan Muaidi and Rasha Al-Tarawneh [48].

Their spell checker is based on n-gram scores. For this purpose, they built eleven matrices to present the combination between the Arabic letters word, with the assumption that the longest valid word in Arabic language has 12 letters. Each matrix concerns in the connection between a 2-grams letters. Each cell in the generated matrix is assigned an integer value 2, 1 or 0. The cell is assigned the value 2 if the word is ended by these two letters and assigned 1 if there is a connection and the word is not over yet, and is assigned 0 otherwise. On the other hand searching process for any word was done by extracting each pair of letters in the word then examines the value for each pair when the corresponding value is zero. Then the spell checker will consider the test word as wrong, otherwise it will check if it is assigned with 1 (which

indicates that there is a connection), and it will continued until it reach the value of 2, to determine that the word is correct.

Finally, they used corpus of around 101,987 word types to train and test the proposed spell-checker. The main purpose of their test is to determine the number of correctly spelled words accepted by the system. To test the performance of the proposed system, they calculated accuracy using the success rate measure (SR). This measure is calculated as $SR = (C_W/N)*100\%$, where C_W is the number of valid words that are accepted correctly, and **N** is the size of the testing dataset. The proposed spell checker system accuracy was 99.7% using the training dataset. While the accuracy of the proposed system reached 40.4% using the testing dataset. The above discussion clarifies that the accuracy mainly depends on the number of words in the corpus, i.e. to increase the accuracy of the proposed spell checker; the number of words should be increased. The evaluations of the works have also shown that the system recognizes many of incorrect words as correct words.

3.3. Spell Checker for Filipino

Filipino is the national language of the Philippines. A spell checker for Filipino (*SpellCheF*) was developed by Cheng et al. [50]. The researchers used a hybrid approach in detecting and correcting misspelled words in a document. Their approach was composed of a dictionary-lookup, n-gram analysis, Soundex and character distance measurements. SpellCheF is composed of three modules, namely, the lexicon builder, the detector and the corrector. These three modules used both manual-formulated and learned rules to carry out their tasks.

The system was tested using three different documents. The test results showed that the lexicon builder was able to correctly categorize words based on the spelling rules used. The detector module had an overall error rate of 7% in identifying misspellings. The corrector module had a 94% accuracy rate in generating word suggestions. The results also showed that using the Soundex code, more suggestions were generated compared to the use of n-gram analysis.

3.4. Spell Checker for English

English is one of the well researched languages in the area of natural language processing. Spelling checker systems for English were developed by many researchers and now they are

integrated with so many search engines and commercial software like Microsoft Office Word and Office.org in UNIX operating system. This section discusses few of them.

3.4.1. SPELL

The UNIX® spell checker SPELL [51] was originally written by S. C. Johnson in 1979 and was subsequently re-written several times. It takes an input file and breaks the file content into words. These words are sorted in alphabetical order and duplicates are discarded. The sorted words are then compared to a dictionary. Words that are not found in the dictionary are flagged and reported. As Bentley [52] pointed out, in McIlroy's version of SPELL, a superior word list was used instead of a simple desk dictionary. This word list was built and compiled from various sources, such as the Brown Corpus and Webmaster's New International Dictionary of the English language. This word list contains 75,000 words. An affix analysis was employed to reduce the size of the dictionary. The goal of affix analysis is to reduce an inflected word down to its word stems. The affix analysis reduced the original 75,000 list to a 30,000 word list. The lookup process starts by affix stripping the word being searched for, and the word itself and the stem are looked up in the stop list. If a match is found, a stop flag is attached to it. The word is also looked up in the original word list. If the word has a stop flag, it is assumed to be correct if it also appears in the original word list. Words that neither occur among, nor are derivable from (by applying affix analysis) words in the word list are output as misspelling. The SPELL had an overall error recall rate of 100% in identifying misspellings. SPELL only addresses the spelling error detection problem.

3.4.2. CORRECT

CORRECT was devised by Kernighan, Church and Gale [53] and further research was carried out by Church and Gale [54]. The CORRECT program combined a lookup in the word list and probabilistic techniques for correcting single-error misspellings. CORRECT is designed to take a list of lower-case words rejected by the UNIX® SPELL program, generate candidate corrections for these misspelled words, and sort these candidates by their probabilities. In CORRECT, it is assumed that a misspelled word is the result of a single error. Moreover, CORRECT assumes that such single spelling errors occur with particular probabilities. For instance, it may be more likely that "**ed**" is derived from a transposition of "**de**" rather than from a substitution of "**ad**".

In seeking a candidate correction that differs from the input (i.e. misspelled word) by a single spelling error transformation, CORRECT searches a particular word list. This list was composed from various sources, such as word list in SPELL and corpus collected from various sources. The lists of suggested words are ranked by probability scores using a noisy channel model and Bayes'rule [55]. Thus, in order to score the candidate corrections, the probability of occurrence of the suggested word is multiplied by the probability of occurrence of the specific error by which the suggested word differs from the misspelled word [53]. The limitation of CORRECT is that it only addresses the spell correcting problem; the spell checking problem was taken care of by the SPELL program.

3.5. Spell Checker for Chinese

As stated by Li et al. [56] there are significant differences between Chinese and English. Chinese does not have evident word boundaries, such as a white space character, as in English. The Chinese character set contains more than 6,700 characters. Thus, before any analysis can be performed on a Chinese text, it must be segmented into words. CInsunSpell which was developed in 2002 is a more recent Chinese-specific spell checking and correcting system [57]. It achieves spell checking and spelling correction in two different processes. For spell checking, character trigrams within a fixed-size window are used to locate the misspelled words in a specific area. This process reduces memory consumption as the Chinese character set is relatively large. Edit distance techniques are adopted by CInsunSpell for spell correcting in order to obtain the information of word construction in a set. CInsunSpell thus achieves the spell correcting task by comparing task by combining the language model construction process (applying Bayes' rule, minimum edit distance technique, and the weight distribution calculation). The suggestions are output in descending order according to the possibilities of these candidates. The corpus to train the word tri-gram language model is about 200**M**bytes. The system was tested using a text that contains 535 errors. They have obtained 56% lexical recall and 64% error recall.

3.6. Spell Checker for Amharic

Amharic is the working language of the Federal Government of Ethiopia and some regional governments of the country. Getaneh W. developed Hunspell based Uninflected Typographic Amharic spell checker [60]. The researcher used the model of Hunspell as a solution for Amharic

spelling errors detection and correction. In his work, Getaneh first generates typographic errors for Amharic language, and then selects a position to start the error generation randomly. To do this, a module assigned for error generation purpose generates 200 errors for each transposition, insertion, substitution, and deletion error types randomly. For the purpose of comparison, the correct word for the generated misspelled words is stored in the file. By doing this, 800 errors are generated for the testing purpose with the corresponding correct words.

In the implementation part of this work, the Hunspell was modified by removing Hungarian languages specific function calls, capitalization checking removal, and truncation of affixation rules. In addition, a new word list was generated from the 13,740 word list to avoid inflected words from the list.

Later, Shewangizaw [59] designed and implemented a spell checker for Amharic that works on inflected Amharic words. In his work, the researcher studied spelling error trends and morphological variants of words in Amharic text. This spell checker has the following components: Input component, Normalization Component, Error Detection Component, Morphological Analyzer Component, Error Correction and Suggestion Component. An input component takes an input from a file or another system and creates tokens which are separated by Amharic punctuation marks and space character. Then it passes the input word to the normalization component for further processing.

The normalization component is responsible for accepting words from the input component and changing Amharic characters having different representation but same pronunciation to a common alphabets (to alleviate syllographic redundancy problem).The Error detection component is responsible for whether the input word is misspelled or not. The Error detection component works first by looking at the input word in the lexicon. When the input word exists in the lexicon, the spell checker does nothing. Otherwise, if this component returns 'not exist', the misspelled word will be sent to the morphological analyzer component for further processing.

The error correction and suggestion component inputs a word from the error detection component, searches all the possible list of corrections from the lexicon as a suggestion, and ranks this list of words. There are different methods for error correction and suggestion as it was

discussed in chapter two. In his work, Shewangizaw has used Levenshtein edit distance for both correcting and ranking the suggestion.

Finally, they evaluated the system with two sets of data. The first set of data was a randomly selected news papers. The second set of data was randomly selected valid and misspelled words. Using the first dataset containing 881 valid Amharic words, 658 were accepted as valid. The rest 223 words were flagged as misspelled words by the spell checker. Out of these 223 flagged words, 175 words are due to the absence of root words in the lexicon, whereas the remaining 48 words have root words in the lexicon but the spelling checker did not recognize the inflection rules. They have obtained 74.7% and 94.7% lexical recall with first and second test dataset respectively.

3.7. Summary

In this Chapter we presented a number of spell checker works by focusing on six languages: English, Indian, Arabic, Chinese, Amharic, and Filipino. While reviewing, we focused on the techniques (i.e. approaches), corpus size used and performance of those works. The general working principle of all spell checkers are almost the same. Mainly, spelling error detection, listing suggestion and ranking suggestions are incorporated in the architecture of every spell checker. Differences are usually observed in spelling error detection phases. For instance, morphological analysis is being incorporated in a spell checker that is designed for morphologically rich languages. The evaluations of the work shows that, systems based on morphological analysis with dictionary-lookup approach obtained better performance than those systems that are done based on normal dictionary lookup techniques. It also reduces the dictionary size significantly, since it stores only the root words. The evaluation of the work also shows that using n-gram analysis requires a very large corpus to train the system or to calculate the probability of each n-gram. Also it may recognize misspelled word as a correct one. Taking what we obtained from the review of the related work and the morphological properties of the Afaan Oromo language, we propose a solution to spell check Afaan Oromo words. The proposed solution was a morphology based (i.e. a dictionary look-up and morphological rules) spell checker.

Chapter Four: Afaan Oromo

4.1. Introduction

Afaan Oromo belongs to the Lowland East Cushitic sub-family of the Afro-asiatic super-phylum. Among the Cushitic language families to which it belongs, Afaan Oromo ranks first by the number of its speakers [61]. The indigenous speakers are uninterruptedly distributed from Southern Tigray in the North to Northern Kenya in the South, and from Harar in the East to Gidaamii in the West [62]. Geographically, except in the Northern, Afaan Oromo is found in Eastern, Southern, Central and Western Cushitic Language families by retaining its homogeneity. The Oromo of all these areas could communicate in this language without dialectical barriers [63].

Afaan Oromo has five major dialects: Rayya (Northern), Borana (Southern), Tulama (Central), Harar (Eastern), and Mecha (Western) [65]. For instance, the Mecha dialect predominantly uses ***koo*** to mean 'my, mine', whereas other dialects use ***kiyya***. At present, Afaan Oromo has different official functions in the Oromiya Regional State and also in Oromiya Zone of the Amhara Region. It is the regional official language, a medium of instructions in primary schools, teacher training institutions and colleges and it is a field of study in higher education institutions.

Before 1991, Ge'ez script was used for writing Afaan Oromo documents. A Latin-based alphabet called Qubee has been adopted and became the official script of Afaan Oromo since 1991. There are about twenty-six consonants and ten vowels (five short and five long) in the language [65, 66]. Afaan Oromo consonants and vowels are depicted in Table 4.1 and Table 4.2 respectively. The schematized syllable format of Afaan Oromo can be represented as: CV (V) (C), where C is a variable for 'consonant', V is a variable for 'vowel', VV represents a long vowel, and items in parentheses are optional [69].

Double consonant will cause variation in the meaning of Afaan Oromo words. For instance, the word ***badaa*** means 'bad' whereas ***baddaa*** implies 'highland'. The language's alphabet (single letter) is constructed from one character symbol or digraph (double character symbol) like ***dh***, ***ny***, ***ph***, ***sh***. Gemination is allowed for single (one character) symbol like ***hoffaa*** which means 'light' whereas gemination for digraph (double character) symbol is not allowed in the language.

For instance, the word ***qophii*** 'readinesses' cannot take the form ***qopphi***. The five long vowels can be obtained by doubling the corresponding short vowels. The difference in length of the vowel induces difference in the meaning. For instance, the word ***laafe*** means 'to become weak' while ***lafee*** is 'bone'.

Table 4.1: Afaan Oromo consonants

Labial	Alveolar/dental	Palatal	Velar	Glottal
P/p	T/t	Ch/ch	K/k	'
B/b	D/d	J/j	G/g	H/h
PH/ph	X/x	C/c	Q/q	
F/f	DH/dh	SH/sh		
V/v	S/s	NY/ny		
M/m	Z/z	Y/y		
W/w	N/n			
	L/l			
	R/r			
	I/i			

Table 4.2: Afaan Oromo vowels

Short vowels	Long vowels
A	Aa
E	Ee
I	Ii
O	Oo
U	Uu

In Afaan Oromo, digits from zero to nine are specific words, namely ***duwwaa*** (0), ***tokko*** (1), ***lama*** (2), ***sadii*** (3), ***afur*** (4), ***shan*** (5), ***jaha*** (6), ***torba*** (7), ***saddeet*** (8), and ***sagal*** (9). The tens are formed by adding the suffix ***/- tama/*** to the matching multiplier digit root, with the exception of ten, twenty and thirty: ***kudhan*** (10), ***digdama*** (20), ***soddoma*** (30), ***afurtama*** (40), ***shantama*** (50), ***jahatama*** (60), ***torbaatama*** (70), ***saddeetama*** (80), and ***sagaltama*** (90). Composed numbers from eleven to ninety-nine are constructed by writing the modified ten first, followed

by the digit separated with a hyphen. The composed ten either loses its ending *-n* (*kudhan* (10) gives ***kudha-tokko*** (11)) or modifies its ending from *-a* to *-ii* (***digdama*** (20) gives ***digdamii-lama*** (22)).The word for hundred is ***dhibba***, and the word for thousand is ***kuma***.

4.2. Afaan Oromo Morphology

Morphology is the branch of linguistics that studies patterns of word formation within and across languages, and attempts to formulate rules that model the knowledge of the speakers of those languages [2]. For example, English speakers recognize that the words *dog*, *dogs*, and *dog catcher* are closely related. Also they recognize these relations from their implicit knowledge of the rules of word formation in English. They infer intuitively that *dog* is to *dogs* as *cat* is to *cats*; similarly, *dog* is to *dog catcher* as *dish* is to *dishwasher*. The rules understood by the speaker reflect specific patterns (or regularities) in the way words are formed from smaller units and how those smaller units interact in speech.

As stated in [10] like a number of other African languages, Afaan Oromo has a very rich morphology. In agglutinative languages most of the grammatical information is conveyed through affixes and other structures. Therefore, the grammatical information of the language is described in relation to its morphology. This makes it very hard to create grammar checker, spelling checker and any other natural language processing task. Although Afaan Oromo words have some prefixes and infixes, suffixes are the predominant morphological features in the language.

4.2.1. Types of Morpheme

In linguistic, the minimum unit of morphology is called morpheme. Afaan Oromo morphemes are divided into two: ***Dhamjecha walabaa*** 'free morphemes' and ***Dhamjecha hirkataa*** 'bound morphemes' [67]. Free morphemes are those morphemes that can stand alone, that is, not attached to some other morpheme to give a meaning. By contrast, bound morphemes have to be attached to some other morpheme to give a meaning. For instance, in ***bishaaniin*** 'by water', **/-iin/** is a bound morpheme and ***bishaan*** 'water' is a free morpheme.

Free morphemes

Afaan Oromo free morphemes are classified into two: ***Dhamjecha hiikaa*** 'lexical morpheme' and ***Dhamjecha tajaajilaa*** 'functional morpheme'. Lexical morphemes are free morphemes that have their own content and stand for one meaning. For example, if we use the word ***Bishaan*** 'water' as a subject or as the object of a sentence its meaning never changes. In contrast, functional morphemes generally perform some kind of grammatical role, carrying little meaning of their own. Functional morphemes specify a relationship between other morphemes. For instance, words like ***Kun, Sun, and Kana*** are all functional morphemes.

Bound morphemes

Based on their content, bound morphemes are classified into two: ***hundee hirkataa*** 'bound root' and ***fufii*** 'affix' [67].

a. Bound root

Bound roots are morphemes that make the most precise and concrete contribution to the words meaning. For instance, words like ***beeka, beeki, beekte, beeku, beekti, beekumsa, and beeka*** have a bound root ***beek-***. Most of the root words in Afaan Oromo are bound root [67].

b. Affix

Affix is a bound morpheme that attaches to bases. The classification of Afaan Oromo affix can be done based on the position of the affix and in terms of the shift of word class [9]. Based on their location, Afaan Oromo affixes are classified into four: ***fufii duree*** 'prefix', ***fufii giddee*** 'infix', ***fufii naannee*** 'Circumfix' and ***fufii duubee*** 'Suffix'. Again based on their grammatical functionality and the type of word class they change, affixes are classified into two: ***fufilee yaasaa*** 'derivational affixes' and ***fufilee hortee*** 'inflectional affixes'.

Derivational morphemes derive a new word by being attached to root morphemes or stems. They change the meaning and optionally change the category of the word class. They can be both suffixes and prefixes in Afaan Oromo. Common derivational affixes in Afaan Oromo are: ***-ummaa, -maata, -aa, -ee, -tuu, -eenya, -ii, -uu, -s-, -at, -siis-, -sis-, -sa, -a, -eenna, hin-, -ina, -aatii, et.*** For instance, adding suffix **/–eenya/** to the adjective ***adii*** 'white' will drive the noun ***addeenya*** 'whiteness'. Whereas inflectional morphemes indicate grammatical information such as number (plural), tense, possession and so on. They may be found in suffix or circumfix in Afaan Oromo. They never change meaning and the syntactic category of the words or morpheme

to which they are attached. Common inflectional affixes in Afaan Oromo: ***-e***, ***-lee***, ***-lii***, ***-t***, ***-oo***, ***-tii(f)***, ***-ch-***, ***-oota***, ***-dhaa(f)***, ***-n***, ***-tuu***, ***-u***, etc. For instance, adding suffix **/-oota/** to the word ***sa'a*** 'cow' will generate inflected word ***sa'oota*** 'cows'.

4.2.2. Word Formation Process in Afaan Oromo

As stated by Getachew [70], Afaan Oromo has five word classes: noun, adjective, adverb, conjunction and verb. But most of the teachers, writers, and linguists classify Afaan Oromo words into ***maqaa*** (Noun), ***maqibsa*** (Adjectives), ***durduubee*** (Affix), ***qabsiistuu*** (conjunction), ***bamaqaa*** (Pronoun) and ***gochibsa*** (Verb) [67]. This section discusses the word formation processes in Afaan Oromo, particularly derivation, inflection and compounding. The discussions and examples in this section are based on the discussion and analysis from [10, 67, 68, 69, 70, 71, and 72].

4.2.2.1. Derivation

Derivation is a process of word formation in which one or more affixes is attached to a root word (and stem) to produce a new word known as derived word. Usually, derivation will change the part of speech of the root word to which a suffix is added. This process of word formation is very productive. Afaan Oromo is derivationally rich language [73].

I. Nouns Derivation

Nouns are any word that can be used to name or identify place, object or ideas. The process of driving a noun from the other word class is called nominalization, and the types of affixes used for this purpose is called nominalizars. In Afaan Oromo, there is a large stock of nominals derived from adjectival, verbal and nominal bases.

Suffixes involved in the derivation of nouns in Afaan Oromo are classified into different groups based on the type of word class they change into nouns.

Group 1 */-eenya/ and /-ina/*

These suffixes are used to derive nouns from adjectives as the following set of examples illustrate.

Adjectives	Gloss	Affix	Derived Noun	Gloss
adii	white	-eenya	addeenya	whiteness
dhiyoo	near	-eenya	dhiyeenya	closeness
bareedaa	beautiful	-ina	bareedina	beauty

Group 2 */-a/, /-aa/, /-ee/, /-ii/, /-sa/, /-sa/, /-aatii/, /-cha, /-choo/, /-maata/, /-nsa/, /-noo/*

These suffixes are used to derive nouns from verb. The following examples indicate the derivation of such nouns.

Verbs	Gloss	Affix	Derived Noun	Gloss
ibse	make it clear	-aa	ibsaa	light
lole	he fought	-a	lola	war
dhuge	he drank	-aatii	dhugaatii	drink

Group 3 */-ummaa/ and /-ooma/*

These suffixes are used to derive nouns from other nouns. The following examples indicate the derivation of such nouns.

Noun	Gloss	Affix	Derived Noun	Gloss
bilisa	free	-ummaa	Bilisummaa	Freedom
garba	slave	-ummaa	garbummaa	slavery
fira	relative	-ummaa/-ooma	firooma	relationship

II. Verb Derivation

Verbs are doing words or action words (for instance, ***figi*** 'run', ***rukuti*** 'hit', ***konkolachisi*** 'drive', and ***nyaadhu*** 'eat'), but some verbs show a 'state of being' (for instance, ***mullate*** 'appear' and ***fedhe*** 'like'). Like that of nouns, Afaan Oromo verbs are derivational. Similarly suffixes involved in the derivation of verbs (verbilizer) in Afaan Oromo are classified into different groups based on the type of word class they change into verbs.

Group 1 */-oom-/, /-aa'-/, /-a'-/*

These verbilizer are used to derive verbs from nouns and adjectives. The following examples indicate the derivation of such verbs. Examples include:

Word	Verbilizer (Affix)	Derived verb
Arjaa	-oom-	Arjoome
gurraacha	-a'-	gurraacha'ee
qulqulluu	-aa'-	qulqullaa'e

Group 2 */-at-/, /-am-/, /-sis-/, /-siis/, /-s-/*

These verbilizer are used to derive verbs from adjectives and another verb. Examples include:

Word	Verbilizer (Affix)	Derived verb
Mure	-at-	Murate
diimaa	-at-	diimat-e
hidhe	-am-	hidhame
dhuge	-siis-	dhugsiise

III. Adjectives Derivation

Adjectives are words that describe or modify nouns (and pronouns). Forming adjectives from other lexical categories is termed as adjectivization. From stative verb like **/diim-at/** 'become red' one can derive the adjective **/diim-at-aa (-tuu)/** 'reddened'. In Afaan Oromo adjectives can be formed from verbs by taking adjectivizers like **/-aa/, /-tuu/, /-eessa/, and /-eettii/.** The following examples indicate the derivation of such adjectives.

Word	Affix	Derived Adjectives
sodaate	-aa/-tuu	sodaataa/sodaattuu
iyye	-essa/-etti	iyyeessa/iyyeettii

Afaan Oromo adjectives are not derivatives as noun and verbs, this is because there are a few number of adjectivizers in the language [67, 69, 70].

4.2.2.2. Inflection

The words in a given class vary their forms for grammatical reasons. Inflection does not form a new word.

I. Noun Inflection

Almost all Afaan Oromo nouns in a given text have person, number, gender and possession markers which are concatenated and affixed to a stem or singular noun form. Afaan Oromo nouns are inflected to indicate different grammatical functions like gender, number, definiteness and case [75]. Inflectional suffixes are combined with stem usually resulting in a word of the same class as the original stem. Both Afaan Oromo nouns and adjectives are highly inflected for number and gender. The principles of noun inflection here apply to verbs and adjectives.

Cases

Afaan Oromo nouns can be inflected to indicate accusative, instrumental, direct object, indirect object, and nominative cases. The nominative cases are used for nouns that are the subjects of clauses. In Afaan Oromo nominative case indicators are **/-n/** and **/-i/**. For instance, in ***Magarsaan Lataa rukute*** 'Magarsa hit Lata', the nominative case indicator is **/-n/.** Afaan Oromo nouns are inflected for nominative, objective, and instrumental cases.

Gender

Oromo has two grammatical genders, masculine and feminine in similar way to other Afro-Asiatic language. There is no neutral gender as in other language like English. Small number of nouns end with ***/-eessa/*** and ***/-eettii/*** to indicate a gender as in ***obboleessa*** 'brother' and ***obboleettii*** 'sister'. Grammatical gender normally agrees with biological gender for people and animals; thus nouns such as ***abbaa*** 'father', ***ilma*** 'son', and ***sangaa*** 'ox' are masculine, while nouns such as ***haadha*** 'mother' and ***intala*** 'girl' are feminine. However, most names for animals do not specify biological gender. Names of astronomical bodies are feminine as in ***aduu*** 'sun' and ***urjii*** 'star'. The gender of other inanimate nouns varies somewhat among dialects. Suffixes like ***/-ch/*** and **/-tti/** are used to show gender in some cases.

Number

In contrast to the English plural marker ***/–(e)s/***, there are more than 12 major and very common plural markers in Afaan Oromo[65]. Plural nouns are formed through the addition of suffixes. The most common plural maker suffixes in Afaan Oromo are: ***/-oota/***, ***/-ooliil/***, ***/-een/***, ***/-lee/***, ***/-wwan/***, ***/-yyii/***, ***/-eetii/***, ***/-ii/***, ***/-oo/***, ***/-n/***.

Definiteness

There is no indefinite article (such as *a, an, some)* in Afaan Oromo like in English. The definiteness article '*the'* in English is */-icha/* for masculine nouns and */-ittii/* for feminine nouns in Afaan Oromo. Vowel endings of nouns are dropped before appending these suffixes. For instance, ***laga*** 'river' becomes ***lagicha*** 'the river', ***haroo*** 'lake' becomes ***harittii*** 'the lake'. In Afaan Oromo definite suffixes appear to be used less often than in English, and they seem not to co-occur with the plural suffixes [67].

Afaan Oromo verbs are also highly inflected for gender, person, number and tenses. Examples:

1. Inni kaleessa dhuf**e**. (He came yesterday)
2. Isheen kaleessa dhuf**te**. (She came yesterday)
3. Inni dhufaa jir**a**. (He is coming)
4. Isheen dhufaa jir**ti**. (She is coming)

In the above four sentences the gender and tense of the sentences are described through suffix which is attached to the verb stems */dhuf-/* and */jir-/*.

4.2.2.3. Compounding

Compounding is the process of putting words together to build a new one that does not denote two things, but one and that is pronounced as one unit. The difficulty with compounds is to work out which words are more heavily pronounced in their first and which ones in their second part. Another problem, also for native speakers, may be to detect which compounds are written how, because some compounds are hyphenated, others are written separately and some are written as one word. This section describes Afaan Oromo lexical compounds and the characteristics which distinguish them from others.

I. Compound Nouns

In Afaan Oromo, compound nouns are productively formed by a combination of different POS.

Noun Compounds

Two nouns can combine to form various kinds of compound nouns. For example, the noun **/abbaa/** 'father' or **/haadha/** 'mother' can occur as a first member as given in the following examples to give a compound noun:

Noun	Gloss	Noun	Gloss	Compound Noun	Gloss
Abbaa	father	buddeena	food	Abbaa buddeenaa	step father
Haadha	mother	mana	house	Haadha manaa	wife

Noun-Adjective compounds

Compound nouns can also be formed by combining noun and adjectives. The process is not productive. Examples include:

Noun	Gloss	Adjective	Gloss	Compound Noun	Gloss
sanbata	sabath	guddaa	big	sanabata guddaa	Sunday
muka	tree	gurraacha	black	mukaa gurraacha	black tree

Adposition-Noun Compounds

A combination of an adposition and a noun may result in a compound noun. In such compounds the nominal element is always a derived form. Examples include:

Adposition	Gloss	Noun	Gloss	Compound Noun	Gloss
gadi	down	qabaa	having	gadiqabaa	oppression
dura	front	ta'aa	sitting	durata'aa	chairman

Verb-Noun Compounds

In this pattern, nominals are formed by combining verbs with other nominals. Examples include:

Verb	Gloss	Noun	Gloss	Compound Noun	Gloss
qotee	plough	bulaa	live	qotee-bulaa	farmer

II. Compound Adjective

Compound adjectives are not as productive as compound noun. But we have instances of them followed by a combination of different categories.

Noun-Noun Compounds

Compound adjectives can be formed by combining two nominals of which the first is **abbaa** 'father' or **haadha** 'mother' and the second a nominal designating some attributes. Baye [71] also mentions the existence of such compounds in Afaan Oromo. The following examples can be given.

Nominal	Nominal	Compound Adjective	Gloss
Abbaa/Haadha	boyichaa	Abbaa/Haadha boyichaa	crying
Abbaa/Hadha	dubbii	Abbaa/Haadha dubbii	talkative
Abbaa/Haadha	hirribaa	Abbaa/Haadha hirribaa	sleepy

Noun-Adjective compounding

In this type of compounding the second member is a modifier of the first. Examples include:

Word	Gloss	Word	Gloss	Adjective	Gloss
bifa	color	badii	useless	bifa-badii	ugly
garaa	stomach	qulqulluu	clean	garaa qulqulluu	clean hearted
ija	eye	jabeessa	strong	ija-jabeessa	shameless

Adjective-Noun Compounds

In such compounds the nominal qualifies the adjective by showing degree. Examples include:

Adjective	Gloss	Noun	Gloss	Compound Adjective	Gloss
gurraacha	black	cilaattii	charcoal	gurraacha cilaattii	charcoal-black
jabaa	strong	mukaa	wood	jabaa mukaa	very strong/strong as wood

Adjective-Adjective Compounds

A combination of two adjectives can form a compound adjective in which the second qualifies the first in terms of degree. Examples include:

Adjective	Gloss	Noun	Gloss	Compound Adjective	Gloss
adii	white	qulqulluu	clean	adii qulqulluu	pretty-white
aja'aa	stinky	tortoraa	rotten	aja'aa tortoraa	rotten-stinky

III. Compound Adpositionals

Compound adposition can be formed from adverbs of time and the direction. Compounds of this type are not very common. Examples include:

Word	Gloss	Word	Gloss	Adjective	Gloss
asii	here	gadi	down	asiigadi	downwards
achii	there	asi	here	achii-asi	towards here

4.3. Afaan Oromo Spelling Error Pattern Analysis

This section reviews the analysis of spelling and grammar errors of Afaan Oromo conducted by Fikadu [83]. The main purpose of his study is to analyze spelling and grammar error of Afaan Oromo modules prepared for teaching Afaan Oromo courses. He used three data gathering techniques for this purpose: text analysis, questionnaires and interview. Text analysis was used to identify and classify spelling errors in the modules. Questionnaires and interview were employed to gather data from the respondents. Both qualitative and quantitative methods of data analysis were used to collect information from students. The finding of his study depicts the existence of spelling errors such as omission, addition, analogy, substitution, transposition, spacing, inconsistency and unclassifiable error types. Examples and discussion for each type of the errors are given below.

4.3.1. Omission

Omission (or deletion) of vowels

Two types of omission of vowels are identified: Errors of shortening long vowels and complete omission of vowel (s). Examples include:

Misspelled word	Correct Word	Gloss
yero	yeroo	time
konkolataa	konkolaataa	car
ibs	ibsi	describe

Omission of consonant

Two types of omission of consonant (s) are identified: Omission of one letter from geminated consonants and complete omission of consonants.

Omission of one letter from geminated consonants

This is a case where a single consonant is used instead of double consonants as in the following examples.

Misspelled word	Correct Word	Gloss
cicimoo	ciccimoo	strong
iratti	irratti	on something
xiqoo	xiqqoo	small

Complete omission of consonant (s)

This results in juxtaposition of either similar vowels or different vowels depending on the type of vowels before and after the omitted consonant (s). Examples include:

Misspelled word	Correct Word	Gloss
buaa	bu'aa	profit
afaa	afaan	language

4.3.2. Addition (or insertion)

Addition of vowels

Further vowel additions are classified into three: addition of identical vowels, addition of vowel between consonant clusters and other types of addition of vowels. Addition of identical vowels (includes error of using double vowels in place of single vowel), and using more than two vowels instead of short or long vowels. Other types of addition of vowels include errors of vowels beside other different vowels that result in something like diphthongal sounds. It also includes vowel addition at the end of words that end in consonant. Examples include:

Misspelled word	Correct Word	Gloss
kitaaboota	kitaabota	books
yoookaan	yookaan	or
caamisaa	caamsaa	May

Addition of consonant

It can be addition of identical consonant(s), geminating one of the consonant clusters, or addition of consonant clusters.

Addition of identical consonants

This includes errors of gemination and using more than two identical consonants as in the following examples.

Misspelled word	Correct Word	Gloss
yookaann	yookaan	or
arriitii	ariitii	speed

Addition that results in consonant clusters

The occurrence of such error at word initially and word finally breaks the spelling convention and represents no sound in that particular word as in the following examples.

Misspelled word	Correct Word	Gloss
dubishuu	dubbisuu	to read
qabxtii	qabxii	point

Geminating one of the consonant clusters

In Afaan Oromo spelling convention, clusters do not geminate as in the following examples.

Misspelled word	Correct Word	Gloss
injjifannoo	injifannoo	defeating
gorssa	gorsa	advice

4.3.3. Analogy

Errors of analogy occur due to the knowledge that one has in two or more languages, particularly with English words. Regarding analogy, the following types of spelling errors were identified:

Adopting letter sequence of English in Afaan Oromo writing

Examples include:

Misspelled word	Correct Word	Gloss
graafii	giraafii	'graph'
projeektii	pirojektii	'project'
biblographii	bibiliyoografii	'bibliography'

Adopting English orthography to represent sounds in Afaan Oromo

This is using the orthographic symbol in English that does not represent sounds in Afaan Oromo. Examples include:

Misspelled word	Correct Word	Gloss
giraaphi	giraafii	'graph'
addugnaa	addunyaa	'word'

4.3.4. Substitution

In most cases, this error occurs between consonants that have some identical features and substitution between vowel and consonant as in the following examples.

Misspelled word	Correct Word	Gloss
guunnamtii	quunnamtii	'communication'
harreeffamuu	barreeffamuu	'to be written'
tottaalee	tooftaalee	'techniques'

4.3.5. Transposition

Transposition errors occur due to interchanging or misplacing of letters as in the following examples.

Misspelled word	Correct Word	Gloss
baa'yee	baay'ee	'many'
dnada'a	danda'a	'be able'
adbii	abdii	'hope'

4.3.6. Spacing

Spacing is spelling one word as separate words and words as one word as in the following examples.

Misspelled word	Correct Word	Gloss
babal lisa	baballisa	'widening'
kan afuu	kanaafuu	'for this reason'
saffisaadubbisuu	saffisaan dubbisuu	'to read fast'
yerooheddduu	yeroo hedduu	'most of the time'

4.3.7. Unclassifiable errors

These are errors that are difficult to classify because of their complicated nature. Examples are: **ildalte**, **gkeekta**, **guea**.

Table 4.3 summarizes a work by Fikadu [83] showing the overall types of errors observed in the modules. As depicted by the researcher in [83], carelessness and lack of knowledge which results from absence of practice, good Afaan Oromo academic background and sufficient experience were found to be the causes of the errors in the modules.

Table 4.3: Spelling errors by major categories

NO	Major categories	Error count	%
1	Omission	534	39.79
2	Addition	410	30.55
3	Analogy	27	2.01
4	Substitution	196	14.61
5	Transposition	15	1.12
6	Spacing	86	6.41
7	Unclassifiable	74	5.51
	Total	1342	100

4.4. Summary

This Chapter has discussed Afaan Oromoo word formation process, especially for adjectives, verbs and nouns through inflections, derivations and compounding. Afaan Oromo spelling error patterns are also reviewed. Understanding all this will help us to design and implement language specific rules to our system. The next Chapter discusses the design and implementation of **AOSC**.

Chapter 5: Design and Implementation of AOSC

In this Chapter, first we will discuss the design approach and requirement. Then, the general overview of the proposed system architecture from the perspective of the system's flow of operations in the form of processes will be discussed. In a similar way the implementation of the algorithms for each component in the architecture will be discussed. Finally, the implemented prototype will be explained.

5.1. The Approach Used

Taking what we obtained from the review of the related work and the morphological complexity and resource scarceness of Afaan Oromo language, we proposed a morphology based spell checker (i.e. a dictionary look-up with morphological rules). A morphology based spell checker has advantages such as its ability to reduce the dictionary size drastically and the ability to recognize new words (i.e. inflected, derivated or compounded words) that are not included in the dictionary. Morphological rules address word categories and their possible inflections, derivation and compounding. Further, the approach can be drawn upon in building grammar checkers. A morphological rule developed for the spell checker is also a stepping-stone for other NLP applications.

5.2. Design Requirements

Knowledge of the language plays an important role in order to design a morphology based spell checker. Such knowledge can be obtained in various ways. In this study, the knowledge for morphological rule and lexicon design was obtained from the analysis of Afaan Oromo text books, published papers, Master's Thesis and discussion with Language professionals. In particular, the lexicon designed as a knowledge base is prepared by using the criteria forwarded in [67, 68, 69, 75, 81]. However, an effort has been made to minimize the number of grammar rules of the language. This is because that the knowledge of the grammar plays a central role in developing an efficient spell checker.

The problems encountered in the design of the programming structures that determine whether a word is correctly spelled or not will differ from language to language. Of particular interest are languages with a more complex morphology. These languages often require additional

morphological processing during the error detection and suggestion prediction phase, since simple increase in the size of the lexicon is not only time consuming, but also error-prone. Specific and accurate analysis at word level is therefore a necessity [76]. Having efficient and effective lexicon look up mechanisms is also one of the requirements in designing every spell checker.

There are several other issues that are closely related to spell checking and correcting processes: dictionary partitioning schemes, coverage of the language of interest in terms of both lexical coverage and coverage of morphological and orthographic phenomena, and word boundary issues. Dictionary partitioning schemes were motivated by memory constraints in many early spell checkers. Word boundaries in most spelling error detection and correction techniques are often defined by inter-word separation, such as spaces and punctuation marks. However, it is largely dependent on the language under consideration. For instance, in Chinese there is no space between words to act as word boundary delimiter [80]. But this is not an issue in languages like Amharic, English, and Afaan Oromo since word boundaries are defined. All the other issues are considered during the design and implementation of **AOSC**.

5.3. Lexicon Design

In order to build a morphology based spell checker, one would require a powerful dictionary for reference in the word error detection and suggestions prediction phases. Creating a dictionary having all the words of the language is a laborious task and infeasible considering the large variation of affix combinations in Afaan Oromo. To handle this problem, we used a morphological analyzer and a dictionary of root word. For example, for a single verb root word ***deem***-'go', over 60 valid word forms can be formed (see Appendix A).

We adopt the Hunspell dictionary and affix file format to design a lexicon. Hunspell is an open source spell checker [79]. It has been designed especially for languages with rich morphology and complex system of word compounding, originally for Hungarian [79]. It's a spell checker for Mozilla Firefox and Mozilla Thunderbird. Even though our spell checker is a standalone application, storing both the root words and affix in this format will help us to directly integrate our system into Apache OpenOffice (Apache OpenOffice is the leading open-source office software suite for word processing, spreadsheets, presentations, graphics, databases and more).

The next subsection discusses root word classification issues and the design of lexicon needed to develop **AOSC**. The design process is based on the morphological properties of the language discussed in Chapter Four and the assumptions and approaches discussed in Chapter Two and Three.

5.3.1. Root word classification

If we are to define rules for permitted combinations of affixes and root words, it is necessary to observe the conditions under which affixes accept certain roots (or vice versa) while they don't produce a meaningful combination with others. If we succeed in understanding this system, it is possible to create a spell checker which would not only recognize all words which could be found in sources it was based on, but also valid compounds and combinations of root words and affixes which may be untypical, but still perfectly correct and usable in the language. Also, implementing such a system of root word classification could result in a significantly shorter word list file.

The big challenge to classify a word stem (or root word) is the grouping of nouns and verbs in such a way that the members of the same group have similar inflections and derivations. Classification of stems and its usefulness for implementing rules of Afaan Oromo morphology has already been touched by Abebe in his morphological synthesizer [75]. He listed 17 different inflectional types which are divided into parts of speech as follows: 7 for the nouns and 10 for the verbs. After listing of stems and affixes, for each stem class there should be a corresponding set of affixes with which it can combine to form variants of words. This association is called signature. Signatures are used to organize the stems (also root words) and suffixes in such a way that the stems in the lexicon or new stems can be organized with appropriate suffixes.

For instance, two syllable nouns ending in short vowels and having end consonants like **k**, **g**, **d**, **r**, **b**, and **n** have the same signature. They take plural making suffix after doubling the last consonant and deleting the last vowel. ***Mana*** 'house', ***laga*** 'river', ***muka*** 'tree' can be taken as examples. The suffixes of this class include: /-een/, /-ni/, /-a/.

Sample nouns and verb signatures are included in Appendix G. The classification rules forwarded by Abebe [75] were adopted for our system with a slight modification.

5.3.2. Design of Affix Lexicon

The affix file contains list of affixes along with their context restrictions and effects. It also serves as a setting file for the dictionary, containing all rules for adding and stripping affix from the root word. An affix file (.aff extension) is a collection of instruction, each on one line, which describes the meaning of the affix classes present in the word list and set different options which influence the behavior of the spell checker, such as character encoding, character set used in suggestions, explicit lists of often misspellings and easily interchangeable letters, etc. For each affix class, a set of rules is defined, which describe the possible derivations, inflections and compounding the affix may produce. The general format of our Affix file adopted from Hunspell [79] is depicted in Figure 5.1.

1. **<Option_Name> < Flag> <Cross_Product> <Line Count>** //The fields of an affix class
2. **<Option_Name> <Flag> <Stripping> <Affix> <Condition>** // Fields of an affix rules

Figure 5.1: The general format of Hunspell Affix file

Description

Option_Name: can be PFX (prefix) or SFX (suffix)

Flag: is the name of the affix class

Cross_Product: is permission to combine prefixes and suffixes.

Possible values: **Y** (yes) or **N** (no)

Line Count: line count of the following rules (i.e. the total number of rules in this class).

Stripping: defines letter or letters to be stripped from the end (for SFX) or from the beginning (for PFX) of the non-affixed form. If no stripping will be necessary, the value of this flag is '0'.

Affix: character or characters added to the end (for SFX) or to the beginning (for PFX).

Condition: regular expression type character, string, or set of characters stating under what conditions the suffix or prefix can be added.

Examples:

1. SFX N1 Y 1
2. SFX N1 0 iin n
3. PFX P2 Y 1
4. PFX P2 0 gag gab

In the above example, there are two affix classes, where class **N1** defines that suffix **/-iin/** will be added to a word if the last character of the word is **n**. For instance, this rule may be applied to the noun ***bishaan*** 'water' and ***ilkaan*** 'teeth'. The second class **P2** defines that the prefix ***/gag-/*** will be added to the word if the first three characters of the word is ***gab***. For instance, this rule may be applied to the adjective ***gabaabaa*** 'short'. Sample suffixation and prefixation rules for Afaan Oromo are included in Appendix C.

Our Affix file also contains language dependent replacement rules. Based on analysis from Afaan Oromo spelling error pattern analysis in [83, 84], we prepared more than 100 replacement rules (sample replacement rules are found in Appendix D). With this replacement rules, the spell checker can suggest the right forms for the typical faults of spelling when the incorrect form differs by more than 1 letter from the right form.

5.3.3. Design of Dictionary Lexicon

The dictionary file contains root words with optional information about morphological affix classes to combine with the root words. Each word in the dictionary may be associated with an arbitrary number of classes separated by a slash (sample Afaan Oromo dictionary files are included in Appendix E). The General format of our dictionary is:

<**Word** (probably a root word)>/<**Affix_Classes** (one or more classes)>

Example:

1. bishaan/N1
2. fi
3. na

In the above example ***N1*** indicates the affix class of this word. Based on this ***bishaaniin*** 'by water' is a possible derivation for the word ***bishaan*** 'water'. In Afaan Oromo, there are some words that don't undergo inflection and derivation, for this reason they are called ***jechoota maseenaa*** 'functional words' [67]. In Afaan Oromo word class like conjunctions, exclamation, post position, pronouns and the like are classified under functional words. For instance ***fi***, ***na*** and ***nu*** in the above example are all functional words (more functional words are included in Appendix B).

5.4. Architecture of the System

Considering the behavior of Afaan Oromo language, the proposed architecture for **AOSC** is shown in Figure 5.2. The architecture has eight components: Tokenizer, Knowledge base, Error detection, Morphological analyzer, Error correction, Morphological generator, Suggestion ranker and Word assembler.

Tokenizer component

This component split a block of text into individual words, digits and punctuation marks. In Afaan Oromo, like in other languages, the blank character (space) shows the end of one word. Moreover, parenthesis, brackets, quotes, etc are being used to show a word boundary. Furthermore, sentence boundaries and punctuations are almost similar to English language (i.e. a sentence may end with a period (.), line break, a question mark (?), or an exclamation point) [82]. Thus, space marks are used as the explicit delimiters or token separator. Every time a space is encountered, the word after the space becomes a token. The output of this module (list of tokens) becomes an input to error detection module.

Knowledge Base

All information (or rules) required for spelling error detection, morphological analysis and error correction are stored in this module. It contains a root word and affixes for different POS. The knowledge base component designed for our system is based on the discussion given in Chapter 4 and analysis from a research conducted on Afaan Oromo spelling error pattern analysis by Fikadu [83] and Ijigu [84].

Error Detection component

Error detection component is responsible for checking whether the input word (from the Tokenization component) is misspelled or not. The error detection component works first by looking the input word in the root word dictionary. If the input word exists in the root word dictionary, the spell checker does nothing. Otherwise (i.e. if this component returns 'not exist'), the input word will be sent to the morphological analyzer component for further processing. The morphological analyzer component decomposes the input word into the possible roots and affixes (based on their signature) and then passes them to the error detection component.

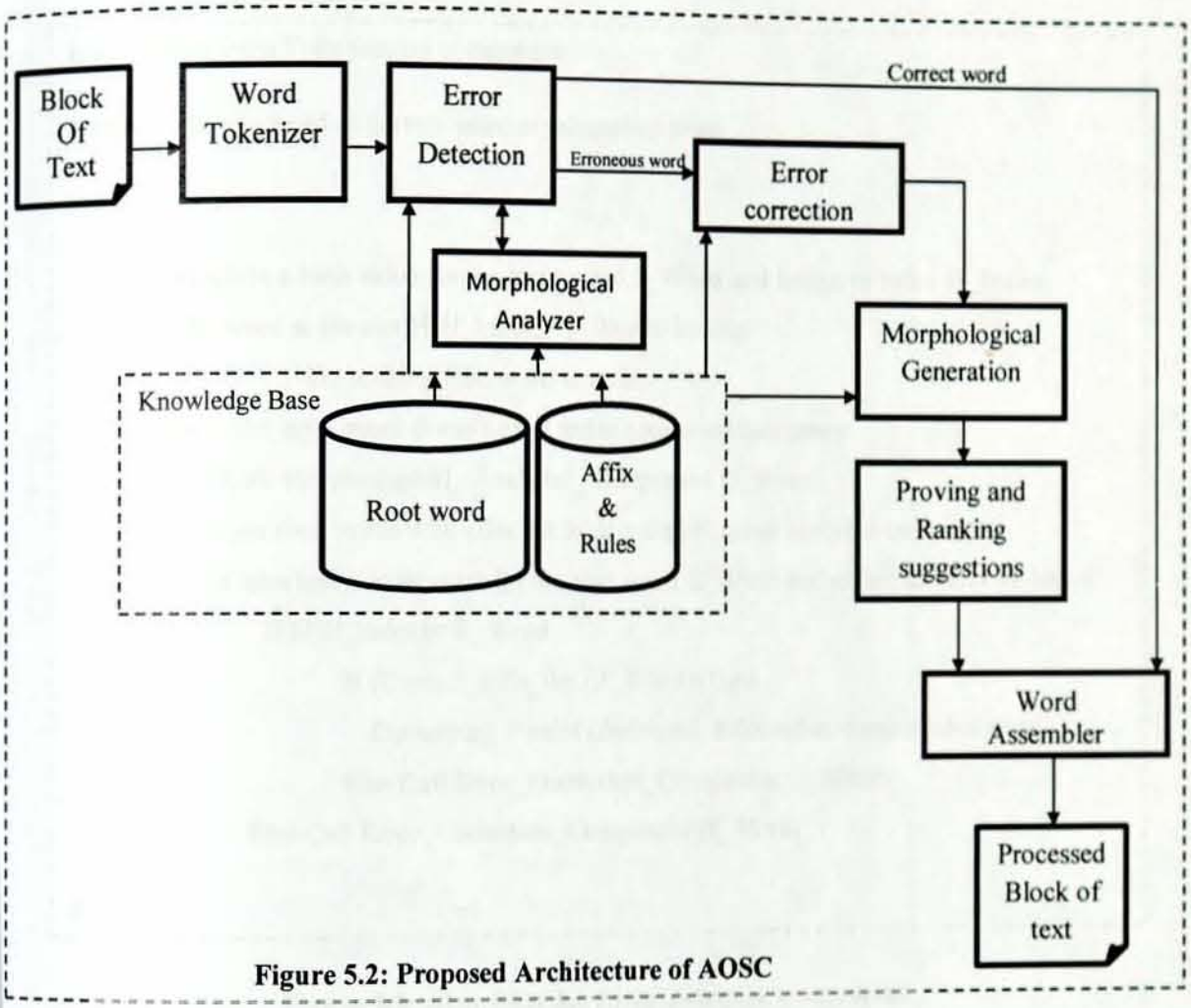


Figure 5.2: Proposed Architecture of AOSC

Then, the error detection component once again lookups the knowledge base to check whether the returned root word and affix exists in the knowledge base or not. If they do not exist, the error detection component will recognize that it's a misspelled word and then passes them into the error correction component. If both the root word and affix are found in the knowledge base, the system cannot automatically say this word is valid, for the root word may not be inflected or derivated for this affix. Finally, to determine if this word is an acceptable word, the class of this root word is checked in the root word dictionary. If it has this affix flag, the system will recognize it as a valid word (i.e. no further processing is needed), otherwise the error detection component will recognize it as misspelled word. The error detection algorithm is depicted in Figure 5.3.

Input: word from Tokenization component

Output: classify word as correct word or misspelled word

Start

1. **Calculate** a hash value for the input word **I_Word** and assign to index **H_Index**
2. **If** the word at the slot H[H_Index]= I_Word// lookup

   Do nothing //the word is valid

   Else // the input word doesn't exist in the root word dictionary

   2.1. **Call** Morphological_ Analyzer_ component (I_Word)

   2.2. **Get** root words with affix list from morphological analyzer component

   2.3. **Calculate** a hash value for the root word R_Word and assign to index H_Index

   If H[H_Index]= R_Word

   If (Correct_affix_list (R_Word)) then

   Do nothing // valid (derivated, inflected or compounded word)

   Else Call Error_Correction_Component (I_Word)

   Else Call Error_Correction_Component (I_Word)

END

Figure 5.3: Algorithm for Error detection component

We adopt the Hunspell [85] hashing algorithm for lookup. As discussed in Section 2.5.1 Hashing is a well-known and efficient lookup strategy. If the word stored at the hash address is the same as the input string, there is a match. However, if the input word and the retrieved word are not the same or the word stored at the hash address is null, the input word is indicated as a misspelling. The random-access nature of hash tables eliminates the large number of comparisons required for lookups.

Morphological Analyzer component

The task of this component is to accept a list of words from error detection component and then decompose each word into root words and affixes (based on their signature) and then pass them

to the error detection component. To develop an algorithm for this component, we first reviewed a research work by Assefa [69] (Morphological analyzer for Afaan Oromo text). He used a machine learning approach. First he trained the system with thousands of words, and then he let the system to perform a morphological analysis. Since we are using a dictionary lookup with morphological rules to develop **AOSC** system, we are forced to develop our own knowledge based morphological analyzer algorithm. The proposed morphological analyzer algorithm is depicted in Figure 5.4.

Input: word I_Word from Error detection component

Output: list of affix and root words

Start

1. **Scan** input word from right to left and left to right to look for valid suffix and prefix
   For each valid suffix in I_Word strip them and store result in a buffer
   For each valid prefix in I_Word strip them and store result in a buffer
   //pass list of affix and stems to the error detection module
   Return root and affix
   //If there is no valid suffix and prefix
2. **Scan** input word from left to right and right to left and then look for a possible roots
   For each valid roots in I_Word strip the root and store result in a buffer
   //pass list of possible roots to the error detection module
   Return root and affix //valid roots and invalid affix

END

Figure 5.4: Algorithm for Morphological Analysis

This algorithm (Figure 5.4) makes use of rules stored in the knowledge base to strip a given word into its root words and affix. In this process, each individual word is scanned from right to left (and right to left) in the affix file and root word dictionary. Upon finding a valid affix, it is stripped from the word. However, the exact affix stripping is possible only for a correctly spelled word. In the case of misspelled word, as there is an ambiguity as to whether the error exists in the root or affix, only probable affix stripping can be done.

To illustrate how the error detection and morphological analyzer works, consider the unknown word ***bishaaniin*** 'by water'. The error detection module will first check if *bishaaniin* is found in the root word dictionary. Since it is not found in the dictionary, the system cannot automatically say that it is misspelled, for it may be inflected, derivated or compounded. Then the error detection component will call the morphological analyzer. To determine if this word is acceptable, morphological analysis will be done. The morphological analysis algorithm will first scan from the right to left and left to right to search for a valid suffix and prefix. Since ***/-iin/*** is a valid suffix, the morphological analyzer component will strip /-iin/ from ***bishaaniin*** and returns suffix */-iin/* and unknown morpheme ***bishaan***. Now the error detection component checks the unknown morpheme ***bishaan*** for its presence in the root word dictionary. Since it is found in the root word dictionary, the system cannot automatically say ***bishaaniin*** is a valid word, for the root ***bishaan*** may not be inflected or derivated for the suffix */-iin/*. Finally, to determine if the unknown word ***bishaaniin*** is an acceptable word, all the rules required to append suffix */-iin/* will be checked in the affix file (e.g. any nouns ending with consonant 'n' can append suffix */-iin/*). Now the error detection component will recognize *bishaaniin* as a valid word, and no further processing is needed. The same process will be done for a misspelled word.

Error Correction component

This component performs two tasks. First it accepts the pairs obtained after affix stripping in morphological analyzer from error detection component to classify the errors into one of the following classes:

- Valid prefix, invalid root, and invalid suffix
- Valid prefix, invalid root, and valid suffix
- Valid prefix, valid root, and invalid suffix
- Invalid prefix, valid root, and valid suffix
- Invalid prefix, valid root, and invalid suffix
- Invalid prefix, invalid root, and valid suffix
- Invalid prefix, invalid root, and invalid suffix
- Valid prefix, valid root, and valid suffix // incorrect combination

After classification, this component will make a probable correction to the misspelled morphemes based on their error classes. A single erroneous word may be classified under two or more classes and each of the error is handled separately. For instance, in the case of a valid affixes (prefix and suffix) and invalid root, the valid affix will give us the specific category of the possible root words. Similarly on finding that the suffix is invalid and root is valid, valid root will give us the specific category of the possible words. Then using the replacement rule and Levenshtein edit distance (LED) possible correction will be done. The general algorithm of our error correction component is depicted in Figure 5.5.

```
Input: list of morphemes M from error detection component
Output: List of valid stem and valid affix
Start
        Accept M from error detection component
        Identify their error class
        Apply replacement rule
        Call Levensthein_Distance // to get nearest matches of the morpheme
        Make a correction
End
```

Figure 5.5: Error correction Algorithm

Rules in the knowledge base and LED techniques have been used for error correction. The less the edit distance between two strings is, the more similar are the strings to each other (details of Levenshtein edit distance was given in section 2.6.1.1). The pseudo code for LED that takes two strings, s of length m, and t of length n, and computes the distance between them is given in Figure 5.6.

```
Int Levensthein_Distance (char s[1..m], char t[1..n])
// d is a table with m+1 rows and n+1 columns
Declare int d[0..m, 0..n]
  For i from 0 to m
    d[i, 0] := i
  For j from 1 to n
    d[0, j] := j
  For i from 1 to m
    For j from 1 to n
      If s[i] = t[j] then cost = 0
                Else cost = 1
      d[i, j] := minimum(
            d[i-1, j] + 1, // deletion
            d[i, j-1] + 1, // insertion
            d[i-1, j-1] + cost // substitution
                )
Return d[m, n]
```

Figure 5.6: Pseudo code of LED

Morphological Generator component

This component will be called to put together the corrected parts (morphemes) from the error correction component to re-build the complete word form. In addition, based on the rules in the knowledge base it determines the various words that can be generated from a given valid morphemes. This component is again used to correct and generate suggestion for those errors resulted from valid root and valid affix (invalid combination of roots and affixes). If no candidates can be found after error correction and morphological generator, no candidates will be generated.

To develop the morphological generator algorithm, it is worth to consider the work of Abebe [75]. He used the rule based approach to develop a prototype named **HORSIISAA** 'morphological synthesizer'. His algorithm takes the valid input stems of either verbs or nouns,

and then output all the possible word forms. But the input to our morphological component is not only the stem. It can be any valid morphemes (i.e. it can be a suffix, a prefix, a root word or combination of them). Taking this into account, we proposed a new knowledge based morphological generator algorithm for Afaan Oromo, which takes any combination of morpheme (s) as the input. The concept for generating a word forms from a single stem is taken from HORSIISAA [75]. The detail of **AOSC** morphological analyzer component algorithm is depicted in Figure 5.7.

Input: any valid morphemes from error correction component

Output: list of correct words

Start

1. **Get** input root (or stem) or affix
2. **If** only root (or stem) is given
   Identify the root word class
   Generate word forms according to the root word class
3. **If** only affix is given
   Generate word forms having this affix class (or flag)
4. **If** both root words and affixes are given
   Identify root word or affix class
   If the root word class matches the given affix
   Append affix
   Else take root word to step 2 and affix to step 3
   Return

End

Figure 5.7: Algorithm for Morphological generation

This word form generation algorithm takes root (or stem) or affix as an input, then it identifies the class of each root words and retrieves corresponding affixes to generate a valid word form.

Providing and Ranking Suggestions

Once the process of error correction and morphological generation was done, the next step is to provide and rank suggestions for the detected error. Hence upon detecting the error, the user will be provided with a list of probable correct words which the user can select to update the misspelled word. It may also be possible that the correct word expected by the user may not be listed in the suggestions; if there is no possible root word and affix. The algorithm for ranking suggestion is depicted in Figure 5.8.

```
Input: list of words L_Words (suggestions)
Output: Top N ranked words (suggestions)
Start
    Accept L_Words from morphological generator component
    If the suggestion is formed by applying replacement rule on L_Word
        Set its rank to 1 //top
    Else
      Call Levensthein_Distance
        If two or more suggestions have the same LED
          Call Character_Distance// similar_letters
            Rank the suggestion list// by Character_Distance
        Else
          Rank the suggestion list// by LED
    Order the list in ascending order of their rank
    Display N words from the list
End
```

Figure 5.8: Algorithm for ranking suggestion

Here (Figure 5.8) we used keyboard layout (i.e. character distance), replacement rule in the affix file (sample replacement rules are included in Appendix D), and LED [77] to rank the suggestion. The character distance method uses a Pythagorean-type metric to measure the distance between a misspelled word and a possible correction, based on the **QWERTY** keyboard layout [78].

The QWERTY keyboard is represented as two-dimensional arrays as shown in Figure 5.9. The suggested word with the shortest distance to the misspelled word is considered as the best suggestion.

i/j	0	1	2	3	4	5	6	7	8	9	10	11	12
0	Q	W	E	R	T	Y	U	I	O	P	[	]	\
1	A	S	D	F	G	H	J	K	L	;	'		
2	Z	X	C	V	B	N	M	,	.	/			

Figure 5.9: Two-dimensional representation of QWERTY Keyboard

According to the Euclidean distance formula, the distance between two points in the plane with coordinates (x, y) and (a, b) is given by:

$$\textbf{dist}\ ((x, y), (a, b)) = \sqrt{(x-a)^2 + (y-b)^2} \qquad \textbf{(5.1)}$$

For example, based on the character distance chart in Figure 5.9, the distance between **w** located at (0, 1) and **n** located at (2, 5) is 4.47 and the distance between **n** located at (2, 5) and **m** located at (2, 6) is 1. This indicates that there is highest probability to mistype **n** as **m** than **w**.

If the erroneous word is corrected by rules in the replacement table, word formed by this rule will take the top rank in the suggestion list. If two or more possible words are generated using the replacement rule, the top rank would be given to the one with the smaller LED. Character distance is considered only if there are words having the same LED.

To illustrate how this component works, consider the misspelled word ***hindeemi***. This word can be analyzed into the following pairs: valid prefix **hin** 'don't' and valid root **deem-** 'go' and valid suffix **-i**. We have the following rules in our knowledge base:

1. Any verb stem ending with consonant **m** can append suffixes **–e,-a,-i, -u, -ti** and the like without any criteria. This indicates that ***deemi*** is a correct word.
2. After appending suffix like –e, -a, -u, -ti etc any verb stem ending with consonant **m** can also append prefix ***hin-***. This indicates that *hindeemi* is incorrect combination.

In this case, the morphological generator will generate too many suggestions. By considering the error is because of prefix, words like ***deemi, deeme, deema, deemte, deemteetti*** and the like may be generated for suggestion. Again by considering the error is because of suffix ***/-i/***, words like ***hindeeme, hindeemti, hindeeme, hindeemu*** and the like may be generated for suggestion. The same process is done for the root word. Listing and displaying all the possible suggestions to a user makes confusion, hence ranking and trimming of suggestion is needed. Table 5.1 shows the number of edit operation required to convert misspelled word ***hindeemi*** to candidate word.

Table 5.1: LED required for converting hindeemi to candidate words

Candidate word	Insertion	Deletion	Substitution	Transposition	Total
deemi		3			3
deeme		3	1		4
deemte	1	3	1		5
hindeeme			1		1
hindeemti	1				1
hindeema			1		1
hindeemu			1		1

As shown in Table 5.1, the LED of the first three words is higher, thus they are trimmed and the last four candidates will be selected for ranking. Since all of them have same LED, another criterion is needed to rank them. As observed from Afaan Oromo spelling error pattern analysis, the probability of making insertion and omission errors are higher than the other types (see Section 4.3 for details). Thus, if two or more words have the same LED we give the top rank for insertion and omission errors, for this reason candidate word ***hindeemti*** will rank number 1.

Now it's time to consider keyboard layout to rank the three words left. They differ from the erroneous word by one character, so we need to find the distance between the last characters of misspelled word ***hindeemi*** '***i***' and the last character of all the other three words.

Based on their score in Table 5.2, the one with the smallest character distance will be ranked first. Finally for the misspelled word ***hindeemi***, suggestion provider and ranker will return **hindeemti, hindeemu, hindeeme** and **hindeema** as the top four suggestions.

Table 5.2: Ranking suggestion for *hindeemi* with their character edit distance

Candidate word	Last character	Distance from i (0, 7)	Rank
hindeeme	e (0, 2)	5	3
hindeema	a (1, 0)	7.07	4
hindeemu	u (0, 6)	1	2

In table 5.2, ranking was started from 2 this because rank number 1 is for the candidate word ***hindeemti***.

Word Assembler component

The task of this component is to combine correct word with the one flagged as misspelled word.

5.5. Prototype Development

Developing a prototype to demonstrate the validity and usability of the proposed **AOSC** is one of the objectives of this work. The prototype of Afaan Oromo spell checker is developed using Microsoft Visual C# 2010. The snapshots of **AOSC** are depicted in Figures 5.10 – 5.13. The input for the prototype is a text file. The text file can be browsed or typed directly into the textbox. The system will check the spelling automatically after the space bar is pressed or automatically after the saved text file was exported. A word that the system believes to be misspelled is flagged with a wavy red underline and suggested corrections are available in a pop-up menu after right clicking on the erroneous word.

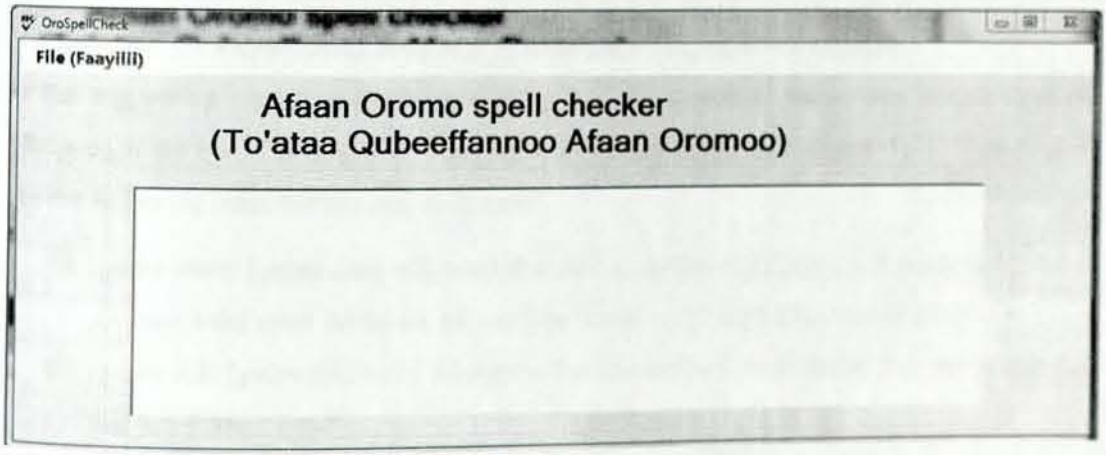


Figure 5.10: Snapshot of Afaan Oromo spell checker

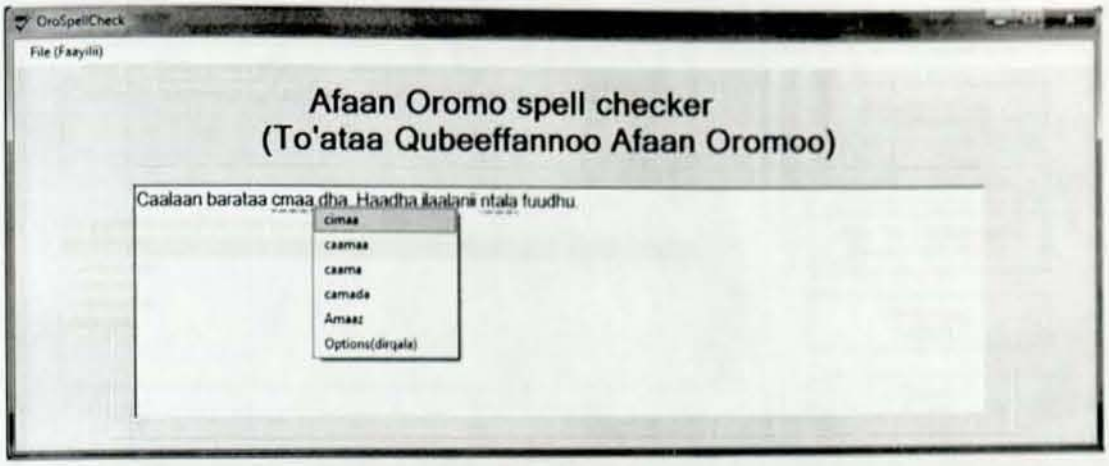


Figure 5.11: Snapshot of AOSC while spell checking and generating suggestions

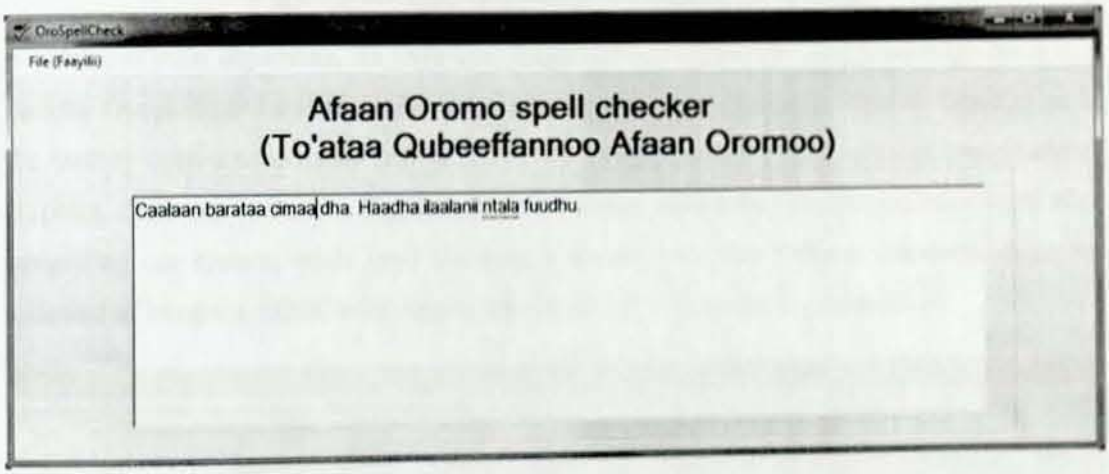


Figure 5.12: Snapshot of AOSC after changes are effected

In the suggestion popup menu (Figure 5.11) there is a context menu item named **Options (dirqala)** at the end. Clicking it will display a new window shown in Figure 5.13. This window has the following main buttons and functions:

- Ignore once: Ignore once will mark that part as spelled right (even if it really isn't), but if you make the same mistake a second time it will warn you on the second one.
- Ignore All: Ignore this word throughout the document. It will assume that you misspelled that word intentionally.
- Add to dictionary: Include this word in the program's dictionary.
- Change: Use the suggested word to replace the detected error (single word).

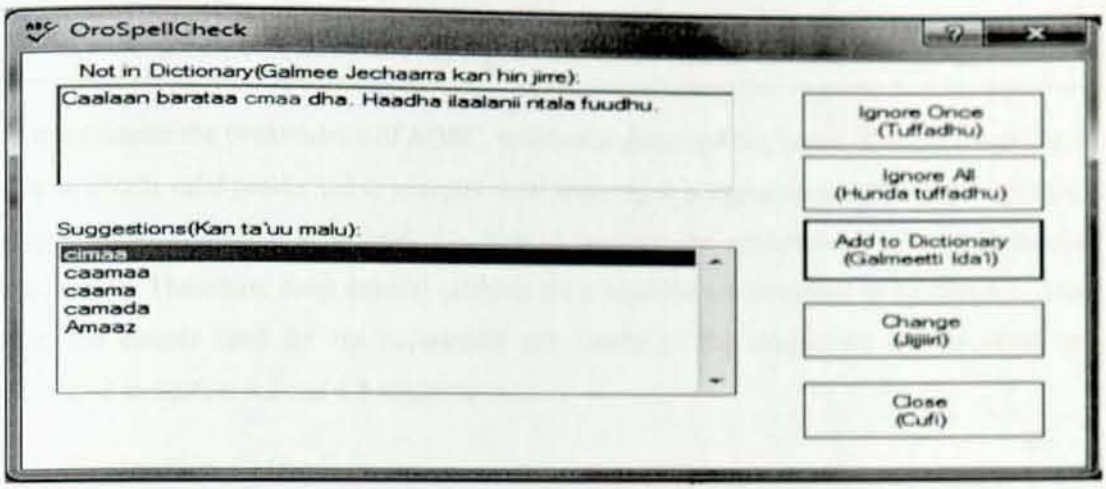


Figure 5.13: Snapshot of AOSC Suggestion window

Besides our main objectives, we have also integrated our system (i.e. the knowledge base) into Apache OpenOffice 3.4.1 windows version for demonstration purpose. Apache OpenOffice is the leading open-source office software suite for word processing, spreadsheets, presentations, graphics, databases and more. Figure 5.14 shows screen shots taken from OpenOffice word after integrating our system; while spell checking a sample text. The different sequential steps we followed to integrate **AOSC** with Apache OpenOffice 3.4.1is given in Appendix H.

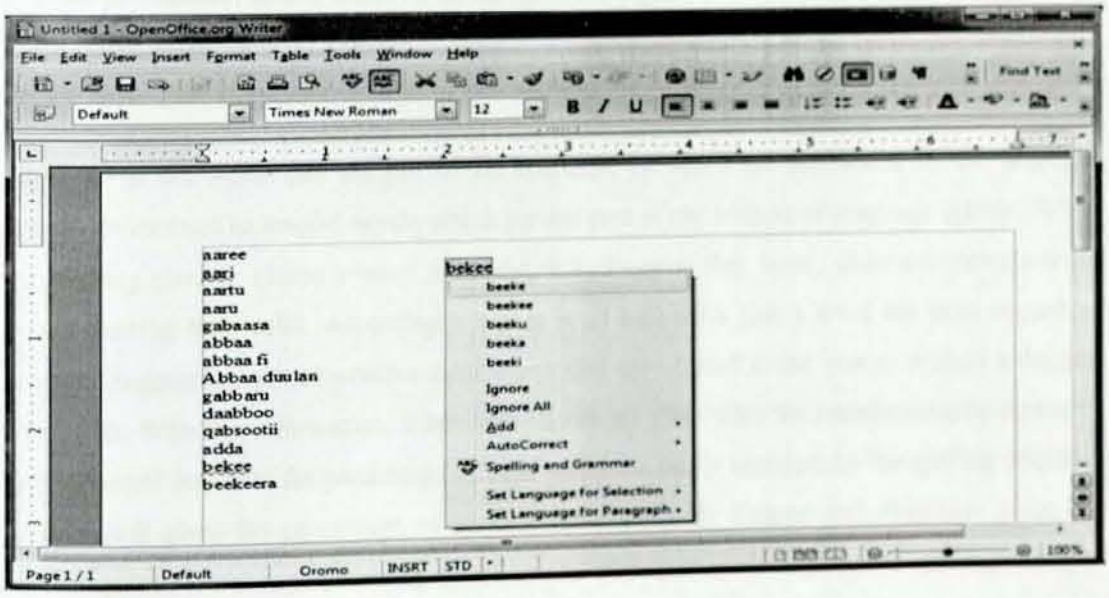


Figure 5.14: Screen shot of OpenOffice

CHAPTER 6: Experiment and Discussion

In this Chapter the performance of **AOSC**, which was described in Chapter 5, is evaluated. To be able to obtain valid results and to interpret them properly, it is important to know how to evaluate a spell checker system in general, i.e. how to measure the performance of a spell checker application. Therefore, some general methods for evaluation are described in Section 6.1. After that, the dataset used for the experiment and results of the experiment are described and discussed in Section 6.2 and 6.3 respectively.

6.1. Evaluation Criteria

The designed system must be evaluated to test its effectiveness. In the literature, several methods for evaluating spell checker system have been proposed. A work done by Kukich [8] proposed lexicon size, test set size, correction accuracy for single and multi error misspellings, and type of errors as evaluation criteria for a spell checker tool. A research work by Paggio et al. [86] recommend error recall, precision recall, interface and suggestion adequacy for the evaluation of a spell checker algorithm. The above stated evaluation metrics were used in different research works of spell checkers [3, 36, 59, and 87]. Some of the measurements are subjective and difficult to evaluate. In this research work, more directly measurable parameters such as lexical recall, error recall, and precision are used to evaluate the developed system. We will follow Paggio et al. [86] in their definitions of the metrics.

Valid words are words that are part of the language, or which are sanctioned by the language system, in contrast to *invalid words*, which are *not* part of the lexicon or language system. When the spelling checker claims a word is invalid, it is *flagging* that word, while *accepting* a word means treating it as valid. Accordingly, a *flag* is an indication that a word has been tagged as invalid. *Suggestions* are alternative valid words that are offered to the user to replace a flagged word with. With this information, it becomes intuitively clear what the metrics actually measure. *Lexical recall* indicates the percentage of valid words correctly accepted by the spelling checker, *Error recall* gives the percentage of invalid words correctly flagged and *Precision* gives the percentage of correct flags (correctly found invalid words) over all flags by the spelling checker. Description about how these measurements are calculated is given in Table 6.1.

Table 6.1: Evaluation metrics

Metric	Measurement Method
Lexical Recall	# of valid words accepted / # of valid words
Error Recall	#of invalid words flagged / # of invalid words
Precision	#of correctly flagged invalid words / # of words flagged

It should be noted that the evaluation methods presented in Table 6.1 works best in a controlled environment (i.e. where the test data and the knowledge base are from the same or a similar source), but this work was evaluated on two dataset randomly taken from different sources (i.e. the dataset is different from the one used for constructing the knowledge base). The first dataset is prepared to evaluate the number of valid compounded, inflected and derivated Afaan Oromo words accepted by the system. The second dataset is prepared to evaluate the number of valid words accepted by the system (i.e. coverage test). In the next section, data collection technique and process on the sample data is briefly discussed.

6.2. Datasets used

Dataset A

Initially, we randomly selected sentences (and paragraphs) from stories and papers which produced 6521 words. To trim repeated words and select derivated, inflected and compounded words Alchemist 2.0 has been used. This reduced the 6521 to 1464 unique words including compound words, words with derivational morphemes, inflected words, and functional words. The word lists are printed out and then manually spell checked by three postgraduate linguistic students. We found that, the data set consists of 1385 correctly spelled words and 79 misspelled words. In addition, we manually generated and added some inflection and derivation variants for the root word ***deem-*** **'go'**, ***qab-***'hold' and ***beek-***'know', a total of 347 unique words to the dataset; making the total words 1811.

The main focus on this test has been to evaluate the number of valid inflected, derivated and compounded Afaan Oromo words accepted by the system and the number of correctly flagged invalid words by the system. A summary of this sample data is shown in Table 6.2. The dataset was opened using **AOSC** tool. All words claimed as misspelled by the spell checker are automatically flagged.

Table 6.2: Summary of Dataset A

Description	No.
Total count of words	1, 811
Total valid words	1, 732
Total misspelled words	79

All type of spelling errors discussed in Section 4.4 such as omission, addition, analogy, substitution, transposition, spacing, inconsistency and unclassifiable error types are observed in the dataset A.

Results of Dataset A

In this sample, out of 1,732 valid Afaan Oromo words, 1,535 were accepted as a valid word; 197 words were flagged as misspelled words by the spell checker. Out of the 197 incorrectly flagged words 186 words are due to the absence of root word in the lexicon, whereas the remaining 11 words have root words in the lexicon, but the spelling checker did not recognize the inflection, derivation and compounding rules (incomplete rules in the knowledge base). On the other hand, all the 79 misspelled words were flagged. Thus the total words flagged become 276 (i.e. incorrectly flagged valid words plus correctly flagged invalid words). The result of sample data set A is shown in Table 6.3.

Table 6.3: Evaluation result of sample A

Description	Value
Lexical recall	(1,535/1,732)*100 = 88.62%
Error recall	(79/79)*100 = 100%
Precision	(79/276)*100 = 28.62%

In addition to the three metrics discussed in Table 6.1, we could also test how the spell checker generates a preferred suggestion. Ideally, the spell checker should only suggest the preferred correction (i.e. maximum five suggestions). However, in practice it is sometimes unclear what the preferred correction really is. For example, what should be the correction for the invalid word: "**deem**"? It could be "**deemi**", "**deeme**", or "**deemu**", among possible others. It depends on the context. However, we only looked at each misspelled word and then check whether a right

suggestion is generated or not based on our algorithm (i.e. without considering the context). The test shows that for all the flagged words a possible suggestion was generated. Sample suggestion generated for a misspelled words in the dataset **A** are included in Appendix F.

Dataset B

To perform the second experiment, paragraphs from news papers belonging to several domains were selected. From these papers around 12,000 words including regular dictionary words, domain-specific terms, proper names, technical terminologies, acronyms, jargons, and expressions are extracted manually. Initially, those papers did not contain any misspellings; however, for evaluation purposes several words were arbitrarily changed, resulting in non-word errors in the text. These introduced misspellings were approximately 1% of the original text; and thus, they are around 120 spelling errors. The main aim of preparing this dataset is to test the lexicon coverage (i.e. the number of valid words correctly accepted by the system). A summary of this sample data is shown in Table 6.4.

Table 6.4: Summary of Dataset B

Description	No.
Total count of words	12, 000
Total valid words	11, 880
Total misspelled words	120

Results of Dataset B

Spell checking the test data using **AOSC** tool resulted in 9,082 accepted valid words; 2798 incorrectly flagged valid words and 120 correctly flagged misspelled words. As a result, around 76.44% of correct words are accepted by the system and 100% of total errors were successfully flagged. Table 6.5 depicts the obtained results.

Table 6.5: Evaluation result of Dataset B

Description	Value
Lexical recall	(9082/11880)*100 = 76.44%
Error recall	(120/120)*100 = 100%
Precision	(120/2918)*100 = 4.11%

Out of 2, 798 incorrectly flagged valid words, 1036 (37%) of them are country names, proper names, scientific words, instrument names, and sport team names that was not found in our knowledge base. 37(1.3%) of the incorrectly flagged valid words have a root word in the knowledge base, but they are incorrectly flagged due to coverage of rules and exceptional cases.

6.3. Discussion

As it is shown in Table 6.3 and Table 6.5, we have obtained 88.62% and 76.44% lexical recall, with dataset A and dataset B respectively. The error recall in both dataset is 100%. With linguists involved in evaluating the outputs of the system, we identified the following reasons as the factors that degrade the performance of **AOSC**:

- The lexicon used in this work was not full-fledged; as a result some of the correct words are flagged as incorrect word. Afaan Oromo has many dialects. So if most of them are considered, tremendous number of valid words can be recognized by the system.

- The developed affixation rules were not exhaustive to identify each inflected and derivated Afaan Oromo words. For instance, if the second consonant of verbal stem is double, then reduplication of the stem takes the form CVCV, where C=consonant and V=vowel, pattern rather than CVCCV pattern unusually. For example, ***barreesse*** 'he wrote' becomes ***babarreesse*** 'he wrote several time' rather than ***babbarreesse***.

- Affix rules were integrated based on their POS and signature; however, some of the lexicon entries have no POS. Hence, affix rules were not integrated to those words.

- There were some exceptions for affix rules of having the same POS. Thus, these exceptions needed to be identified and adjusted. For instance, the suffix ***/-eenya/*** in the class V3 of root words works perfect for roots like **qab-+-eenya**= ***qabeenya***, but it gives no sense for root word like **bad-+-eenya**= ***badeenya***, rather it should be ***badaa*** or ***baduu***.

Generally, from the result and the feedbacks and suggestion of the linguists who evaluated the system, we observed that enhancement in the knowledge base will improve the accuracy of the system. Considering our knowledge base lexicon size, the obtained result is satisfactory. Moreover, we have observed that morphology in Afaan Oromo is complex, so it needs to be studied linguistically to promote the computational aspect.

Chapter 7: Conclusion and Recommendation

7.1. Conclusion

Since most computers work in English and other few languages, people who do not speak such languages are either forced to access computers in those languages or will not use them at all. This has its own impact on the socio-economic development of that country. In order to increase the usability of computer devices and let people express their ideas using their native languages, these devices need to be localized into native languages. Hence, it is necessary to do research to alleviate these problems. Spell checker is one potential candidate to this. Use of computers for document preparation is one of those many tasks undertaken by different organizations. Introducing texts to word processing tools may result in spelling errors. Hence, text processing application software has spell checkers. Integrating spell checker into word processors reduces the amount of time and energy spent to find and correct the misspelled word. However, these tools are not available for Afaan Oromo language.

This study proposed the designed and implementation of a system called **AOSC** to solve Afaan Oromo word spell checking and correcting problem. The system is designed based on a dictionary look-up with morphological rule based (i.e. morphology based spell checker). To develop morphology based spell checker, the knowledge of the language morphology is necessarily required. Accordingly, the morphological properties of Afaan Oromo have been studied. Morphological generation was used to derive all the possible inflection and derivation of a given root word. And also simple compounding rules are included. Our knowledge base contains more than 4000 lexicons. The knowledge base is developed using **Hunspell** standard. Most of the algorithms are designed from scratch as there are no previously designed algorithms for this purpose based on the morphological properties of the language under consideration. But, some of the algorithms are adapted from the research work of other languages.

In order to test the performance of the algorithms developed, two experiments have been conducted. The system was evaluated using two datasets. The first experiment was conducted in a corpus of size around 1, 811 words. The result shows 88.62% lexical recall, 100% error recall, and 28.62% precision. The second experiment was conducted in a corpus of size around 12, 000 tokens. The result shows 76.44% lexical recall, 100% error recall, and 4.11% precision.

In general, given different constraints, it is possible to say that, the performance obtained is good compared with resource rich languages. We hope that this research paves a way for a full fledged Afaan Oromo spell checker for those who want to pursue conducting research in natural language processing for Afaan Oromo language.

7.2. Contribution of the work

The main contributions of this study are summarized as follows:

- The first spell checker for Afaan Oromo,
- Development of a new morphological analyzer and morphological generator algorithm that works with both the correct and incorrect words of Afaan Oromo.

7.3. Recommendations and Future Work

Spell checking and correcting have become a part of everyday life for today's generation. Whether it is working with text editing tools, such as MS Word, or typing text messages on one's cellular/mobile phone, spell checking and correcting are an inevitable part of the process.

Developing a full- fledged and efficient spell checker needs a group of experts from linguistic and computational perspectives. Additional features can be added or the existing components of our spell checker can be modified in order to increase the performance. Spell checkers are vital components for any word processor applications and search engines. They are helpful in detecting and correcting spelling errors.

The developed **AOSC** has portions that require further improvements that we want to recommend them as future work. Hence, the following recommendations are made for further research and improvement:

- The lexicon coverage of our knowledge base is small. One can increase the lexicon size and add more word formation (i.e. inflection, derivation, and compounding) rules to the knowledge base to obtain a desired result. The prefixes and suffixes are almost exhaustive. There may be some prefixes, suffixes and rules missing during manual compilation, and they can be added to the knowledge base rules. Dialect consideration is crucial to have more word formation rules.

- Classification of root words into their POS and inflection and derivation class was done manually. Interested researcher can also integrate POS tagger developed by others into the system.

- Our spell checker only detects and corrects non-word errors, but Afaan Oromo documents also exhibit real-word errors. Hence, there is a need for detection and correction of real-word errors that can occur in Afaan Oromo documents.

- The **AOSC** developed involves many things starting from linguistic study to developing the prototype. As the ambition of this research is to develop a computational framework, the program does not aspire to account for the algorithm execution time. Enhancing the algorithm to minimize the time complexity is also an interest in future works.

- The morphological processes considered in this thesis are inflections, derivations and simple compounding rules. Further linguistic study should be considered to include infix, circumfix and complex compounding.

- The system developed in this research work is just a prototype. Any interested person can do a project to develop a full-fledged Afaan Oromo spell checker that can be easily integrated into different Afaan Oromo NLP works like;
  - ✓ Grammar checkers
  - ✓ Search engines
  - ✓ Machine translation
  - ✓ Dictionaries and the like

- There are a lot of holes in the linguistic study of the language in general, and in morphology and phonology in particular. Linguists should give due consideration to intensively study the language structure and make it available for use in developing computational models.

References:

[1] Jurafsky, D. and Martin, J. H., Speech and Language Processing, An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition Prentice Hall, *1st edition*, 2000.

[2] Tan veer, Tiwary U., Natural Language Processing and Information Retrieval, Oxford university press, 2008.

[3] Bruno Martins, Mário J. Silva, Spelling Correction for Search Engine Queries, EsTAL, pp. 372-383, 2004.

[4] Mansour Sarr, Improving Precision and Recall Using a Spellchecker in a Search Engine, Master's Thesis, Department of Computer Science, Stockholm University, 2004.

[5] Sait Ulaş Korkmaz, G. Kırçiçeği Y. Akıncı, Volkan Atalay, A Character Recognizer for Turkish Language, In Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR), IEEE, 2003.

[6] Li Zhuang, TaBao, Xiaoyan Zhu, Chunheng Wang, Satoshi Naoi, A Chinese OCR Spelling Check Approach Based on Statistical Language Models, International Conference on Systems, Man and Cybernetics, Hague, Netherlands, pp: 4727-4732, IEEE, 2004.

[7] Damerau, F.J., A Technique for Computer Detection and Correction of Spelling Errors, In Communications of ACM, 7(3):171-177, 1964.

[8] Karen Kukich, Techniques for Automatically Correcting Words in Text, ACM Computing Survey, **24**(4):377–439, 1992.

[9] Census Report, Ethiopia's population now 76 million, (2008) available at: http://ethiopolitics.com/news.

[10] G. Q. A. Oromoo, Caasluga Afaan Oromo Jildi I, Komishinii Aadaaf Turizmii Oromiyaa, Finfinnee, Ethiopia, pp: 105-220, 1995.

[11] G. Mangion, Spelling Correction for Maltese, Technical report, Department of Computer Science and Artificial Intelligence (CSAI), University of Malta, 1999.

[12] R. Mizzi, The Development of a Statistical Spell Checker for Maltese, Technical report, of Computer Science and Artificial Intelligence (CSAI), University of Malta, 2000.

[13] Xian Tong and David A. Evans, A statistical approach to automatic OCR error correction in context, In Proceedings of the Fourth Workshop on Very Large Corpora, **4**, pp: 88–100, 1996.

[14] Kazem Taghva, Eric Stofsky, An interactive spelling correction system for OCR errors in text, International Journal of Document Analysis and Recognition, **3**, 2001.

[15] Okan Kolak and Philip Resnik, OCR error correction using a noisy channel model, In Proceedings of the second international conference on Human Language Technology Research, HLT '**02**: 257–262, San Francisco, USA, 2002.

[16] Li Zhuang, Ta Bao, Xioyan Zhu, Chunheng Wang, and S. Naoi, A Chinese OCR spelling check approach based on statistical language models, IEEE International Conference, **5**, pp: 4727 – 4732, 2004.

[17] Masaaki Nagata, Context-based spelling correction for Japanese OCR, In Proceedings of the 16th conference on Computational linguistics, **2**, pp: 806–811, Stroudsburg, USA, 1996.

[18] Masaaki Nagata, Japanese OCR error correction using character shape similarity and statistical language model, In Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, **2**(98): 922–928, Stroudsburg, USA, 1998.

[19] Walid Magdy and Kareem Darwish, Arabic OCR error correction using character segment correction, language modeling, and shallow morphology, In Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, **6**, pp: 408–414, Stroudsburg, PA, USA, 2006.

[20] Surapant Meknavin, Boonserm Kijsirikul, Ananlada Chotimongkol, and Cholwich Nuttee, Combining trigram and window in Thai OCR error correction, In Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics , **2**(98): 836–842, Stroudsburg, PA, USA, 1998.

[21] Peterson, L.J, A Note on Undetected Typing Errors. *In Communications of ACM*, **29**(7): 633-637, 1986

[22] Zobel J., Finding Approximate Matches in Large Lexicons, 1st edition, 1994.

[23] Brill, E. and Moore, R. C, An Improved Error Model for Noisy Channel Spelling Correction, in proceedings of 38th Annual meeting of Association for Computational Linguistics, pp: 286-293, 2000.

[24] Toutanova, K. and Moore R. , Pronunciation Modeling for Improved Spelling Correction, in proceedings of 40th Annual meeting of Association for Computational Linguistics, pp: 144-151, 2002.

[25] Richard Domeij, Joachim Hollman, Viggo Kann, and Mikael Tillenious, Implementation Aspects and Applications of a Spelling Correction Algorithm, Numerical Analysis and Computing Science, Sweeden, 1998.

[26] U. Pfeifer, T. Poersch, and N. Fuhr, Searching proper names in databases, In proceedings of Hypertext - Information Retrieval - Multimedia, Synergieeffekte elektronischer Information systeme, pp: 259–276, Konstanz, Germany, 1995.

[27] R. A. Wagner and M. J. Fischer, The string-to-string correction problem, *JACM*, **21**(1):168–173, 1974

[28] R. Lawrence and Wagner, An extension of the string-to-string correction problem, *JACM*, **22**(2):177–183, 1975.

[29] Stanier, and Alan, How accurate is Soundex matching, Computers in Genealogy, **3**(7): 286-288, 1990.

[30] Holmes, David and McCabe, M. C., Improving precision and recall for Soundex, IEEE transactions on knowledge and data engineering, **15**(6), 2003.

[31] J. Zobel and P. Dart, Finding approximate matches in large lexicons. Software Practice and Experience, **25**(3):331–345, 1995.

[32] Kernighan A., Spelling Correction Program Based on Noisy Channel Model, In Proceedings of COLING-90, the 13th International Conference on Computational Linguistics, **2**, 1990.

[33] Angell, R. Freund, and G. Willet, Automatic Spelling Correction using a Trigram Similarity Measure, Information Processing and Management, **19**(4): 255–261, 1983.

[34] M. K. Odell and R. C. Russell, U.S. Patent Numbers (1,261,167) (1918) and 1,435,663. U.S Patent Office, Washington D.C., USA, 1922.

[35] Peter C., "Soundex – can it be improved?", *Computers in Genealogy*, **6**(5), 1998.

[36] Jennifer Pedler, Computer Correction of Real-word Spelling Errors in Dyslexic Text, Birkbeck, London University, 2007.

[37] R. Murthy, V. Madi, and R.Kumar, A non-word Kannada spell checker using morphological analyzer and dictionary lookup method. *International Journal of Engineering Sciences & Emerging Technologies*, **2**(2): 43-52, 2012.

[38] B. H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM, 137(7):422-426, 1970.

[39] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, Second Edition, MIT Press, Massachusetts, 2003.

[40] D. E. Knuth, The art of Computer Programming, **3**: Sorting and Searching, Second Edition, Addison-Wesley Professional, 1998.

[41] B. A. Sheil, Median split trees, a fast look-up technique for frequently occurring keys, Communications of the ACM, **21**(11): 947-958, 1978.

[42] D. Comer, A note on median split trees, ACM Transactions on Programming Languages and Systems (TOPLAS), **2**(1): 129-133, 1980.

[43] A. V. Aho and M. J. Corasick, Efficient string matching: An aid to bibliographic search, Communications of the ACM, **18**(6):333-340, 1975.

[44] B. W. Watson, A new algorithm for the construction of minimal acyclic DFAs, Science of Computer Programming: **48**(2-3):81-97, 2003.

[45] Leon Davidson, Retrieval of misspelled names in an airlines passenger record system, Communications of the ACM, pp: 169–171, 1962.

[46] R. Kaur and P. Bhatia, Design and Implementation of SUDHAAR-Punjabi Spell Checker, *IJITT*, **1**(1), 2010.

[47] P.Kundra and B.B Charudhari, Error pattern in Bangla text, International Journal of Dravidian Linguistics, 1999.

[48] H. Muaidi and R. Al-Tarawneh, Towards Arabic Spell-Checker Based on N-Grams Scores, *IJCA*, **53**(3), 2012.

[49] Alqrainy S., Ayesh A., and Muaidi H, Automated tagging system and tag set design for Arabic text, *IJCLR*, **1**, pp:55–62, 2010.

[50] Cheng, Charibeth, Alberto, Cedic Paul, Vazir Joshia, "SpellCheF", *JRSCE*, **4**(3), 2007.

[51] D. Ritchie, UNIX spell checker (SPELL), Private Communication, 2006.

[52] J. Bentley, "Programming Pearls", *Second Edition*, Addison-Wesley, Massachusetts, Private Communication, 2006.

[53] M.D Kerninghan, K. W. Church, and W. A. Gale, A spelling correction program based on a Noisy channel model, in Proceedings of the 13th International Conference on Computational Linguistics, **2**, pp: 205-210, 1990.

[54] K. W. Church and W. A. Gale, Probability scoring for spelling correction, Statistics and Computing, **1**(2):93-103, 2006.

[55] R. L. Winkler, Introduction to Bayesian Inference and Decision, Second Edition, Probabilistic Publishing, 2003.

[56] J. Li, X. Wang, and Y. Sun, The research of Chinese text proof reading algorithm, High Technology Letters, **6**(1): 1-7, 2000.

[57] J. Li and X. Wang, Combine trigram and automatic weight distribution in Chinese spelling error correction, *JCST*, **17**(6):915-923, 2002.

[58] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals", Technical report, Soviet Physics Doklady, 1966

[59] Shewangizaw Gulilat, Design and Implementaion of spell checker for Amharic language, Master's thesis, Addis Ababa University, Department of Computer Science, 2009.

[60] Getaneh Woldeyesus, Hunspell Based Uninflected Typographic Amharic Spelling Checker, Master's Thesis, Graduate School of Telecommunications & Information Technology, Addis ababa, 2007.

[61] Tesfaye Tolessa, Early History of Written Oromo Language up to 1900, *Star Journal*, **1**(2):76-80, 2012.

[62] Greenfield Richard. "Ethiopia: A New Political History." Pall Mall Press, London, (1965).

[63] Baxter John. "Ethiopia's Unacknowledged Problem of the Oromo in African affairs." *Journal of the Royal Society*, 77(308): 287-299, 1978.

[64] Kebede Hordofa, The Varieties of Oromo, Ethiopian Languages Research Center Working papers, Addis Ababa University, pp:134-149, 2005.

[65] Wakshum Mekonnen, Development of a Stemming Algorithm for Afaan Oromo Text, Master's thesis, Addis Ababa University, School of Information Studies, 2000.

[66] Diriba Megersa, An Automatic Sentence Parser for Oromo Language using Supervised Learning Technique, Master's thesis, Addis Ababa University, School of Information Studies, 2002.

[67] Addunyaa Barkeessaa, "Sanyii Jechaafi caasaa Isaa (Word and Its structure)", Alem Printing Plc., 2011.

[68] Aberra Nefa. "Oromo verb inflection." Unpublished MA Thesis, Institute of Language Study, Addis Ababa University, 1982.

[69] Assefa W/mariam, Developing Morphological Analysis for Afaan Oromo Text", Thesis, School of Graduate studies, Addis Ababa University, 2005.

[70] Getachew Mamo. "Automatic Part Of Speech Tagging for Afaan Oromo Language", Thesis, School of Graduate studies, Addis Ababa University, 2009.

[71] Baye Yimam. "The Phrase Structure of Ethiopian Oromo." Doctoral Thesis, University of London, 1986.

[72] Debela T. and Ermias A, Designing a rule based stemmer for Afaan Oromo text, *IJCL*, **1**(2), 2010.

[73] Mohammed Ali, Trends in Oromo Lexicon and Lexicography, Studies in Linguistic Sciences, **19**(2), University of Illinois, 1989.

[74] Gragg, G. (1976). "Oromo of Wollega, the Non-Semitic Language of Ethiopia." Michigan State University Press, pp: 166-195, 1976.

[75] Abebe Abeshu, Automatic Morphological Synthesizer for Afaan Oromo, Master's Thesis, School of Graduate studies, Addis Ababa University, 2010.

[76] Aduriz E., and Agirre I., A Word-Grammar based morphological analyzer for agglutinative languages, In the proceeding of the 18th International Conference on Computational Linguistics (COLING), Saarbrucken, Germany, 2000.

[77] Hanada H., A practical comparison of edit distance approximation algorithms, Granular Computing (GrC), IEEE International Conference, 2011.

[78] Min, K., Wilson, W., and Moon. Y, Syntactic and Semantic Disambiguation of Numeral strings using an n-gram Method, Advances in Artificial Intelligence, Springer, Berlin. Pp: 82-91. 2005.

[79] Peter Halacsy, Open language resources for Hungarian, In proceedings of Language Resources and Evaluation Conference (LREC04), European Language Resources Association, 2004.

[80] R. Sproat, W. Gale, C. Shih, and N. Chang, A stochastic finite-state word-segmentation algorithm for Chinese, Computational Linguistics, **22**(3):377-404, 1996.

[81] Temesgen Negassa, "Oromo word formation", Institute of Language Study, AAU, Unpublished MA Thesis, 1993.

[82] Taha R., MODERN AFAAN OROMO GRAMMAR, ISBN: 9781468515060, 2004.

[83] Fikadu Balda, "Analysis of spelling and grammar errors of teaching materials prepared for Afaan Oromo students in Haramaya University", Institute of Language Study, AAU, Unpublished MA Thesis, 2010.

[84] Ijigu Gaja'a, "Qaaccasa Qubeessuu Afaan Oromo Barattoota muummee Afaaniifi Saayinsii Umamaa barattoota kollejjii kamisee", Institute of Language Study, Addis Ababa University, Unpublished MA Thesis, 2012.

[85] Hsuan L. Liang. "Spell checker correctors: A unified Treatment." Master's Thesis University of Pretoria, South Africa, 2008.

[86] Paggio P. and Music B., Evaluation in the SCARRIE project, in the proceedings of the first International conference on language resources and evaluation, 1998.

APPENDICES

Appendix A: Verb Inflection

Some inflection variants of the root word **deem-**'go'

1	deema	16	deemte	31	deemneerra	46	hindeemta
2	deemaa	17	deemtee	32	deemneetu	47	hindeemtaa
3	deeman	18	deemteef	33	deemteettaa	48	hindeemtan
4	deemanii	19	deemti	34	hindeema	49	hindeemtan
5	deeme	20	deemtii	35	hindeemaa	50	hindeemtee
6	deemee	21	deemtiif	36	hindeeman	51	hindeemteef
7	deemna	22	deemtu	37	hindeemanii	52	hindeemti
8	deemnaa	23	deemtuu	38	hindeeme	53	hindeemti
9	deemnaaf	24	deemtuuf	39	hindeemee	54	hindeemtiif
10	deemneef	25	deemu	40	hindeemna	55	hindeemtu
11	deemnu	26	deemuu	41	hindeemnaa	56	hideemtuu
12	deemnuu	27	deemuuf	42	hindeemnaaf	57	hindeemtuuf
13	deemta	28	deemeera	43	hindeemneef	58	hindeemu
14	deemtaa	29	deemeeraa	44	hindeemnu	59	hindeemuu
15	deemtan	30	deemeeraaf	45	hindeemnuu	60	hindeemuuf

Appendix B: Functional Words

Afaan Oromo functional words (***Jechoota maseenaa***)

1	inni	8	akka	15	ta'us	22	ishoo!
2	nu	9	gara	16	kanaafuu	23	elaa!
3	isaan	10	waa'ee	17	garuu	24	ajab!
4	na	11	irraa	18	yookaan	25	muu!
5	ishiin	12	Jala	19	ta'uyyuu	26	magan!
6	ishii	13	wajjin	20	fi	27	uggum!
7	iseen	14	yagguu	21	hata'u malee		

Appendix C: Affixation Rules

Sample suffixation and prefixation rules in our Affix file

#class N2 Nouns ending in short vowel and the last syllable and consonants like {l, r, m}
SFX N2 Y 9
SFX N2 a lan [aeiou'][aeiou]la
SFX N2 a li [aeiou'][aeiou]la
SFX N2 a ran [aeiou'][aeiou]ra
SFX N2 a ri [aeiou'][aeiou]ra
SFX N2 a man [aeiou'][aeiou]ma
SFX N2 a mi [aeiou'][aeiou]ma
SFX N2 a ota [aeiou'][aeiou][lmr]a
SFX N2 a olee [aeiou'][aeiou][lmr]a
SFX N2 a olii [aeiou'][aeiou][lmr]a

#class N3 two syllable nouns ending with short vowels having end consonants {k, g, d, r, b, n}
SFX N3 Y 8
SFX N3 a keen [^aeiou][aeiou]ka
SFX N3 a geen [^aeiou][aeiou]ga
SFX N3 a deen [^aeiou][aeiou]da
SFX N3 a reen [^aeiou][aeiou]ra
SFX N3 a been [^aeiou][aeiou]ba
SFX N3 a neen [^aeiou][aeiou]na
SFX N3 a ni [^aeiou][aeiou][kgdrbn]a
SFX N3 0 a [^aeiou][aeiou][kgdrbn]a

#classn N4 Nouns ending in "eessa, eeysa, eecha, eettii,eeytii, essa"
SFX N4 Y 12
SFX N4 eessa eeyyii eessa
SFX N4 a i eessa
SFX N4 eeysa eeyyii eeysa
SFX N4 a i eeysa
SFX N4 eecha eeyyii eecha
SFX N4 a i eecha
SFX N4 eettii eeyyii eettii
SFX N4 a i eettii
SFX N4 eeytii eeyyii eeytii
SFX N4 a i eeytii
SFX N4 essa eeyyii essa
SFX N4 a i essa

Verb stem prefix (hin)
PFX P1 Y 1
PFX P1 0 hin

Appendix D: Replacement Rule

Sample replacement rules taken from Affix file.

// **REP** stands for Replace

For instance, **REP aa a** means replace **aa** by **a** if the erroneous word contains **a.**

REP ph f
REP f ph
REP dh d
REP d dh
REP ch c
REP c ch
REP ny n
REP n ny
REP a aa
REP aa a
REP e ee
REP ee e
REP i ii
REP ii i
REP o oo
REP oo o
REP u uu
REP uu u
REP ds ch
REP dhs ch
REP dhn n
REP tn nn
REP xs cc
REP tdh dh
REP dht t
REP ls ch
REP bt bd
REP st ft
REP dt dd
REP ln ll
REP gt gd
REP xt xx
REP ct cc
REP jt jj
REP rn rr
REP sn fn

Appendix E: Dictionary

Sample root words with their possible affixation class

Abarraa/N7	baas/F1W2V5	bohaar/F1W2V7	edah/F1W2V2
Abbayyaa/N7	baat/F1W2V4	bokok/F1W2V1	fal/F1W2V6
Baacaa/N7	baatam/F1W2V1	boo'/F1W4	falam/F1W2V1
Caalaa/N7	babal/F1W2V6	booji'/F1W4	falan/F1W2V1
Dabalaa/N7	bad/F1W2V3	boon/F1W2V1	ibs/F1W2V5
Fayisaa/N7	badhaas/F1W2V5	boont/F1W2V4	ibsam/F1W2V1
Gaaddisee/N7	badin/F1W2V1	ce'/F1W3	ibsan/F1W2V1
Jabeessaa/N7	baffam/F1W2V1	ceesis/F1W2V5	id/F1W2V3
aan/F1W2V1	bah/F1W2V5	ceet/F1W2V4	if/F1W2V1
aar/F1W2V7	bakkis/F1W2V5	ciccitt/F1W2V4	ifaaj/F1W2W1
adeem/F1W21	balf/F1W2V1	cich/F1W2V5	ifirr/F1W2V7
af/F1W2V1	balles/F1W2V5	ciibs/F1W2V5	ift/F1W2V4
afadh/F1W2V5	ban/F1W2V1	ciift/F1W2V4	ifteess/F1W2V5
afars/F1W2V5	banaadh/F1W2V5	ciis/F1W2V5	ijaar/F1W2V7
afeel/F1W2V6	baradh/F1W2V5	cim/F1W2V1	laaf/F1W2V1
afeer/F1W2V7	bax/F1W2V8	cims/F1W2V5	laalt/F1W2V4
afuuf/F1W2V1	beek/F1W2V1	cimsan/F1W2V1	laamsha'/F1W3
agar/F1W2V7	beekkan/F1W2V1	cimsit/F1W2V4	laaqam/F1W2V1
aggaam/F1W2V1	beel/F1W2V6	cin/F1W2V1	laat/F1W2V4
agur/F1W2V7	beela'/F1W4	cinc/F1W2V5	labs/F1W2V5
ajaa'ib/F1W2V3	beellam/F1W2V1	ciniin/F1W2V1	lagan/F1W2V1
ajaj/F1W2W1	beelof/F1W2V1	cinqam/F1W2V1	lagat/F1W2V4
ajjeefam/F1W2V1	beeloft/F1W2V4	cir/F1W2V7	lakkaa'/F1W3
ajjees/F1W2V5	been/F1W2V1	ciramt/F1W2V4	lalis/F1W2V5
akeek/F1W2V1	biif/F1W2V1	cirt/F1W2V4	lalist/F1W2V4
alalch/F1W2V5	biift/F1W2V4	cit/F1W2V4	manch/F1W2V2
aman/F1W2V1	bilbil/F1W2V6	coolag/F1W2V3	maqs/F1W2V5
amanam/F1W2V1	bilchaat/F1W2V4	cuf/F1W2V1	mar/F1W2V7
amanat/F1W2V4	billiq/F1W2V8	cuft/F1W2V4	turt/F1W2V4
argit/F1W2V4	birat/F1W2V4	cuqqaal/F1W2V6	tus/F1W2V5
ari'/F1W4	birmat/F1W2V4	cururs/F1W2V5	tuttuq/F1W2V8
asaas/F1W2V5	bit/F1W2V4	cuub/F1W2V3	tuul/F1W2V6
asir/F1W2V7	bitam/F1W2V1	cuubam/F1W2V1	alaf/F1W2V1
asit/F1W2V4	bitamt/F1W2V4	cuunf/F1W2V1	xalal/F1W2V6
atoom/F1W21	bitan/F1W2V1	cuuph/F1W2V8	xax/F1W2V8
awwaal/F1W2V6	bitat/F1W2V4	da'/F1W3	xill/F1W2V6

Appendix F: Sample misspelled words in the Dataset A

Misspelled word	Correct word (from suggestion)	Gloss
Yero	Yeroo	'time'
konkolataa	konkolaataa	'car'
nannoo	naannoo	'environment'
sadarka	sadarkaa	'rank'
dare	daree	'room'
addan	addaan	'in different way'
kitaba	kitaaba	'book'
iddo	iddoo	'place'
graafii	giraafii	'graph'
ibs	ibsi	'describe'
oftti	ofitti	'towards own self'
keess	keessa	'in'
cicimo	ciccimoo	'strong'
iratti	irratti	'on something'
xiqoo	xiqqoo	'small'
bareeffama	barreeffama	'text'
qopheesuu	qopheessuu	'to make ready'
qulquluu	qulqulluu	'clean'
odeeffanoo	odeeffannoo	'information'
fayada	fayyada	'serve'
keni	kenni	'give'
dabauu	dabaluu	'to add'
hirrisa	hir'isa	'decrease'
buaa	bu'aa	'profit'
afaa	afaan	'language'
ba'inaan	bal'inaan	'widely'

Kitaaboota	Kitaabota	'books'
dabareee	dabaree	'turn'
filaachuu	filachuu	'to choose'
feedhiin	fedhiin	'by interest'
yoookaan	yookaan	'or'
tooraa	toora	'order'
caamisaa	caamsaa	'may'
beekune	beekne	'knew'(we)
namani	namni	'human'(Nominal)
ibisi	ibsi	'describe'
obisi	obsi	'tolerate'
dandaeessisu	dandeessisu	'to enable'
hangama	hangam	'how'
fakkaeenya	fakkeenya	'example'
yookaann	yookaan	'or'
ffayyadu	fayyadu	'to serve'
yerroo	yeroo	'time'
arriitii	ariitii	'speed'
dubbishuu	dubbisuu	' to read'
dandhabaa	dadhabaa	'weak'
qabxtii	qabxii	'point'
injjifannoo	injifannoo	'defeating'
atumtti	atumti	'yourself'
gorssa	gorsa	'advice'

Appendix G: Sample Afaan Oromo word signatures

Noun Signatures

Class N1: Nouns ending in long vowels.

For instance, nouns such as *aadaa, aarsaa, amantii, daandii,* and *haroo* are grouped under this category. The suffixes of this class include: *-wwan,- lee , -n,-tiin,-dhaa, -dhaan, -dhaaf,* and *–tii.*

Class N2: Nouns ending in short vowel '***a***' in the last syllable and consonants like ***l***, ***r***, and ***m*** are of this category. For instance, ***wasiila*** 'uncle' and ***daa'ima*** 'child' are nouns of this class. The suffixes of this class include: *-an, -i.*

Class N3: Two syllable nouns each ending in short vowels and having end consonants like ***k***, ***g***, ***d***, ***r***, ***b*** and ***n*** are classified here. For instance, ***laga*** 'river', and ***muka*** 'tree' are nouns of this class. The suffixes of this class include: *-een, -ni, -a*

Class N4: Nouns ending in ***"eessa, eeysa, eecha, eettii, eeytii, essa, eensa"*** take the plural suffix by deleting these endings. Examples are ***jaldeessa*** 'monkey', ***hiyyeessa*** 'poor', ***dureettii*** 'rich woman', and ***bineensa*** 'wild animal'. The suffixes of this class include: *-eeyyii, -i.*

Verb Signatures

Class V1: Stems ending in letters ***f***, ***k***, ***m***, ***n***

For instance, ***daf***, ***adeem*** and ***akeek*** are verb stems of this class. The associated suffixes set for this class is *-na*, *-ne* and *–nu.*

Class V2: Stems ending in ***dh***, ***h***, *apostrophe* (').

The suffixes corresponding to this class are: *-atini, -achise, -achisa, -achisan, -achiste, -achisna, -achistan,-aniiru, -anna, -anne, -annu, -ata, -atani, -ate,-atine, -atte, -attu, - attan, -atu*

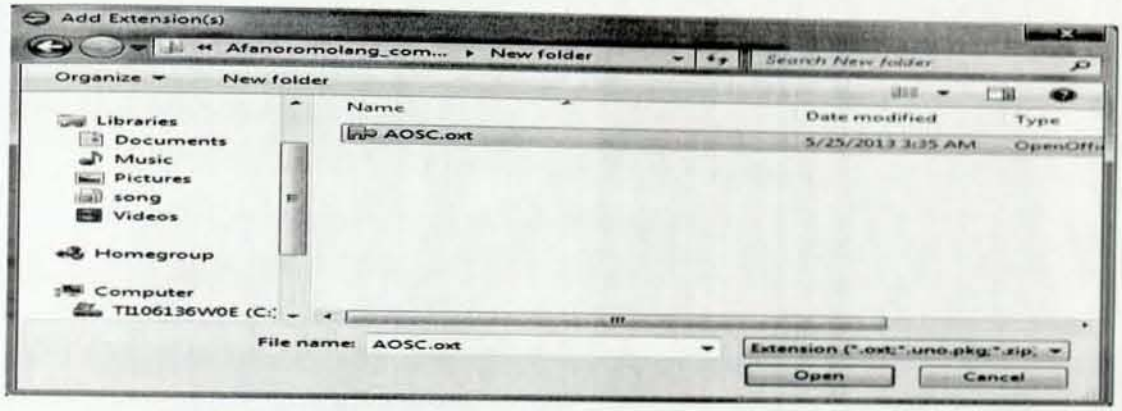
Class V3: Stems ending in ***b***, ***d***, ***g***

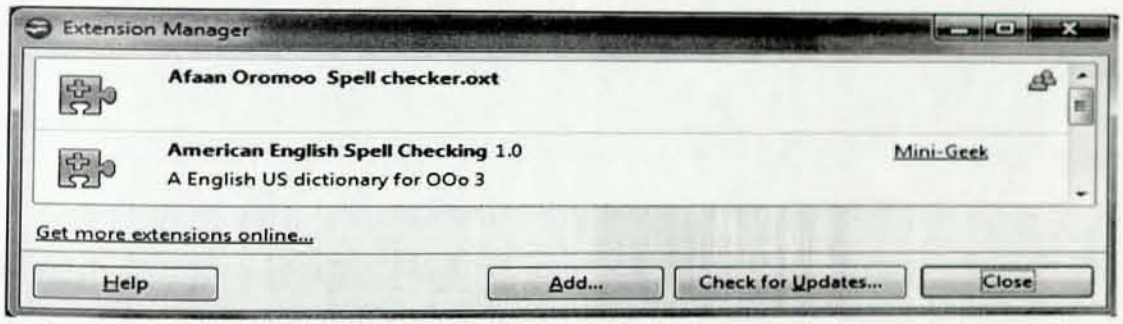
These phonemes have similar phonological characteristics in that they all are voiced groups, and hence they behave similarly in suffixation process [69, 75]. The associated suffixes set for this class is *-da, -di, -dani, -de, -du, - eenya*

Appendix H: Steps we followed to integrate AOSC with OpenOffice 3.4.1

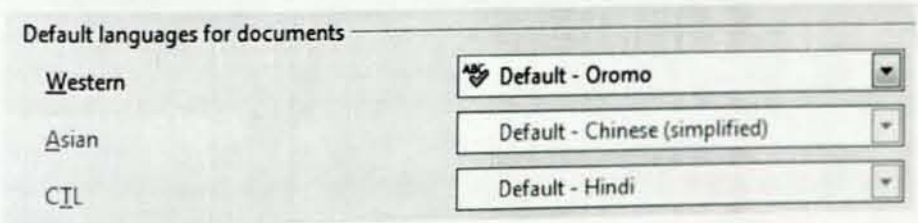
1. Install **Apache OpenOffice 3.4.1**
2. Zipp **AOSC** knowledge base, and then change its file extension to **.oxt** (i.e. **AOSC**.oxt), which was a compatible file type for Apache OpenOffice.
3. Then add **AOSC**.oxt to Apache OpenOffice:

   Go to menu Tools > Extension Manager > Add, then locate **AOSC**.oxt, as indicated below:





4. Restart your PC, and then open **OpenOffice 3.4.1**.
5. Finally configure the general settings. Go to menu Tools>Options>Languages Settings>Languages. Then select Afaan Oromo as default language as shown in the figure below, and then enjoy with it.



Declaration

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

Declared by:

Name: Gadisa O

Signature: _____

Date: 03/07/13

Confirmed by advisor:

Name: Dida Midekso

Signature: Dida M.

Date: 03/07/13

Place and date of submission: Addis Ababa University, June 2013