CHINHOYI UNIVERSITY OF TECHNOLOGY

INSTITUTE OF LIFELONG LEARNING AND DEVELOPMENT STUDIES

CENTRE FOR LANGUAGE AND COMMUNICATION STUDIES

M.PHIL PROGRAMME

Out Of Vocabulary Words in Spell-checking for Southern Bantu Languages: a morphological analysis-based approach for Shona

A dissertation by

Farayi Kambarami

(C18135499V)

A dissertation submitted to the Institute of Lifelong Learning's Centre for Language and Communication Studies in Fulfilment of the Requirements for the Award of the Master of Philosophy Degree in Computational Linguistics

The Approval page:

DECLARATION

I, Farayi Kambarami do hereby declare that this dissertation is the result of my own investigation and research, except to the extent indicated in the acknowledgements, references and by comments included in the body of the report, and that it has not been submitted in part or in full for any other degree to any other university.

……...	…….	…….	……
Student signature	Date	Supervisor	Date

Acknowledgement

I came to the field of research knowing that my knowledge of the subject that I wanted to investigate was sparse. Despite this, I thought that I was prepared for the journey ahead. How wrong I was! To say that the last three and a quarter years have been challenging is an understatement. At the same time, they have been a period of great growth. Paradoxically, I am much clearer that I know even less than I presumed to do at the start of that journey. This is all thanks to the amazing supervisory team that I have had the privilege of being associated with.

When I decided that I was going to work on the natural language processing of Shona, I knew that I wanted Prof Chimhundu to help me walk me through this path. Given his stature in the field, I was not sure that he would be willing to take me under his wing. Again, I misjudged him. Not only did he accept the challenge to mentor me, he has also been a calming influence, patiently directing me through the research process, even offering his own resources to ensure that I succeed. Prof, I can never thank you enough.

Drs Dube and McLachlan have challenged me and gifted me with a deeper appreciation for academic rigour – both in terms of the actual research process itself and, even more importantly, in terms of how this research is finally presented to the relevant research communities. Whilst I have tested the patience of both men, they have been both gracious, continuing to offer support and guidance even when it was clear that I was probably a lost cause…

I also found an academic home with Dr Bozic and his other research students. Listening to the challenges that other (more advanced) students are grappling with and eavesdropping on the research processes that others are following has helped to keep me motivated against the odds. The one-on-one meetings that I have had with Dr Bozic as well as those with the group have provided me with an alternative perspective of what it means to be an academic.

This journey would not have been possible without the support of my line manager, Ceri, our former MD, Spencer Sonn, who encouraged me to take this leap and the team of my direct reports who have had to bear the consequences of my limited availability to support them at times.

Finally, and definitely not least, to my wife Codilia and our two sons Dante Anesu and Anopa Ethan. Even though I am fully alive, they have had to experience the life of a virtual widow and virtual orphans as I have been mostly unavailable to them, especially over the last six months. They have had

to experience this whilst going through multiple other challenges which have not made it any easier for them. To you all, please know that despite all appearances to the contrary, I really love you, and I appreciate your patience and support for me. May God continue to bless and guide you. My gratitude to him is given through my appreciation of your support for me.

December 2021

Abstract:

Spell-checking can be reduced to a dictionary search for the given word in a comprehensive dictionary of the target language. Previous research on South African Southern Bantu Languages (SBLs) has demonstrated that this approach does not work well for conjunctively written agglutinative languages. It is not possible to create comprehensive dictionaries for such languages because their morphology allows them infinite possibilities for creating spoken and written words in real context. In the standard dictionary, the headwords of the entries are very often not complete words but morphemes around which words are built by inflection and compounding. Therefore, when developing spell-checkers, alternative approaches have had to be developed to counter this. In the absence of larger data sets and dictionaries, most of these approaches aim to enhance dictionary sizes synthetically by using various heuristics. Lately a data driven approach has shown promise in delivering effective results without requiring an increase in dictionary size. However, there is limited research on the effectiveness of all these approaches in dealing with out of vocabulary words (OOV). Words are considered to be out of vocabulary if the system is built without being exposed to them. Such words are highly prevalent within conjunctively written languages which include Shona.

This research had two broad aims. First, it seeks to establish the way in which developers of spell checkers have addressed the question of out of vocabulary words within Southern Bantu Languages. Second, it aims to develop a new method for conducting spell-checking of Shona that utilizes morphological analysis to optimize their performance on out of vocabulary words.

A meta-narrative review of the literature on the spell-checking of conjunctively written agglutinative languages was conducted. This revealed the lack of research focus on the question of how spell checkers handled out of vocabulary words. Following this, a finite state transducer based morphological analyser for Shona was developed. Verbs, nouns, and pronouns were prioritised for inclusion in the morphological analyser due to their complexity and relative prevalence. This morphological analyser is called ***M***orphological ***A***nalysis of ***Sh***ona using ***K***nowledge and ***O***bservations (*MAShoKO*). A spell checker for Shona which checks for the validity of Shona spellings in two phases was built based on *MAShoKO*. It starts with a dictionary lookup and then follows this with a morphological analysis for OOV words. OOV words that are not morphologically well formed are flagged as invalid, whilst those that conform with Shona morphology are accepted. This spell checker's performance was then tested against a character trigram language model (CTLM) based spell checker.

The MAShoKO based spell checker outperforms the CTLM spell checker on OOV words for the parts of speech that were encoded into it. However, it does not perform as well on those words whose structure is not encoded in the morphological analyser.

The study concluded that morphological analysis is effective for increasing the effectiveness of spell checkers to handle out of vocabulary words in conjunctively written agglutinative languages. This, however, requires that all the parts of speech be adequately encoded in the morphological analyser.

Pfupikiso

Tsvakiridzo ino ine chinangwa chekuongorora nzira dzinoshandiswa nevagadziri vezviperengeso kana kuti zvirongwa zvinobatsira vanyori kutsvaga mazwi asina kunyatsoperengwa zvakanaka mumitauro yekumaodzanyemba kweAfrica inodaidzwa kuti ma*Southern Bantu Languages* (SBL). Mitauro yeVanhu vekuMaodzanyemba (MVM) iyi inoshandisa nzira dzekunyora mazwi dzinobatanidza zviumbamazwi zvakasiyana-siyana pakunyora kwezwi roga roga. Nokudaro hazvikwaninisiki kuti mazwi ose anoumbika mukutaura nokunyora aiswe muduramazwi. Tsvakurudzo ino inoedza Kukunda dambudziko rekugona kuziva mazwi kwawo angaumbwa mukutaura kana kunyora asi iwo asiri mumatura emazwi anoshandiswa kugadzira zviperengeso izvi. Mitauro yeSBL (MVM) ndeimwe yemitauro inodaidzwa kuti ma*agglutinative languages*. Izvi zvinoreva kuti mazwi mazhinji emitauro iyi anoumbwa nezviumbamazwi zvakawanda uye zvinogona kunyatsopatsanurwa. Pamusana pechikonzero ichi, hazvigone kuti kuunganidzwe duramazwi rine mazwi ose angawanikwa mumutauro werudzi urwu. Nokudaro kunofanira kutsvagwa imwe nzira yekuti vagadziri vezvirongwa zvinobatsira kuperengera zvigone kuziva mazwi anenge akanyorwa zvakanaka chero zvazvo asina kumboonekwa muduramazwi rakashandiswa kugadzira chiperengeso chacho. Tsvakurudzo ino yakatanga nekuongorora nzira dzakamboshandiswa nevamwe vagadziri vezvirongwa zvekuperenga kuti vakunde dambudziko irori. Zvakawanikwa mutsvakiridzo ino ndezvekuti mamwe mabasa ose akaitwa nevamwe aiwanzosimbirira pakuwana nzira dzekuvandudza mazwi ari mumaduramazwi nechakare, nekushandisa nzira dzekugadzira mazwi angawanikwa mumutauro wavanenge vachishanda nawo. Izvi vanozviita nekugadzira zvirongwa zvinogadzira mazwi angawanikwa muduramazwi

Donzvo rechipiri retsvakurudzo ino nderekugadzira nzira itsva dzingashandiswa kuita kuti zviperengeso zvigone kukwanisa kuziva mazwi asiri mumaduramazwi azvo. Tinogadzira chiperengeso chinoshandisa nzira yekuongorora maumbirwo emazwi tichishandisa nzira inodaidzwa kuti *Morphological Analyser for Shona based on Knowledge and Observations* (MAShoKO). Mashandiro echiperengeso ichi anoenzaniswa nemashandiro echimwe chiperengeso chakaumbwa nenzira yakashandiswa pane rudzi rweChiZulu. Zvinoshamisa ndezvekuti mashandiro ezviperengeso izvi haana kunyanyosiyana - izvi zvinoonekwa pakugona kwazvinoita kupenengura mazwi ari mumaduramazwi azvo pamwechete neayo asimo mumaduramazwi azvo. Izvi zvinoshamisa zvakabuda mutsvakurudzo iyi zvinoratidza kuti nzira yekushandisa zvidimbu zvinoumba mazwi kugadzira chiperengeso chemutauro wese zvawo werudzi rwemaSBL, iyo inoshandisa mhando dzekunyora dzinobatanidza zviumbamazwi, inokwanisa kubatsira kuti zviperengeso zvinonyatsoshanda mumitauro iyi zvigadzirwe nekukasika. Chiperengeso chakaumbwa mutsvakurudzo ino chakashandisa semienzano yacho mazwi emhando dzinoti mazita, zviito

nezvirevamwene. Mubvunzo wasara uchiri kuda mhinduro ndewekuti zviperengeso zvakadai zveMVM zvingavandudzwa sei kuti zvishandewo kupenengura mazwi asiri muduramazwi uyezve ari emhando dzemazwi dzisina kushandiswa semienzano yekugadzirisa chiperengo cheChiShona icho chatakabuda nacho mutsvakiridzo ino.

List of Abbreviations

Abbreviation	Meaning
ACL	Association of Computational Linguistics
ACM	Association of Computing Machinery
ADR	Action Design Research
ALLEX	African Languages Lexical Project
AR	Action Research
CL	Computational Linguistics
CS	Computing Science
CTLM	Character Trigram Language Model
CV	Consonant Vowel

Abbreviation	Meaning
CWAL	Conjunctively Written Agglutinative Language
CWSBL	Conjunctively Written Southern Bantu Languages
DGS	Duramazi Guru reChiShona
DGS	A Descriptive Grammar of Shona
DSR	Design Science Research
DSRM	Design Science Research Methodology
DSRPM	Design Science Research Process Model
DWAL	Disjunctively Written Agglutinative Language
FN	False Negative

Abbreviation	Meaning
FP	False Positive
FSA	Finite State Automata
FST	Finite State Transducer
FV	Final Vowel
HLT	Human Language Technologies
HMM	Hidden Markov Chain Model
IEEE	Institute of Electrical and Electronic Engineers
IS	Information Science
IT	Information Technology

Abbreviation	Meaning
L1	First Language
L2	Second Language
LC	Leipzig Corpus
MAShoKO	Morphological Analysis of Shona using Knowledge and Observations
NEG	Negation
NLP	Natural Language Processing
OC	Object Concord
OCR	Optical Character Recognition
OOV	Out of Vocabulary Words

Abbreviation	Meaning
PADR	Participatory Action Design Research
PAP	Personal Absolute Pronoun
PRP	Personal Reflexive Pronoun
RAMESES	realist and meta-narrative evidence synthesis
RQ1	Research Question 1
RQ2	Research Question 2
RQ3	Research Question 3
RQ4	Research Question 4
SBL	Southern Bantu Language

Abbreviation	Meaning
SC	Subject Concord
SDRM	System Development Research Methodology
SDSM	Soft Design Science Methodology
TAM	Tense, Aspect, Mood
TN	True Negative
TP	True Positive
WALS	World Atlas of Language Structures

List of Tables

List of Figures:

List of Code snippets:

Table of Contents

"If you talk to a man in a language he understands, that goes to his head.
If you talk to him in his language, that goes to his heart"
Nelson Mandela.

Chapter 1: Introduction

The use of digital language aides by professional academic writers is extensive for both first language (L1) and second language (L2) English and German speakers (Schcolnik, 2018). Whilst a different study found that the users of such tools report having mixed feelings about their intrusiveness, they nonetheless lauded their utility in the writing process. (Ching, 2018). Despite their reported value, such tools are not available for all of the world's languages (Neubig, et al., 2020). The situation is more pronounced for those languages spoken on the African continent (Mabuya, Ramukhadi, Setaka, Wagner, & Zaanen, 2020). Some work has been done on a few African languages, but it has been slow and has only affected a limited number of languages. Some of the most extensive of this has been on the South African indigenous languages (Moors, Wilken, Calteaux, & Gumede, 2018). However, as will be shown later, this work has not adequately addressed the question of how spell checkers that have been produced as part of this work perform in the *out of vocabulary* (OOV) context. The phrase out of vocabulary is widely used in the speech recognition community and it refers to those words that are not included in the training and/or development of a system (Creutz, et al., 2007; Nijat, Hamdulla, & Tuerxun, 2019; Yang, Zhu, Sachidananda, & Darve., 2019). As a result, these words may not be correctly processed by such a system. This research aims to close this gap by investigating the development of a spell checker that is optimised with the capability to correctly identify and suggest previously unseen words for Shona, one of the languages spoken in Zimbabwe, for which there is no extant spell checker or corrector.

The rest of this chapter is organised as follows: Starting with a broad overview of the topic and context of the study, the background of the study as well as its relevance and significance are then presented. This is followed with the statement of the problem. A section on the research questions and objectives then follows. The nature of the research is described in the next section leading into a brief section covering the assumptions, limitations, and delimitation of the study The final section summarises this chapter and provides an overview of the rest of the study.

1.1. Topic and Context

The existence of appropriate language technology tools like spelling and grammar checkers is of critical value for the development of all language communities. This is because they enable the use of these languages in the digital domain. The digital world is increasingly becoming the locus of most human communication (Twenge & Spitzberg., 2020; Venter, 2019) Tools of equal quality and ability do not exist for all language types and even individual languages (Joshi, Santy, Budhiraja, Bali, & Choudhury, 2020). Agglutinative languages such as Shona, a member of the Southern Bantu Language (SBL) family, are one example of a language group that is inadequately supported by modern applications like Microsoft Word and Google Translate. Like many agglutinative language families, the words in SBL are formed from a relatively small set of building blocks, called *morphemes*, through repeated prefixing and suffixing, a feature referred to as having a *highly productive morphology*. This aspect makes the creation of an exhaustive dictionary for them impossible as a new word can be formed by adding an additional affix to any other pre-existing entry.

The simplest way to approach spell checking is to compare every word to the entries in a dictionary. Since it is not possible to have a dictionary of all words in an Agglutinative language, such static dictionary-based approaches for spell checking are generally of poor quality (de Schryver & Prinsloo, 2004). To remedy this, some existing spell checkers for these difficult languages use various approaches to enhance the size of the dictionaries (Prinsloo & Schryver., 2004). These methods have had some relative success but still suffer from the challenge of not being able to address *out of vocabulary words*. More sophisticated approaches utilise a data driven approach to the identification of misspelt words. This is done in two broad steps. First a language model is built. In most cases this is a statistical representation of how words are formed, however it can take other forms. The second step is to compare words in a document to the language model. Only words that have a high probability of being generated by the language model are accepted as valid. At least one spell checker or an SBL has used such a method. Real-world evaluation of this spell checker demonstrated the shortcomings in its performance with respect to OOV (Keet & Khumalo, 2017). This thesis aims at addressing this shortcoming by developing a morphological analysis based approach to the spell checking of Shona. This approach enhances dictionary lookup with *syllabification* and *morphological analysis* to perform spell checking of Shona. Morphological analysis, which breaks down a word into its smallest "*meaning bearing units"* that are referred to as *morphemes,* aims to optimise the capability to identify and suggest OOV words as corrections for misspelt words (Seidenberg & Gonnerman, 2000). Syllabification is the process of breaking down a word into its constituent syllables and can be utilised by the morphological analyser to parse valid Shona words.

1.2. Focus and Scope

At the time of registering this research project, the aim was to study language models for the Southern Bantu languages. The focus and scope of the study was narrowed from that of addressing this question after some initial work had been carried out on the original question. This study is now focused on the development of a computational method to address the spell checking of highly productive agglutinative languages. The particular emphasis on their performance on OOV words. More specifically, this study addresses the case of Shona. Shona is an SBL of more than 10 million native users (David M, Simons, & Fennig, 2021). The research will be conducted through the investigation of the use of morphological analysis to increase the ability of spell checkers to correctly identify OOV words.

1.3. Relevance and Background

1.3.1. Relevance

The right to communicate in one's own mother tongue is increasingly being recognised and asserted (Milambiling, 2018; De Varennes, 2017). Most communication now takes place in the digital realm. The key to enabling mother tongue communication is the availability of appropriate supporting tools and technologies to enable the unhindered use of all the world's languages. The SBL in general, and Shona in particular, have limited language support in modern technologies. This has the effect of hindering their *frictionless* use on digital devices (Drake, 2019; Frischmann, 2016). This means that even though one can use existing tools and technologies to process these languages, this requires significant effort on the part of the user. For example, the default mode is that all Shona words are marked as incorrect. A user must thus either add every word that they use into the tool's custom (or user defined) dictionary at least once per word or they must switch off the proofing tools whilst using it. Contrast this with the situation for languages such as English. In this case the language tools work so well that users largely trust their suggestions. They also have a comparatively lower need to add unrecognised words into their custom dictionaries. In fact, *proofing tools* are so well integrated into the *document authoring* and other communication tools that they are only noticed when they do not work well or are absent due to some other fault.

There have been some spell checkers developed for the South African SBLs (Grobbelaar & Kinyua, 2009; Prinsloo D. & de Schryver, 2003; Prinsloo & Schryver., 2004; Prinsloo & Eiselen, 2005; Bosch & Eiselen, 2005). However, these developments have not benefited any of the other related languages. An analysis of the utility of one of these spellcheckers through the "*evaluation of its effect on the intellectualisation of isiZulu* " found poorer performance on OoV words as one of the key limitations

of such existing tools (Keet & Khumalo, 2017). Several valid words which did not exist in any of the language's dictionaries had been added to the spell checker's lexicon by its users. This problem of OoV words is important for agglutinative languages with highly productive morphologies. This is because the likelihood of encountering such words in real world scenarios is high, as was previously highlighted. Whilst (Keet & Khumalo, 2017) also report that their tool was able to recognise some previously unseen words, the extent to which it could do this is not quantified. There are also no comparative studies showing how other spell checkers cope with this specific problem.

One of the key digital tools used to communicate these days is the mobile phone (de Bruijn & Brinkman, 2018). Two tools offer similar functionality to spell checkers on these devices. These are i) built-in auto-correctors that are native to each mobile phone platform and ii) third party-built keyboard applications (Vertanen, et al., 2019). A high-level analysis of online reviews of the latter for some SBLs, including Shona, shows that the issue of "missing words" is a recurring theme in the user feedback. This suggests a need for an alternative paradigm to address the spell-checking issue for these languages. This thesis aims at addressing this problem by developing a novel approach to the correct identification as well as the generation of suggestions for corrections of previously unseen words using a combination of a statistical as well as a rules based morphological analysis methods.

1.3.2. Background

At a high level, the problem is that there is no *useful* Spell checker or auto-corrector for Shona. This is despite the apparent *vitality* of the language as well as the clear need for such a spell checker. Let us address language vitality before we consider the usefulness of a spell checker. Language vitality refers to the extent to which a language is used by native speakers and other communities (Fishman, 1991). A vital language is one that has widespread usage in various fields of human endeavour. Shona has both a large speaker population and is *sustained by institutions beyond the home and community* (David M, Simons, & Fennig, 2021).

In this thesis a spell checker is considered to be *useful* if it can correctly identify misspellings when it is used in the real world. For a language like Shona this means that, among other things, such a spell checker needs to be able to work well with and for words that it encounters for the first time when it is in use. No spell checker with these features has been documented for Shona at the time of initiating this research. There are possibly several reasons why this could be the case. Some of these have to do with the nature of the language. Others may have to do with the availability of willing and able developers for such tools.

If we consider the nature of the language, we observe the following: i) Shona is an agglutinative language and ii) it has a conjunctive writing system. Being an agglutinative language means that its words are made up of several clearly identifiable meaning bearing units called *morphemes.* These morphemes can be combined to form new words, whose meanings can be easily deduced from that of the constituent components. A *conjunctively written* agglutinative language is one in which the written form of the words tends to combine these morphemes into one *orthographic word* rather than split them into separate ones. An orthographic word is the word as it is written in text. Some agglutinative languages use a *disjunctive writing* system. In such a system some morphemes forming the *morphological word* are written as separate orthographic words. More detailed definitions for both orthographic and morphological words will be given in Chapter 2. For now it suffices to state that morphological words are words that are defined based on the way that they are formed.

Apart from being agglutinative and having a conjunctive orthography, Shona has a *highly productive derivational* and *inflectional morphology*. *Derivational morphology* refers to the way that languages form words with different syntactic categories or meanings than the roots from which they are derived from through the use of affixes (Bauer, 2008). An example of this can be seen in the creation of the noun <*mutyairi*> (driver) from the verb <*tyaira*> (drive) through the addition of the derivational prefix <*mu*> and the substitution of the terminal vowel <*a*> with <*i*>. *Inflectional morphology* modifies existing words to fit specific contexts without changing the core meaning of the word (Marzi, Blevins, Booij, & Pirrelli, 2020). Noun class prefixes perform this function in Shona as can be seen with the words <*mukomana*> (boy) and <*vakomana*> (boys). Here, the substitution of the prefix <*mu*> with <*va*> only changes the number of the noun, but the meaning of the noun as referring to "human, young male children" is retained. A language is highly productive when the "*number of words that can*" be formed from the morphemes "i*s large*" (GÜNGÖR, GÜNGÖR, & ÜSKÜDARLI, 2019). Since Shona is agglutinative, conjunctively written and has a highly productive derivational and inflectional morphology the number of possible words in the language is very high. This situation is similar to that of the other conjunctively written SBLs.

Despite its high levels of vitality, Shona is considered to be a *less resourced language* in terms of language technology. This also contrasts with the fact that it has been the subject of a lot of well documented scholarly work. Among these is the highly influential Report by Doke which led to the formal adoption of the name Shona as the identifier for the group of related dialects spoken on the Zimbabwean plateau (Doke, 2005). Since then, a number of texts expounding various aspects of the language including its grammar have been written. These include comprehensive grammars by the well-respected Bantu grammarian George. Fortune (Fortune, 1985) as well as a more recent one by a number of largely first language Shona speaking academics (Mpofu, Ngunga, Mberi, & Matambirofa.,

2013). Furthermore, in the 1990s, the language was one of the key focuses of the African Languages Lexical Project (ALLEX) which operated at the University of Zimbabwe and developed a number of monolingual dictionaries for the language as well as a 2 million word corpus (Grønvik & Chimhundu, 1998).

A report by the ALLEX project in June of 1996 noted that the corpus that they had developed should "*ideally have a Shona (and soon an Ndebele) parser*" as well as a '*Shona (and soon an Ndebele) spell checker*" (Grønvik O. , 1996). The language of that report is very hopeful, and there is an air of imminence that is communicated in the transcript. It is now almost 25 years since that report was penned. No "*parser*" or "*spell checker*" for Shona has been realised in this time frame. At the time of the report's writing, Shona was a leader in the field of corpus linguistics. It boasted the largest corpus among the SBLs at 2 million words. Since then, it has lost this leadership position to the other languages of the sub-region. These have since overtaken it in the development of tools that are enabled by the corpora that they compiled years after the Shona Corpus was completed (Khumalo, 2017).

The question can be asked as to why Shona lost its leadership position? Why did it fail to realise the promise of 1996 when it was on the cusp of a major transformation? It appears as if the answers to these are linked to two main problems. The first problem relates to the availability of people with a software engineering/development background who were willing and able to work on the development of the parser(s) and spell checkers mentioned above. For all its successes, the ALLEX project appears to not have attracted sufficient local interest in the computing faculty of the University of Zimbabwe, where it was based (Grønvik O. , 1996; Grønvik & Chimhundu, 1998). Almost all the technical artefacts that it produced were developed by members of the team that were not from Zimbabwe. When these members left, they took their expertise and interest in the development of the tools that had been anticipated with them. The evidence of this is found in a follow up report written in 1998 in which Chimhundu and Grønvik report that whilst there was some interest expressed in getting the local MA and PhD students to take over the morphological analyser that had been developed "*to date tangible steps have not been taken*" (Grønvik & Chimhundu, 1998).

In their report on the status of *Human Language Technologies* (HLTs) for South African Bantu Languages produced in 2010, (Grover, Van Huyssteen, & Pretorius, 2010), found that the development of HLTs for the various languages were at different levels. Specifically, they found that isiZulu had the highest activity, followed by the other languages. They then proposed three explanations for these variances in the development of HLTs for the South African Bantu languages. The first one was to do with the availability of expert knowledge in both linguistics and HLTs. The

second one was to do with the availability of data whilst the third one was due to market forces. Looking at these three, it is clear that each one played a role in the development of the situation described by (Grønvik & Chimhundu, 1998) above. Arguably the issue of the availability of expert knowledge in both linguistics and HLT could have been easily remedied as both types of expertise are known to have been available in the University at the time. The second issue concerning the availability of data was one that the ALLEX project itself was addressing, so it should not have been a factor. The third factor is more difficult to assess for the period in question, but it is known that there is now market interest in HLTs for indigenous languages. This is evidenced by the availability of some tools, albeit rudimentary ones, that attempt to address this need on cell phones.

A second, and much more important problem that has stalled the development of the much needed and anticipated tools is the surprising difficulty of the whole enterprise. Spell checking is a deceptively simple looking problem. The spell checker program is meant to identify incorrectly spelt words in a given text and then propose a ranked list of correct candidate words (Damerau, 1964; Levenshtein, 1966; Kukich, 1992). That second part of the process is actually a spell corrector. Despite this, the two components are usually found together. This has led some researchers to include both pieces of functionality in their definition of the spell checker (Mohammed & Abdellah, 2018) Given this simple description of the problem, a reasonable assumption to make is that the dictionary lookup method should be able to solve this problem. Dictionary lookup-based spell checking has a reasonable level of effectiveness for languages such as English and other less morphologically rich languages. Unfortunately, it does not work as well for a language with a complex morphology like Shona as previously explained. This was established in experiments that were done for isiZulu and other Southern Bantu Languages (SBLs) when the first and second generation of spell checkers were being developed for the South African SBLs (de Schryver & Prinsloo, 2004).

Once the limitations of dictionary lookups for spell checking conjunctively written languages like Shona researchers were understood, researchers started seeking alternative solutions. The majority of this work has been done on the related SBLs. Most of it has taken a rules based approach (Prinsloo & Schryver., 2004; Prinsloo D. & de Schryver, 2003; Grobbelaar & Kinyua, 2009). The key aim has been that of augmenting the sizes of the dictionaries to be used in the actual spell-checking task. This spell checking is still done via a dictionary lookup. At least one study took the alternative data driven approach to the problem (Ndaba, Suleman, Keet, & Khumalo, 2016).

The data driven approach produced some promising results, leading the authors to conclude that *a data-driven approach toward spellchecking is indeed feasible for at least isiZulu, and, by extension of the approach that is essentially language-independent, all Bantu languages* (Ndaba, Suleman, Keet,

& Khumalo, 2016). However, a subsequent study found that the system failed to identify some valid words that did not exist in the language's dictionaries during real world use cases (Keet & Khumalo, 2017). Furthermore, there has not been a significant uptake of this approach for other languages - at least not in the published literature. It is possible that this could be because, as they also conclude, *the accuracy of such a spellchecker depends on the text corpus used for training the model as well as on the text document or corpus with which it is tested* (Ndaba, Suleman, Keet, & Khumalo, 2016). Such good quality *gold standard* corpora are not readily available for all languages, and even where they exist, many are not current. The same study showed poorer performance of the spell checkers when used with older corpora.

The main concern of this research is on the performance of spell checkers on OOV words like those encountered by (Keet & Khumalo, 2017)'s spell checker in real world usage. None of the other studies report on the performance of their spell checkers in such contexts. However, based on the description of the approaches that they take, it can be assumed that they would do equally badly. This means that their usefulness in real world situations is not as good as they could potentially be, which is a significant limitation. Previous work in spelling for SBLs was initially reported (Prinsloo & Schryver., 2004; Prinsloo D. & de Schryver, 2003; Grobbelaar & Kinyua, 2009). This was followed up by (Prinsloo & Schryver., 2004) who improved on their results by adding what they termed *clusters of circumfixes* to generate additional words for their spell checker, whilst (Bosch & Eiselen, 2005) were also able to produce better results when they bolstered their dictionary look up with the use of regular expressions for some morphological analysis. Their spell checker was able to recognise words that did not exist in the lexicon. However, they noted that it could only make suggestions of words that were part of the dictionary's lexicon.

1.4. Problem Statement

This project addresses the problem of the correct identification of previously unseen correctly spelt words. It also addresses the generation of accurate suggestions of previously unseen words in spell checkers for conjunctively written agglutinative languages, using Shona as an example. A solution which addresses this problem using a morphological analyser-based spell checker will be presented and evaluated. This problem is an important one as spell checkers that do not address OOV words are bound to have limited utility in real world settings for languages such as Shona. The likelihood of encountering new words during the use of the spell checker is high, thus the need to be able to correctly handle them. More importantly, there is currently no widely used spell checker for Shona, so the development of any spell checker is a significant milestone in the digital development of the language.

1.5. Research Objectives and Questions

The objective of this research study is to determine the effectiveness of using morphological analysers to increase the performance of spell checkers on previously unseen words. This will be achieved through the development and evaluation of a Morphological Analyser for conjunctively written Southern Bantu languages such as Shona. This research objective can be further broken down into the following sub-objectives:

1.5.1. Objectives

RO1. Determine the challenges encountered in the development of spell checkers that aim to maximise the correct identification of OOV words for SBLs.

RO2. Determine the previous approaches utilised in the development of spell checkers that aim to maximise the correct identification of OOV words for SBLs.

RO3. Design a morphological analyser for Shona.

RO4. Design a Shona spell checker that utilises morphological analysis and embedded knowledge to optimise its performance on OOV words.

RO5. Implement the morphological analysis based spell checker.

RO6. Implement a basic character n-gram language model-based spell checker for Shona.

RO7. Evaluate the general performance of the morphological analysis-based spell checker against a basic character n-gram model-based spell checker.

RO8. Evaluate the performance of the morphological analysis based spell checker for Shona on OOV words against a simple character n-gram model.

1.5.2. Research Questions:

The research objectives will be addressed by answering the following research questions.

RQ1. What are the challenges with the previous approaches used to maximise the correct identification of OOV words for SBLs?

RQ2. What are the approaches that have been utilised to develop spell checkers that aim to maximise the correct identification of OOV words in SBLs?

RQ3. What is a good design for a morphological analyser that can correctly identify Shona verbs and nominals Shona?

RQ4. What is a good design for a Shona Spell checker that utilises a morphological analyser in order optimise its performance on OOV words?

RQ5. What is the overall performance of a Shona spell checker that utilises the morphological analyser as well as knowledge of the language to develop a sub-word aware spell-checking engine for Shona words across the most frequent word types?

RQ6. What is the performance of a morphological analysis based spell checker on OoV against a simple character n-gram model?

1.6. Significance of the Study

There is currently no documented spell checker for Shona, which makes this study of great significance to the language. Furthermore, this research will contribute a few key tools to both the language and to the body of knowledge on spell checking for agglutinative languages. The key contributions of this study are as follows:

1. The study will produce the first academically documented and research-based Spell Checker for Shona.
2. It will also introduce a framework to evaluate the performance of spell checkers for conjunctively written agglutinative languages on out of vocabulary or previously unseen words.
3. A new method to perform spell checking through the utilisation of a knowledge-based approach [checking of open syllables as well as permitted consonants] to identify misspelt words based on their non-adherence to the language's syllabic inventory will also be introduced.
4. An improvement to the existing methods of utilising sub-words to determine the correctness of words, especially those that do not exist in the dictionary will also be developed.
5. Finally, a method to utilize sub-words to propose correct words including those that may not have been encountered in the training data will be introduced and utilised in the development of a working spell checker and corrector.

1.7. Approach and Methods

This study was conducted through a combination of a detailed literature review as well as Design Science. RO1 and RO2 required the determination of the previous approaches as well as the challenges encountered in the development of spell checkers that aim to maximise their performance on OoV words. These were addressed through an extensive meta-narrative survey on the development of spell checkers for SBLs and other agglutinative languages. This entailed the identification of search terms and the subsequent search for such literature. The resultant literature was then be filtered for

relevance before being subjected to analysis for key themes and issues. The resultant themes and issues were then presented in the form of a systematic literature review of the development of spell checkers for conjunctively written SBLs.

To satisfy RO3 a finite state transducer based morphological analyser for Shona was designed and built. An appropriate grammar for Shona was then selected from the existing published grammars for the language. The grammar rules were encoded into a finite state transducer which was then incorporated into a spell checker for Shona in. This spell checker was designed and implemented to fulfil RO4 and RO5 The morphological analyser can be used for two functions within the spell checker. It can be used to check the validity of presented words. It can also be used to create suggestions for misspelt words.

RO6 was met by developing a simple n-gram based spell checker and corrector, similar to the one that is reported by (Ndaba, Suleman, Keet, & Khumalo, 2016). The performance of this spell checker on previously unseen words was then compared against the performance of the Morphological Analyzer based spell checker developed to meet the requirements of RO7 and RO8. The results of this experiment will be tabulated, presented, and discussed.

1.8. Assumptions

This study is based on several assumptions. The first key assumption is that the grammars that are available for the language provide an adequate description of the language for use in the development of their computational morphological models. It is also assumed that the corpora used are representative of the language's current usage.

1.9. Summary and organisation of the remainder of the study

The rest of this thesis is set out as follows: Chapter 2 reviews the literature on spell checking as it pertains to conjunctively written agglutinative languages and introduces the concepts and terms covered in the thesis. Chapter 3 then discusses the methods used in Computing Science research and provides justification for the use of the Design Science Research Methodology in this research. The details of how each step of this methodology were applied in this thesis are also presented in this chapter. In Chapter 4 the material and methods used to conduct the experiments reported in this study are reported on. This is then followed by chapter 5 which is a presentation of the results of the experiments that were conducted in this research. The presentation of these results is then followed by

a discussion and analysis of the results in chapter 6. Chapter 7 concludes this thesis with a summary and some concluding remarks.

"Most controversies would soon be ended, if those engaged in them would first accurately define their terms, and then adhere to their definitions."

Tryon Edwards.

Chapter 2 - Literature Review

2.1. Introduction

It has been argued that human "*language is inherently ambiguous*" (Lebeaupin, Rauzy, & Roussel, 2017). This ambiguity which is partly caused by the multiple uses and meanings of words has also been shown to make human communication more efficient (Piantadosi, Tily, & Gibson., 2012). Despite this, ambiguity can be costly, especially in scientific literature (Fischhoff, 2013). It is thus important that the language used in this domain is precise. The purpose of the present chapter is to enable understanding of the terms and concepts that will be utilised in this thesis as well as to place this study within the greater context of similar studies on spell checking for agglutinative languages.

2.2. Context to terms and concepts

This research is focused on spell checking of Shona in the OOV context. Figure 2.2-1 maps the general setting within which spell checking for conjunctively written agglutinative languages operates. First the problem of spell checking applies to a specific language. The focus of this thesis is on conjunctively written agglutinative languages (CWALs). These will be fully defined in section 2.4.2. In these languages, as in all known human languages, the basic unit of speech or text is the sentence. Such sentences are composed of words. Section 2.4.4 discusses the concept of words within the CWALs. Words in CWALs can be further broken down in several ways. First, they can be decomposed into their constituent morphemes using morphological analysis as described in section 2.6. Second, they can be broken down into short sequences of the characters from which they are formed as covered in section 2.5.2. Third, for a language like Shona where every word is made up of open syllables, each word can be broken down into the syllables from which it is formed. Syllabification is described in section 2.6.3. Critically, a spell checker is tasked with the job of evaluating whether any given word is a valid word in the language of the spell checker. Section 2.4.2 discusses the question of what constitutes a valid word before introducing other concepts related to spell checking for CWALs.

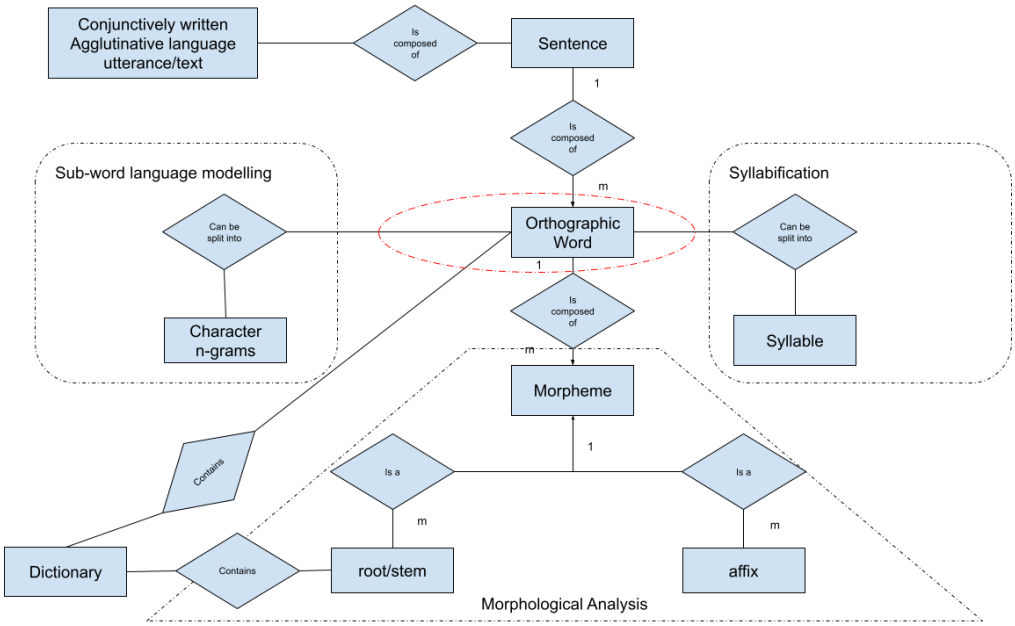


Figure 2.2-1 Research Context

2.3. Spell Checkers

A spell checker is a piece of software that checks for the presence of incorrectly spelt words within a text (Kukich, 1992). It may incorporate a spelling corrector, which suggests corrections for every incorrect word encountered. Some types of spell checkers implement this functionality as an auto-corrector, which, as the name implies, automatically selects the most probable candidate correction, and uses it to replace the incorrectly spelt word. The definition of what constitutes a valid word in any given language is a matter of continuing scholarly debate. For this reason, the next section provides a cursory explanation for some linguistic ideas that pertain to the identification of what a word is and how this affects the development of spell checkers for different language types.

2.4. Language

There are many kinds of languages in the world. Some of these, including programming languages, are man-made and every aspect of them have precise specifications. This study is about ***human*** or ***natural language,*** also referred to simply as ***language***. (Sapir, 1921) defines language as "*a purely human and non instinctive method of communicating ideas, emotions, and desires by means of a system of voluntarily produced symbols.*" Another definition says that it is "*a system of communication based upon words and the combination of words into sentences.*" (Eifring & Theil, 2015). Whilst it has some limitations, this is the definition that will be used in this research. Spell

checkers are designed to check the validity of words within human languages. One of the key distinctions between artificial and natural languages is that natural languages do not have the same level of formal specifications that artificial languages have. This poses some technical challenges for the developers of rules-based NLP tools and the rules as will be discussed later.

2.4.1. Sentences

(Eifring & Theil, 2015)'s definition sees language as being composed of ***words*** and ***sentences*** and these being the means of achieving communication. There is thus a need to clarify what sentences and words are. Whilst the key focus in this thesis is the word, we start by looking at the sentence. The Oxford Dictionary defines a sentence as "*a set of words that is complete in itself, typically containing a subject and predicate, conveying a statement, question, exclamation, or command, and consisting of a main clause and sometimes one or more subordinate clauses.*" In natural language processing, a sentence has a deceivingly simple definition. According to (Grefenstette & Tapanainen., 1994) "*sentences end in punctuation*". However, as they also observe, determining which punctuation marks terminate sentences is not a trivial matter. The spell-checking problem addressed here occurs at the individual word level, so these challenges will have little or no bearing to the subsequent discussion. However, it is important to note that a production scale spell checker should be able to differentiate words from sentences.

2.4.2. Words

A spell checker deals primarily in words. Developers of spell checkers need to understand what constitutes a word in each language. Words have a much more complex definition than appears on the surface. (Haspelmath, 2011) considers the question of what constitutes a word and identifies four different criteria that could be used to decide on what a word is. These are the ***semantic***, ***orthographic***, ***phonological*** and ***morphosyntactic*** criteria. When faced with the same question, (Bejan, 2017) expanded these criteria into five categories by splitting the morphosyntactic into ***morphological*** and ***syntactic*** criteria.

The Semantic word

(Packard, 2000) states that most traditional definitions of what a word is take the *semantic* or lexical view. Quoting Sapir, they define a word as "t*he outward sign of a specific idea, whether of a single concept or image or of a number of such concepts or images definitely connected into a whole"* (Sapir, 1921). However, they argue that the semantic word is *only minimally useful, because reducing concepts to their semantic primitives is a notoriously difficult exercise* (Packard, 2000)*;* hence the

need for other ways to define what a word is. Whilst more sophisticated NLP tools may need the capability to identify semantic words, the spell checker discussed in this research does not require it.

The Orthographic or Graphemic word

The ***orthographic*** or ***graphemic word*** is the word as defined by writing conventions. In those languages that use the Roman script, it is defined as a string of letters that are found between spaces or punctuation marks in written text (Krause, 2012). This is the word that is found in written texts and is the primary concern of this study. It is important to note that the orthographic word in most languages is the result of convention. Language authorities determine how words should be written. This idea will be discussed further in the section on writing systems.

The Prosodic word

(Dixon, 1977) introduced the concept of a ***prosodic*** or ***phonological word*** (Hildebrandt, 2015). This refers to the word that is defined according to phonological properties of a given language in accordance with the prosodic hierarchy hypothesis. A discussion of the prosodic hierarchy is beyond the scope of this study. What is key to note is that one of the challenges that speaker-writers of any given language have is to correlate the way that they speak the language and how it is written. Some challenges with English spelling are to do with the way that the language's orthography differs with its prosody (Enderby, Carroll, Tarczynski-Bowles, & Breadmore, 2021).

The Morphological word

Anderson, quoted by (Packard, 2000), defines the ***morphological word*** as '*a base together with the expression of the [grammatical] categories appropriate for its part-of-speech class*'. In the same work, (Packard, 2000) provides a simpler definition which says that it is the "*proper output of word-formation rules in the language*". What is important to note here is that the morphological word in this case presupposes the existence of some clear word-formation rules in each language. This means that these rules can be automated and used both for the generation and validation of words in the language. It will be shown later that the morphological word has great importance and significance for languages like Shona in which the inventory of words is open and increasing due to the way that they are formed.

The Syntactic word

The definition for the syntactic word is even more dense than that for the morphological word. The **syntactic word** is defined as "a form that can stand as an independent occupant of a syntactic form class slot, in other words, a syntactically free form, commonly designated in the literature as Xº" (Packard, 2000). Syntacticians do not yet have a consensus as to the precise definition of what a syntactic word is (Svenonius, 2018). Whilst higher order, context sensitive spell checkers may need to be able to distinguish between syntactic words, the spell checkers that will be addressed in this study do not need to meet this requirement as will be explained in section 2.5.1.

2.4.3. Morphemes

Regardless of how words are identified, they can be further analysed in several possible ways. One such way is based on their building blocks, which are called morphemes or morphs. A **morpheme** is the smallest meaning bearing unit of a word and a word may be made from one or more such morphemes (Anderson, 2015). Morphemes can be divided into types. One set of morphemes are called ***bound*** or ***free morphemes*** (Martini, 2016).

Free or unbound morphemes are those morphemes that can form valid words without being combined with any other morphemes (Martini, 2016). An example of such a morpheme is "*kind*". It cannot be broken down to any other smaller meaning bearing unit and it is a valid word on its own. Bound morphemes on the other hand cannot form valid words on their own (Martini, 2016). They can only be used together with a free morpheme. For example, the morpheme "*ness*" in "*kindness*" cannot form a valid word, but it changes the meaning of the word "*kind*" when it is appended to it. It will be shown later that the spell-checking task for conjunctively written agglutinative languages can be made easier if the morphemes that make up each word can be identified - since spell checking in these languages can be viewed as morphological analysis, among other conceptualisations.

Roots and Stems

Bound morphemes play different roles within a language. One set of bound morphemes are what are referred to as ***roots*** and ***stems*** of words. (Bakovic, 2003) defines these as follows: "*(i) a stem is any morphological constituent to which an affix may attach"* and a root as *"the innermost stem.*" Roots give the core meaning of the word. One way in which speaker-writers of Shona can identify valid Shona words is through observing whether a given word includes a valid root or stem. Some roots/stems can also be realised as free morphemes in some Shona dialects. For example, the root "*mbwa*" (dog) is a complete word in ChiKaranga. Similarly, "*she*" (king/lord) is also a full word in

that dialect even though both morphemes require the stabiliser "*i*" to form valid words in the Zezuru and other Shona dialects.

Affixes

A second set of bound morphemes are called ***affixes*** (Miti, 2006). Most languages have two types of affixes, these being ***prefixes*** and ***suffixes***. Prefixes are those affixes that can be appended in front of word roots whilst suffixes follow word roots. Some other languages also have ***circumfixes*** and ***infixes***. Circumfixes are combinations of prefixes and suffixes that appear together (Hendrikse & Mfusi, 2011), whilst infixes are those affixes that may be added to a root/stem of a word (Umar, 2020). Most Shona words consist of at least one affix and a stem. Even when the word is unknown, checking to see if it contains a valid set of affixes attached to a valid root/stem can confirm its correctness. The process of doing this is referred to as *morphological analysis* and will be discussed in detail in section 2.6.

Derivational Morphemes

Affixes perform one of two key functions within words. They are either ***derivational*** or ***inflectional***. *Derivational* morphemes change the meaning or part of speech of the free morpheme whilst *inflectional* morphemes provide additional grammatical meaning to the free morpheme (Tariq, et al., 2020). For example, in Shona, noun prefixes are inflectional. Their primary function is to provide information about the number and class membership of a specific noun. Thus, the prefixes "*mu*" and "*va*" in the words "*mukomana*" and "*vakomana*" indicate that the first word is a singular noun whilst the second one is a plural noun and that both of them refer to people.

Inflectional Morphemes

Shona verbs can be inflected by both prefixes and suffixes as in the following example: "*akazoendeswa*". Here the root/free morpheme "*end*" (go) is inflected by adding the prefixes "*a*" (first person singular), "*ka*" (remote past tense) and "*zo*" (then) which carry the meaning "s/he eventually did something in the remote past". The suffixes "*es*" (causative mood) and "*w*" (passive mood) bring the additional meaning that this was caused to happen to the passive subject of this verb, thus the full word means "s/he was eventually caused to go → s/he was eventually taken [to some place]"

Nominalisation

It is also possible to ***nominalise*** a verb by appending specific prefixes and suffixes to it. For example, the Shona noun "*mutyairi*" (driver) is formed by adding the prefix "*mu*" to the verb stem "*tyair*" (to drive) and the terminal vowel "*i*". Similarly, the verb stem "*nyor*" is transformed into the noun "*chinyoreso*" (pen, lit. something to write with) by appending the noun prefix "*chi*" (inanimate single object) and the causative verb extension suffix "*es*" together with the terminal vowel "*o*".

Lemma

If a human speller wishes to look up the spelling of a word in a dictionary, they typically lookup for the *lemma* in a dictionary. (Burchfield, 1985) defines a lemma as "*a set of grammatical words having the same stem and/or meaning and belonging to the same major word class, differing only in inflection and/or spelling*". For instance the Shona words "*rova*", "*kurova*", "*rohwa*" and "*kurohwa*" have the same lemma "-rova". In the Shona dictionary, it is the lemma that is the headword in the entry that carries the basic definition. The headwords therefore are not always full words, i.e. inflected.

Lexical and Surface forms of words

(Bonami, Boyé, Dal, Giraudo, & Namer, 2018) assert that linguists distinguish between two types of words, stating that "*Those that constitute dictionary entries are usually called lexemes."* An alternative definition holds that the lexical form of a word, refers to "*a lemma (that also codes for syntactical markings such as gender and number)*" (Ralph & Lambon, 2001). Whilst these are the basis upon which surface words are formed, this may not always be transparent in every language and writing system.

Two examples can serve to illustrate the concept of a lexical word. The word "*gava*" (fox), when pluralised becomes the word "*makava*" (foxes). Similarly, the word *gudo* (baboon) becomes *makudo* (baboons) when in the plural. Linguists postulate that there is a process which is used to get from *gava* to the expected regular plural form *magava* and then finally to the proper plural *makava*. This process can be conceptualised as being the existence of a set of phonological rules which determines the sounds that can follow each other. In the case of *magava* or *magudo*, these presumed rules change the *g* sound to the *k* sound whenever these are preceded by the plural class marker *ma*. "*Magava"*, which is not a real word in Shona, is the lexical or the analytical form of the word *makava*.

The surface form of a word is the string of characters as they are found in actual texts (Schütze, 1992). For example, the surface form of the word "kindness" is the word itself. In the previous Shona

example of *gava*/*makava*, *makava* is the surface form of the word *gava*. Similarly, *makudo* is one of the surface forms of the word *gudo*.

In the context of spell checking, it is necessary that they have the capability to correctly identify surface forms of words even though these may not always be found in the dictionary. This may require them to be able to deduce the surface forms from the set of lexical forms that may be available within their lexicons.

2.4.4. Language Typology

A key question for the developers of NLP tools is the extent to which solutions developed for one language can be transferred to other languages. The answer to this question can be obtained by observing the ways in which the languages of the world bear similarities with one another. An outcome of this method of inquiry is the identification of what are termed "*linguistic universals*". As (Miti, 2006; Greenberg, 1960) state, there are a number of such universals leading to the classification of languages into various types - referred to as "*language typology*" (Kashyap, 2019). One of the oldest topologies, which is of interest to this study and developers of NLP tools in general, is that of morphological ***typology*** (Greenberg, 1960). This classifies languages based on how their morphological words are formed.

If we consider the ease with which it is possible to separate the morphemes within the words of any language, we come up with a classification of the world's languages into three main groupings. These three groupings are that of the 1. *isolating*, 2. *agglutinative* and 3. *fusional* languages (Greenberg, 1960). It has been shown that these categories correlate strongly with the difficulty to model a given language, and thus to develop NLP tools for them (Gerz, Vulić, Ponti, Reichart, & Korhonen, 2018).

Isolating Language

Isolating languages are those languages in which every word is made of only one free/unbound morpheme (Greenberg, 1960). There are no bound morphemes, so it is easy to identify the boundaries of each morpheme in every word because it is the same as the word boundary. The task of conducting spell checking for these languages is comparatively easier as they tend to have more fixed vocabularies.

Agglutinative Languages

According to (Miti, 2006), agglutinative languages are those languages that on average have more than one morpheme per word and where each of these morphemes is easy to identify. The morphemes within agglutinative languages each also carry only one meaning per morpheme. Spell checking as well as other language modelling tasks are more complex for the subset of the agglutinative languages that utilise a conjunctive writing system rather than a disjunctive one.

Disjunctively versus Conjunctively written Agglutinative Languages

In some agglutinative languages, the morphological word does not correspond to the orthographic word. This is the case for languages such as SeSotho and SeTswana. In these languages bound morphemes that belong to the same morphological word are written separately as if they are free morphemes. For example, the morphological "word" *"kea u rata*" (I love you) is rendered as three separate orthographic words. Contrast this with the Shona rendering of the same phrase which is "*ndinokuda*". In both "words" the morphemes "*kea*" and "*ndino*" have the exact same meaning - they roughly translate to "*I do <what the root says - in the present continuous tense>*". Similarly, the morphemes "*u*" and "*ku*" both mean "*you*" whilst the morphemes "*rata*" and "*da*" both mean "love". The first form of writing is referred to as a disjunctive writing system whilst the latter is a conjunctive writing system. Sotho is a *disjunctively written agglutinative language* (DWAL) whilst Shona is a *conjunctively written agglutinative language* (CWAL). More specifically, it is a conjunctively written Southern Bantu language CWSBL. CWSBLs are a subset of CWALs. It was shown by (de Schryver & Prinsloo, 2004) that dictionary lookup based spell checkers are very effective for DWALs but they perform dismally on the CWSBLs.

Fusional Languages

(Miti, 2006) further states that fusional languages are those languages which also have more than one morpheme per word. However, unlike agglutinative languages, each morpheme may have more than one meaning and syntactic role within the word. This category of language types is only included here for completeness' sake as they have no bearing on the rest of the work presented in this thesis.

Analytic Languages

An alternative way to classify languages is based on the degree of internal complexity of the words (Aikhenvald, 2007). This classification places the world's languages into two broad categories: one for analytic languages and a second one for synthetic languages. The category of the ***analytic***

languages coincides with that of the isolating languages. They are characterised by having words with very little internal complexity.

Synthetic Languages

On the other hand, ***synthetic*** languages are those which have more than one morpheme per word. The former categories of agglutinating and fusional languages all fit into this category. A third category of ***polysynthetic*** languages is sometimes added to these groupings. In this system of categorisation, Shona and other SBLs are synthetic languages.

Morphological Complexity

These above methods of morphological classification can be applied together to provide a richer understanding of the complexity of given languages. They correlate quite well with an additional concept of ***morphological complexity***. According to (Pirinen, 2014), one of the key features of morphological complexity is the average number of morphs per word which can be measured using a morph-to-word ratio. Related to this first feature is morphs-per-morpheme. Then there is '*the rate of productive derivation and compounding that produces new ad hoc words and word forms that are hard to predict using finite word lists*' (Pirinen, 2014). A language that has both features is considered to be morphologically complex. According to all these criteria, Shona and other conjunctively written SBLs are morphologically complex. This suggests that methods developed for less morphologically complex languages may not be suitable for it as was reported in (Gerz, Vulić, Ponti, Reichart, & Korhonen, 2018).

Summary position on words

The preceding linguistic discussion sought to clarify the concept of *wordhood* and how it impacts the performance of spell checkers. This study is on the spell checking of Shona, a conjunctively written, agglutinative, Southern Bantu language. Since the language uses a conjunctive writing system, it has a large inventory of words and thus spell checking cannot be performed using simple word lookup algorithms. A system based on such an algorithm would encounter a high proportion of OOV words.

The question that this thesis will attempt to answer is whether the use of the understanding of the components of the morphological word and the conventions that determine orthographic words in Shona can improve the performance of spell checkers on these OOV words. The next section considers the broad language technology concepts that apply to the spell-checking problem.

2.5. Language Technology

Language technology is a broad term that refers to all the computational tools that are used to process human language (Uszkoreit, 2000). These range from simple tools such as "spell and grammar" checkers like the ones discussed in this thesis, to even more sophisticated tools such as speech recognition and synthesis tools. The language technology that this thesis is investigating is that of spell checkers.

2.5.1. Spelling Errors and Spell Check types

One of the key reference works on spell checking is the survey by (Kukich, 1992). Pertaining to the various types of spell checkers that can be developed, it states that automatic word correction research focuses "*on three increasingly broader problems:(1) non word error detection; (2) isolated-word error correction; and (3) context-dependent word correction.*" This research focuses primarily on the first two types of spell checkers, with the greater focus being on the first one.

Non-Word Errors

Spelling errors generally fall into two main categories. The first category is that of ***non-word errors***. These occur when an author of a given text inputs a word that does not exist in the lexicon of the specific language. For example, an author could type the word "*errar*" instead of "error". The token "*errar*" is an instance of a non-word spelling error as it does not exist within the English lexicon. The same would apply in Shona if one typed "*vsna*" (meaningless in Shona) instead of "*vana*" (children/ four/ with).

Real word errors

A second type of spelling error is that of ***real word errors***. This is the kind of error that occurs when the misspelt word is a valid word in the language of the text. An English example of this is the use of the word "*their*" in place of "*there*" in the sentence "*Their they are.*" A Shona example is of this is the word "achienda" in the sentence *Ndakavaona achienda kubasa* (I saw them (or honorific singular) going (singular, non honorific verb) to work). Whilst there are some authorities who hold that detection and correction of real word errors falls outside the remit of true spell checking and fits rather into the category of grammar checking, the taxonomy from (Kukich, 1992) places them within the broader spell checking framework. This study will not attempt to identify real world errors.

Non-word error detection

In its most basic form, the goal of non-word error detection is relatively simple: it is to identify any words that do not exist in a predefined word list or dictionary. Essentially it boils down to a pattern matching problem. The problem becomes complex when the target language is morphologically complex. Whilst it is possible to develop a comprehensive dictionary of most of the words that occur in the analytic/isolating languages - the same cannot be said of the agglutinating languages - especially those with greater morphological complexity.

Where morphological complexity limits the effectiveness of dictionary lookup-based spell checkers, alternative approaches, including the use of ***finite state transducers*** to generate the words to be looked up, need to be considered. This research is aimed at finding solutions to the detection of non-word errors when dictionary lookup is inadequate.

Isolated word error correction

First generation spell checkers typically aim to achieve non-word error detection. A second level of maturity within the development of spell checkers is that of the ***isolated word error correction.*** This is an extension of the initial sub-problem. The goal of this area of research is to develop spell checkers that are able to correct non-word errors without considering the context within which they occur. Once a non-word has been identified by the previous step, a corrector attempts to generate correct words from the incorrect input before ranking and presenting them to the user as candidate corrections for the misspelt word.

Context-dependent word correction

The highest level of spell-checking functionality and maturity is that of ***context-dependent word correction***. This considers the word and the context within which it occurs. Some authorities like (Paggio, 2000) posit that context-dependent spell checking is synonymous with ***grammar checking,*** contrasting with the view espoused by (Kukich, 1992).

Out of Vocabulary Words (OoV)

Words are Out of Vocabulary for an NLP system if they were never encountered during training and are not available within its lexicon. An NLP system has an OoV problem if it cannot generate any of the OoV words when required to do so. The aim is therefore to minimise the incidence of such words.

Alternatively, the aim is to develop methods that can generate the OoV terms without having been exposed to them.

2.5.2. Evaluation Metrics for Spell Checkers

The performance of a spell checker is measured using the set of standard metrics that is used in most machine learning two class, classification problems. These metrics are defined below with specific reference to the spell-checking problem.

True Positives

A word is a *true positive* (TP) if it is correctly spelt and the spell checker correctly identifies it as such. The goal of a good spell checker is to maximise such true positives.

True Negatives

True negatives (TN) are those words that are incorrectly spelt which the spell checker correctly flags as being incorrect. An ideal spell checker ensures that the value of these is also maximised.

False Positives

False positives (FP) occur when a word that is incorrect is marked as being correct by a spell checker. In other words, this is an error in which the spell checker fails to identify an incorrectly spelt word as such. Spell checkers usually aim to minimise the incidence of false positives.

False Negatives

When a spell checker marks a correct word as being incorrect, this is referred to as a *false negative* (FN). A high incidence of false negatives is undesirable; thus, the goal of spell checker designers is to minimise their incidence as well.

Lexical Recall or Sensitivity

The lexical recall (R_l) or sensitivity of a spell checker refers to its ability to correctly identify all the correct words within a document. The formula for R is to divide the true positives by the sum of the true positives and false negatives. It is usually expressed in percentage terms. The resultant percentage

indicates the proportion of correct words that the spell checker is able to correctly identify as such. The formula for lexical recall is shown in Equation 2.5-1-Calculation of Lexical recall.

$$R_l = \frac{TP}{TP + FN}$$

Equation 2.5-1-Calculation of Lexical recall

Error Recall or specificity

Error recall (R_e) or *specificity* measures the ability of a spell checker to flag incorrectly spelt words. The calculation is shown in Equation 2.5-2- Calculation of Error Recall. It is the quotient of the true negatives by the sum of the true negatives and the false positives expressed as a percentage.

$$R_e = \frac{TN}{TN + FP}$$

Equation 2.5-2- Calculation of Error Recall

Precision

Precision (P) measures the extent to which the words that a spell checker marks as correct are actually correct. It is calculated by dividing the number of true positives by the sum of the true and false positives as shown in Equation 2.5-3.

$$P = \frac{TP}{TP + FP}$$

Equation 2.5-3- Calculation of Precision

Negative Predicted Value

The *negative predicted value* (P_n) is an analogue to precision, but for the incorrectly spelt words. It measures the extent to which the words that the spell checker marks as incorrect are actually incorrect. A high negative predicted value means that the spell checker can be relied upon to flag incorrectly spelt words. The calculation for this metric is given in Equation 2.5-4.

$$P_n = \frac{TN}{FN + TN}$$

Equation 2.5-4- Calculation for Negative Predicted Value

Accuracy

The proportion of words that are correctly flagged as either correct or incorrect as a percentage of all the words evaluated is referred to as the *Accuracy* (A) of the spell checker. The calculation for this metric is given in Equation 2.5-5

$$A = \frac{TP + TN}{(TP + FN) + (FP + TN)}$$

Equation 2.5-5- Calculation of Accuracy

F-Measure or F-Score

The *F-measure* or *F-Score* (F_1) is defined as the harmonic mean of the precision and recall as given in Equation 2.5-6 (Powers, 2014). It provides one number that gives an indication of how well a spell checker performs across both of these. This score gives assumes that precision and recall have the same value and importance to the system being measured. If the importance of either recall or precision is viewed different, the alternative F_β score which is given in Equation 2.5.7 is used. The value of β gives greater weight to precision or to recall. Values greater than 1 give greater weight to precision whilst those lower than 1 give greater weight to precision.

$$F_1 = 2\frac{P \,.\, R_l}{P + R_l} = \frac{2TP}{2TP + (FP + FN)}$$

Equation 2.5-6- Calculation of F-Score

$$F_\beta = \frac{(1 + \beta^2).TP}{(1 + \beta^2).TP + \beta^2.FN + FP}$$

Equation 2.5-7- Calculation of F_β Score

Confusion Matrix

A confusion matrix is a 2 x 2 grid which tabulates the actual values against the predictions provided by the spell checker. The columns of the matrix represent the actual values while the rows represent the predicted. Table 2.5.1 is an example of a confusion matrix with a third column and third row added for the calculations that depend on the values in the respective row or column. The part of the table that is within the bold borders is the confusion matrix. The rest of the table shows how the values in the confusion matrix relate to the evaluation metrics described in the previous subsections.

	Words correctly spelt	**Words incorrectly spelt**	**Calculations**
Words Flagged as Correct	TP	FP	Precision $P = \frac{TP}{TP + FP}$
Words Flagged as Incorrect	FN	TN	Negative Predicted Value TN/(FN+TN)
Calculations	Recall/sensitivity $R_l = \frac{TP}{TP + FN}$	Specificity $R_e = \frac{TN}{TN + FP}$	Accuracy (TP+TN)/ (TP+FN) +(FP+TN)

Table 2.5.1- Confusion Matrix - Summarising methods to evaluate spell checkers

2.5.3. Language Models

In the absence of a comprehensive dictionary with all the words for a given language, the spell-checking task is transformed into a question of comparing a given word with the most representative model of the words in the target language. There are various ways in which the words of a given language can be modelled. Traditionally these are split into rules-based models on the one side and data-driven or statistical models on the other side. Rules based approaches include the development of finite state transducers based on the grammar of given languages. Statistical approaches can take the

form of using statistical language models. These have various levels of sophistication. Their key task is to deduce the probability that a given word is valid within a specific language.

The meaning of the term language model has morphed within the recent past. Traditionally, a language model is a computational representation of a given language. More formally, a language model "*refers to a mathematical function that utilises statistical analysis to estimate the probability of a given word within a specific context"*. Whilst the traditional approach in natural language processing (NLP) is to limit the definition of a language model to a data driven artefact, language models can be either rules based/knowledge based or they can be data driven, or as in the case of the one that will be introduced in this thesis, they can take a hybrid approach. There has been an increasing tendency to refer to the extremely complex transformer based natural language generation models as language models. (Bommasani, et al., 2021) have clarified this issue by coining the newer term, Fundamental Models, to refer to those language models that perform more than the task of providing the likelihood of a given string of characters or word occurring within a given context.

Rules Based/Knowledge Based Approaches

Computational models of language can be built using handwritten rules provided by experts. This approach is alternatively called the rules based or the knowledge based approach. It is called rules based because it relies on the codification of the specific rules that define a given language. The name Knowledge is attached to it because this presupposes the existence of a knowledge base which is then converted into a form that can be utilised by a computer to make predictions about a specific language. *Finite State Transducers* (FSTs) are the primary way of developing rules based models of language.

1. Finite State Transducer

Morphological Analysers are realised through Finite State Transducers. A finite state transducer is a special type of Finite State machine which, apart from just moving from one state to the next given a specific input, generates some output. In the case of Morphological Analysis, a Finite State Transducer can emit the part of speech tag of the sub-components of a given word as it goes through the valid transitions of the word.

Data Driven Approaches

In recent times, there has been a tendency to utilise more empirical and data driven approaches. What this means is that the model is built from learning the structure of a given language from large amounts of data of the language without being explicitly told what the structure of the language is.

Full word language models

The initial approach to modelling all languages was to utilise full words as the unit of modelling. The simplest form of a full word model is the n-gram model, where n is an integer typically greater than one. For instance, a bigram language model is one in which n is equal to two. It can be developed by enumerating the number of times that pairs of words co-occur within a corpus. These frequencies are then used to estimate the likelihood that the second word occurs after the first one. This process works well for languages with lower morphological complexity. Since agglutinative languages and other morphologically complex languages tend to have large numbers of words, full word language models do not do well for them.

Sub-word language models

Some language models of language utilise information about the morphological composition of a language to produce a digital version of the language. There are various types of sub-word language models. As with full word language models, the simplest type of sub-word language model is the n-gram. The simplest sub-word n-gram is the character n-gram. It breaks each word into consecutive chunks of n-characters each and uses these to estimate the probability of that subsequence being found within the words of a given language. More sophisticated sub-word language models use syllables, morphs, or morphemes as the modelling units.

2.6. Morphological Analysis

Morphological Analysis is the process of determining the constituent components of a given word. It is the process of recovering the lexical form of a word, given its surface form. For example, when given a word like "*nyamutambanemombe*" (s/he who plays with the cows), a morphological analyser produces the following analysis <*nya*> (possessive genitive prefix) <*mu*> (class 1 noun prefix) tamb (verb root – meaning to play) <*a*> (verb final vowel) <*ne*> (conjunctive affix meaning with) <*mombe*> (cow).

2.6.1. Finite State Automata

Conceptually a finite state automaton (FSA) is a machine which is made up of a number of states including a start state and at least one final or accepting state. Such a machine takes a finite string of symbols from a well-defined alphabet as input and as it processes the string, it moves from the start state to any of the internal states based on the input that it receives. If the machine ends up in a final or accepting state, it is said to have accepted the input string. The list of permissible transitions from one state to the other are described through a transition function.

Formally, a deterministic FSA M is specified by a 5 tuple as follows:

$M = (Q, \Sigma, q_0, \delta, F)$

Where Q is a finite state of states;

Σ is the set of permissible input characters called the input alphabet

$q_0 \in Q$ is the start state.

$F \subset Q$ is the set of final or accepting states

$\delta: Q \times \Sigma \rightarrow Q$ is the transition function which indicates the valid transitions from one state to another state for a given input. In a deterministic finite state automaton, there is only one valid transition from each state on any given input symbol.

Among their many uses, FSA are used to develop compilers for programming languages. Computational linguists have also deployed them to address spell checking as well as grammar checking. In the context of spell checking, an FSA can be designed to accept only those words that are valid in a given language. Each word is broken down into a set of morphemes which are then given to the FSA. The use of FSAs for spell checking will be further discussed in Chapter 5.

2.6.2. Finite State Transducers

Finite State Transducers (FSTs) are a class of finite state automata which have both an input tape and an output tape. An FST produces some output for each transition that it makes from one state to

another. As such they may be used to translate from the input language to the output language. Alternatively, they can be used to provide a label to the input strings. Within Computational linguistics, FSTs are used to perform morphological analysis, among other uses. When used in this way, the FST takes in words and outputs the labels for each of the morphemes of the given word. Within this thesis, modified versions of FSTs will be used to conduct spell checking.

2.6.3. Syllabification

A syllabifier is a software application that can be used to decompose words of a given language into its constituent syllables. The complexity of this task is dependent on the morphology of the specific language. This is because the rules that determine the identification of syllables differ across languages. The process of morphological analysis can be aided by the use of a syllabifier as some of the morphemes for a language may consist of syllables as is the case with some Shona verb and noun prefixes. Section 2.8.1 describes the nature of syllables for Shona.

2.7. Two level Formalism

The two-level formalism is a method for developing computational models that can recognise word forms in morphologically complex languages (Koskenniemi, 1984). It is built from two key parts: "*a lexicon system and two level rules*" (Koskenniemi, 1984). The two-level rules describe the relationships between surface and lexical forms of words. A detailed explanation of this formalism and how it applies to the spell checking task will be given in the methods chapter. However, in the present chapter, we consider the use of this formalism on the previously discussed example of the words *gudo/makudo* and *gava/makava*.

It was previously stated that *makava* is the surface form of the inexistent lexical form *magava* in the same way that *makudo* is the ungrammatical surface form of *magudo*. The two level formalism for a language would have the list of lexical words that are valid in the language - in this case, *gudo, gava, gororo (outlaw/robber)*, etc. It would then also have a set of rules that apply at both the lexical and surface levels in parallel. In this case, the rule is that Shona nouns in class 5 that begin with the consonant <*g*> when pluralised take on the noun class 6 prefix <*ma*>. Whenever the phoneme /*g*/, follows the noun class *ma*, it is changed to the phoneme /*k*/. Similarly, there is a rule that says if a word is in class 6 and has the noun prefix <*ma*> followed by the morpheme *k*-, then the singular form of that word is in class 5. This singular form of the word does not include the <*ma*> prefix and replaces the /*k*/ phoneme with the /*g*/ phoneme. It must be stated at this point that it is very difficult to identify a compact set of rules that can apply to all word forms for a given language. For instance, in

the case of Shona, the above rule falls flat for some surface words like *gamba* (hero) where the lexical (*magamba*) and surface forms (*magamba*) are identical and do not conform to the preceding rules which would imply that the plural would be the ungrammatical *makamba*.

The power of this formalism is that it can be used to derive words that do not exist within the lexicon using a small set of rules. This is particularly helpful for spell checking a language where it is not feasible to have a dictionary of all the words as is the case for the CWALs

2.8. Shona Grammar

Shona is a Bantu language. It is classified as a Southern Bantu Language by Doke and listed in its own category by (Janson, 1991-92) and Ethnologue (David M, Simons, & Fennig, 2021). Like all other Bantu languages, it is an agglutinative language which has a noun class system. This section presents an overview of the main components of Shona Grammar that have a bearing to the spell checking problem.

2.8.1. Shona Orthography

One can consider a spell checker to be a validator of the conformance of a written text to a given orthography. It is therefore important that the language's orthography be properly understood before the development of such a spell checker is attempted. The current orthography for Shona is based on a 1967 revision of the one that was proposed by Doke in 1931. This initial proposal went through the first revision in 1955 before the Shona language committee reviewed it in 1967. The main characteristic of this orthography is the use of the conjunctive writing system, the Roman alphabet, and a set of principles to guide word division (Doke, 2005). Whilst they were provided to help speaker-writers, these principles have been the source of much debate and confusion. As a result, the orthography is not without its detractors and there are at least two proposals to revise it including that of (Magwa, 2008). Discussion of these issues is outside the scope of this thesis. What is of interest to this study are the issues of the alphabet, the nature of syllables, as well as the rules pertaining to word division that are specified in this orthography. These are briefly discussed below.

The alphabet

Shona uses a subset of the Roman alphabet which excludes the letters l, q, and x. The full list of letters and digraphs that are considered to be part of Shona's alphabet are given in

Table 2.8.1.

Shona Alphabet
a, b, bh, ch, d, dh, e, f, g, h, i, j, k, m, mh, n, nh, ny, n', o, p, r, s, sh, sv, t, u, v, vh, w, y, z, zh, zv (Doke, 2005)

Table 2.8.1 – Shona Alphabet – Letters and Diagraphs used in Shona orthography

Syllables

(Mpofu, Ngunga, Mberi, & Matambirofa., 2013) quoting (Ducrot & Todorov, 1972) define a syllable as "a phonemic group constituted by a phoneme called syllabic and, optionally, by other non-syllabic phonemes. The first constitutes the peak of a syllable whereas the others form its margins". Shona has two types of syllables. The first one is what they term V-syllables. These are syllables composed of a single vowel like the *i* in *ishe*. The second type of vowels are referred to as CV-syllables. These are made up of a series of valid consonants followed by a vowel. A syllable with a vowel at the end is referred to as an open syllable. All Shona words are composed of open syllables - meaning that no valid words end up in a consonant. Formally, a Shona word can be viewed as being a string with the following form:

$w=V^{[0|1]}CV^{*}$

where $V^{[0|1]}$, indicates that a word can start with zero or 1 vowels and

CV^{*} means that a valid Shona word is composed of zero or more open syllables.

Equation 2.8-1 Syllabic formulation of a Shona word

The implication of this for a spell checker is that it is possible to deduce the correctness of a word by checking to see if any of the constituent syllables do not conform to this pattern.

This thesis uses the grammar described by (Mpofu, Ngunga, Mberi, & Matambirofa., 2013) as a reference for both the orthography and grammatical information used to develop the spell checker for Shona.

2.8.2. Noun Morphology

Apart from having the syllabic form described in the preceding section, each part of speech has a well-defined structure or morphology. This section considers the noun. It is one of the most important word types in Shona morphology as it determines the form of all other words within a given sentence through a property referred to as concordial agreement. Shona nouns, like those of other Bantu languages, have a relatively simple morphology. They are made up of two parts - a class prefix, which can be the null string, and a noun stem. Shona has 20 of the 23 numbered noun classes attested in Bantu languages. These are classes 1 to 19 and class 21. Words in each class can be identified by the class prefix. Some noun classes have the null string as the prefix. Table 2.8.2 is a schematic of the normal form of the Shona noun as well as some examples for each of the classes.

Class Number	**Class Prefix**	**Example Noun Stem**	**Example Noun**	**Gloss**
1	mu-	-komana	mukomana	boy
1a	∅	Baba	baba	father
2	va-	-komana	vakomana	boys
2a	va-	-tete	vatete	paternal aunt

Class Number	Class Prefix	Example Noun Stem	Example Noun	Gloss
2b	a-	-mai	amai	mother
3	mu-	-ti	muti	tree
4	Mi	-ti	miti	trees
5	Ø	Gava	gava	fox
6	ma-	-gava	makava	foxes
7	chi-	-ngwa	chingwa	bread (single)
8	zvi-	-ngwa	zvingwa	bread (plural)
9	Ø[i]	Mbavha	mbavha	thief

Class Number	Class Prefix	Example Noun Stem	Example Noun	Gloss
10	Ø[dzi]	Mbavha	mbavha	thieves
11	Ru	-kova	rukova	river
11	Ru	-oko	ruoko	hand
12	ka-	-mbuyu	kambuyu	insect
13	Tu	-mbuyu	tumbuyu	insects
14	u-	-chi	uchi	honey
15	ku-	-famba	kufamba	to walk
16	pa-	-seri	paseri	behind

Class Number	Class Prefix	Example Noun Stem	Example Noun	Gloss
17	(ku-)	-seri	kuseri	behind
17a	∅	Zvimba	Zvimba	Place name
18	mu-	-seri	museri	behind
19	svi-	-nhu	svinhu	small thing
21	zi-	-nhu	zinhu	huge thing

Table 2.8.2 - Shona Noun Morphology

2.8.3. Verb Morphology

The verb has the most complex morphology within Shona. A generic verb is composed of any number of optional prefixes followed by a stem and up to three optional suffixes, which include the final/terminal vowel, verb extensions and the clitics. (Mberi, 2006) proposed a thirteen slot system to describe the Shona verbs. This system is reproduced in Table 2.8.3 - Shona Verb Slot system - according to Mberi.

1	2	3	4	5	6	7	8	9	10	11	12	13
Neg/ Mood	**SC**	**TAM**	**NEG**	**TAM**	**NEG**	**TAM**	**Aux**	**OC**	**R**	**Ext**	**FV**	**Clitic and other**
i	SC	i	si	chi	Si	chi	ndo	zvi	R	an	a	e-yi
ha	sca	ch	sa	ka	Ka	ka	mbo			anur	e	e-yi
ha	sco	no		do	Za	zo	ngo			ek	(i)	e-pi
nga	u			ne			zo			enur		e-ka
	mu			nga			fum			er		
	ku						etc			ik		

Table 2.8.3 - Shona Verb Slot system - according to Mberi

KEY		
1. Negatives, Mood	2. Subject Concord	3. Tense-Aspect-Mood (TAM)
4. Negative (Neg)	5. TAM	6. Neg
7. TAM	8. Auxiliary	9. Object Concord
10. Verb Root	11. Extension(s)	12. Final Vowel
	13. Clitics	

The following examples illustrate how this verb system works in practice. The Shona verb enda means go. It is composed of the verb root -end- and the final vowel a. In this base form it instantiates slots 10 and 12.

Table 2.8.4 shows eight examples of how this verb can be transformed through the instantiation of the other verb slots. Example 7 amply demonstrates the complexity of the verb. In this example, seven of the thirteen slots are filled in. Here the word *ngatichimuendeserei* (let us now take it for him/her) has the hortative mood morpheme <*nga*> in slot 1. This is followed by the subject concord morpheme <-*ti*-> which indicates the first person plural. The next morpheme <-*chi*-> marks the tense and aspect of the verb. In this case it indicates the fact that this action will happen in the near future. The object of the verb is indicated by the next morpheme <-*mu*->. This specifies that the action of the verb will be on a human object. As already stated, <-*end*-> is the verb root and this is followed by two verb extension morphemes. The first one <-*es*-> is the causative verb extension whilst the <-*er*-> is the applicative extension.

It should be clear from the above that knowledge of this verb slot system can be used to determine the correctness of a given Shona verb. This will be further discussed in the methods chapter.

#	1	2	3	4	5	6	7	8	9	10	11	12	13	Full word	Gloss
Role	Neg/ Mood	SC	TAM	NEG	TAM	NEG	TAM	Aux	OC	R	Ext	FV	Clitic and other		
1										-end-		-a		enda	Go
2		a								-end-		-a		aenda	S/he has gone
3	ha-	-a-								-end-		-i		haaendi	S/he does not go
4		a-			-ka-			-zo-		-end-		-a		akazoenda	S/he finally went
5		a-	-no-							-end-	-es-	-a		anoendesa	S/he goes with (takes)
6		-nda-			-ka-				-chi-	-end-	-es- -er- -w-	-a-		ndakachiendeserwa	I had it sent for me
7	nga-	-ti-	-chi-						-mu-	-end-	-es- -er-		-ei/-a	Ngatichimuendeser/a?	Let us now send (the things) to him/her
8	ha-	-ku-	-cha-							-end-	-ek-	-i		hakuchaendeki	It is no longer feasible to go there

Table 2.8.4 Example verbs based on Mberi's Shona Verb Slot System

2.8.4. Other Parts of speech

In comparison to nouns and verbs, other parts of speech have relatively simpler morphology and are arguably fewer. This section provides a broad overview of each of these.

Adjectives

According to (Mpofu, Ngunga, Mberi, & Matambirofa., 2013), Shona adjectives fall into two categories: i) the first group are called variable adjectives; and ii) a second group of invariable ones. The variable adjectives have a similar structure to nouns in that they have a prefix and a stem. They take an adjectival prefix that agrees with the noun that they modify. Invariable adjectives do not have a class marker and do not have to agree with the noun that they modify.

Some adjectives can be modified by adding the suffix *-sa* to the adjectival stem in a process called intensification. Table 2.8.5 has some examples of adjectives that are intensified through suffixation by -sa.

Example	**Gloss**
mombe huru**sa**	a very big cow
murume mutema**sa**	a very dark man

Table 2.8.5 Examples of intensification by appending the suffix ***-sa***

Another way in which adjectives can be intensified is through a process called reduplication. This is when the adjectival stem is repeated as shown in Table 2.8.6.

Example	Gloss
munda murefu-refu	an extremely large field
nhasi izuva guru-guru	Today is a very important day (literally a big big day)

Table 2.8.6 - Examples of reduplication of adjectives

Possessives

In Shona, Possessives are a part of speech that modify the noun as part of the broader genitives. Their morphology is remarkably simple, and the set of possessives is easy to enumerate. These include words like *yangu* (mine class 4 noun), *rangu* (mine class 5 noun), *changu* (mine class 7 noun), *chako* (yours class 7 noun), *chedu* (ours class 7 noun).

Demonstratives

Like the possessives, the set of demonstratives in Shona has a small number of members. Words that fall into this category include *uyu* (this one - human subject), *iyo* (that one – class 4 subject), and *ichi* (this one - class 5 subject). However, unlike the possessives, and like the adjectives, demonstratives can be reduplicated in three different ways. These three ways are shown in Table 2.5.4.c.

Example	Gloss
mwana uya uya	that child
mwana uyuyu	this child here
mwana yuyuyu	this child here

Table 2.8.7 - Examples of reduplication of Demonstratives

Another way that demonstratives can be used to modify nouns is in the form of enclitics. This is when the first vowel of the demonstrative is deleted whilst the rest of the word is added to the noun that it

modifies as a suffix. For example, the noun *benzi* (mad person) could be modified by the demonstrative *iro* (that one - class 5) to form the word *benziro* (that mad person). Demonstratives also modify absolute pronouns to form demonstrative pronouns like <*iyeyu*> (this person here) which is the result of combining the pronoun <*iye*> (this person) and the demonstrative <*uyu*> (who is here).

Quantitatives

Quantitative serve the purpose of *qualifying nouns in relation to quantity* and have the structure <*prefix*> + <*stem*> (Mpofu, Ngunga, Mberi, & Matambirofa., 2013). The set of quantitative stems is composed of only three items *-ose/-ese* and *-ga*. The prefixes that they take depend on the noun that each of these quantitative qualifies.

Enumeratives

Shona nouns can also be qualified by the enumeratives. These are words that have the enumerative stem *-mwe*.

Genitivisation of Nouns

Nouns can be turned into adjectives by the addition of the genitive marker -e-, preceded by the relevant class agreement marker. For instance, the noun *mupurisa* (policeperson) can be transformed into the adjective *wemupurisa* ([human subject] of ["belonging" to] the policeperson) as in *mwana wemupurisa* (the policeman/woman's child). Similarly, the noun *hari* (clay pot) can also yield the adjective *yehari* (of the clay pot).

Infinitisation of Verbs

Infinitives can be formed by appending the prefix *ku-* preceded by the genitive marker *-e-* to verb roots. For example, the verb root *-penga-* (be mad) can become the infinitive *zvekupenga* ([things] of madness) by appending the class 8 prefix *zv-*, the genitive marker *-e-* and the prefix *-ku-* to it.

Pronouns

Pronouns can take one of five different forms within Shona. The first form is that of personal absolute pronouns (PAP). The set of PAPs is finite, having one member per noun class. However, these can also go through reduplication. The second set of pronouns are called personal reflective pronouns

(PRP). PRPs take the form of the affix *-zvi-* within verbs as in the word *ndazviruma* (I bit myself). Here the *-zvi-* indicates that the action happened to me. Demonstrative pronouns come in four flavours indicating how far the speaker is from the object being pointed out. They are also a closed set. The same applies to possessive pronouns.

The last set of pronouns is that of interrogatives. Whilst this set has a larger membership, only the following are of interest to this study as they influence orthography and the possible identification of previously unseen words:

1. the suffixation of *-ngani?* to noun prefixes. This set is arguably fixed.
2. the suffixation of *-i*? to noun prefixes.
3. suffixation of *-ei*?
4. suffocation of *-ni* or *-pi*

2.8.5. Summary of Shona Morphology

The preceding sections on Shona morphology have demonstrated the regularity in the words that make up the various parts of speech in Shona. This regularity helps resolve the paradox of having an inexhaustible inventory of words, without overwhelming the speaker-writers of the language. It is also what is used by human spellers to differentiate between valid and invalid spellings of words. This research will look into the effect of codifying this knowledge on the performance of spell checkers on OOV words.

2.9. Synthesis of concepts

Section 2.3 provided a broad definition of the spell-checking problem. In its simplest formulation it can be reduced to a case of string searching. Section 2.7, particularly subsection 2.7.3, showed that this is inadequate for Shona due to the way that words, especially verbs, are formed.

The correctness of a word in Shona and other CWSBLs can be determined in a few ways. The first way is to look it up in a dictionary. If it exists in the dictionary, then the word is valid. However, if it is not found in a dictionary it could still be a valid word. This is because if the word is composed of valid constituent units, the word can still be valid. Still, this can only be true if the units are properly

ordered according to the morphological rules of the language. Such rules can either be explicitly encoded into a spell checker or they can be empirically derived from language data.

To illustrate the validity of a word, consider the word "*[V]anyakutenga*" (noun - the [honorific] one who bought). Even though it is a valid word, there is no entry for it in DGS, and this is not an oversight on the part of its compilers. There is, however, an entry for the verb root "*-tenga*" (buy), from which it is derived. A human conducting a spell checking exercise can utilise their knowledge of the morphology of Shona to validate that "*[V]anyakutenga*" is a correct word. This would apply even if they had never encountered the verb "*tenga*" before.

Figure 2.9-1 generalises the fact that Shona, like other CWSBLs, is complex. First, it presents the generic form of a morphologically valid word in any conjunctively written agglutinative language. This generic form is then instantiated with examples of the Shona word "[*V]anyakutenga*" using three different approaches ordered by their proximity to what is most intuitive. The first example shows the character trigrams that make up the word. Character trigrams are a subclass of the sub-word language models discussed in section 2.5.3 and are the basis for the spell checker that (Ndaba, Suleman, Keet, & Khumalo, 2016) developed for isiZulu.

The second instantiated example is that of the syllabic decomposition of the word. Last, is the morphological decomposition of the word. Morphological decomposition is closest to the way that humans would check the validity of a given word.

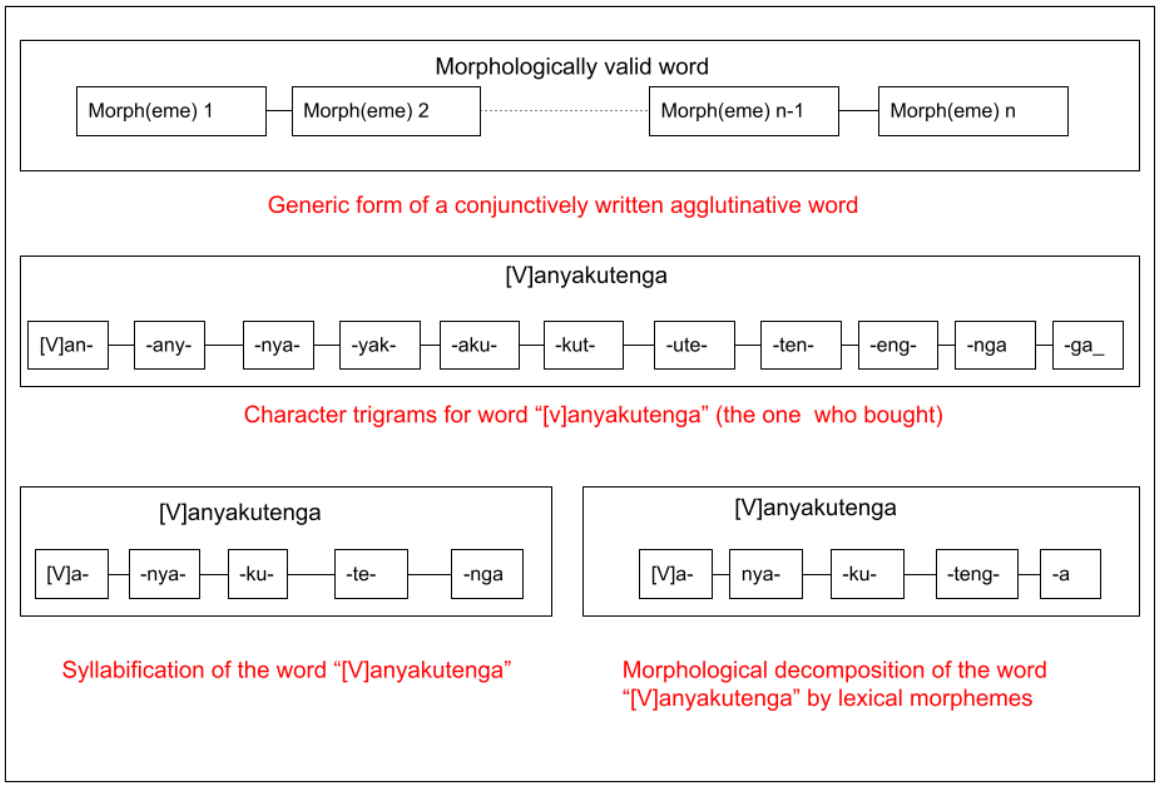


Figure 2.9-1 - Generic form of conjunctively written agglutinative words

2.10. Chapter Summary

This chapter introduced the terms and concepts that will be encountered in the rest of the thesis. It started by defining the linguistics terms and concepts before introducing the terminology utilised in the NLP of agglutinative languages. Finally, a synthesis of the concepts and how they relate to spell checking was provided.

"We don't have to waste our time learning how to make pastry when we can use grandma's recipes."

Orson De Wit.

Chapter 3 - Research Methodology

3.1. Introduction

Every research project is steeped in a particular research methodology. This chapter introduces both the theoretical underpinnings as well as the specific details of the methodology used in this thesis. In presenting this methodology, it begins with a brief overview of the methodologies available for use in computer science research. This introduction leads into a section which discusses the choice of the methodology used in this study. Subsequently, the research design is presented in detail. The method used to collect data, including the data sources and collection process, are presented next. Following this is a section on the measurements used to evaluate the data as well as a discussion on the reliability and validity of the research. The chapter closes with a summary of the methodology chosen for this thesis.

3.2. Research Methodologies for Computing Science

Each scientific discipline has its established methodologies for conducting research. Computing Science (CS) is not an exception to this. (Amaral, 2011) identifies five different methodologies used by researchers in CS. These are i) *Formal approach*, ii) *Experimental approach*, iii) *Build approach*, iv) *Process approach* and v) *Model approach*. Each of these methodologies is best suited to different research problems. Specifically, formal methodologies are used to *prove facts about algorithms and system*(s). Experimental methodologies can be used to *evaluate new solutions for problems*. The build research methodology involves the *construction of either a physical artifact or software system*. It is only considered to be research when it produces something that is completely new or that has novel features. The process methodology is suited for understanding the processes used to *understand a task in computing science*. The last methodology is o*ften used together with one of the previous four methodologies*. It is best suited in situations where researchers are working with *complex systems*. Abstract models are defined to simplify this complexity and enable the researchers to better understand it.

(Dodig-Crnkovic, 2002) propose a simpler taxonomy of CS research approaches. According to them, CS shares many characteristics with what they term the *classical sciences*. It is thus subject to investigation through the classical *scientific methods*. They further characterise research in CS as

falling into three broad categories which are i) *Theoretical Computer Science* - which is closely related to (Amaral, 2011)'s formal methodology, ii) *Experimental Computer Science* which maps to the experimental approach in the earlier theory and iii) *Computer simulation* which is linked to the modelling approach.

Other authors propose different methodologies for use within the CS domain. One such approach, whose usage transcends CS, is *Action Research* (AR). It is an iterative process to research which combines taking concrete actions, reflecting on the outcomes, and optionally iterating through the process as theory and practice are engaged (Bradbury, 2015). Another approach is *Design Science Research* (DSR) (Hevner, March, Park, & Ram, 2004; Pries-Heje, 2007) . With its widespread usage, (Peffers, Tuunanen, & Niehaves, 2018) argued that this is a valid approach which has various genres.

(Warfield, 2010) takes a more traditional approach to the discussion of research methodologies applicable to the Information Technology and Information Systems (IS/IT) domain. They place IS/IT research methodologies into three categories: *Quantitative Research, Qualitative Research* and *Mixed Methods Research*. Within these broad categories, they posit that Quantitative Research takes place within the *positivist paradigm* and can be further broken down into *experimental research*, *quasi-experimental*, *correlational*, and *descriptive* approaches. The positivist paradigm is one of two broad paradigms under which research is conducted. Within this paradigm the main assumptions are that there is an ontological difference between the researcher and the reality that they seek to study. The aim of the researcher is to objectively reveal this reality in a way that can be replicated by other researchers. This differs from interpretivism which posits that researchers cannot be separated from the reality that they aim to study.

Within positivism, the key difference between the experimental and the other quantitative approaches is the extent to which the researcher can infer causal relationship among phenomena. Uniting these methods is a five-step process which starts with the determination of the research questions. This first step is then followed with the determination of the study participants and then the selection of a method to answer the research questions. Statistical analysis tools to analyse the collected data are then selected before they are interpreted.

Qualitative research takes place within the interpretivist paradigm. Five general designs for qualitative design are identified. These are i) *narrative*, ii) *phenomenology*, iii) *grounded theory*, iv) *ethnography* and v) *case study*. Each of these broad approaches is best suited to distinct research question types. Narrative research is for when the *study has specific contextual focus.* Phenomenology is used when

the study *is about the lived experience of a specific concept or phenomenon.* Grounded theory is used to generate or discover theory. Ethnographic research is recommended when *the study is about an entire cultural group* whilst case studies are for c*ontained group within a specific setting or contex*t.

Mixed methods borrow from both Quantitative and Qualitative approaches (Damian, et al., 2020). The use of these methods can be justified by the need to e*nsure participant enrichment, instrument fidelity, treatment integrity, and significance enhancement.* (Warfield, 2010; Leech & Onwuegbuzie, 2010)

DSR is an ideal method for use in researching a problem that can be investigated by designing an artefact to solve it. The practical problem that this study seeks to solve is that of the non-existence of spell checkers for Shona. Conceptually, it also aims to address the problem of the effectiveness of current methods to address the challenges posed by out of vocabulary words. The conceptual problem can be solved through the development of a theoretical model which can be validated via formal means. However, the solution to the practical problem can only be demonstrated through the development of a software application. It is for this reason that the DSR approach to CS research has been chosen as the methodology for this study.

3.2.1. Types of DSR

Several genres of DSR exist. (Pries-Heje, 2007) identify six such genres. These are discussed in the following subsections.

System development Research Methodology

System Development Research Methodology (SDRM) which was proposed by (Nunamaker Jr, Chen, & Purdin, 1990) follows a 5-step process. These are i) Construct a conceptual Framework, ii) Develop a System architecture, iii) Analyse and design the system, iv) Build the (Prototype) system, and v) Observe and Evaluate the system.

DSR Process Model (DSRPM)

DSR Process Model (DSRPM) is a version of DSR that was proposed by (Vaishnavi V. K., 2007; Vaishnavi & Kuechler, 2015). It also consists of 5 steps. The first of these is awareness of the problem. Next comes suggestion. This is followed by development and then evaluation. The process is closed out with a conclusion step.

Design Science Research Methodology (DSRM)

Another genre of DSR is Design Science Research Methodology (DSRM), a six-step process proposed by (Peffers K. T., 2007). The first of these is to identify a problem and motivate. This is then followed by defining objectives and a solution. After this is the design and development step. Following design and development comes demonstration. The next step is evaluation. Communication of the results is the final stage of this process.

Action Design Research (ADR)

Action Design Research (ADR) borrows concepts from both AR and DSR in general. Formulated by (Sein, Henfridsson, Purao, Rossi, & Lindgren, 2011) it is a four-step process. It starts with a problem formulation which is then followed by building, intervention, and evaluation. This step is followed by reflection and learning. The process closes out with the formalisation of the learning step.

Soft Design Science Methodology (SDSM)

Soft Design Science Methodology (SDSM) was conceptualised by (Pries-Heje, 2007). It follows an eight-step process which starts with learning about the specific problem. This is followed by a number of design thinking steps. First there is a step to inspire and create the general problem and general requirements. After this it is intuit to and abduce the general solution. Then follows Ex Ante Evaluation and then Designing a specific solution for the specific problem. Following the Design thinking steps comes the Ex-Ante evaluation of the specific solution, construction, and final Ex Post evaluation.

Participatory Action Design Research Approach (PADR)

The last genre of DSR is the Participatory Action Design Research Approach (PADR). Proposed by (Bilandzic & Venable, 2011) it is more suited to the development of solutions for multi-stakeholder projects. It is a five-step process which starts with Diagnosing and formulating the problem. Next comes action planning. Then comes design which is conceptualised as action taking. After this comes impact evaluation. Finally, the process closes with reflection and evaluation.

3.3. Methodology Selected

This study primarily uses the DSRM methodology. This is because it is suited to the development of an artifact that can be used to answer the research questions posed in this study. Specifically, DSRM is well suited to address research objectives 3 to 8 as these require the development and evaluation of an artefact to demonstrate that the problem has been solved. The first two research objectives can be addressed through a literature review. The following section will also show that these objectives can also be covered during the first step of the DSRM process.

3.4. DSRM

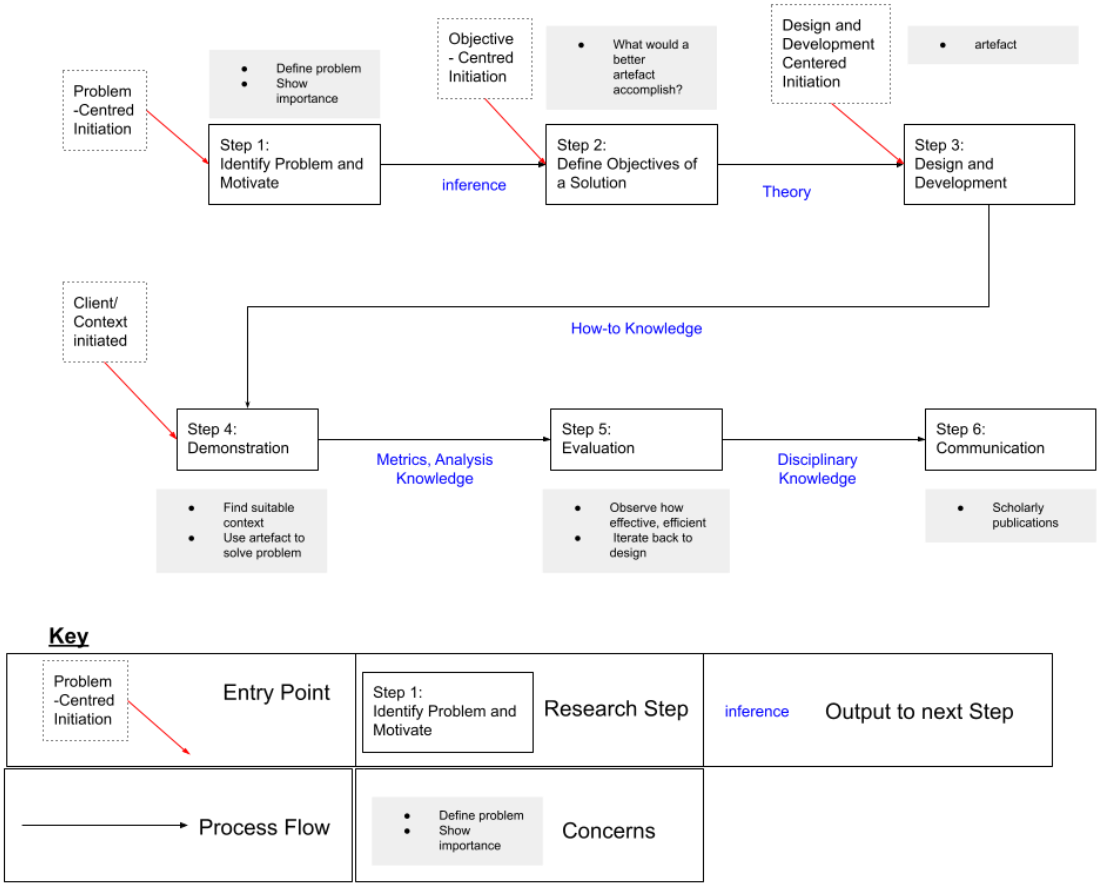


Table 3.4.1The Design Science Methodology Research (DSRM)

Table 3.4.1 shows the process flow proposed by (Peffers K. T., 2007). A key aspect of this process is that it consists of six different activities. These activities are shown occurring sequentially, but they can be performed iteratively. The following paragraph describes each of these steps in detail.

3.5. Step 1: Problem Identification and motivation

In this first step the specific research problem must be clearly defined. Justification also needs to be provided for the value of its solution. During this step, the researcher can also explicitly transform the problem into system objectives or meta requirements. As Table 3.4.1 shows, the entry point into this first step is problem centred.

Within this study, the identification and motivation of the problem was conducted through the use of a meta-narrative review of the literature on spell checking for CWSBLs. The following sections present the details of this meta-narrative review.

3.5.1. Introduction to meta-narrative review

As with most research traditions, the DSRM Guidelines emphasise a deep understanding of the problem as the entry point to any research endeavour. This subsection presents the first step in the DSRM process for this project. This step consists of the identification of the problem and the motivation for the research. Within this thesis, this step is conducted using a literature review of on spell checkers for the CWSBLs with a particular emphasis on the approaches taken to OOV words. The review was conducted using the RAMESES (**R**ealist **A**nd **ME**ta-narrative **E**vidence **S**yntheses: **E**volving **S**tandards) guidelines for meta-narrative reviews (Wong, Greenhalgh, Westhorp, Buckingham, & Pawson, 2013). As a result, the subsequent sections that describe the review follow the recommended format for a meta-narrative review as per these guidelines. The remainder of this introductory subsection presents the rationale for conducting the review. This is followed by a subsection which gives the objectives and the specific focus for this review.

The next sections elaborate on the meta-narrative review process as follows: First, there is a methods subsection which starts by highlighting the changes to the review process from what was initially envisaged to what was finally accomplished is presented. This is followed by a justification for using the meta-narrative review format and a discussion of the evidence that this specific review meets the guidelines for these kinds of reviews. Subsequently the processes used to scope, search and select are each presented in a separate subsection. A subsection on the data extraction leads to one on the analysis and synthesis of the data thus closing out the methods section. After presenting the methods, the next key section is that of the results. This encompasses the flow diagram for the process, the characteristics of the documents that were encountered as well as the main findings of the review. The last major section discusses the findings, first summarising them and then looking at the strengths and weaknesses of the previous studies, comparing them with previous literature before making some final conclusions and recommendations.

Rationale for reviewing spell checking of CWSBLS

The utility of spell checkers is not in dispute. That the majority of the word processing software applications include spell checking demonstrates their ubiquity (Yunus & Masum, 2020). However, this ubiquity is deceiving. Spell checkers are not available for many languages. There are several explanations for this disparity. Among these are the fact that NLP methods and rules developed for the well-resourced languages do not always have broad application to other languages (Gerz, Vulić, Ponti, Reichart, & Korhonen, 2018). It has also been recently shown that *morphology matters* when developing multilingual models (Park, et al., 2021). Unfortunately, most model developers do not appear to take this into account.

Development of spell checkers for under-represented languages is an ongoing area of research (Gezmu, Nürnberger, & Seyoum, 2018; Ahmadi, 2021; Himoro, 2020). Despite this, there is a dearth of prior work on Shona. Shona has neither a widely available nor functional spell checker. Where attempts have been made by large technology companies like Google and Microsoft on other related technology like machine translation, the results have been less than stellar[1][2]. It is known that development of an accurate spell checker for the CWSBL is not a trivial endeavour. This is because simple approaches do not produce satisfactory results.

The challenges associated with the development of spell checkers for CWALs in general and CWSBLs in particular was briefly touched on in the morphological typology section of the previous chapter. The next few paragraphs provide additional detail with some examples. As discussed there, it emanates from the languages' ability to recursively create new words from relatively small inventories of roots/stems and affixes. This means that any spell checker relying on the standard approach of a finite dictionary will fall short as it is bound to encounter words used in real world settings that do not exist in the lexicon. This finding was previously established experimentally (Prinsloo & Schryver., 2004).

To illustrate the issue with an example, consider the Shona verb stem ***-simba*** (be strong/powerful). It can be inflected to form new words like ***akasimba*** (s/he is strong), ***achasimba*** (s/he will become strong), ***ndamusimbisa*** (literally - I have made him strong - colloquially and figuratively, I have encouraged him), and ***akazosimbiswa*** (literally, s/he was made strong, colloquially, and figuratively he was encouraged). On seeing the above pattern, a human could easily reason that

[1]See https://www.dutchtrans.co.uk/how-accurate-is-google-translate-for-shona/ (extracted on 3 August 2021)
[2]See also https://www.techzim.co.zw/2020/06/why-is-google-translate-so-bad-at-translating-shona-ndebele (extracted on 3 August 2021) for a discussion on the problem of translation.

akazobatsirwa (s/he was helped) is a valid word without prior exposure to it. This would be truer if they would have been exposed to the words ***batsira*** (help), ***achabatsira*** (s/he will help) and ***akabatsira*** (s/he helped) in addition to at least one other analogous extension of ***simba***. Such a language learner could also conceivably handle an even more complex inflection like ***paakazobatsirwa*** (where/when s/he was eventually helped) which is also a valid word. Experimentation has resolved that none of the preceding can be assumed with many of the current approaches used to develop spell checkers and other NLP tools for SBLs. This point is illustrated using the following mini experiment performed on Google Translate.

First, two Shona verbs with very similar constructions are presented to Google Translate. The first word is ***famba*** (walk). The second one is ***ona*** (see). Next, each of these verbs are inflected into their continuous present tense forms ***anofamba*** (s/he walks) and ***anoona*** (s/he sees) respectively. After this, the negated form of this present continuous tense forms, ***haafambe*** (s/he does not walk) and ***haaone*** (s/he does not see), are presented to Google translate. The results of this experiment are presented in the Table 3.5.1. Screenshots from Google Translate showing the actual results of this experiment can be found in the Appendix - Appendix 2 – Results of Mini Experiment on limitations of Google Translate.

Shona	**Gloss**	**Google Translate – English translation**
Famba	Walk	Walk
Ona	See	She
Anofamba	s/he walks	He walks
Anoona	s/he sees	He sees

Shona	**Gloss**	**Google Translate – English translation**
Haafambe	s/he does not walk	He does not walk
Haaone	S/he does not see	haaone

Table 3.5.1 - Results of the mini experiment demonstrating Google Translate's limitations with utilising the morphology of Shona to inform its translations

It is clear from the above that Google Translate's translation engine is not "learning" the inter-relationships between the morphemes that make up the various Shona words. Whilst translation and spell checking are distinct problems, they are actually related in that both need to be able to distinguish between valid and invalid words in the target and source languages.

Outside of the development of spell checkers, lexicographers working on dictionaries for CWAL such as the Shona *Duramazwi Guru ReChiShona* (DGS) have always been aware of the impossibility of compiling a list of all possible word forms. As a result, in the case of DGC, they limited the headword entries in the dictionary to only show non-inflected versions of most of the complex words, like verbs and some nouns. However, in the front matter of the dictionary, they explain how the language works, how headwords are selected and defined, and how the entries are constructed for the different word classes. In the dictionary proper, they leave it to the user to find the correct *lemma* for an unknown word, and to figure out how to extend a specific root form for specific contexts (Chimhundu, 2001; Mpofu N. .., 2007) The key take-away from this observation is that even human spell checkers do not have access to a dictionary of all the words in the language. They have to resort to a different heuristic in order to determine the validity of a new word when they encounter it.

Within the realm of computational solutions, since full word list solutions are inadequate to address the needs of spell checkers for CWSBLs, alternative solutions must be devised. To this end, research on such alternative methods has been conducted, leading to the development of spell checkers for isiZulu and other South African SBLs which rely on other modalities. These approaches use knowledge of the fact that words in these languages can be decomposed into smaller units and that these smaller units combine in well-defined ways. Broadly, three strands of attacking the problem

have arisen, and these are: i) augmenting the dictionaries used in the spell checkers by synthetically generating additional words through an understanding of these rules as in (de Schryver & Prinsloo, 2004); ii) using handwritten morphological rules to dynamically evaluate the correctness of words using regular expression based morphological analysers as per (Bosch & Eiselen, 2005) and iii) utilising sub-word (character) statistical language models (n-grams) to check the correctness of given words as in the work of (Ndaba, Suleman, Keet, & Khumalo, 2016; Mjaria & Keet, 2018).

The three approaches mirror the key development in natural language processing (NLP) which has been moving from more human input intensive rules and knowledge-based approaches to less human input intensive data driven approaches. There are clear trade-offs between the two extremes of this scale (Vincent, 2019). A key challenge with approaches that require substantial amounts of human input is that it takes longer to produce spell checkers using this method. The methods are also not easily generalisable or transferable to other languages, even when they are related. On the other end of the spectrum, the most advanced approaches utilise the extremely data and computational power hungry recently named class of ***foundational models*** (Bommasani, et al., 2021)**.** These do not easily lend themselves for use by resource poor researchers working on less resourced languages (Chau & Smith, 2021; Goetze & Abramson, 2021). One way that they can be used for these languages is when the approach is to fine-tuning pre-existing models which have been developed elsewhere (Doddapaneni, Ramesh, Kunchukuttan, Kumar, & Khapra, 2021)

The question that arises is whether any of the methods that are available for CWSBLs can robustly handle unknown words. It can be inferred that a reasonable goal for a decent quality spell checker for Shona and other CWSBLs is that it would be able to correctly handle most OOV words, especially those for which related roots/stems are already recognised. Ideally this should be enabled without the need for explicit training, or at best with minimal human supervision/input. It is not clear that existing methods meet this requirement. This is despite the fact that one of the stated goals of each of these approaches has been to improve *performance* on the spell-checking task. The link between the methods of performance evaluation and the effectiveness of the OoV problem is not well established.

In this research study we aim to evaluate the extent to which the approaches that have been previously developed address the OoV problem. Specifically, we seek to find out the methods that have been used to determine if such spell checkers can cope with such previously unseen words and how they have been optimised to deliver improved performance on these words. A spell checker that can correctly identify OoV words has a higher likelihood of real-world user acceptance than one which is limited to the lexicon that it was trained on.

Objectives and focus of review

This review aimed to evaluate the extent to which the developers of spell checkers for CWALs in general but more specifically the CWSBLs have addressed the need for their spell checkers to be able to correctly handle OOV words. The overarching objective for this review, linked to those of this research, were as follows:

RO1. Determine the challenges encountered in the development of spell checkers that aim to maximise the correct identification of OOV words for SBLs.

RO2. Determine the previous approaches utilised in the development of spell checkers that aim to maximise the correct identification of OOV words for SBLs.

The above objective were met by answering the following research questions:

RQ1. What are the challenges with the previous approaches used to maximise the correct identification of OOV words for SBLs?

RQ2. What are the approaches that have been utilised to develop spell checkers that aim to maximise the correct identification of OOV words in SBLs?

The following search questions were used to address the above research questions, for each paper that was reviewed

1. Which approaches did the paper utilise for spell checking?
2. Did the paper address the question of previously unseen/OOV words?
3. How did the approach perform on OOV words?
4. What were the key challenges faced with these approaches with respect to previously unseen words?
5. What methods were developed to mitigate these challenges?

3.5.2. Method

Changes in the review process

The RAMESES guidelines require that any changes from the originally envisaged process be reported on upfront. There were no material changes to the approach that was used to conduct this review from

the one that was initially planned and therefore none can be reported here.

Rationale for using meta-narrative review

" meta-narrative review is a way of systematically reviewing literature, that was developed to probe subject areas that have been studies by diverse researchers from different angles. (Wong, Greenhalgh, Westhorp, Buckingham, & Pawson, 2013). It is a qualitative method which aims at identifying the main ideas in the different research communities linked to the study topic. With specific reference to the question of spell checking for CWALs, various researchers have looked at the spell-checking problem. The key focus of these researchers has been on improving the reported accuracy, recall and precision of these spell checkers. There has been progress towards the achievement of these goals in laboratory conditions. However, the question of how well the spell checkers work with OOV does not appear to have been answered. Since the researchers have approached this problem from diverse perspectives, the meta-narrative review approach is considered suitable for such a review.

Evidence for adherence to guiding principles

The RAMESES guidelines for conducting meta-narrative reviews strongly recommend that they follow six guiding key principles. These principles are as follows

1) The first principle is that of *pragmatism*. This principle holds the studies that need to be included are *not self-evident* and as such reviewers need to apply judgement in selecting those sources that will be *most useful to the intended audience.* This principle guided the search and selection process for this thesis. For example, whilst conducting this review it became clear that the literature on spell checking for the SBLs is limited. Research in the areas has been conducted by a small group of researchers. Care has been taken to widen the scope of the search to learn from other similar language groups that have similarities to the CWSBLs as will be demonstrated later in this chapter.
2) *Pluralism* is the key principle which holds that a topic must be explored from *"multiple angles and perspectives, using the established criteria appropriate to each*". It is required that when performing a review of a given study the reviewers should apply the same paradigm to judge its findings. The guidelines caution against the use of methods developed for other research to evaluate other different research traditions. In this study, we evaluate each study based on its own assumptions and approaches to the spell-checking problem.

3) The principle of *historicity* holds that it is best to present each tradition in chronological fashion. Whilst this is not strictly followed in this review, the results section does treat each thematic tradition according to this principle.
4) *Contestation* as a principle is also applied by ensuring that ideas that do not agree are presented from opposing research traditions. This is one of the tenets of this specific principle.
5) It is required by the principle of *reflexivity* that the reviewers should reflect on the emerging findings of the review as they work on it. Whilst this review was carried out by only one reviewer, care was also taken to reflect on the themes that emerged as the work proceeded.
6) Finally, the principle of peer review holds that the emerging findings should be presented to an external audience for "*further guidance and analysis*". This thesis is one way in which the findings of this review will be disseminated to a wider audience.

Scoping the literature

A high-level scoping and research territory mapping was deductively conducted. This started with a broad search of the literature on spell checking to identify the key research strands around their development for all languages. This was then narrowed to a search of the approaches on agglutinative languages. Based on initial results for the South African Bantu languages which identified the distinction between conjunctively written and disjunctively written languages, the review was further narrowed to just the conjunctively written languages. Through application of this method, a focused subset of the literature that dealt exclusively with the challenges of developing spell checkers for conjunctively written agglutinative languages was acquired. Through this search it also became clear that there was also a need to review work on the development of related concepts even in studies in which they were not utilised in spell checkers. This process culminated in the development of the following research territory map.

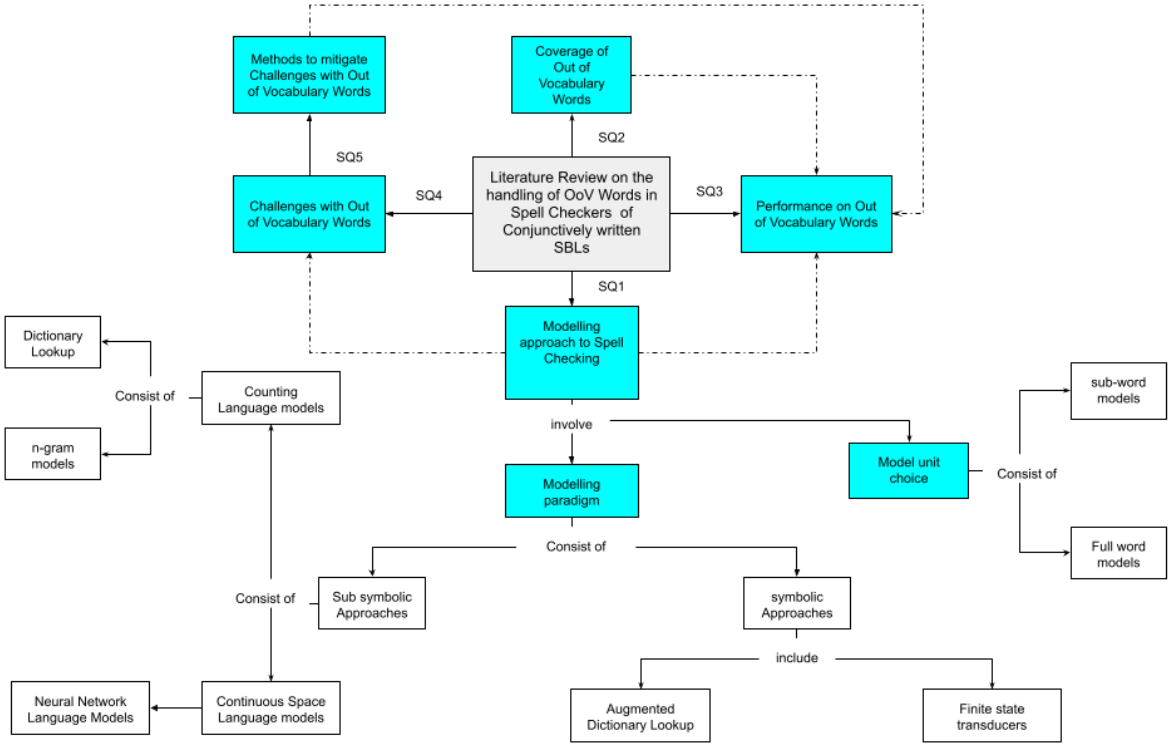


Figure 3.5-1Research territory map

Searching processes

Using the insights gleaned from the scoping exercise, the following search strategy was developed. All the searches were performed using JabRef's web search functionality on the following sites via the JabRef interface: Google Scholar, the Association of Computing Machinery (ACM)'s ACM Portal, the Collection of Computer Science Bibliographies, SearchAll and the Institute of Electrical and Electronics Engineers (IEEE)'s IEEExplore websites. Further searches were manually done on the following websites and then added to JabRef: Google Scholar, the Association of Computational Linguistics (ACL),

The key search was for literature on spell checkers for CWSBLs. Additional searches were then performed on the non-spell-checking literature for each of the methods utilised to improve their performance on OoV words on other agglutinative languages.

Table 3.5.2 summarises the purpose and the details of each of the search terms that were used to conduct the search process.

Purpose of Search	Search Terms used
Identify Studies with Spell checking for CWSBLs	"Spell Check" +isiZulu Or "Spell Check" +isiXhosa Or 'Spell Check" +isiNdebele Or "Spell Check" + siSwati
Identify modelling paradigms used on CWSBLs	"Language Model" +[isiZulu/isiNdebele/siSwati] Or "Neural Language Model" +[isiZulu/isiNdebele/siSwati]
Identify Studies dealing with OoV words for agglutinative languages	"Out of vocabulary words" + "Language model" + "agglutinative language"

Table 3.5.2 - Search terms used to search for literature

Selection and appraisal of documents

The references to all the documents that met the search terms were downloaded into JabRef. A review of the abstracts of each of the found papers was then conducted. Only papers that explicitly dealt with spell checking of CWSBLs were marked for inclusion in the list for final review, whilst the remaining papers were excluded.

After the filtering process was concluded, the remaining papers then underwent a detailed review. First, the list of papers was exported to Excel. Then the resulting Excel spreadsheet was amended to include an additional set of columns, one for each of the key questions that each paper was subjected to.

Data Extraction

For each paper that met the required criteria, the following details were extracted:

1. The bibliographic details of the paper
2. The language that the paper addressed
3. Which modelling paradigm the paper utilised
4. Whether it explicitly addressed the question of OoV words
5. What approach it took to optimise the performance of the spell checker
6. How the spell checker was measured - which metrics were utilised
7. How the spell checker performed against those metrics
8. A thematic summary of the paper.

Analysis and synthesis processes

Emerging out of the data extraction, a timeline of the research and the development of spell checkers for CWSBLs as well as the key shifts in the approaches used to implement them were drawn up. This helped conceptualize the historical development of the field as well as foreground the main strands of thought that have permeated the field from the onset. These are further described in the next section.

3.5.3. Results

Document flow diagram

Figure 3.5-2 provides a visual summary of the process that was used to conduct this review.

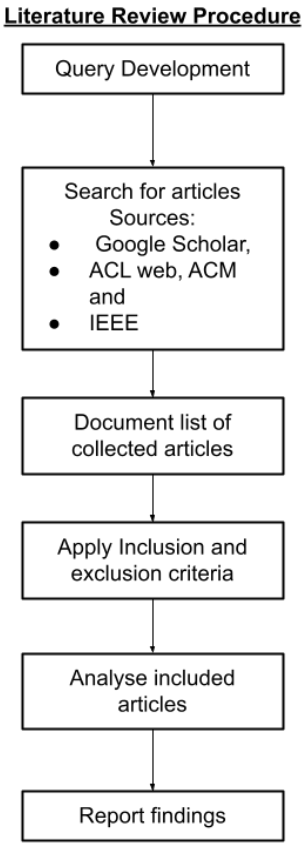


Figure 3.5-2 -Flow diagram for the meta-narrative review process

Document characteristics

Figure 3.5-3 is a network analysis of the relationships between the various researchers who have conducted research on CWSBLs. This is conducted on two axes. As the legend shows, the colour indicates the years that the researchers were active, whilst the links show the strength of collaboration among different authors. The earliest paper was written in 2003 whilst the youngest paper was published in 2018. Four of the papers were written by the duo of Prinsloo and de Schryver. The next most prolific author is Keet who has collaborated with Khumalo, Ndaba and Mjaria on different occasions of the papers

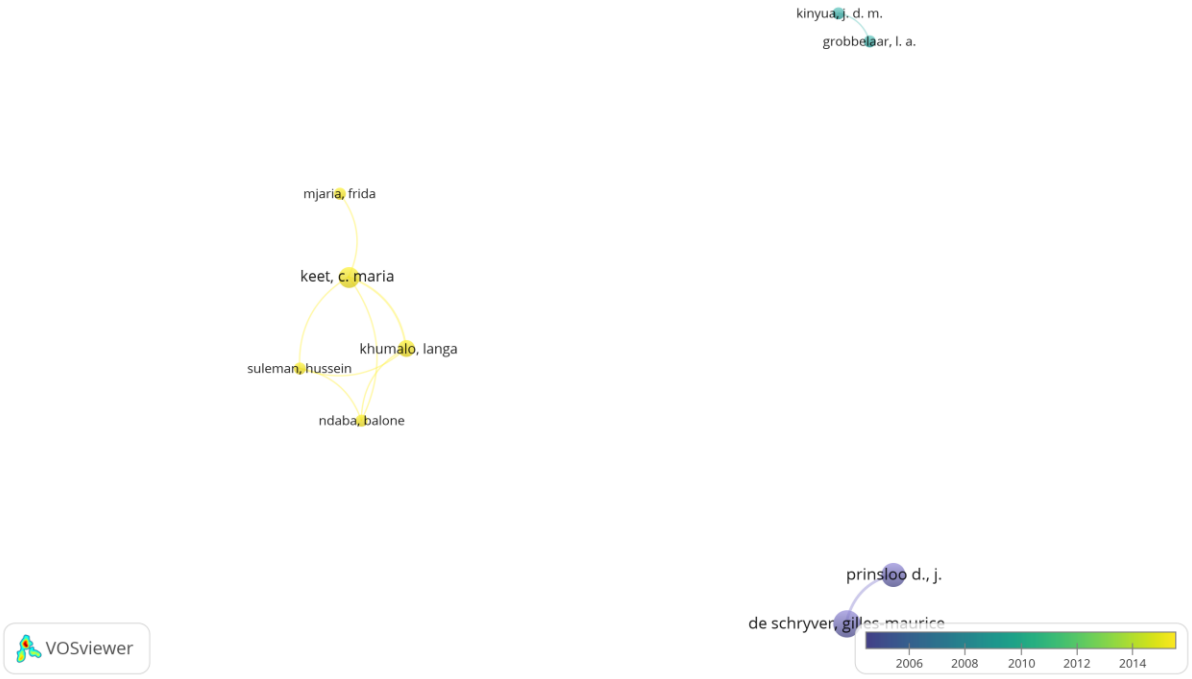


Figure 3.5-3 Network analysis of the key researchers on spell checking for CWSBLs

Main findings

This review sought to answer two very specific questions. The first one concerns the identification of the challenges and approaches that researchers working on spell checkers for SBLs have encountered. First, it is clear that the researchers identified the problem with pure dictionary lookup approaches right from the beginning of the development of spell checkers for SBLs. The first approach that they utilised to solve this problem was that of augmenting the available dictionaries by synthetically generating more words (Prinsloo & Schryver., 2004; Prinsloo & Eiselen, 2005). Two issues arise with the synthetic generation of words. First, the spell checker is still limited to a finite list of words from which to search. Second, there is a danger of what the authors term over-generation. This means that some words that are synthetically generated are invalid. To remedy this later issue, they employed linguists to filter the list of synthetically generated words leaving only valid isiZulu words.

The next approach that was used to address the problem of OOV words was the use of morphological rules (Bosch & Eiselen, 2005). Using regular expressions to model the morphological rules of isiZulu, a morphological analysis-based spell checker of isiZulu was developed. This approach was shown to perform much better than increasing the words in the lexicon. However, it also suffered from the problem of over-generation.

Data driven approaches to the spell checking of SBLs were first reported in 2016 by (Ndaba, Suleman, Keet, & Khumalo, 2016). Departing from the rules based approached that had been previously used, they instead used a character trigram-based language model to detect incorrectly spelt words in isiZulu. Whilst their main concern was to understand the impact that the corpus had on the effectiveness of such a spell checker, they found that their spell checker performed comparably well with the rules-based approaches previously mentioned. A subsequent study by (Keet & Khumalo, 2017) which aimed at evaluating the impact of this spell checker on the intellectualisation of isiZulu, found that it had some problems with some OoV words.

All the spell checkers attested in the literature have only concentrated on the non-word error detection problem. None of them have moved on to the real word error detection problem. However, work on non-word error correction has also begun.

Figure 3.5-4 illustrates the key paradigms that have been applied to the spell-checking problem for the CWSBLs as well as a few other CWALs. It also gives an indicative view of the amount of human effort required to develop each of these methods versus the amount of data and computation required to enable each approach.

Apart from the reviews on CWSBLs, additional reviews were conducted on spell checkers that have been developed for other CWALs. A similar pattern emerged in that the vast majority of the literature on spell checking for CWALs utilises rules based approaches to address the challenges brought by their morphologies. Only one of the studies that were surveyed for these non CWSBL CWALs utilised a language model approach.

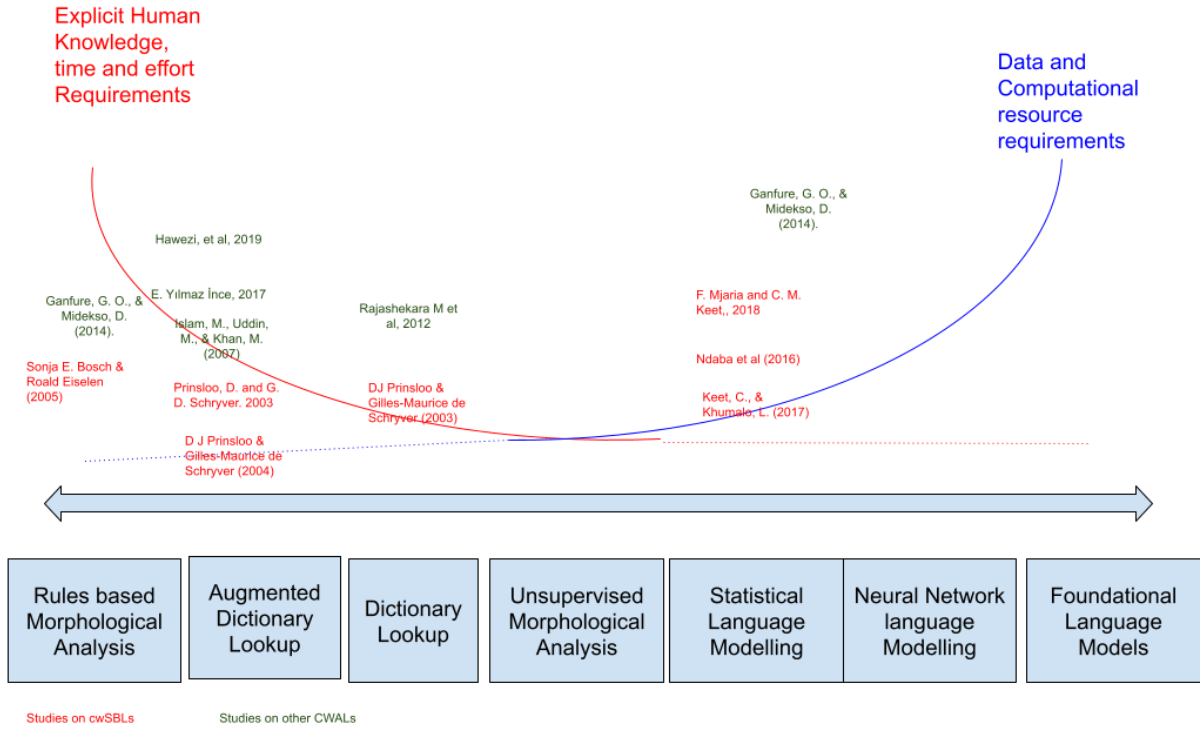


Figure 3.5-4 Approaches to Spell Checking for CWSBLs

3.5.4. Discussion

Summary of findings

Whilst it can be argued that every researcher who has worked on the spell-checking problem of the CWSBL is acutely aware of the OOV problem, none of them have explicitly looked at the question of how to improve the performance of spell checkers on OOV words. The main reason for this was initially pragmatic (de Schryver & Prinsloo, 2004). Recently there has also been a narrow focus on meeting the performance benchmarks against limited test documents without explicitly considering how the spell checkers would perform on words that are foreign to their lexicon. This approach is not entirely without merit due to the Zipfian nature of language. However, since CWSBLs have a long tail of words that occur very infrequently, the need to address them is critical. One of the key challenges identified is that researchers must balance the need to produce solutions that can be made available to the language communities within the shortest amount of time with the need to develop comprehensive

models that take even longer to develop. Pragmatism triumphs and, as such, the tools developed are known not to be as effective as they could be. It is also not clear if the desire to be able to adequately address OOV words is shared by all in the research community. There appears to be a mistaken view that achieving high levels of accuracy and recall during the development process indicates equally good recall and accuracy with OOV words. However, since these metrics are measured against the whole test documents without paying due regard to OOV words, it is unclear if the misses are all OOV words, or if any of them are correctly identified. Whilst dictionary lookup-based spell checkers will fail on OOV words, the same is not necessarily true of the other approaches.

Strengths and limitations and future research

The key weakness of this review is that it was conducted by one student and was thus unable to follow all the guidelines for the RAMESES meta-narrative review process.

Comparison with existing literature

The field of spell checking for CWAL outside of the CWSBL is more advanced. There have been other reviews conducted on the general spell-checking problem. These include the work of work of (Gezmu, Nürnberger, & Seyoum, 2018; Ahmadi, 2021; Himoro, 2020). In comparison to those studies, this review was focused on CWAL with a particular focus on CWSBLs. It confirmed the same methodological approaches to spell checking as were previously reported on by (Kukich, 1992) and many others since then.

3.5.5. Summary of Meta-narrative review

This review sought to answer two questions pertaining to the performance of spell checkers for CWSBLs. It established that there has been an increasing movement towards more sophisticated approaches to the handling of the OOV problem. However, there has not been any explicit focus on evaluating the performance of the spell checkers based on these more sophisticated methods. Based on this it is clear that the OOV problem is not yet adequately addressed for CWSBLs.

3.6. Step 2: Define the objectives for a solution

The second step within the DSRM methodology is the clear articulation of the objectives for the solution. Key inputs into this step are the outputs of the initial step as well as an understanding of the technological status quo as well as the domain of the problem. With this understanding, the objectives of the solution can be inferred. As a result, the following subsections will start with the identification of the objectives for the solution. Following this the Design and development of the solution will be described in the next subsection.

Objectives of Solution

The problem faced with spell checking for CWSBLs is that there is limited data that is available to feed large data driven language models. The largest publicly available corpus for any SBL only has just over 20 million tokens (or running words), compared to that of English which has more than 2 billion running words. Most SBLs have corpora whose sizes are less than a million words. Given the type to token ratio of the CWSBL, these corpora have severe limitations when it comes to supporting data driven language models. The challenge for these languages is to improve the performance of language tools built on top of these corpora without requiring them to be expanded.

Broadly speaking, the ultimate objective of this project is to develop a spell-checking algorithm which increases the chances of recognising OoV words without increasing the size of the input dictionary. The key aim is to maximise Error recall, the Negative Predicted Value as well as the suggestion accuracy without using a larger dictionary to drive the spell checker.

Background to the requirements for MAShoKO

In this section we present the background to the requirements for MAShoKO. There is a need for a software application to support spell-checking for Shona within a number of other software applications that may receive input in Shona. Such a system should be able to check the validity of given Shona words, indicate possible corrections for incorrectly spelt words and provide and analysis of a given Shona word. The purpose of this application is to minimise the reliance on human editors to pick up incorrect spellings in Shona documents that are captured in computer systems. It needs to be able to do this without requiring access to an infinitely large dictionary of Shona words, which is not available.

Requirements for MAShoKO

The generic requirements for a spell checker are well understood. At a minimum, it should be able to take in a word and output a statement of its validity or not. Other requirements include the ability to accept new words into its lexicon based on user input, as well as the ability to suggest corrections for incorrectly spelt words.

Detailed Requirements

The system needs to be able to do the following

1. Take in a string of text
2. Check if the string is a valid Shona word
3. Return the validity status of the text

UML Use Cases for MAShoKO

Figure 3.6-1 documents the use cases that MAShoKO should be able to execute. In the following passage, these use cases are described in detail. There are three main uses as shown below.

Check Spelling: In this use case the user provides a word whose spelling is to be checked. Executing this use case initiates first the Check Syllabic Correctness use case and then depending on the outcome of that use, the Search Dictionary use case, and then the Analyse verb and Analyse Noun use cases.

Correct Spelling: A user can request that the spell checker corrects a given word. This use case begins with the Check spelling use case. If the word is flagged as incorrect, the Get Suggestions use case is executed. This in turn includes the Create suggestions use case. After the user is presented with a list of suggestions, they may choose to add the original word to the custom dictionary, which executes the Add word to Custom Dictionary use case. This use case optionally executes the Add Custom Dictionary use case.

Analyse Word: A user can provide a word for which they require a morphological analysis. This use case utilises the Analyse Verb and Analyse Noun use cases.

Discussion of Requirements

As previously stated, the requirements of a spell checker are relatively well understood. The main requirement is for a user to be able to provide check the correctness of a word as well as to suggest that a given word be added into the custom dictionary. In this version of MAShoKO, it is assumed that words are presented individually, and their correctness is independent of the context within which they are found.

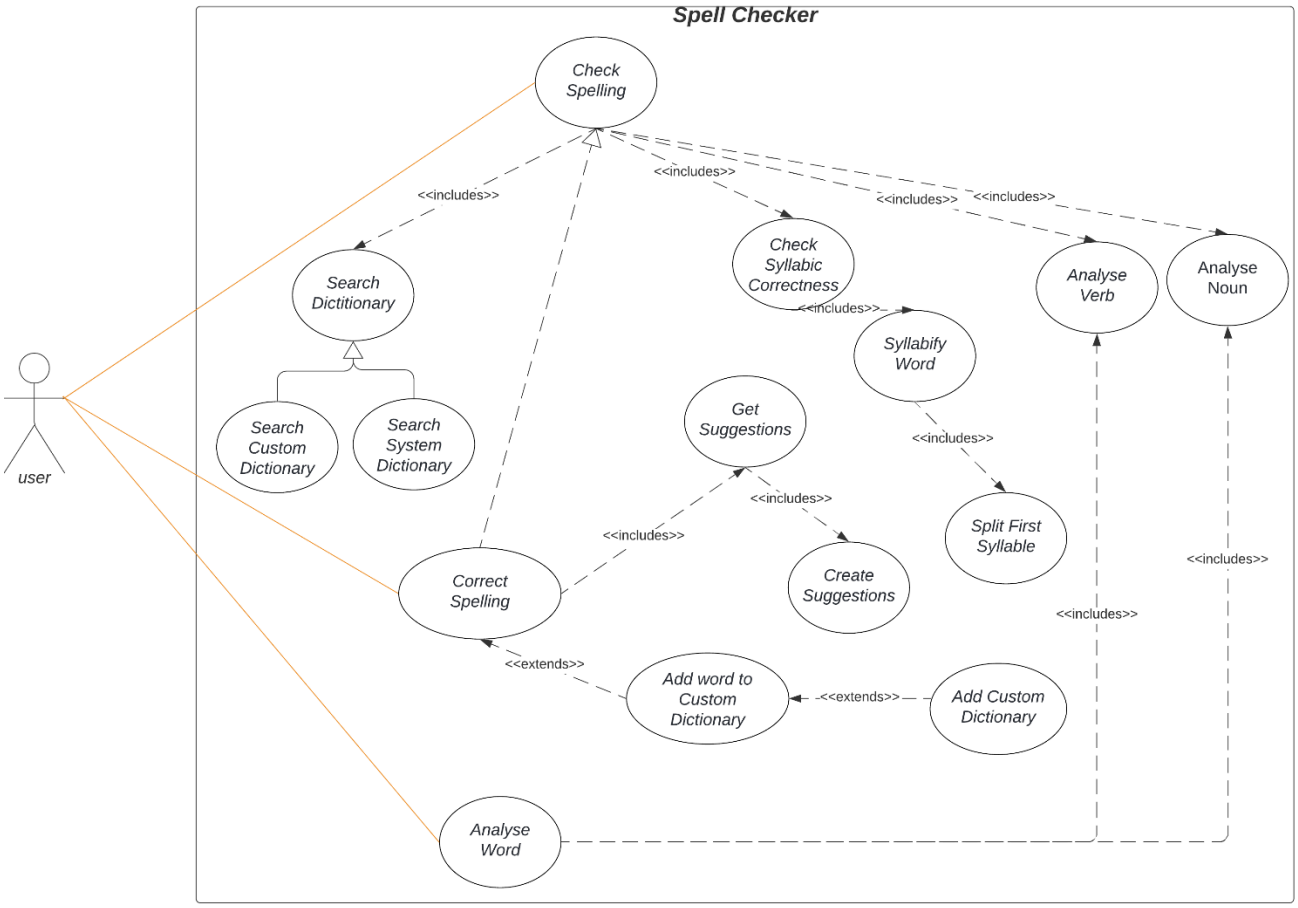


Figure 3.6-1 Use Case Diagram for MAShoKO

3.7. Step 3: Design and Development

Within the DSRM methodology, problems can be solved by the use of various artefacts. These can include *constructs, models, methods, and instantiations.* Design is an exercise in making choices and taking trade-offs between potentially conflicting options. Such choices need to be made around the features, architecture, and performance of the solution.

In this thesis, two key artifacts need to be developed to realise the objectives of this research. The first one is a ***morphological analyser*** for Shona. The morphological analyser forms the basis for the word recognition engine within the spell checker. The second key component required in this research is the actual ***spell checker***. Whilst it can be argued that the morphological analyser is a subsystem of the spell checker, it is important to note that the morphological analyser envisaged in this research can be used as a stand-alone application for the purposes of evaluating the morphology of Shona words. It

can also be used as the basis for a Shona parser by extending the model to cover complete sentences and not just words. Other important considerations in the development of a knowledge based morphological analyser include the number of word types that the finite state automata should cover. Covering more word classes increases the accuracy of the morphological analyser. However, this comes at the cost of more time and effort on the part of the researcher. The converse uses less time but sacrifices accuracy. The right balance which delivers better accuracy than other methods need to be established.

3.7.1. The Design of MAShoKO

This section presents the design of MAShoKO. Linked to the objectives of this thesis, the main emphasis of this section is on the design of the algorithm for the spell checker. Key to the design of MAShoKO is the adoption of the Spell Checking as Morphological Analysis (SCaMA) approach. Spell checking is taken as a special case of morphological analysis. The argument is that all morphologically valid words are also correctly spelt. It therefore follows that if a morphological analyser can correctly segment a given word, that word can be assumed to be correct.

3.7.2. The MAShoKO approach to Shona Spell Checking

Figure 3.7-1 shows the high-level flow chart for the MAShoKO Spell Checking module. This is also depicted in *Figure 3.7-2* using a UML sequence diagram showing the various objects that are involved in the spell-checking process.

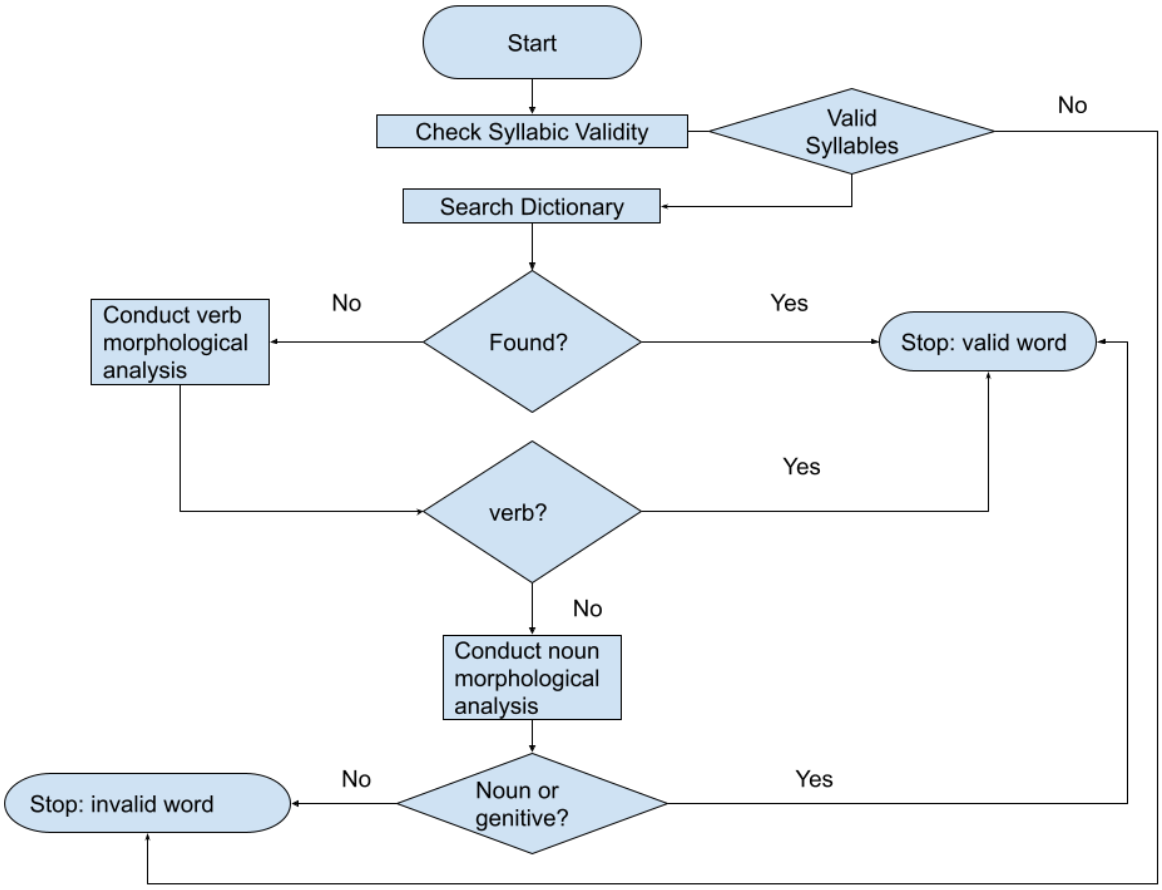


Figure 3.7-1 Flow Diagram for MAShoKO Spell Checker

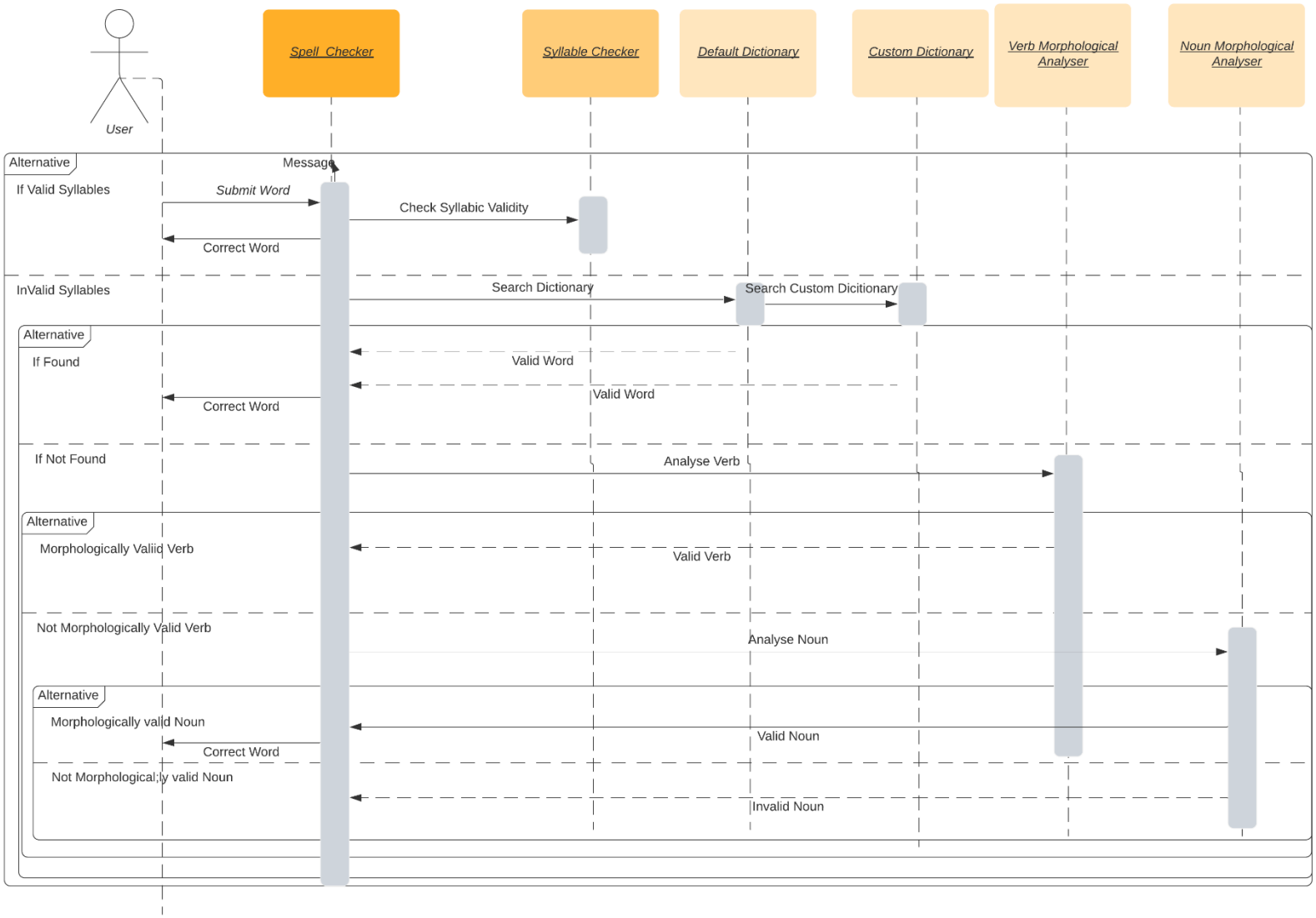


Figure 3.7-2- UML Sequence Diagram for MAShoKO Spell Checking Module

As the two diagrams show, MAShoKO performs spell checking in a number of increasingly complex steps. The first step is to conduct a check on the validity of the syllabic composition of a word presented to the spell checker. This is a computationally cheap operation and provides a quick way to root out clearly misspelt words. Any word that contains invalid syllables is flagged as incorrectly spelt.

The next step within MAShoKO is to check if the submitted word exists in either the built or the user defined dictionary. This operation is relatively inexpensive in terms of computational time, being in the order of O(n). If the word is found in the dictionary, the spell-checking process concludes with a verification of the word as being correctly spelt. However, if the word is not found in either dictionary, the more computationally intensive part of the process begins. First a morphological analysis of the word, assuming that it is a verb is conducted. If a successful parse is completed, the word is presumed to be correct. Should it fail this analysis, it is sent for further morphological analysis – this time against the specification of nouns. Within the current implementation of MAShoKO, if it fails both morphological analyses, the word is assumed to be incorrect and flagged as such.

3.7.3. High level system design

MAShoKO was designed to be available via three different interfaces within this research project. The first manner in which it can be interacted with is via the command line. Second, a RESTful Application Programmer's Interface (API) was developed to expose the core capabilities of the spell-checking engine. Third, MAShoKO was incorporated into a mini application which was used to compare its performance to the Character Trigram Language Model (CTLM) based Spell Checker.

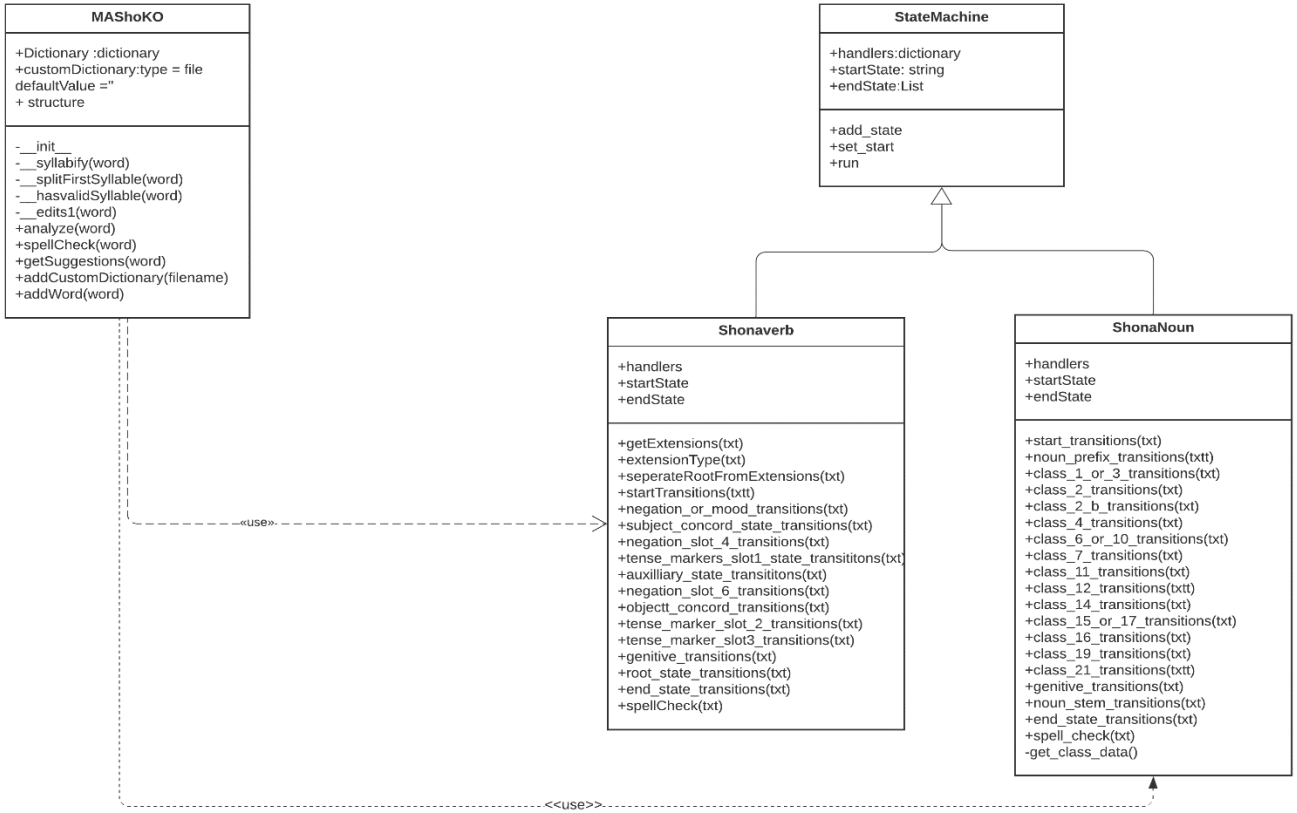


Figure 3.7-3- MAShoKO Class Diagram

Figure 3.7-3 shows the classes that are implemented in MAShoKO. The main class is the eponymous class MAShoKO. This class provides the functions required of the spell checking application. It has two public properties and one private one. The private property is the name and location of the default dictionary. The name and location of the Custom Dictionary as well as the structure of the last analysed word are available as public properties.

The MAShoKO class exposes five public methods. The first method provides the functionality to analyse a given word as previously described. This method is linked to the spell checking method. The third method generates suggested corrections for a given incorrect word. The last two methods provide the ability to add a custom dictionary as well as to add words to it respectively.

In order to give effect to the above a number of Shona specific routines needed to be developed. The first one of these is a Tokeniser for Shona text which is described in the next subsection.

3.7.4. The Tokenizer

Within the context of a spell checker, a tokenizer splits the contents of a text document into individual words. Conceptually this process is trivial as it is based on an understanding of the orthography for a specific language. Many tokenizers exist for languages such as English, and if one were developing a spell checker for English, there would be no need to develop a new one. However, Shona has an orthography that differs from that of English. As a result, an English tokenizer does not always tokenise Shona words correctly. One particular set of morphemes poses challenges for generic tokenizers when they are applied to Shona: this is the *n'* phoneme. English tokenizers consider the apostrophe to be an indicator of the possessive and as such they separate the words that include this morpheme into two parts using the apostrophe as a separator. In Shona, this tokenisation would be incorrect. Apart from this, the rules for splitting a Shona text document into individual words is similar to that for English. Specifically, Shona words are separated by white space, and punctuation marks excluding the apostrophe that follows the letter *n*. Table 3.7.1 provides the pseudocode for the tokenizer which applies this rule to tokenise a given document of Shona text.

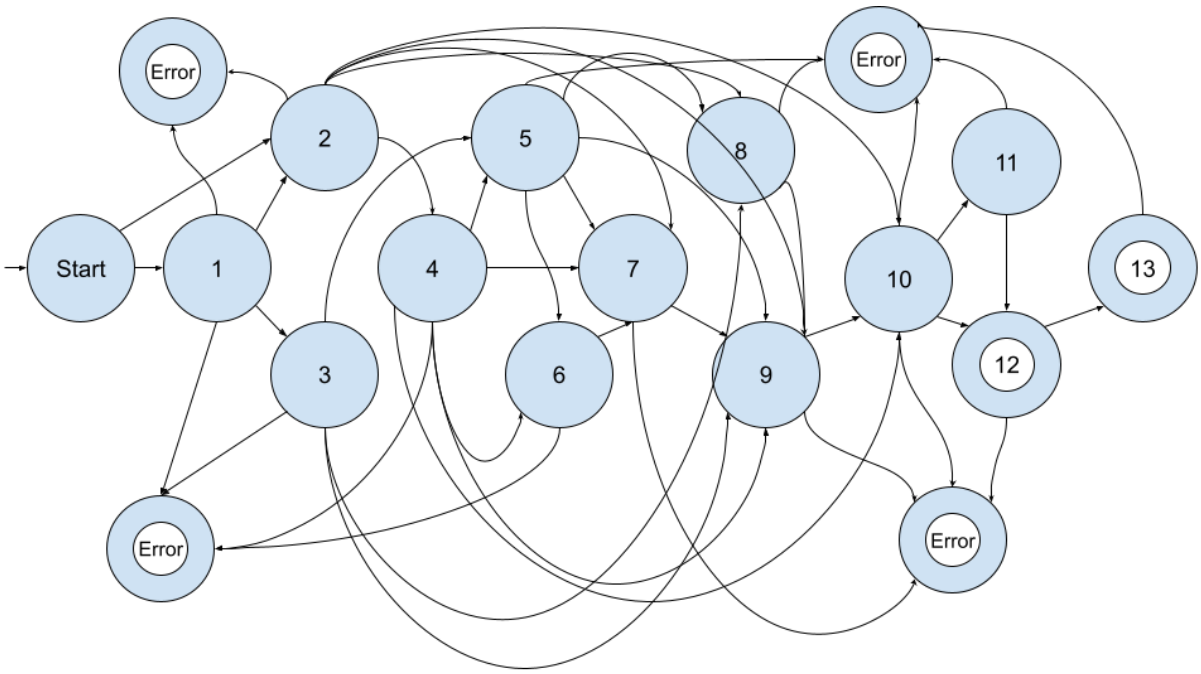
```
i = 0
word(i) ← null string

While not end of file(document)
   nxtChar ← Get next character in document
    While nxtChar not (white space or punctuation mark excluding apostrophe)
        Word(i) ← Concatenate (word, nxtChar)
        nxtChar ← Get next character in document
    end
    i = i+ 1
return word
```

Table 3.7.1 Pseudocode for Shona Tokenizer

3.7.5. The Shona verb Morphological Analyser

The verb is the most complex morphological word type in all Bantu languages including Shona. Each Shona verb is made up of up to 13 distinct morphemes as defined by Mberi in his 13-slot system. A finite state representation of this 13-slot system was implemented as shown in *Figure 3.7-4*. Each state of the FSA represents one of the slots in the verb slot system. Valid transitions show permissible combinations of morphemes from one slot to the next. The elements that make up all the 13 slots, except for the verb roots are finite and were easy to deduce. They are all monosyllabic in composition, so the FSA consumes them one syllable at a time. The challenge is with the verb root. First verb roots are not composed of a full set of open syllables. For example, the verb root <*famb*> (root of to walk) has one full open syllable <*fa*>- and the *syllable margin* <-*mb*>. Since the length of verb roots is not easy to identify ahead of time, the Morphological analyser switches its direction and analyses verbs in backward fashion – consuming all the verb extensions, whose lengths are more determinate. Once all the verb extensions have been consumed, the remaining text is considered to be a verb root. This is checked against a dictionary of verb roots. If it matches, the word is accepted, otherwise it is rejected.

In theory, the list of verb roots is also finite. However, it is possible that a given corpus or dictionary may not have a comprehensive listing of all potential verb roots. This implies that a FSA which aims to recognise verbs may not be able to recognise all verb forms if it has to rely on a finite dictionary of verb roots. Since the goal of this research is to maximise the recognition of OOV words, a heuristic was developed to enable the FSA to accept verb roots that did not exist within the lexicon. Instead of implementing the Shona Verb Morphological Analyser as a true FSA, the automaton is allowed to read the last part of a verb starting from the back. This allows it to consume all the verb extensions and the final vowel. After all these elements have been consumed, the remaining part of the verb is checked against a dictionary of verb roots. If it is among these, the word is accepted as valid, otherwise it is rejected.



1	2	3	4	5	6	7	8	9	10	11	12	13
Neg Mood	SC	tma	Neg	tma	Neg	tma	Aux	OC	R	Ext	FV	clit & oth
i	sc	i	si	chi	si	chi	ndo	zvi	R	an	a	eyi
ha	sca	cha	sa	ka	ka	ka	mbo			anur	e	e-yi
ha	sco	no		do	za	zo	ngo			ek	(i)	e-pi
nga	u			ne			zo			enur		a-ka
	mu			nga			fum			er		e-ka
	ku						etc			ik		
										inur		
										is		
										onor		
										unur		
										w		

Figure 3.7-4 - - Finite State representation of Mberi's 13 slot representation of the Shona Verb

The ShonaVerb Class implements the 13 slot system as a Finite State Machine. Starting with a start state, it represents each of the slots as a state within the FSM. Each state has an associated public method as shown in the class diagram in *Figure 3.7-3*. The method that runs the FSA is named spellCheck, although it performs more than just a spell check. It walks through the FSA, storing morphological information about each node that has been visited into the structure property. This is the information that the analyse method of MAShoKO uses when it is run in morphological analysis mode.

3.7.6. Shona Noun Morphological Analyser

The ShonaNoun Class is similar in anatomy to the ShonaVerb class. It is based on the same StateMachine class and shares the same public and private properties with it. Given that Shona nouns fall into one of 20 classes with most of the classes paired, the class has a few more methods than the verb one. Each verb class is treated as a separate node within the FSM. Since nouns can be genitivised by appending any one of a number of prefixes, an additional node for genitivising nouns is added to this class.

3.7.7. Other parts of speech

One of the key design decisions made was to not implement morphological analysers for the other parts of speech as their morphology is relatively simple. As a result, the likelihood of encountering most of the words belonging to these classes in a dictionary is high.

3.7.8. MAShoKO RESTful API

A Simple RESTful API for MAShoKO was implemented using the flask web application framework. The API implements two of the core functions of MAShoKO: i) the spell-checking function and ii) the correction or word suggestion function.

3.7.9. MAShoKO and CTLM Comparison App

In order to enable the comparison of the performance of MAShoKO to the CTLM based spell checker, a simple graphical user interface (GUI) app was developed. This application takes in as input a text file, or user typed in text. It then runs spell checking on that text using both the MAShoKO and CTLM spell checking engines, saving the results to two text boxes below the main input text box. It also provides an indication of the number of OOV words that it encounters so that the error rate of the two spell checkers against OOV words can be calculated. Use of this application will be further discussed in section 4.5.

3.8. Step 4: Demonstration

The fourth step within the DSRM is to demonstrate the effectiveness of the delivered solution to solve the problem for which it was designed. The guidelines for the methodology offer several possible ways in which such a demonstration can be conducted These include the conduct of experiments, simulating the problem, conducting case studies, or generating proofs. Within this study, the approach that will be used to demonstrate the spell checker will be that of experimentation. The morphological analysis-based spell checker's performance will be compared with that of other models built using the traditional approaches. Chapter 5 presents the materials and methods that will be used to conduct the experiments to demonstrate the efficacy of this solution to resolve the problems identified in the literature review.

3.9. Step 5: Evaluation

There is a strong link between demonstration and the evaluation of the solution. Evaluation requires that the actual observed performance of the solution be reviewed against some expected or desired benchmark. In conducting this evaluation, appropriate metrics need to be utilised for the specific problem domain. Furthermore, such metrics have to be compared against meaningful benchmarks where these are available.

The results of the evaluation of the solution developed in this thesis will be presented in Chapter 7. This will be done using the metrics presented within Chapter 2. However, one of the challenges with these metrics is that they do not explicitly consider the case of OOV words. Addressing this question is one of the key focuses of Chapter 7 as the results of the evaluation are presented.

3.10. Step 6: Communication

The DSRM guidelines end with a recommendation that exhorts all researchers to communicate their work to the *appropriate research community*. Whilst this thesis is one of the ways in which this study will be communicated to the computational linguistics community working on CWSBLs, components of the research will be submitted to relevant journals and conferences. Conclusion

The aim of this chapter was to provide an overview of the research methodology used to answer the research questions for this study. It started with a general presentation of the methodologies that are applicable to CS research and then narrowed down to the selection of Design Science Research as the specific approach that will be used in this research. The DSRM variant of DSR was selected as the specific approach to be utilised for this study. The methodology was chosen because of its fitness to the specific objectives and the questions that this research seeks to answer. Finally, the steps of how this research is to be conducted and how these tie in with the rest of this thesis was presented.

3.11. Chapter Summary

This chapter introduced the various methodologies that are available for use in Computing Science Research. Considering the nature of the research question, the Design Science methodology is chosen. There are several ways in which Design Science can be conducted. Each of these were considered, after which the Design Science Research Methodology was chosen for this research. After this, the procedure for conducting this type of research was presented in detail, leading to this conclusion.

"The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them."

William Lawrence Bragg.

Chapter 4 - Materials and Methods used to demonstrate and evaluate the solution

4.1. Introduction

This chapter will describe the materials and methods that were used to evaluate the performance of the spell checker that was developed in the previous chapter. In doing this, this it will be setting the stage for meeting research objectives 7 and 8. To recap, the progress made so far; of the eight research objectives, RO1 and RO2 were addressed in Chapter 3 using the meta-narrative review of the spell-checking literature for CWSBLs, RO3 to 6 were also covered in Chapter 3 as part of the description of the design and development step of the DSR methodology.

The purpose of this chapter is to describe the materials and methods that were utilised to conduct the experiments that were used to evaluate the performance of the MAShoKO spell checker against the CTLM based spell checker.

The rest of this chapter proceeds as follows: first the Data collection method will be described. After this, the experimental set up for testing the efficacy of the MAShoKO based spell checker against the CTLM based spell checker will be described. This includes the types of measurements and metrics that will be used to compare the performance of the two spell-checkers. The chapter will conclude with a summary of the content covered here.

4.2. Data Sets - Change to method for collecting data

Three main data sources were used to develop and to test the spell checkers. The words used to develop each of the spell checkers were acquired from *Duramazwi Guru ReChiShona* (DGS) (Chimhundu, 2001) whilst the text from the one hundred thousand word Leipzig Corpus (LC) (Goldhahn, Eckart, & Quasthoff, 2012) were used to test the spell checker. For the knowledge based morphological analyser, the book *A Descriptive Grammar of Shona* (ADGS) by (Mpofu, Ngunga, Mberi, & Matambirofa., 2013) was used as a source for the grammatical rules.

DGS is a monolingual Shona dictionary which, in its printed form, comprises of three main sections: i) a front matter section which introduces the dictionary, provides guidance on its usage, and introduces some key aspects of Shona grammar; ii) the main part of the dictionary which consists of *lexemes* and their definitions; and iii) a section with various reference materials related to Shona culture and language use. The data for the development of the spell checkers presented here was sourced from the main dictionary section of DGS. This main section was provided as a pdf document. The reason for this is that this is the section that speakers of Shona would refer to if they needed to lookup any given word's meaning.

The LC Collection is a publicly available set of corpora for about two hundred languages. Each of the corpora are presented in the *same format* and they are sourced from *comparable sources.* Made up of randomly selected sentences from each selected language, they contain materials taken from newspapers and other randomly selected text from the web. Care is also taken to remove sentences that are not in the language of the corpus as well as those that are not deemed to be sentences. All this means that the corpora have a likelihood of being representative of the contemporary usage of their content language on the internet. It is for this reason that the one hundred-thousand-word Shona LC was chosen.

ADGS was written to cater for a diverse audience of readers with a special emphasis on university students and lecturers, secondary school teachers, and researchers of Shona and Bantu languages in general. It presents a descriptive rather than a prescriptive grammar. This means that it is a good reference for the language as it is used by the speakers and writers making it a good reference for the development of tools that are meant to be used by native speakers of the language.

The following subsections describe the process that was undertaken to acquire this data.

4.3. Data Pre-Processing - change to method for pre-processing data

DGR was obtained in pdf format and needed to be converted to a format that could be used to develop the spell checkers. The following subsections describe the process used to extract the text from the dictionary into the format required for the spell checker.

4.3.1. Extract Text from Dictionary

Each page of the dictionary section comprises of two columns of text. The first column is headed by left justified text. This heading has the lexeme that is the first entry on that page. The second column is also headed by a lexeme which is right justified. This lexeme indicates the last entry on that page. A python script to extract the contents of DGR from the pdf document and convert it into a conventional ascii text file was developed. This utility was built in using the *PyPDF2* library for optical character recognition (OCR). Listing 4.3-1 is the pseudocode for the logic applied by this utility to extract the text from DGR.

```
Initialise Converted Text to null string

Open pdf document

Convert pages of pdf file to images

For each page image created

Convert page image into text

Append Text of appended page to Converted Text

Save Converted Text to Text File
```

Listing 4.3-1 – Convert PDF to Text

4.3.2. Extract List of words from Dictionary

A second utility to extract a list of all the words found in the dictionary was developed. This utility takes the output of the first script as its input. It then tokenises this text document. After tokenization three files are generated: i) a file that contains all the distinct words encountered in the dictionary; ii) another file that contains each distinct word followed by a count of its incidence in the dictionary; and iii) a file of all the sub word lexemes found in the dictionary as well as the counts of their incidence.

4.3.3. Extract Word List by part of speech

Another script to extract the lexemes defined in the dictionary was also implemented in python. First the utility extracts all the verbs from the text document. The location of all the verbs that are defined in the dictionary is established by checking for the transitive verb indicator symbol <*it*>' or the intransitive verb indicator <*itik*>. After indexing these locations, the program parses through the dictionary text and extracts all the entries at these locations. A similar procedure was also used to identify the locations of all the other parts of speech types in the dictionary. The output of each of these searches was saved into separate files for each part of speech.

4.3.4. Get text data from Leipzig Corpus

Text from the Leipzig Corpus (LC) was used as the test data for the experiments. Word lists and exemplar sentences for Shona were extracted from the LC. From the 2018 Shona LC, we extracted *sna-zw_web_2018_100_k-words, the one-hundred-thousand word Corpus.*. This file is, like all of the other LC files is formatted as follows: i) each entry in the file is comprised of four tab delimited columns. ii) The first column contains the entry or line number. iii) Following the tab delimiter is the *presumed word*. They are presumed words because not all the entries in each line are valid orthographic words. iv) Column three is a repeat of column 2 and contains the same presumed word. v) Last is a number indicating the frequency of the entry on that line. Each of these files was imported into Microsoft Excel as tab delimited files. The words obtained were then compared with the words acquired from DGS. Those words that did not occur in DGR were marked as OOV. Calculations of the ratio of OOV to known words were done and the results tabulated.

4.4. Experimental Setup - Change to method to evaluate spell checkers

All three experiments used the same data sets, took the same measurements, and calculated the same metrics

4.4.1. Overview

Each experiment consists of a test in which the spell checker is tasked with spell checking the text of the Shona 100k word LC. Unlike in some previously reported work, no attempts were made to generate synthetic errors into the text. Instead, the texts are to be tested in their original state. The performance of each spell checker on the full set of words as well as on those words that do not occur

within DGR is noted and recorded as described in section 2.5.1. Performance metrics are then calculated also as detailed in section 2.5.2.

4.4.2. Evaluation

The following measures were used to evaluate the performance of each of the spell checkers.

Measurements

We measure the number of words in DGR as well as those in the 100k word Shona LC. The number of words occurring in the 100k word Shona LC that were not in DGR were noted and counted. For each experiment, the number of words that each spell checker can correctly identify were also counted. A similar count was also done of all the words that each was unable to identify. Counts of the number of out of vocabulary words that each spell checker could correctly classify were also done.

Metrics

The previously discussed measurements were used to calculate the Standard metrics as defined in Section 2.5.2. A key difference is that these metrics are also calculated just for the OoV words in order to determine how well each spell checker performs on them. Additionally, these metrics are also calculated by part of speech type.

4.5. Conducting the comparative spell checking experiments

The program *10 Compare MAShoKO to CTLM.py* is executed. This program opens up the interface shown in the screenshot in Figure 4.5-1.

Following this, the Open button is clicked, and this opens up a file dialogue box. From this the test file "Shona_100k_words.txt" is selected. At this point, the interface of the program should look like Figure 4.5-2.

The "Spell Check" button is now clicked. Once this is done, the program immediately starts to write some output to the console as it analyses the words from the test file. After it has completed the spell check task, the program presents a message box indicating the completion status. Clicking OK on this status reveals the interface shown in Figure 4.5-3.

Conducting

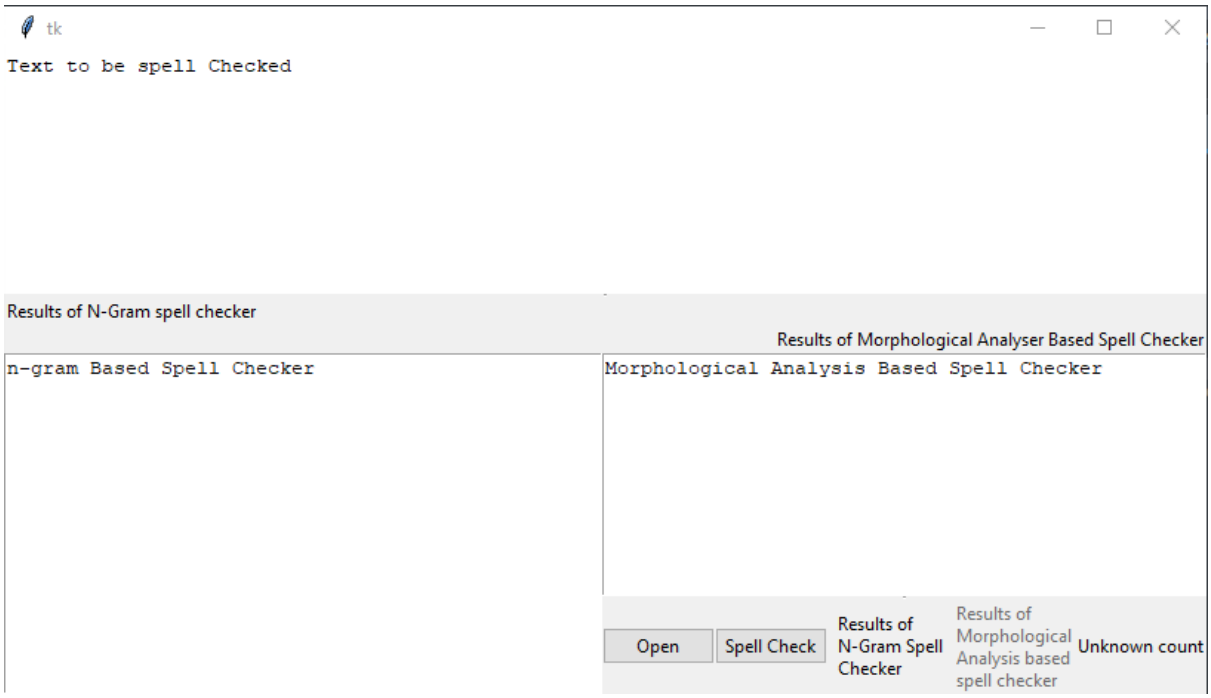


Figure 4.5-1-Screenshort of the Program "10 Compare MAShoKO to CTTLM" before a file is opened

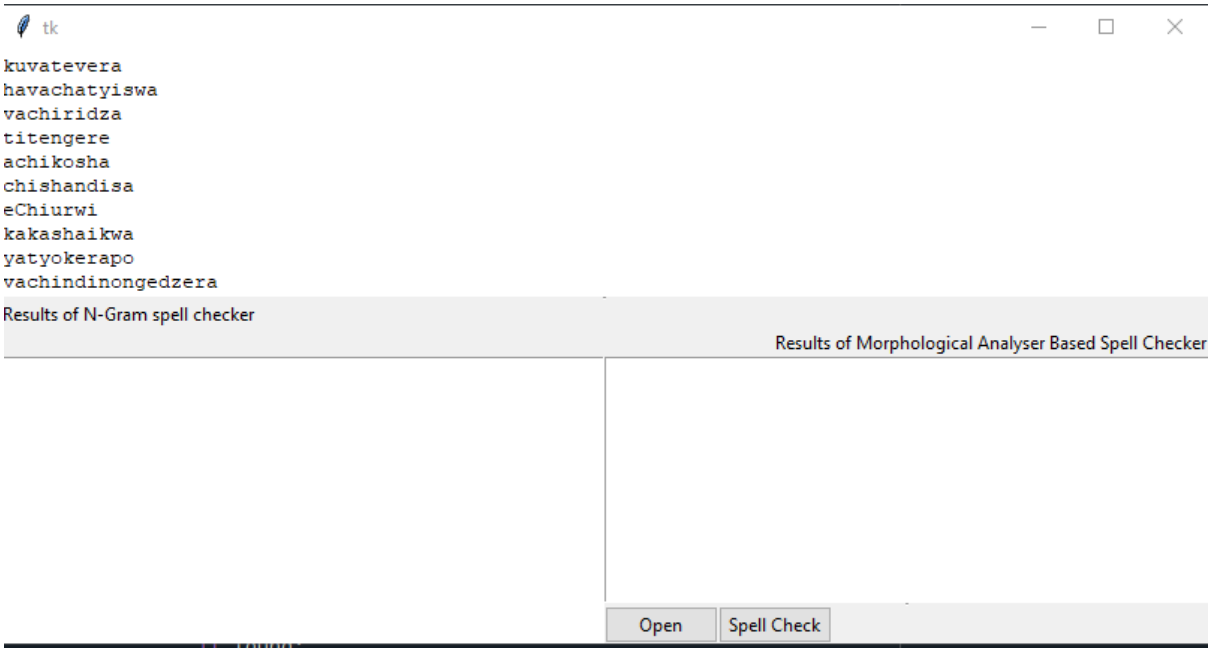


Figure 4.5-2-Screenshort of the program after the test file has been opened

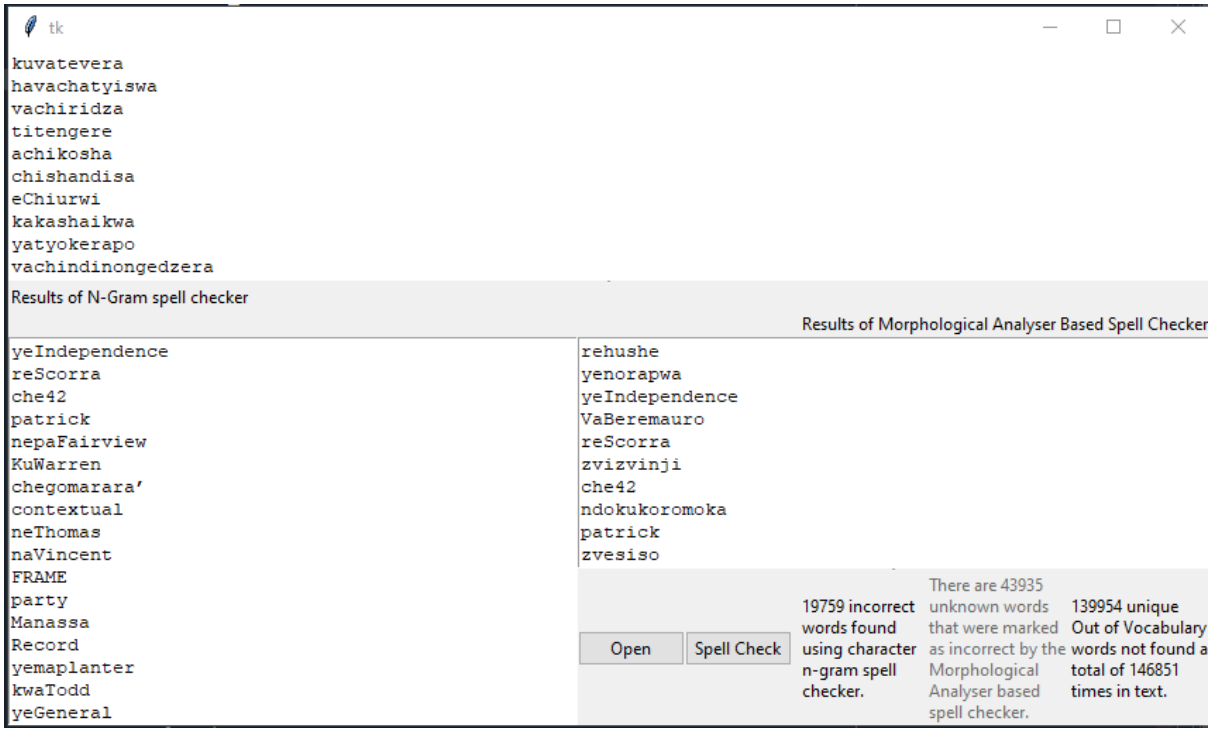


Figure 4.5-3 - Screenshot of the comparison program after spell checking the test document

After the program is done, the list of words identified as misspelt is copied from each of the two boxes which represent the results of the N-gram and the MAShoKO based spell checker respectively. These are copied into a spreadsheet containing all the words labelled based on their part of speech as well as whether they are valid or invalid Shona words. A pivot table which gives the values for a confusion matrix of each of the two methods is then generated and prepared.

4.6. MAShoKO and CTLM Source Code

The source code for all the classes, applications and the API developed are included as Appendix 1 of this thesis. The data used to test the application as well as the results reported in this are archived on GitHub at the following link https://github.com/Farayi/MAShoKO.

4.7. Reliability and Validity

The aim of this chapter is to present a method that evaluates the performance of a morphological analyser-based spell checker as well as to compare it against an existing method. The experimental method in the preceding sections adequately addresses both questions and should provide a reliable and viable method to achieve these stated aims.

4.8. Chapter Summary

This chapter detailed the software artefacts and data required to evaluate the performance of the MAShoKO spell checker against the CTLM benchmark that was previously developed by (Ndaba, Suleman, Keet, & Khumalo, 2016). The procedure was to set up the experiment as well as conduct the experiment. This was then followed by a discussion on the reliability and validity of the methods presented here.

"Results! Why, man, I have gotten a lot of results! I have found several thousand things that won't work."

Thomas A. Edison

Chapter 5 - Results

5.1. Introduction

This chapter presents the results of the experiments that were conducted as part of this research. The first section details the global findings about the datasets used in the experiments. Each of the subsequent sections gives the detailed results of each of the experiments that were conducted on the two baseline spell checkers as well as on the two morphological analysis-based spell checkers. After this is a summary section which consolidates all the results before the conclusion.

5.2. Recall

Figure 5.2-1 shows the recall performance of the MAShoKO Spell checker versus the CTLM Spell Checker on various word types as described in the previous chapter. The recall is expressed as a percentage, and it is calculated for all the words in the sample of LC used to evaluate the performance of the two spell checkers.

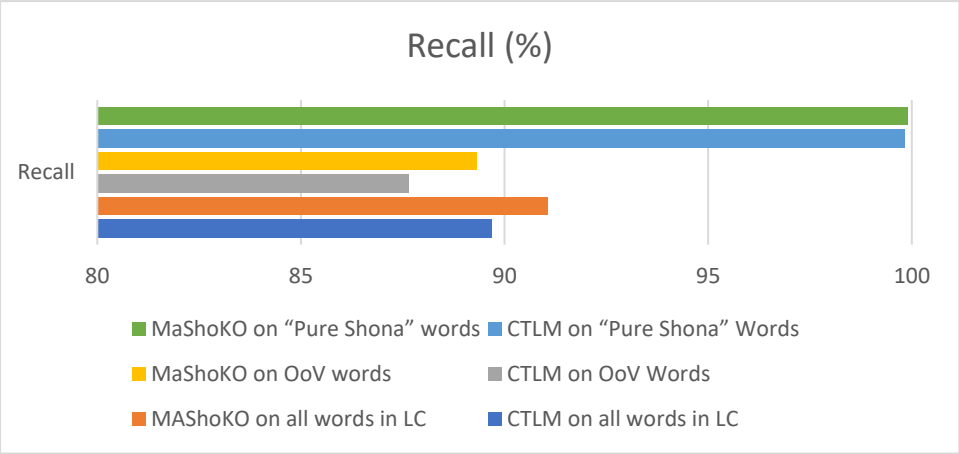


Figure 5.2-1- Comparison of the performance of the two spell checkers on various categories of Shona words

5.3. Specificity

The specificity of the two spell checkers were calculated for each of the various word types as well as for the full data set. The results of these calculations were graphed and shown in Figure 5.3-1.

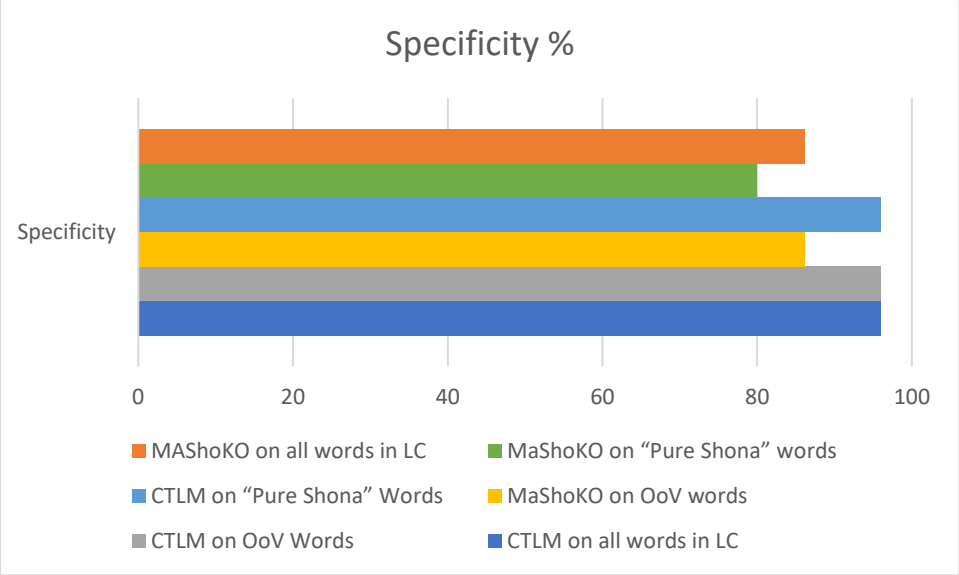


Figure 5.3-1-Specificity of the two spell checkers on various categories of Shona words

5.4. Precision

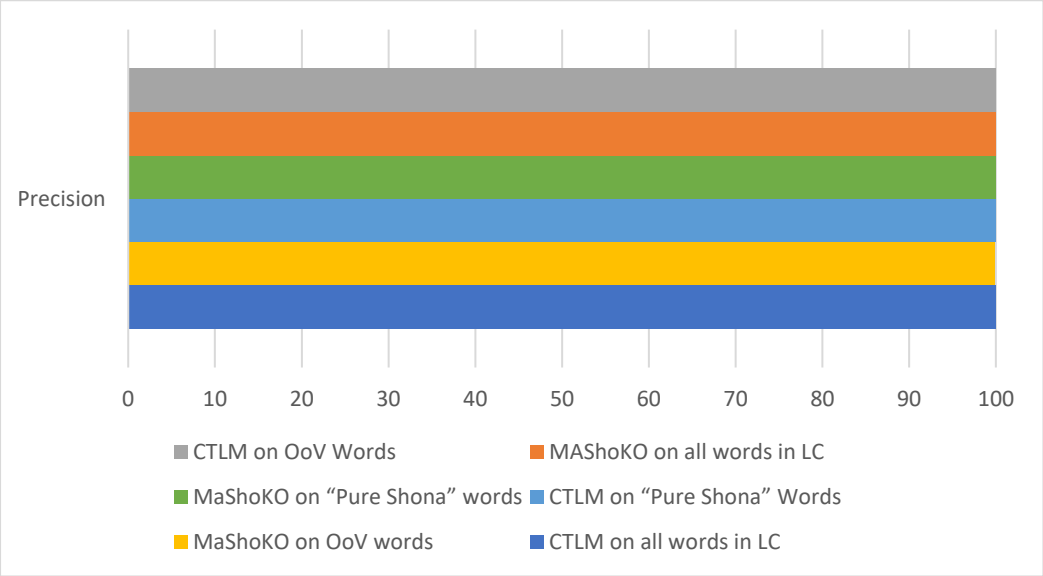


Figure 5.4-1- Precision on various categories of Shona words

The precision of each of the two spell checkers is presented in Figure 5.4-1.

5.5. Negative Predicted Value

After the negative predicted values were calculated, the results were tabulated and graphed. The graph of these results is presented in Figure 5.5-1.

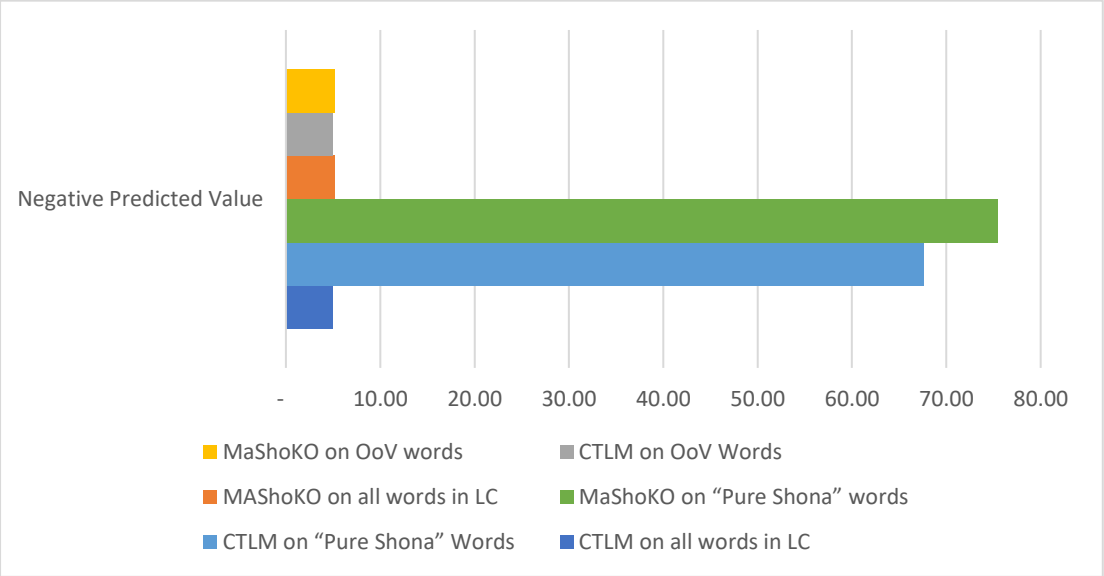


Figure 5.5-1 - Negative Predicted value on various categories of Shona words

in the results.

5.6. Accuracy

The accuracy of the two spell checkers on each of the specified word categories is presented in Figure 5.6-1.

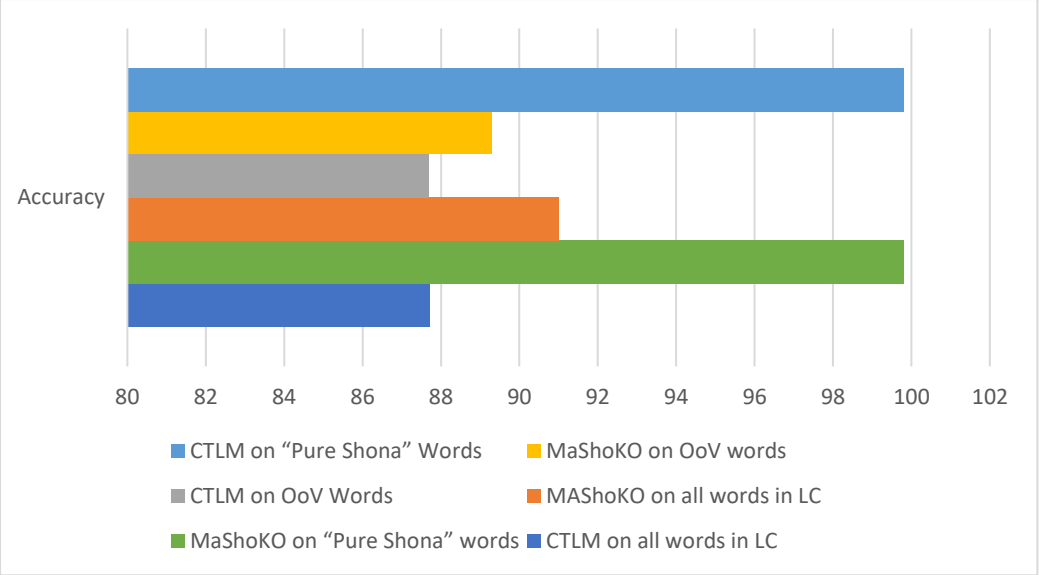


Figure 5.6-1-Spell checker accuracy for various categories of Shona words

5.7. F1 Score

The F1 scores for the two spell checkers are shown in Figure 5.7-1.

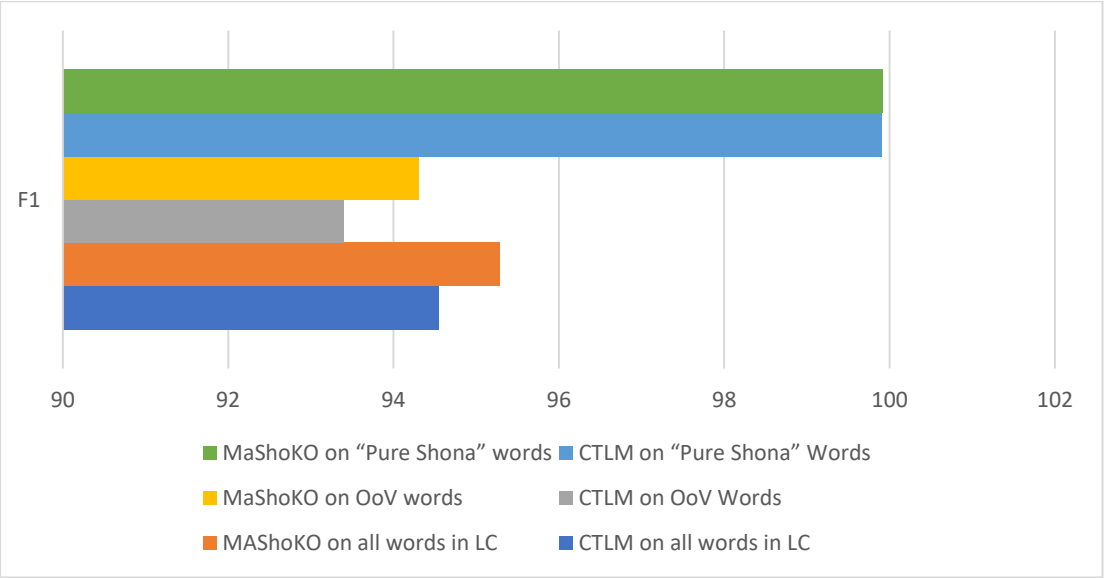


Figure 5.7-1-F1 Scores foe the two spell checkers across different word types

5.8. Description of the results

The performance of the two spell checking methods were evaluated using six standard metrics. Each metric was however extended to consider only the OOV words. The first metric that was used is recall which was described in section 2.5.2. Comparison of recall for authentic Shona words are compared against those on the OOV words and the subset of words that were used to evaluate the LC. The next metric to be evaluated is specificity. This was also evaluated for the same segments of data as the recall. Precision, Negative Predicted Value, Accuracy and the F1 score were calculated in similar fashion for the same data partitions.

5.9. Chapter Summary

The results of the spell-checking experiments using the two engines were presented and described, the next chapter is a discussion of these results.

"Curiosity begins as an act of tearing to pieces or analysis."

Samuel Alexander

Chapter 6 - Discussion

6.1. Introduction

This chapter reviews and discusses the results that were presented in the previous chapter. It starts by looking at the comparative performance of the two spell-checkers on various metrics, then some analysis is performed on the similarities and differences between their performance. Next the key similarities and differences and their implications on the future development of spell checkers are discussed in detail. The chapter closes with a summary of the key findings of these experiments.

6.2. Overview

The results of the experiments that were carried out to compare the performance of two spell checkers on a subset of the LC show that the hybrid approach developed in this thesis marginally outperforms the CTLM that was previously used on the related isiZulu language on the full data set. Once the data set is narrowed down to a mirror the assumptions on which both these models were built, the gap between the MAShoKO based spell checker and the CTLM based one widens. This gap applies to all metrics except that of error recall or specificity, where the CTLM outperforms MAShoKO by a significant margin. The data used to train the model this performance improves significantly. The current version of MAShoKO has a higher likelihood of misidentifying incorrect words than the CTLM based checker does.

6.3. Limitations

A decision was taken to limit the number of word types that MAShoKO would be able to recognise. It was specifically decided to only handle verbs, nouns, and some of the genitive forms of the nouns. These choices were made to ensure that the scope of the project would be manageable. As a result of this, there is no expectation that the current version of MAShoKO would have the ability to properly identify OOV words other than nouns and verbs. Among the word types that it would not be able to identify are the ideophones as well as any code switched words.

It was also decided to use the text of the monolingual DGR as the training corpus for both CTLM and MAShoKO. Due to its monolingual nature, it only has *pure Shona words* and only those borrowed words that are written in the official Shona orthography. Here pure Shona refers to only words that are native to the language and all its dialects. Shona has a specific way of assimilating borrowed words into its lexicon. Some of these methods include at least one of the following modalities: i) all syllables are converted to open vowels - for example, the English word "paraffin" is converted to the Shona word *parafini* where the last syllable changes from the closed syllable *n* to the open one *ni*; ii) the letter *b* becomes the digraph *bh* - an example of this is in the word "bell" which, although its proper name is *dare* in some dialects, can be rendered as *bhero*; iii) likewise *d* is replaced by *dh*, for example "dollar" becomes *dhora*; iv) *v* by *vh* as in "visa" which becomes *vhiza*; v) all *L*s are converted into *R*s as we saw in the "bell" to *bhero* example above; vi) and all *Q*s into *K*s.

The two spell checkers were tested on LC, which is based on Shona text collected from the web. Such Shona text does not always conform to the official orthography as will be discussed later. Both spell checkers had poor error recall on this text, and this was mainly because it contains foreign words that are written in ways that do not conform to the official orthography

6.4. Performance of Spell Checkers

The seventh objective of this project was to evaluate the performance of a spell checker that utilises a morphological analyser as well as knowledge of the language. This objective is tied to the eighth objective which is to establish how well such a spell checker performs against character n-gram language model-based spell checkers. It has already been established that the performance of a spell checker can be measured using a few standard metrics. This previous discussion also showed how prior research did not explicitly call out the performance of spell checkers on out of vocabulary words.

From a user's perspective a good spell checker is one that can correctly identify words that are misspelt. It is as important for it to correctly identify wrongly spelt words as it is to not flag correctly spelt words as being incorrect. Four standard metrics are typically used to measure the performance of spell checkers in the literature. These are Lexical Recall - the extent to which the spell checker can identify correctly spelt words. The second one is Error recall, which is sometimes referred to as Specificity. This refers to the extent to which the spell checker can correctly pick up spelling errors. It

is an indicator of the number of incorrect words flagged by the system as a proportion of all the words that are incorrect.

Precision is an indicator of proportion of well spelt words that are not flagged as incorrect. A high precision indicates that the spell checker does not incorrectly identify correctly spelt words as misspellings. Error precision, also referred to as Negative predicted value precision is like precision except it is defined for the misspelt words. It gives the percentage of misspelt words that the system was able to correctly identify. Two other measures are also used. The first one is accuracy, which indicates the percentage of predictions that the spell checker made that were correct as a proportion of all the calls that the system made. Finally, the F1 score is used to balance the precision and recall of the system, although it does not have an intuitive explanation. However, a higher number is better than a lower one.

The literature review showed that there is usually no explicit focus on the performance of spell checkers on OOV words. This research includes a review of the usual metrics, but it adds a comparison of these metrics for just OOV words. In this vein, the review of the performance of the two spell checkers has been conducted in layers. First, their performance on the full LC is reviewed. Then the analysis is narrowed down to only the OOV words. Last, and more contentiously, the review is further narrowed down to just pure Shona words. The last choice is contentious as it could be argued that the whole goal of building a model is to ensure that it generalises beyond the training data that it is provided with. However, in this case, it is clear that the two types of Shona represented in the training data and the one found in the LC are significantly different. This will be further expounded in the following sections.

6.4.1. MAShoKO outperforms CTLM on Lexical Recall and matches it on Precision

The MAShoKO based spell checker performed better than the CTLM one on lexical recall. This means that, if a word is correct, MAShoKO has a higher chance of leaving it unflagged than the CTLM spell checker does. On precision, the performance of the two spell checkers is indistinguishable. When this evaluation is continued on OOV Words, MAShoKO's performance has a slightly bigger difference to that of CTLM. However, the difference is not as high when it comes to OOV Words that conform to the DGS orthography.

6.4.2. Higher accuracy on MAShoKO

Comparing the accuracy of the spell checkers, MAShoKO outperforms CTLM on the full sample dataset as well as on the OOV words. However, the performance of MAShoKO is marginally worse than that of CTLM on the orthographically valid Shona words.

6.4.3. Poor Error Precision

The error precision of MAShoKO was 5.16% whilst that of CTLM was only 5% on the full 17,970-word subset of the LC used to evaluate the two spell checkers. This means that either spell checker correctly identified only that proportion of incorrectly spelt words. On this basis alone, the spell checkers did not perform well. To answer this question, some further analysis needs to be performed. It boils down to two key elements. The first one is the nature of the vocabulary used in LC versus the vocabulary founds in DGS. LC contains more colloquial language which is written in a more informal register which has a higher incidence of code switching, whilst DGS is more formal and is strictly monolingual. - which is purer and more conformant to the orthodox orthography. Subsequent sections will address this issue further. For now, suffice it to say that this poor showing on lexical recall improves significantly when the subset of words that are morphologically similar to those in DGS are used. The error precision metric was calculated for both MAShoKO and CTLM to evaluate their performance on just the Shona words in the LC. In this case, CTLM's error recall increases to 67.61% whilst that of MAShoKO goes up to 75.47%

6.4.4. Lower Error Recall

The key area in which CTLM outshines MAShoKO is that of specificity, otherwise known as error recall. Not only does CTLM do better than MAShoKO, but the gap between their performance is also significant. This gap is 16% points when the subset of words that is similar to those in DGS is used, whilst it is around 10% for all the other scenarios. This indicates that the CTLM model has a great ability to correctly identify misspelt words in comparison to MAShoKO.

6.4.5. The incidence of foreign words in LC

The question of why both spell checkers fared badly on error recall is one that requires additional attention. The tentative answer that was previously provided is that the words on which the two models were trained are significantly different from those on which they were tested. This appears to

be a non-answer as one would (rightly) expect this to be the case. The value of a model is in the fact that it can generalise to model previously unseen phenomenon. In this case the language used to develop the spell checkers that used to evaluate it are significantly different from each other. This indicates a need to review future approaches to developing and evaluating spell checkers for the language. Specifically, it is important that the spell checker be developed using language that is in the register that matches the texts that it will be used to spell check. This result is not new as similar observations were made by (Keet & Khumalo, 2017). Reverting to the present study, we note that the first key difference between the two is the higher incidence of foreign words in LC than those encountered in DGS. Worse still, not all of these words conform to the expected Shona orthography. Instead, they make heavy usage of code switching

The statistics of this situation should help illuminate the issue further. 827 (4.6%) of the 17,970 words used to evaluate the performance of the two spell checkers are foreign words. The vast majority of these are English words. Of these, many are proper nouns, or as they are referred to within NLP, named entities. The non-English words are also dominated by proper nouns. A few of the remaining words are abbreviations.

A second category of foreign words are found in genitive constructions in which the first part is a class marker, and the rest of the word is the foreign word. Many of these do not follow the official orthography in that they do not change the spelling to render the foreign words as Shona. 919 (5.11%) of the words fall into this category. There is also a level of inconsistency in the way these words are spelt by the different authors whose writings have been included in LC. Some of them put quotation marks around the foreign words whilst others just include them with valid Shona class markers as if they are proper Shona words. Table 7.4.5 provides an overview of the distribution of word correctness categories encountered within LC. These categories were used as the gold standard to evaluate the two spell checkers.

Error Classification of word	Count of Words	Percentage of Words
Valid - Shona word	15,981	88.93%

Valid - Shona plus Borrowed Word	919	5.11%
Valid - Borrowed Word	827	4.60%
Valid - Shona plus Number	103	0.57%
Incorrect Shona Word	51	0.28%
Incorrect - tokenisation error	43	0.24%
Valid - Numerical Value	25	0.14%
Valid Shona - *ny'* phoneme	6	0.03%
Invalid - Incorrect merge of words	4	0.02%
Valid - Shona Slang word	4	0.02%

Incorrect - Borrowed Word	3	0.02%
Valid - Shona word?	2	0.01%
Valid Borrowed word - unconventional orthography	2	0.01%
Grand Total	**17,970**	**100.00%**

Figure 6.4-1 Distribution of Categories of Word correctness within the sample of LC used to evaluate the spell checkers

6.4.6. Handling of Borrowed Words

Of the 3,555 words that the CTLM based spell checker marked as incorrect, 96% were false negatives. Of these 89% were constructions that were composed of either valid borrowed words or genitive and other combinations of Shona and borrowed words. This points to the need to incorporate borrowed words into the spell checkers. This also means that the corpus used to train the spell checkers needs to have more borrowed words, especially when they occurred in the genitive constructions.

6.4.7. Handling of Borrowed Words

The convention in Shona when mentioning numbers is to concatenate the number to the genitive marker or the specific adjective that it is being used with. For instance, the phrase "he has taken a second wife" can be translated to "*akatora mukadzi wechi2*". Here the "two" is included in the word "*wechi2*". Whilst the CTLM spell checker was able to correctly accept a number of these, the MAShoKO one did not do as well. This is because the spell-checking engine within it was not optimised to handle this specific scenario.

6.4.8. Handling of the *nyn'* phoneme by MAShoKO

The MAShoKO based spell checker did not do well with all the words that included the *nyn'* sound as this was not included in the lexicon used to develop it. There is some debate around the validity of this construction. Shona LC sourced most of its content from news websites like *Kwayedza*. That the writers of such a newspaper use this construction is telling. At the very least it calls for additional consideration for the validity of this phoneme. The grammar book that was the key reference for this project aims to be a descriptive rather than a prescriptive one. The authors state that their intention is to describe the language as it is used rather than how it ought to be. It therefore seems fair to assume that faced with language data which shows such a high prevalence of this phoneme's use, they would incorporate it into the current orthography. The key issue for the designers of spell checkers for Shona and other SBLs is to have a close working relationship with grammarians and lexicographers to inform the choice of constructions that can be built into the language models.

6.5. Implications for the development of spell checkers for CWSBLs

One question that can be asked from the results of the experiments is whether it is worthwhile investing the time and energy required to build a hand-written morphological analyser if it performs so poorly on error recall? This is a fair question. For the time and effort required to build it, CTLM does reasonably well. In fact, its performance confirms the conclusions of Ndaba et al that building spell checkers for all CWSBL is now feasible. This begs the second question: When is it worthwhile to invest the time and effort required to build a model like MAShoKO? The answer to this and the previous question can be found in reviewing the instances that MASHoKO outperforms CTLM.

The Error Precision of MAShoKO on authentic Shona words is significantly higher than that of CTLM. This means that it is better able to tell incorrect words than the CTLM model – correctly identifying 3 out of every 4 wrong words while CTLM only manages just over 2 in every 3. Considering the difference between the manner in which the two spell checkers determine the correctness of a word that they have never seen before, this suggests that the MAShoKO model does perform better than CTLM on words that it did not previously see. Unsurprisingly, MAShoKO does not do well with word types that were not built into it. This means that the performance recorded here is not the peak performance of MAShoko. Additional rules will further enhance its performance, further widening the gap between it and that of CTLM.

6.6. Closing Comments on Morphological Analyser based spell checkers

MAShoKO demonstrated that it is possible to build a morphological analyser-based spell checker for a CWSBL. The performance of such a spell checker is largely influenced by the rules that it implements. The system's performance on word types that it was not designed to accept will be poor and this can tarnish its image.

6.7. Chapter Summary

This chapter discusses the results of the experiments that were carried out to evaluate the performance of MAShoKO against the CTLM based spell checker. The metrics used to evaluate the two spell checkers are presented, followed by an analysis of what these results mean for the research question.

"Without analysis, no synthesis"

Friedrich Engels

Chapter 7- Summary and Conclusion

7.1. Introduction

This chapter is a review of the research project. It starts by revisiting the research question before sketching the literature review process. After this the design and review of the experiments that were carried out to address the research question is briefly discussed. Finally, some views on possible future directions of this work are given.

7.2. Overview

The motivation for embarking on this research was the observation that there is no extant, useful, and widely available spell checker for Shona. Furthermore, even though there are spell checkers for other related languages, it did not appear as if they fully addressed the challenges that arose from the nature of the writing system used for Shona which lead to NLP systems encountering many new words in real world usage. As a result, this study sought to find out the ways that other researchers have utilised to improve the performance of their spell checkers on these OOV words. It also aimed at developing new methods that would be optimised to perform well on OOV words. Specifically, it would develop morphological analysis-based methods to address this question.

7.3. Review of Previous Approaches

A systematic literature review following the RAMESES meta-narrative review protocol was conducted to evaluate the ways in which previous research projects have addressed the question of OOV words in spell checkers for CWALs. It was found that there have been three broad themes in this space. The first broad theme has been to handle spell checking as dictionary lookup. Within this paradigm, the way in which OOV words have been addressed has been through the enhancement of dictionary sizes using several synthetic word generation approaches. The first of these was the development of morphological analysers to generate new words and then add them to a static dictionary which would then be utilised in the final spell checker. Linguists evaluated the resultant words to ensure that only valid words were added to the dictionaries. Spell checkers designed using

this method performed much better than those that did not have these additional words appended to them. The second theme is similar in nature. However, instead of using morphological analysers to generate the words, these appended new words by applying rules about likely words based on existing words in the dictionary. They also managed to get modest results using this approach. The final and most recent approach was that of using a CTLM to develop a data driven spell checker. Work reported in this work has shown that these perform as well as the rules-based approaches.

7.4. MAShoKO based Hybrid Spell checker

In this research, a Morphological Analyser (MA) for Shona verbs, nouns and some of the genitive nouns was developed. This MA was incorporated into a Hybrid spell checker which uses three modalities to perform spell checking on Shona words. First, it does a dictionary check up. If this fails to find a match, it checks to see if the word has valid syllables, flagging any that do not as incorrect. Finally, it performs a morphological analysis of the words. This process is computationally expensive as it involves several searches. As a result it is only performed on those words that have failed the last two checks.

An experiment to evaluate the performance of MAShoKO against the CTLM based spell checker was conducted. The two spell checkers were trained and/or developed using words obtained from the DGC. After this, they were then presented with the words obtained from the 100k Shona LC. The performance of both spell checkers on OOV as well as the full word lists were tabulated and compared. It was found that the MAShoKO based spell checker did much better than the CTLM spell checker on both sets of words. Specifically, on OOV, it had fewer false positives and false negatives. A key challenge was with borrowed words as well as a number of noun modifications which are not programmed into the FST.

7.5. Possible Future Directions

The research carried out in this thesis has demonstrated that spell checkers based on morphological analysis of CWAL perform better than character n-grams. This morphological analysis requires the manual codification of the language's grammar, a process that is painstakingly slow and demanding of the developers. The data driven paradigm is increasingly preferred for the development of NLP systems. A key question to consider is whether the morphological analyser can be developed in a data driven manner using either some unsupervised or a degree of supervised learning. One way which this can be done is if the morphological analyser that was developed for this project is extended to cover

all parts of speech. Once this is completed, it can then be used to create a training corpus for a data driven morphological analyser by providing labelled data.

A second potential future research direction is to evaluate the performance of these methods to perform context sensitive or real world error detection. Without having conducted any analysis on this question, it appears as if CTLM based spell checkers would fair even worse on this task as they are incapable of maintaining longer term dependencies that would be required to check aspects such as concordial agreement between nouns and other parts of speech in a sentence.

7.6. Conclusion

This research had eight objectives which were stated in Chapter 1. With respect to the first objective, a review of the literature on the spell checking of CWSBLs revealed a paucity in research that specifically addressed the question of OoV words. None of the previous research specifically addressed the question of how to handle them. Neither did any of them attempt to maximise the performance of their spell checkers on such words. This was not due to any lack of awareness of the problem. Rather, previous approaches aimed to reduce the incidence of such words by increasing the sizes of their dictionaries.

Given the above finding, the answer to the second research question, which is tied to the second objective is that there were no approaches found to have been used to achieve this aim.

The third objective of this research was to develop a morphological analyser for Shona verbs and nouns. The approach chosen to design such a morphological analyser was that of utilising a Finite State Transducer, based on a Finite State Machine. Utilising textbooks of Shona Grammar and the author's knowledge of the language, a Morphological Analyser for Shona, dubbed MAShoKO was designed.

The SCaMA framework was utilised to design a spell checker for Shona. This design has three main constituent components. First it has a syllabic checker whose task is to confirm if a given words consists of valid Shona syllables. Any words that have illegal syllables are by definition invalid Shona words and are flagged as such. Those words that pass this initial test are then subjected to a simple dictionary search. Any words that do not exist within the dictionary are then parsed by the morphological analyser. Words that the morphological analyser fails to identify are also flagged as

invalid. The last component of the spell checker is the suggestion engine, which generates words that are within one edit distance of the incorrectly spelt word.

The fifth and sixth research objectives were met by implementing MAShoKO and a MAShoKO based spell checker in Python. Two versions of the spell checker were developed. First a RESTful API to serve MAShoKO was developed using the Flask framework. Second a comparison application which incorporated the MAShoKO spell checking engine and the character trigram based spell checking engine used in previous research in order to compare their performance was developed. This GUI application is based on the TKinter framework.

Utilising the comparison application mentioned above, experiments to compare the performance of MAShoKO and the CTLM based spell checkers were conducted. The first one compared the performance of the two on all words in the LC. On this comparison, the performance of both spell checkers was marginally different.

The second comparison considered the eighth research objective and sixth research question pertaining to the performance of the morphological analyser on OOV words. When compared to the CTLM model, the MAShoKO model did significantly better on OoV words – especially on verbs and nouns. This demonstrated that morphological analysis is an effective way for improving the performance of a spell checker on OoV words for CWSBLs.

In summary, all eight objectives of this research were satisfied. The challenges that can be encountered in the development of spell checkers that aim to maximise correct identification of OOV words in SBLs, as well as the previous approaches used for these languages are now understood. A morphological analyser for Shona was developed and it was used to build a spell checker whose performance was evaluated and demonstrated to be superior to that of a previously developed CTLM based approach.

References

Ahmadi, S. (2021). Hunspell for Sorani Kurdish Spell Checking and Morphological Analysis. *arXiv preprint arXiv:2109.06374*.

Aikhenvald, A. Y. (2007). *Typological dimensions in word-formation.* Cambridge University Press.

Amaral, J. N. (2011). *About computing science research methodology*.

Anderson, S. R. (2015). The morpheme: Its nature and use. *The Oxford handbook of inflection*, 11-33.

Bakovic, E. (2003). *Vowel harmony and stem identity*.

Bauer, L. (2008). "Derivational morphology. *Language and linguistics compass*, *2*(1), 196-210.

Bejan, C. (2017). *English Words: Structure, Origin and Meaning: a Linguistic Introduction.* Addleton Academic Publishers.

Bilandzic, M., & Venable, J. (2011). Towards participatory action design research: adapting action research and design science research methods for urban informatics. *Journal of Community Informatics, 7*(3).

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., Arx, S. v., . . . et. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Bonami, O., Boyé, G., Dal, G., Giraudo, H., & Namer, F. (2018). *The lexeme in descriptive and theoretical morphology*. Language Science Press.

Bosch, S. E., & Eiselen, R. (2005). The effectiveness of morphological rules for an isiZulu spelling checker. *South African Journal of African Languages*, *25*(1), 5-36. doi:10.1080/02572117.2005.10587246

Bradbury, H. (Ed.). (2015). *The Sage handbook of action research.* . Sage.

Burchfield, R. (1985). Frequency Analysis of English Usage: Lexicon and Grammar. By W. Nelson Francis and Henry Kučera with the assistance of Andrew W. Mackie. Boston: Houghton Mifflin. 1982. x+ 561. *Journal of English Linguistics, 18*(1), 64-70.

Chau, E. C., & Smith, N. A. (2021). Specializing Multilingual Language Models: An Empirical Study. *arXiv preprint arXiv:2106.09063*.

Chimhundu, H. (Ed.). (2001). *Duramazwi guru reChiShona*. College Press in conjunction with the African Languages Research Institute, University of Zimbabwe.

Ching, K. L. (2018). Tools Matter: Mediated Writing Activity in Alternative Digital Environments. *Written Communications, 35*(3), 344-75.

Creutz, M., Hirsimäki, T., Kurimo, M., Puurula, A., Pylkkönen, J., Siivola, V. V., . . . Stolcke, A. (2007). Morph-based speech recognition and modeling of out-of-vocabulary words across languages. *ACM Transactions on Speech and Language Processing (TSLP), 5*(1), 1-29.

Damerau, F. J. (1964). "A technique for computer detection and correction of spelling errors. ″ *Communications of the ACM , 7*(3), pp. 171-176.

Damian, A. J., Robinson, S., Manzoor, F., Lamb, M., Rojas, A., Porto, A., & Anderson., D. (2020). A mixed methods evaluation of the feasibility, acceptability, and impact of a pilot project ECHO for community health workers (CHWs). *Pilot and feasibility studies*, 1-11.

David M, E., Simons, G. F., & Fennig, C. D. (Eds.). (2021). *Ethnologue: Languages of the World. Twenty-fourth edition.* Dallas, Texas: SIL International. Online version: http://www.ethnologue.com.

de Bruijn, M., & Brinkman, I. (2018). Mobile Phone Communication in the Mobile Margins of Africa: The ‘Communication Revolution’Evaluated from Below. In *The Palgrave Handbook of Media and Communication Research in Africa* (pp. 225-241). Cham: Palgrave Macmillan.

de Schryver, G.-M., & Prinsloo, D. (2004). Spellcheckers for the South African languages, Part 1: The status quo and options for improvement. *South African Journal of African Languages, 24*(1), 57-82.

De Varennes, F. (2017). "Language Rights as an Integral Part of Human Rights–A Legal Perspective." . In *Democracy and Human Rights in Multicultural Societies* (pp. 115-125). Routledge.

Dixon, R. M. (1977). Some Phonological Rules in Yidin^\rmy.". *Linguistic Inquiry, 8*(1), 1-34.

Doddapaneni, S., Ramesh, G., Kunchukuttan, A., Kumar, P., & Khapra, M. M. (2021). A primer on pretrained multilingual language models. *arXiv preprint arXiv:2107.00676*.

Dodig-Crnkovic, G. (2002). Scientific methods in computer science. In *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia,* (pp. 126-130).

Doke, C. M. (2005). *Report on the Unification of the Shona Dialects: a Photographic Reprint with an Introduction by Herbert Chimhundu.* The Allex Project.

Drake, L. (2019). Frictionless Technologies: The Innovation of Human Obsolescence. *Biennial Conference of the Society for Philosophy and Technology*. Texas. Retrieved from https://posthumanity.ai/wp-content/uploads/2019/06/Laura-Drake-Frictionless-Technologies-Human-Obsolescence-2019.pdf

Ducrot, O., & Todorov, T. (1972). *Dictionnaire encyclopédique des sciences du langage.* FeniXX.

Eifring, H., & Theil, R. (2015). Linguistics for Students of Asian and African Languages. 2005: 3. *Institutt for osteuropeiske og orientalske studier. Web , 5*.

Enderby, J. L., Carroll, J. M., Tarczynski-Bowles, M. L., & Breadmore, H. L. (2021). The roles of morphology, phonology, and prosody in reading and spelling multisyllabic words. *Applied Psycholinguistics*, 1-21.

Fischhoff, B. (2013). The sciences of science communication. *Proceedings of the National Academy of Sciences, 110*(Supplement 3), 14033-14039.

Fishman, J. A. (1991). *Reversing language shift: Theoretical and empirical foundations of assistance to threatened languages* (Vol. 76). Multilinguual Matters.

Fortune, G. (1985). *Shona Grammatical Constructions: Volume 1*. Mercury Press.

Frischmann, B. a. (2016). Utopia: A Technologically Determined World of Frictionless Transactions, Optimized Production, and Maximal Happiness. *UCLA L. Rev. Discourse , 64*, 372.

Gerz, D., Vulić, I., Ponti, E. M., Reichart, R., & Korhonen, A. (2018). On the Relation between Linguistic Typology and (Limitations of) Multilingual Language Modeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (pp. 316-327).

Gezmu, A. M., Nürnberger, A., & Seyoum, B. E. (2018). Portable spelling corrector for a less-resourced language: Amharic. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018).*

Goetze, T. S., & Abramson, D. (2021). Bigger Isn't Better: The Ethical and Scientific Vices of Extra-Large Datasets in Language Models. In *13th ACM Web Science Conference 2021* (pp. 69-75).

Goldhahn, D., Eckart, T., & Quasthoff, U. (2012). Building Large Monolingual Dictionaries at the Leipzig Corpora Collection: From 100 to 200 Languages. In *LREC* (Vol. 29, pp. 31-43).

Greenberg, J. H. (1960). A quantitative approach to the morphological typology of language. *International journal of American linguistics, 26*(3), 178-194.

Grefenstette, G., & Tapanainen., P. (1994). What is a word, what is a sentence?: problems of Tokenisation.

Grobbelaar, L. A., & Kinyua, J. D. (2009, June 29). A spell checker and corrector for the native South African language, South Sotho. *Proceedings of the 2009 Annual Conference of the Southern*

African Computer Lecturers' Association (pp. 50–59). New York, NY, USA: Association for Computing Machinery. doi:10.1145/1562741.1562747

Grønvik, O. (1996). *THE FOURTH ALLEX WORKSHOP UNIVERSITY OF ZIMBABWE.*

Grønvik, O., & Chimhundu, H. (1998). *Annual report for the Allex project in 1998.* African Languages Research Institute, University of Zimbabwe.

Grover, A. S., Van Huyssteen, G. B., & Pretorius, M. W. (2010). South African human language technologies audit.

GÜNGÖR, O., GÜNGÖR, T., & ÜSKÜDARLI, S. (2019). .The effect of morphology in named entity recognition with sequence tagging. *Natural Language Engineering, 25*(1), 147-169. doi:doi:10.1017/S1351324918000281

Haspelmath, M. (2011). "On S, A, P, T, and R as comparative concepts for alignment typology.".

Hendrikse, R., & Mfusi, M. (2011). Circumfixes as emergent linguistic structures. *South African Journal of African Languages, 31*(1), 41-53.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 75-105.

Hildebrandt, K. A. (2015). The prosodic word. *The Oxford handbook of the word*, 221-245.

Himoro, M. Y. (2020). Towards a Spell Checker for Zamboanga Chavacano Orthography. In *Proceedings of the 12th Language Resources and Evaluation Conference* (pp. 2685-2697).

Janson, T. (1991-92). Southern Bantu and Makua. (R. K. Verlag, Ed.) *prache und Geschichte in Afrika, 12/13*, pp. 63-106.

Jones, J., Podile, K., & Puttkammer, M. (2005). Challenges relating to standardization in the development of an isiXhosa spelling checker. *25*, 1-10. doi:10.1080/02572117.2005.10587244

Joshi, P., Santy, S., Budhiraja, A., Bali, K., & Choudhury, M. (2020). The state and fate of linguistic diversity and inclusion in the NLP world. *arXiv preprint arXiv:2004.09095*.

Kashyap, A. K. (2019). Language Typology. *The Cambridge handbook of systemic functional linguistics* , 767-792.

Keet, C. M., & Khumalo, L. (2017, December). Evaluation of the effects of a spellchecker on the intellectualization of isiZulu. Retrieved from http://pubs.cs.uct.ac.za/archive/00001233/; http://pubs.cs.uct.ac.za/archive/00001233/01/KK17alternationSpellcheck.pdf

Khumalo, L. 2. (2017). The Design and Implementation of a Corpus Management System for the isiZulu National Corpus. In *In Abstracts of the 22nd International Conference of the African Association for Lexicography. Conference of the Language Associations of Soutern Africa (CLASA). Rhodes University, Grahamstown, South Africa* (pp. 26-29).

Koskenniemi, K. (1984). A general computational model for word-form recognition and production. In *Proceedings of the 4th Nordic Conference of Computational Linguistics (NODALIDA 1983)* (pp. 145-154).

Krause, T. A. (2012). Multiple tokenizations in a diachronic corpus. In *Exploring Ancient Languages through Corpora Conference (EALC)* (Vol. 14).

Kukich, K. (1992). Techniques for automatically correcting words in text. *Acm Computing Surveys (CSUR) , 24*(4), pp. 377-439.

Lebeaupin, B., Rauzy, A., & Roussel, J.-M. (2017). A language proposition for system requirements. In *2017 Annual IEEE International Systems Conference (SysCon)* (pp. 1-8). IEEE.

Leech, N. L., & Onwuegbuzie, A. J. (2010). Guidelines for conducting and reporting mixed research in the field of counseling and beyond. *Journal of Counseling & Development, 88*(1), 61-69.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady, 10*(8), pp. 707-710.

Mabuya, R., Ramukhadi, P., Setaka, M., Wagner, V., & Zaanen, M. v. (Eds.). (2020). Proceedings of the First Workshop on Resources for African Indigenous Languages. *Proceedings of the First Workshop on Resources for African Indigenous Languages.*

Magwa, W. (2008). Language harmonization in Southern Africa: toward a standard unified Shona orthography (SUSO) for Botswana, Mozambique and Zimbabwe. *Dyke (Gweru, Zimbabwe), 3*(2), 60-76.

Martini, I. D. (2016). Derivational of bound morpheme. *International Research Journal of Management, IT and Social Sciences, 3*(1), 15-22.

Marzi, C., Blevins, J. P., Booij, G., & Pirrelli, V. (2020). Inflection at the morphology-syntax interface. In *In Word Knowledge and Word Usage* (pp. 228-294). De Gruyter Mouton.

Mberi, N. E. (2006). *The categorical status and functions of auxiliaries in Shona.* ALLEX Project, African Languages Research Institute, University of Zimbabwe.

Milambiling, J. (2018). The Universal Declaration of Linguistic Rights. In *Language and Social Justice in Practice* (pp. 208-216). Routledge.

Miti, L. (2006). *Comparative Bantu Morphology and Phonology."* . Cape Town: The Centre for Advanced Studies for African Society.

Mjaria, F., & Keet, C. M. (2018). A Statistical Approach to Error Correction for isiZulu Spellcheckers. (pp. 1 of 9–9 of 9). Gaborone: IEEE.

Mohammed, N., & Abdellah, Y. (2018). The vocabulary and the morphology in spell checker. *Procedia Computer Science, 127*, 76-81.

Moors, C., Wilken, I., Calteaux, K., & Gumede, T. (2018). Human language technology audit 2018: Analysing the development trends in resource availability in all South African languages. *In Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologiests*, (pp. 296-304).

Mosel, U. (n.d.). *A critical analysis of current definitions of lexeme and related linguistic terms.*

Mpofu, N. .. (2007). The User Perspective in Lexicography: The Lemmatisation of Fixed Expressions in" Duramazwi Guru reChiShona. *Lexikos, 17*.

Mpofu, N., Ngunga, A., Mberi, N. E., & Matambirofa., F. (2013). *A descriptive grammar of Shona.* CROBOL Project.

Mukherjee, S. P. (2019). *A guide to research methodology: An overview of research problems, tasks and methods.* CRC Press.

Ndaba, B., Suleman, H., Keet, C. M., & Khumalo, L. (2016, January). The Effects of a Corpus on isiZulu Spellcheckers based on N-grams. Retrieved from http://pubs.cs.uct.ac.za/archive/00001084/; http://pubs.cs.uct.ac.za/archive/00001084/01/afrispeIST16crc.pdf

Neubig, G., Shruti, R., Alexis, P., MacKenzie, J., Li, H. C., Lee, M., & al, e. (2020). *A Summart of the First Workshop on Language Technology for Language Documentation and Revitalization.* arXiv[cs.CL]. Retrieved from http://arxiv.org/abs/2004.13203

Nijat, M., Hamdulla, A., & Tuerxun, P. (2019). The Methods for Reducing the Number of OOVs in Chinese-Uyghur NMT System. *In International CCF Conference on Artificial Intelligence*, (pp. 183-195). Singapore: Springer.

Nunamaker Jr, J. F., Chen, M., & Purdin, T. D. (1990). Systems development in information systems research. *Journal of management information systems*, *7*(3), 89-106.

Packard, J. L. (2000). *The morphology of Chinese: A linguistic and cognitive approach.* Cambridge University Press.

Paggio, P. (2000). Spelling and grammar correction for Danish in SCARRIE. In *Sixth Applied Natural Language Processing Conference* (pp. 255-261).

Park, H. H., Zhang, K. J., Haley, C., Steimel, K., Liu, H., & Schwartz, L. (2021). Morphology Matters: A Multilingual Language Modeling Analysis. *Transactions of the Association for Computational Linguistics 9*, *9*, 261-276.

Peffers, K. T. (2007). A design science research methodology for information systems research. *Journal of management information systems*, *24*(3), 45-77.

Peffers, K., Tuunanen, T., & Niehaves, B. (2018). *Design science research genres: introduction to the special issue on exemplars and criteria for applicable design science research.*

Piantadosi, S. T., Tily, H., & Gibson., E. (2012). "The communicative function of ambiguity in language.". *Cognition , 122*(3), 280-291.

Pirinen, T. A. (2014). *Weighted Finite-State Methods for Spell-Checking and Correction.* Helsingin yliopisto.

Powers, D. M. (2014). *What the F--measure doesn't measure….* Technical report, Beijing University of Technology China & Flinders University, Australia.

Pries-Heje, J. R. (2007). Soft design science research: Extending the boundaries of evaluation in design science research. In *Proceedings from the 2nd International Conference on Design Science Research in IT (DESRIST)* (pp. 18-38).

Prinsloo D., J., & de Schryver, G.-M. (2003). Non-word error detection in current South African spellcheckers. *21*, 307-326. doi:10.2989/16073610309486351

Prinsloo, D. J., & Eiselen, R. (2005). Improving a lexicon-based spelling checker for Sesotho sa Leboa. *25*, 11-24. doi:10.1080/02572117.2005.10587245

Prinsloo, D., & Schryver., G.-M. d. (2004). Spellcheckers for the South African Languages, Part 2: The Utilisation of Clusters of Circumfixes. *South African Journal of African Languages, 24*(1), 83-94.

Ralph, M. L., & Lambon, R. M. (2001). Lexical processes (word knowledge): psychological and neural aspects. In *The International Encyclopaedia of Social and Behavioral Sciences.* Elsevier BV.

Sapir, E. (1921). An introduction to the study of speech. *Language, 1*.

Schcolnik, M. (2018). Digital Tools in Academic Writing. *Journal of Academic Writing, 8*(1), 121-30.

Schütze, H. (1992). Word space. *Advances in neural information processing systems, 5*.

Seidenberg, M. S., & Gonnerman, L. M. (2000). Explaining derivational morphology as the convergence of codes. *Trends in Cognitive Sciences, 4*(9), pp. 353-361. doi:https://doi.org/10.1016/S1364-6613(00)01515-1

Sein, M. K., Henfridsson, O., Purao, S., Rossi, M., & Lindgren, R. (2011). Action design research. *MIS quarterly*, 37-56.

Svenonius, P. (2018). Delimiting the syntactic word. *Linguistics at Santa Cruz*.

Tariq, T. R., Rana, M. A., Sultan, B., Asif, M., Rafique, N., & Aleem, S. (2020). An Analysis of Derivational and Inflectional Morphemes. *International Journal of Linguistics, 12*(1), 83.

Twenge, J. M., & Spitzberg., B. H. (2020). Declines in non-digital social interaction among Americans, 2003–2017. *Journal of Applied Social Psychology, 50*(6), 363-367.

Umar, A. U. (2020). Infixes and Infixation Processes in Hausa Morphology.

Uszkoreit, H. (2000). Language Technology: A First Overview. *German Research Center for Artificial Intelligence, Saarbrücken*.

Vaishnavi, V. K. (2007). *Design science research methods and patterns: innovating information and communication technology*. Auerbach Publications.

Vaishnavi, V. K., & Kuechler, W. (2015). *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. CRC Press.

Venter, E. (2019). Challenges for meaningful interpersonal communication in a digital era. *HTS: Theological Studies, 75*(1), 1-6.

Vertanen, K., Gaines, D., Fletcher, C., Stanage, A. M., Watling, R., & Kristensson, P. O. (2019). VelociWatch: Designing and evaluating a virtual keyboard for the input of challenging text. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (pp. 1-14).

Vincent, R. (2019). *A study of different data-driven and knowledge-based approaches for human language technologies*.

Warfield, D. (2010). IS/IT RESEARCH: A RESEARCH METHODOLOGIES REVIEW. *Journal of Theoretical & Applied Information Technology, 13*.

Wong, G., Greenhalgh, T., Westhorp, G., Buckingham, J., & Pawson, R. (2013). RAMESES publication standards: Meta-narrative reviews. *Journal of Advanced Nursing, 69*(5), 987-1004.

Yang, Z., Zhu, C., Sachidananda, V., & Darve., E. (2019). Out-of-Vocabulary Embedding Imputation with Grounded Language Information by Graph Convolutional Networks. *arXiv preprint arXiv:1906.03753*.

Yunus, A., & Masum, M. (2020). A Context Free Spell Correction Method using Supervised Machine Learning Algorithms. *International Journal of Computer Applications, 176*(27), 36-41.

Appendix 1 – Code Listings

Listing 1 - Finite State Automata – using Bernd Klein's code

```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 10 23:09:56 2020

@author: Farayi Kambarami
"""

#This is the finite State Automaton based on Bernd Klein's implementation
class StateMachine:
    def __init__(self):
        self.handlers = {}
        self.startState = None
        self.endStates = []

    def add_state(self, name, handler, end_state=0):
        name = name.upper()
        self.handlers[name] = handler
        if end_state:
            self.endStates.append(name)

    def set_start(self, name):
        self.startState = name.upper()

    def run(self, cargo):
        self.__init__()
        result = False
        try:
            handler = self.handlers[self.startState]
        except:
            raise InitializationError("must call .set_start() before .run()")
        if not self.endStates:
            raise  InitializationError("at least one state must be an end_state")

        while True:
            (newState, cargo) = handler(cargo)
            if newState.upper() in self.endStates:
                #print(cargo[2])
                if newState.upper() == "error_state".upper():
                    result = False
                    break
                else:
                    result =True
                    return result
                    break
            else:
                handler = self.handlers[newState.upper()]
                result = False
        return result
```

Listing 2 - ShonaVerb.py : Morphological Analyser Shona Verbs

```
# -*- coding: utf-8 -*-
"""
This code implements a morphological analyser for Shona Verbs

"""

negationOrmood ={"i","ha","nga"}
subjectConcords = {"ndi","nda",
"ti","ta","u","wa","mu","ma","a","va","i","ya","ri","ra","a","chi","cha","zvi","zva","i","dzi","dza","ru"
,"ra","rwu","rwa","ka","twu","twa","tu","hu","hwu","hwa","ku","kwa","pa","ku","mu","svi","sva","k
wu","kwa","ku","zi","za"} # "mwa"
tenseMarkers_slot1 = {"i","cha","no"}
negationSlot1 = {"si","sa"}
tenseMarkers_slot2={"chi","ka","do","ne","nga"}
negationSlot2 = {"si","ka","za", "sa"} # The last "sa" does not appear in Dr Mberi's slot system - to
validate with him
#negationSlot3 = {"si","sa"}
tenseMarkers_slot3 = {"chi","ka","zo"}
auxilliarySlot = {"ndo","mbo","ngo","zo","fum"}
objectConcords =
{"ndi","ti","ku","mu","mu","mu","va","u","i","ri","a","chi","zvi","i","dzi","ru","ru","ka","twu","tu","h
u","vu","ku","pa","ku","mu","svi","ku","ri"}
validSyllables = {"a","e","i","o","u"
,"b","bh","bw","bv","ch","d","dh","dy","dz","dzv","dzw","f","g","gw","h","hw","j","k","kw","m","m
b","mw","mv","mbw","mh","n","n'","n’","nd","ndy","ndw","ng","ngw","nj","njw","nh","nhw","nw","
ny","nz","nzv","nzvw","nzw","p","pf","pw","r","rw","s","sh","shw","sw","sv","svw","t","ts","tsw","ts
v","tsvw","ty","tw","v","vh","w","y","z","zh","zhw","zv","zw","zvw"}
vowels ={"a","e","i","o","u"}
genitive_prefixes ={"ne","wa", "we","wo", "va","ve","vo","ya","ye","yo", "ra", "re","ro",
"a","e","cha","che","cho","zva","zve","zvo","dza","dze","dzo","rwa","rwe","rwo","ka","ke","ko","twa
","twe","two","hwa","hwe","hwo","kwa","kwe","kwo","pa","pe","po","ma","me","mo","sva","sve","s
vo"}

from StateMachine import StateMachine # as sm

class ShonaVerb(StateMachine):

    structure = []

    def __init__(self):
        StateMachine.__init__(self)
        #shonaVerbFSA = StateMachine()
        StateMachine.add_state(self,"Start", self.start_transitions)
        StateMachine.add_state(self,"Negation_or_mood_state",self.negation_or_mood_transitions)
        StateMachine.add_state(self,"Subject_Concord_state",self.subject_concord_state_transitions)

StateMachine.add_state(self,"Tense_Markers_Slot1_State",self.tense_markers_slot1_state_transitions
)
        StateMachine.add_state(self,"Negation_Slot1_State",self.negation_slot_4_transitions)
        StateMachine.add_state(self,"Object_Concord_state",self.object_concord_transitions)
```

```
StateMachine.add_state(self,"Tense_Markers_Slot2_State",self.tense_marker_slot2_state_transitions)

StateMachine.add_state(self,"Tense_Markers_Slot3_State",self.tense_marker_slot3_state_transitions)

        StateMachine.add_state(self,"Auxilliary_state",self.auxilliary_state_transitions)

        StateMachine.add_state(self,"Negation_Slot6_State",self.negation_slot_6_transitions)
        StateMachine.add_state(self,"Root_State",self.root_state_transitions)
        StateMachine.add_state(self,"End_State",self.end_state_transitions, end_state=1)
        StateMachine.add_state(self,"error_state", None, end_state=1)

        StateMachine.add_state(self,"genitive_prefixes_state",self.genitive_transitions)

        StateMachine.set_start(self,"Start")
        self.structure = []
        #return ShonaVerbFSA


    def openStemDictionary(self):
        with open('dictionaries/madzitsi_eduramazwi.txt') as f:
            lines = f.read().splitlines()
        return lines

    def openDictionary(self):
        with open('dictionaries/Shona_words.txt') as f:  #'DGR_vocab.txt') as f: # mazwi_eduramazwi
            words_dict = f.read().splitlines()
        return words_dict

    def syllabify(self,txt):
        syllables = []
        firstsyllable, restoftxt = self.splitfirstsyllable(txt)
        while len(restoftxt) > 0:
            syllables.append(firstsyllable)
            firstsyllable, restoftxt = self.splitfirstsyllable(restoftxt)
            ##print(firstsyllable)
        if len(firstsyllable) > 0:
            syllables.append(firstsyllable + txt[-1:])
        return syllables

    def hasValidSyllables(self,txt):
        itdoes = True
        chktxt = txt.lower()
        syllables = self.syllabify(chktxt)

        for syllable in syllables:
            if syllable[:-1] in validSyllables:
                #flash('{} is a valid syllable.'.format(syllable))
                itdoes= itdoes
            elif syllable in vowels:
                #flash('{} is a valid syllable.'.format(syllable))
                itdoes= itdoes
            else:
                #flash('{} is not a valid syllable'.format(syllable))
                itdoes= False
```

```
        #flash('{} has been checked and the statement of whether it  has valid syllables has been found to be {}'.format(txt,itdoes))
        return itdoes

    def hasValidSubwords(self,txt):
        word_list = self.openDictionary()
        found = False
        restoftxt = txt
        while (not found) and (len(restoftxt) > 2):
            firstsyllable, restoftxt = self.splitfirstsyllable(restoftxt)
            #flash('Split into <{}> and <{}>.'.format(firstsyllable, restoftxt))
            subword = restoftxt
            if subword in word_list:
                #flash('{} is a valid subword'.format(subword))
                found = True
            else:
                found=False
        #flash('{} has been checked and the statement of whether it has valid subwords has been found to be {}'.format(txt,found))
        return found

    def hasValidStem(self,txt):
        #words_list = sv.openDictionary()
        stem_list = self.openStemDictionary()
        stem = "-" + txt
        ##print("stem is " + stem)
        found = stem in stem_list
        restoftxt = txt
        if not found:
            while (not found) and (len(restoftxt) > 2):
                firstsyllable, restoftxt = self.splitfirstsyllable(restoftxt)
                #flash('Split into <{}> and <{}>.'.format(firstsyllable, restoftxt))
                stem = restoftxt[:-1]
                stem = "-" + stem + "a"
                if stem in stem_list:
                    #flash('{} is a valid stem'.format(stem))
                    found = True
                else:
                    found=False
        #flash('{} has been checked and the statement of whether it has valid stems has been found to be {}'.format(txt,found))
        return found

    def presumedStem(self,txt):
        #words_list = sv.openDictionary()
        stem_list = self.openStemDictionary()
        stem = "-" + txt
        ##print("stem is " + stem)
        found = stem in stem_list
        restoftxt = txt
        if not found:
            while (not found) and (len(restoftxt) > 2):
                firstsyllable, restoftxt = self.splitfirstsyllable(restoftxt)
                #flash('Split into <{}> and <{}>.'.format(firstsyllable, restoftxt))
                stem = restoftxt[:-1]
```

```
            stem = "-" + stem + "a"
            if stem in stem_list:
                #flash('{} is a valid stem'.format(stem))
                found = True
            else:
                found=False
                stem = " #"
        #flash('{} has been checked and the statement of whether it has valid stems has been found to be {}'.format(txt,found))
        return stem[1:]


    def roteSpellCheck(self,txt):
        words_list = self.openDictionary()
        if txt in words_list:
            #print('Word {} found in our our dictionary'.format(txt))
            found = True
        else:
            if (self.hasValidStem(txt) or self.hasValidSubwords(txt)) and self.hasValidSyllables(txt):
                found = True
            else:
                found = False
            #if not found:
              #print('The word {} is not a known word.'.format(txt))
        return_val = found
        return return_val


    """
        This function splits a Shona word or the remaining part of the word into two components. The first part is the
        first syllable in the word, or word fragmanent and the second part is the remainder of the word or word segment.
        A syllable in Shona is defined as being composed of either a single vowel or a number of consonants followed by a
        vowel.
        This function does not attempt to check for the validity of the syllables returned - by validity here, we refer to
        whether the syllable conforms with the official shona orthography or not.

    """
    def splitfirstsyllable(self,txt):
        firstSyllable=""
        restOftext=""
        vowels={"a","e","i","o","u"}
        if len(txt) ==1 :
            firstSyllable=txt
        else:
            if len(txt) > 0:
                startChar=txt[0]
                ##print("startChar is : " +startChar )
                if startChar in vowels:
                    firstSyllable = startChar
                    restOftext = txt[1:]
                else:
```

```
            i=1
            nextChar=txt[i]
            firstSyllable=startChar
            while (nextChar not in vowels) and ( (i+1) < len(txt)):
               firstSyllable= firstSyllable+nextChar
               i+=1
               nextChar=txt[i]
            if (i+1) != len(txt):
               firstSyllable=firstSyllable+nextChar
               restOftext = txt[len(firstSyllable):]
   return firstSyllable, restOftext



"""
   This function returns the verb root and a list of all the extensions that have been applied to it.
   It does this by searching for the known verb extensions from left to right in each verb.
   A known issue is that this greedy process finds stems even in words that do not have them.

"""
def getExtension(self,txt):
   extensions=[]
   root = txt
   if txt[-4:] in ["erer","oror","urur","inur","enur","onor","arar"]:
      root=root[:-4]
      extensions.append(txt[-4:])
   elif txt[-3:] in ["idz","zvi"]:
      root=root[:-3]
      extensions.append(txt[-3:])
   elif txt[-2:] in ["is", "es","iw","ew","an","ir","er","ik","ek","ek","at","am","ar"]:
      root=root[:-2]
      extensions.append(txt[-2:])
   elif txt[-1:] in ["w"]:
      root=root[:-1]
      extensions.append(txt[-1:])
   return root, extensions


"""

   This function labels the extensions presented to it

"""

def extensionType(self,txt):
   if txt in ("is","es"):
      extnType =("Causative or Intensive/ Yesakiso kana Yenyanyiso")
   elif txt in ("idz"):
      extnType="Causative/ Yesakiso"
   elif txt in ("w","iw","ew"):
      extnType = "Passive/ Yokuitwa"
   elif txt in ("an"):
      extnType = "Reciprocal|Associative/ Yokuitirana|Yokubatana"
   elif txt in ("ir","er"):
      extnType = "Applied|Benefective/ Yokuitira"
```

```
        elif txt in ("erer","oror","urur"):
            extnType = "Repetitive"
        elif txt in ("inur","enur","onor"):
            extnType = "Reversive"
        elif txt in ("ik","ek"):
            extnType = "Potential|Neuter/ Yegoneko|Yekwaniso"
        elif txt in ("at"):
            extnType = "Contactive/ Yokubatika"
        elif txt in ("am"):
            extnType = "Stative/ Yemamiriro"
        elif txt in ("ar","arar"):
            extnType = "Extensive/ Yetambanuko"
        elif txt in ("zvi"):
            extnType = "Reflexive/ Yekuzviitira"
        return extnType



    """

        This function identifies the verb root and the extension for the remaining portion of the verb after
all affix slots up to the Object
        Concord have been stripped from it.

    """

    def seperateRootFromExtensions(self,txt):
        root =txt
        extensions =[]
        noMoreExtns = False
        while not noMoreExtns:
            root, newExtns = self.getExtension(root)
            if newExtns == []:
                noMoreExtns = True
            else:
                extensions.extend(newExtns)
        return root, extensions



    def start_transitions(self,txt):
        firstsyllable, txt = self.splitfirstsyllable(txt)
        ##print(firstsyllable+ " [First Syllable]")
        if  firstsyllable in negationOrmood:
            newState = "Negation_or_Mood_state"
            #print(firstsyllable + " [Negation or Mood]")
            self.structure.append(firstsyllable + " [Negation or Mood]")
        elif firstsyllable in subjectConcords:
            newState = "Subject_Concord_state"
            #print(firstsyllable + " [Subject Concord]")
            self.structure.append(firstsyllable + " [Subject Concord]")

        elif  firstsyllable in genitive_prefixes:
            newState = "genitive_prefixes_state"     #Otherwise write Genitive Transitions method
            #print(firstsyllable + " [Genitive Prefix]")
```

```
            self.structure.append(firstsyllable + " [Genitive_prefix]")

        else:
            newState = "error_state"
        return (newState, txt)


    def negation_or_mood_transitions(self,txt):
        orgTxt = txt
        newState = "Subject_Concord_state"
        firstsyllable, txt = self.splitfirstsyllable(txt)
        if firstsyllable in subjectConcords:
            newState = "Subject_Concord_state"
            #print(firstsyllable + " [Subject Concord]")
            self.structure.append(firstsyllable + " [Subject Concord]")
        else:
            newState = "Root_State"
            txt=orgTxt
       # #print(firstsyllable + " [Subject Concord]")
        return(newState,txt)


    def subject_concord_state_transitions(self,txt):
        orgTxt = txt
        firstsyllable, txt = self.splitfirstsyllable(txt)
        # #print(firstsyllable+ " [First Syllable]")
        if firstsyllable in tenseMarkers_slot1:
            newState = "Tense_Markers_Slot1_State"
            #print(firstsyllable + " [Tense Marker]")
            self.structure.append(firstsyllable + " [Tense Marker]")
        elif firstsyllable in tenseMarkers_slot2:
            newState= "Tense_Markers_Slot2_state"
            #print(firstsyllable + "[Tense Marker Slot 2]")
            self.structure.append(firstsyllable + "[Tense Marker Slot 2]")
        elif firstsyllable in negationSlot1:
            newState = "Negation_Slot1_State"
            #print(firstsyllable + " [Negation Slot 4]")
            self.structure.append(firstsyllable + " [Negation Slot 4]")
        elif firstsyllable in objectConcords:
            newState = "Object_Concord_state"
            #print(firstsyllable + " [Object Concord]")
            self.structure.append(firstsyllable + " [Object Concord]")
        elif firstsyllable in auxilliarySlot:
            if firstsyllable == "bvi":
                firstsyllable, txt = self.splitfirstsyllable(txt)
                firstsyllable = "bviro"
            elif firstsyllable == "fu":
                firstsyllable, txt = self.splitfirstsyllable(txt)
                firstsyllable = "fumo"
            newState = "Auxilliary_State"
            #print(firstsyllable + " [Auxilliary]")
            self.structure.append(firstsyllable + " [Auxilliary]")
        else:
            newState = "Root_State"
            txt=orgTxt
```

```
    return(newState, txt)


def negation_slot_4_transitions(self,txt):
    orgTxt = txt
    firstsyllable, txt = self.splitfirstsyllable(txt)
    if firstsyllable in tenseMarkers_slot1:
        newState = "Tense_Markers_Slot1_State"
        #print(firstsyllable + " [Tense Marker]")
        self.structure.append(firstsyllable + " [Tense Marker]")
    elif firstsyllable in tenseMarkers_slot2:
        newState= "Tense_Markers_Slot2_state"
        #print(firstsyllable + "[Tense Marker Slot 2]")
        self.structure.append(firstsyllable + "[Tense Marker Slot 2]")
    elif firstsyllable in negationSlot2:
        newState = "Negation_Slot6_State"
        #print(firstsyllable + " [Negation slot 6]")
        self.structure.append(firstsyllable + " [Negation slot 6]")
    elif firstsyllable in tenseMarkers_slot3:
        newState = "Tense_Markers_Slot3_state"
        #print(firstsyllable + " [Tense Marker Slot 3]")
        self.structure.append(firstsyllable + " [Tense Marker Slot 3]")
    elif firstsyllable in objectConcords:
        newState = "Object_Concord_state"
        #print(firstsyllable + " [Object Concord]")
        self.structure.append(firstsyllable + " [Object Concord]")
    else:
        newState = "Root_State"
        #print("Not yet handled, the first syllable was ", firstsyllable)
        self.structure.append("Not yet handled, the first syllable was " + firstsyllable)
        txt = orgTxt
    return(newState,txt)

def tense_markers_slot1_state_transitions(self,txt):
    orgTxt = txt
    firstsyllable, txt = self.splitfirstsyllable(txt)
    # #print(firstsyllable+ " [First Syllable]")
    if firstsyllable in tenseMarkers_slot2:
        newState = "Tense_Markers_Slot2_State"
        #print(firstsyllable + " [Tense Marker]")
        self.structure.append(firstsyllable + " [Tense Marker]")
    elif firstsyllable in objectConcords:
        newState = "Object_Concord_state"
        #print(firstsyllable + " [Object Concord]")
        self.structure.append(firstsyllable + " [Object Concord]")
    elif firstsyllable in auxilliarySlot:
        if firstsyllable == "bvi":
            firstsyllable, txt = self.splitfirstsyllable(txt)
            firstsyllable = "bviro"
        elif firstsyllable == "fu":
            firstsyllable, txt = self.splitfirstsyllable(txt)
            firstsyllable = "fumo"
        newState = "Auxilliary_State"
        #print(firstsyllable + " [Auxilliary]")
        self.structure.append(firstsyllable + " [Auxilliary]")
```

```
    else:
        newState = "Root_State"
        txt=orgTxt
        #print(txt + " [Inflected Root]")
    return(newState,txt)


def auxilliary_state_transitions(self,txt):
    orgTxt = txt
    firstsyllable, txt = self.splitfirstsyllable(txt)
    # #print(firstsyllable+ " [First Syllable]")
    if firstsyllable in objectConcords :
        newState = "Object_Concord_state"
        #print(firstsyllable + " [Object Concord]")
        self.structure.append(firstsyllable + " [Object Concord]")
    elif firstsyllable in auxilliarySlot:
        if firstsyllable == "bvi":
            firstsyllable, txt = self.splitfirstsyllable(txt)
            firstsyllable = "bviro"
        elif firstsyllable == "fu":
            firstsyllable, txt = self.splitfirstsyllable(txt)
            firstsyllable = "fumo"
        newState = "Auxilliary_State"
        #print(firstsyllable + " [Auxilliary]")
        self.structure.append(firstsyllable + " [Auxilliary]")
    else:
        newState = "Root_State"
        txt=orgTxt
    return(newState,txt)


def negation_slot_6_transitions(self,txt):
    orgTxt = txt
    firstsyllable, txt = self.splitfirstsyllable(txt)
    if firstsyllable in tenseMarkers_slot3:
        newState = "Tense_Markers_Slot3_state"
        #print(firstsyllable + " [Tense Marker Slot 3]")
        self.structure.append(firstsyllable + " [Tense Marker Slot 3]")
    else:
        newState = "Root_State"
        txt = orgTxt
    return(newState,txt)


def object_concord_transitions(self,txt):
    newState = "Root_state"
    ##print(txt + "[need to figure out what to do here]")
    return(newState,txt)


def tense_marker_slot2_state_transitions(self,txt):
    orgTxt = txt
    firstsyllable, txt = self.splitfirstsyllable(txt)
    if firstsyllable in tenseMarkers_slot3:
        newState = "tense_markers_slot3_state"
```

```
            #print(firstsyllable+ " [Tense marker slot 3]")
            self.structure.append(firstsyllable+ " [Tense marker slot 3]")
        elif firstsyllable in objectConcords:
            newState = "Object_Concord_state"
            #print(firstsyllable + " [Object Concord]")
            self.structure.append(firstsyllable + " [Object Concord]")
        elif  firstsyllable in negationSlot2:
            newState = "Negation_Slot6_State"
            #print(firstsyllable + " [Negation slot 6]")
            self.structure.append(firstsyllable + " [Negation slot 6]")
        elif firstsyllable in auxilliarySlot:
            if firstsyllable == "bvi":
                firstsyllable, txt = self.splitfirstsyllable(txt)
                firstsyllable = "bviro"
            elif firstsyllable == "fu":
                firstsyllable, txt = self.splitfirstsyllable(txt)
                firstsyllable = "fumo"
            newState = "Auxilliary_State"
            #print(firstsyllable + " [Auxilliary]")
            self.structure.append(firstsyllable + " [Auxilliary]")
        else:
        # ToDo: Add other transitions here
            newState = "Root_State"
            txt=orgTxt
        return(newState,txt)

    def tense_marker_slot3_state_transitions(self,txt):
        orgTxt = txt
        firstsyllable, txt = self.splitfirstsyllable(txt)
        if firstsyllable in objectConcords:
            newState = "Object_Concord_state"
            #print(firstsyllable + " [Object Concord]")
            self.structure.append(firstsyllable + " [Object Concord]")
        else:
            newState = "Root_State"
            txt = orgTxt
        return(newState, txt)

    def genitive_transitions(self,txt):
        orgTxt = txt
        firstsyllable, txt = self.splitfirstsyllable(txt)
        if firstsyllable in subjectConcords:
            newState = "Subject_Concord_state"
            self.structure.append(firstsyllable + " [Subject Concord]")
        elif  firstsyllable in objectConcords:
            newState = "Object_Concord_state"
            self.structure.append(firstsyllable + " [Object Concord]")

        else:
            newState = "error_state"
            txt = orgTxt
        return(newState, txt)


    def root_state_transitions(self,txt):
```

```
        ##print("In root transitions with text = ", txt)
        finalVowel = txt[-1:]
        extensions=[]
        root = txt[:-1]
        root, extensions = self.seperateRootFromExtensions(root)
        if self.presumedStem(root+"a") == (root+"a"):
        #if self.hasValidStem(root + "a"):
           #print(root + " [Root]")
           self.structure.append(root + " [Root]")
        elif extensions:
           if self.hasValidStem(root + extensions[-1] + "a"):
              root = root + extensions[-1]
              extensions.remove(extensions[-1])
              #print(root + " [Root]")
              self.structure.append(root + " [Root]")
        #elif self.presumedStem == txt.lower():   #Added line
        #    self.structure.append(txt + "[root]")
        #    newState ="End_State"
        else:
           newState = "error_state"
           #self.structure.append(root + "[Invalid Verb Root]")

           return (newState, txt)
        if extensions != []:
           extensions.reverse()
        num_extns = len(extensions)
        for i in range(num_extns):
           #print(extensions[i], " [Extension ", i+1,"]", self.extensionType(extensions[i]))
           extens = extensions[i] + " [Extension " + str(i+1)  +"]" + str(
self.extensionType(extensions[i]))
           self.structure.append(extens)
        if finalVowel not in vowels:
           newState = "error_state" # "End_State" #
           self.structure.append(finalVowel + " [Invalid Final vowel]")
        else:
           #print(finalVowel+" [Final Vowel]")
           self.structure.append(finalVowel+" [Final Vowel]")
           newState ="End_State"
           #package = (root, extensions,finalVowel)
        return(newState,txt)

    def end_state_transitions(self,package):
        done =1
        #print("Verb Accepted with root ", package[0], " and extensions ",package[1], "and final vowel
",package[2])

        return done

    def spellCheck(self,txt):

        retValue = self.run(txt)
        if retValue:
           retValue = True
        else:
           retValue = False
```

```
return retValue
```

Listing 3 – ShonaNoun.py : Morphological Analyser for Shona Nouns

```
# -*- coding: utf-8 -*-
"""
This program implements a morphological analyser for Shona Nouns

"""

negationOrmood ={"i","ha","nga"}
subjectConcords = {"ndi","nda",
"ti","ta","u","wa","mu","ma","a","va","i","ya","ri","ra","a","chi","cha","zvi","zva","i","dzi","dza","ru"
,"ra","rwu","rwa","ka","twu","twa","tu","hu","hwu","hwa","ku","kwa","pa","ku","mu","mwa","svi","
sva","kwu","kwa","ku","zi","za"}
tenseMarkers_slot1 = {"i","cha","no"}
negationSlot1 = {"si","sa"}
tenseMarkers_slot2={"chi","ka","do","ne","nga"}
negationSlot2 = {"si","ka","za", "sa"} # The last "sa" does not appear in Dr Mberi's slot system - to
validate with him
#negationSlot3 = {"si","sa"}
tenseMarkers_slot3 = {"chi","ka","zo"}
auxilliarySlot = {"ndo","mbo","ngo","zo","fu","bvi"} #"fumo","bviro"}
objectConcords =
{"ndi","ti","ku","mu","mu","mu","va","u","i","ri","a","chi","zvi","i","dzi","ru","ru","ka","twu","tu","h
u","vu","ku","pa","ku","mu","svi","ku","ri"}
validSyllables =
{"b","bh","bw","ch","d","dh","dy","dz","dzv","f","g","h","hw","j","k","kw","m","mb","mw","mbw","
mh","n","n'","nd","ndy","ndw","ng","ngw","nh","nw","ny","nz","nzv","nzw","p","r","rw","s","sh","s
w","sv","t","ty","tw","v","vh","w","y","z","zh","zhw","zv","zw"}
noun_prefixes = {"mu","va","mi","ri","ma","chi","zvi","i","dzi","ru","ka","tu","ku","pa","svi","zi"}
noun_stems = {"komana","sikana","adhivhoketi","dhokota","chiremba","basa","gwaro","chikwari"}
vowels ={"a","e","i","o","u"}
genitive_prefixes ={"ne","wa", "we","wo", "va","ve","vo","ya","ye","yo", "ra", "re","ro",
"a","e","cha","che","cho","zva","zve","zvo","dza","dze","dzo","rwa","rwe","rwo","ka","ke","ko","twa
","twe","two","hwa","hwe","hwo","kwa","kwe","kwo","pa","pe","po","ma","me","mo","sva","sve","s
vo"}


from StateMachine import StateMachine # as sm
try:
  import cPickle as pickle
except:
  import pickle as pkl

class ShonaNoun(StateMachine):

    structure = {}
    class_1_nouns = []
    class_1a_nouns = []
    class_1b_nouns = []
    class_2_nouns  = []
    class_2a_nouns = []
```

```
class_2b_nouns = []
class_3_nouns = []
class_4_nouns = []
class_5_nouns = []
class_6_nouns = []
class_7_nouns = []
class_8_nouns = []
class_9_nouns = []
class_10_nouns = []
class_11_nouns = []
class_12_nouns = []
class_13_nouns = []
class_14_nouns = []
class_15_nouns = []
class_16_nouns = []
class_17_nouns = []
class_17a_nouns = []
class_18_nouns = []
class_19_nouns = []
class_21_nouns  = []

#nounStems = []
class_1_stems = []
class_1a_stems = []
class_3_stems = []
class_5_stems = []
class_7_stems= []
class_9_stems= []
class_10_stems = []
class_11_stems = []
class_12_stems = []
class_14_stems = []
class_14_stems = []
class_16_stems = []
class_18_stems = []
class_21_stems = []


verbStems = []

def __init__(self):
    StateMachine.__init__(self)
    #shonaVerbFSA = StateMachine()
    StateMachine.add_state(self,"Start", self.start_transitions)
    StateMachine.add_state(self,"noun_prefixes_state",self.noun_prefix_transitions)
    StateMachine.add_state(self,"noun_stem_state",self.noun_stem_transitions)
    StateMachine.add_state(self,"Class_1_or_3_state",self.class_1_or_3_transitions)
    StateMachine.add_state(self,"Class_2_state",self.class_2_transitions)
    StateMachine.add_state(self,"Class_2b_state",self.class_2b_transitions)
    StateMachine.add_state(self,"Class_6_or_10_state",self.class_6_or_10_transitions)
    StateMachine.add_state(self,"Class_4_state",self.class_4_transitions)
    StateMachine.add_state(self,"Class_7_state",self.class_7_transitions)
    StateMachine.add_state(self,"Class_11_state",self.class_11_transitions)
    StateMachine.add_state(self,"Class_12_state",self.class_12_transitions)
    StateMachine.add_state(self,"Class_14_state",self.class_14_transitions)
```

```
        StateMachine.add_state(self,"Class_15_or_17_state",self.class_15_or_17_transitions)
        StateMachine.add_state(self,"Class_16_state",self.class_16_transitions)
        StateMachine.add_state(self,"Class_19_state",self.class_19_transitions)
        StateMachine.add_state(self,"Class_21_state",self.class_21_transitions)

        StateMachine.add_state(self,"genitive_prefixes_state",self.genitive_transitions)



        StateMachine.add_state(self,"End_State",self.end_state_transitions, end_state=1)
        StateMachine.add_state(self,"error_state", None, end_state=1)
        StateMachine.set_start(self,"Start")
        self.structure = []
        nounClasses = pkl.load(open("e:/data/programming/python/Shona NLP/Language
Modelling/Dictionaries/nounClasses.p","rb"))

        self.class_1_nouns, self.class_1a_nouns,self.class_1b_nouns,self.class_2_nouns,
self.class_2a_nouns, self.class_2b_nouns, self.class_3_nouns,self.class_4_nouns, self.class_5_nouns,
self.class_6_nouns, self.class_7_nouns, self.class_8_nouns, self.class_9_nouns, self.class_10_nouns,
self.class_11_nouns, self.class_12_nouns, self.class_13_nouns, self.class_14_nouns,
self.class_15_nouns, self.class_16_nouns, self.class_17_nouns, self.class_17a_nouns,
self.class_18_nouns, self.class_19_nouns,  self.class_21_nouns = nounClasses

        nounStems = pkl.load(open("e:/data/programming/python/Shona NLP/Language
Modelling/Dictionaries/nounClass_stems.p","rb"))

        #class_stems = [class_1_stems, class_1a_stems, class_5_stems, class_7_stems, class_9_stems,
class_10_stems]

        self.class_1_stems,self.class_1a_stems, self.class_3_stems,
self.class_5_stems,self.class_7_stems, self.class_9_stems, self.class_10_stems, self.class_11_stems,
self.class_12_stems, self.class_14_stems, self.class_15_stems, self.class_16_stems,
self.class_18_stems, self.class_21_stems = nounStems

        #return ShonaVerbFSA


    def openStemDictionary(self):
        with open('dictionaries/madzitsi_eduramazwi.txt') as f:
            lines = f.read().splitlines()
        return lines

    def openDictionary(self):
        with open('dictionaries/Shona_words.txt') as f: #'DGR_vocab.txt') as f: # mazwi_eduramazwi
            words_dict = f.read().splitlines()
        return words_dict

    def syllabify(self,txt):
        syllables = []
        firstsyllable, restoftxt = self.splitfirstsyllable(txt)
        while len(restoftxt) > 0:
            syllables.append(firstsyllable)
            firstsyllable, restoftxt = self.splitfirstsyllable(restoftxt)
            ##print(firstsyllable)
        if len(firstsyllable) > 0:
```

```
            syllables.append(firstsyllable + txt[-1:])
        return syllables

    def hasValidSyllables(self,txt):
        itdoes = True
        chktxt = txt.lower()
        syllables = self.syllabify(chktxt)

        for syllable in syllables:
            if syllable[:-1] in validSyllables:
                #flash('{} is a valid syllable.'.format(syllable))
                itdoes= itdoes
            elif syllable in vowels:
                #flash('{} is a valid syllable.'.format(syllable))
                itdoes= itdoes
            else:
                #flash('{} is not a valid syllable'.format(syllable))
                itdoes= False
        #flash('{} has been checked and the statement of whether it  has valid syllables has been found to
be {}'.format(txt,itdoes))
        return itdoes

    def hasValidSubwords(self,txt):
        word_list = self.openDictionary()
        found = False
        restoftxt = txt
        while (not found) and (len(restoftxt) > 2):
            firstsyllable, restoftxt = self.splitfirstsyllable(restoftxt)
            #flash('Split into <{}> and <{}>.'.format(firstsyllable, restoftxt))
            subword = restoftxt
            if subword in word_list:
                #flash('{} is a valid subword'.format(subword))
                found = True
            else:
                found=False
        #flash('{} has been checked and the statement of whether it has valid subwords has been found to
be {}'.format(txt,found))
        return found

    def hasValidStem(self,txt):
        #words_list = sv.openDictionary()
        stem_list = self.openStemDictionary()
        found = False
        restoftxt = txt
        while (not found) and (len(restoftxt) > 2):
            firstsyllable, restoftxt = self.splitfirstsyllable(restoftxt)
            #flash('Split into <{}> and <{}>.'.format(firstsyllable, restoftxt))
            stem = restoftxt[:-1]
            stem = "-" + stem + "a"
            if stem in stem_list:
                #flash('{} is a valid stem'.format(stem))
                found = True
            else:
                found=False
```

```
        #flash('{} has been checked and the statement of whether it has valid stems has been found to be
{}'.format(txt,found))
        return found


    def roteSpellCheck(self,txt):
        words_list = self.openDictionary()
        if txt in words_list:
            #print('Word {} found in our our dictionary'.format(txt))
            found = True
        else:
            if (self.hasValidStem(txt) or self.hasValidSubwords(txt)) and self.hasValidSyllables(txt):
                found = True
            else:
                found = False
            #if not found:
                #print('The word {} is not a known word.'.format(txt))
        return_val = found
        return return_val


    """
        This function splits a Shona word or the remaining part of the word into two components. The
first part is the
        first syllable in the word, or word fragmanent and the second part is the remainder of the word or
word segment.
        A syllable in Shona is defined as being composed of either a single vowel or a number of
consonants followed by a
        vowel.
        This function does not attempt to check for the validity of the syllables returned - by validity
here, we refer to
        whether the syllable conforms with the official shona orthography or not.

    """
    def splitfirstsyllable(self,txt):
        firstSyllable=""
        restOftext=""
        vowels={"a","e","i","o","u"}
        if len(txt) ==1 :
            firstSyllable=txt
        else:
            if len(txt) > 0:
                startChar=txt[0]
                ##print("startChar is : " +startChar )
                if startChar in vowels:
                    firstSyllable = startChar
                    restOftext = txt[1:]
                else:
                    i=1
                    nextChar=txt[i]
                    firstSyllable=startChar
                    while (nextChar not in vowels) and ( (i+1) < len(txt)):
                        firstSyllable= firstSyllable+nextChar
                        i+=1
                        nextChar=txt[i]
```

```
            if (i+1) != len(txt):
                firstSyllable=firstSyllable+nextChar
                restOftext = txt[len(firstSyllable):]
    return firstSyllable, restOftext




def start_transitions(self,txt):
    orgTxt = txt
    firstsyllable, txt = self.splitfirstsyllable(txt)
    # #print(firstsyllable+ " [First Syllable]")
    # if orgTxt  in self.class_1_nouns:
    #     newState = "Class_1_or_3_state"
    #     #print(firstsyllable + " [Class 1 or 3]")
    #     self.structure.append(firstsyllable + " [Class 1 or 3 Noun_prefix]")

    if  firstsyllable == "mu":              #noun_prefixes:
        newState = "Class_1_or_3_state"
        #print(firstsyllable + " [Class 1 or 3]")
        self.structure.append(firstsyllable + " [Class 1 or 3 Noun_prefix]")
    elif firstsyllable == "a":
        nxtsyllable, rst = self.splitfirstsyllable(txt)

        if nxtsyllable == "nya":
            newState = "noun_stem_state"
            #print(firstsyllable + nxtsyllable + " [Class 2b]")
            self.structure.append(firstsyllable + " [Class 2b Noun_prefix]")
            #txt = rst
        else:
            newState = "Class_2_state"
            #print(firstsyllable + " [Class 2]")
            self.structure.append(firstsyllable + " [Class 2 Noun_prefix]")

    elif firstsyllable == "va":
        nxtsyllable, rst = self.splitfirstsyllable(txt)
        if nxtsyllable == "na":
            newState = "Class_2b_state"
            #print(firstsyllable + nxtsyllable + " [Class 2b]")
            self.structure.append(firstsyllable + nxtsyllable + " [Class 2b Noun_prefix]")
            txt = rst
        elif nxtsyllable == "nya":
            newState = "noun_stem_state"
            #print(firstsyllable + nxtsyllable + " [Class 2b]")
            self.structure.append(firstsyllable + " [Class 2b Noun_prefix]")
            #txt = rst
        else:
            newState = "Class_2_state"
            #print(firstsyllable + " [Class 2]")
            self.structure.append(firstsyllable + " [Class 2 Noun_prefix]")
    elif firstsyllable == "ma":
        nxtsyllable, rst = self.splitfirstsyllable(txt)
        if nxtsyllable == "dzi":
```

```
            newState  = "Class_6_or_10_state"
            #print(firstsyllable + nxtsyllable + " [Class 6]")
            self.structure.append(firstsyllable + " [Class 6 Noun_prefix]")
            txt = rst
        else:
            newState = "Class_6_or_10_state"
            #print(firstsyllable + " [Class 6 or 10]")
            self.structure.append(firstsyllable + " [Class 6 Noun_prefix]")
    elif firstsyllable == "mi":
        newState = "Class_4_state"
        #print(firstsyllable + " [Class 4]")
        self.structure.append(firstsyllable + " [Class 4 Noun_prefix]")
    elif firstsyllable in ("chi","cha"):
        newState = "Class_7_state"
        #print(firstsyllable + " [Class 7]")
        self.structure.append(firstsyllable + " [Class 7 Noun_prefix]")
    elif firstsyllable in ("zvi","zva","zvu"):
        newState = "Class_7_state"
        #print(firstsyllable + " [Class 8 prefix]")
        self.structure.append(firstsyllable + " [Class 8 Noun_prefix]")
    elif firstsyllable in ("ru","rwa","gwa", "rwe", "gwe", "gu"):
        newState = "Class_11_state"
        #print(firstsyllable + " [Class 11 prefix]")
        self.structure.append(firstsyllable + " [Class 11 Noun_prefix]")
    elif firstsyllable == "ka":
        newState = "Class_12_state"
        #print(firstsyllable + " [Class 12 prefix]")
        self.structure.append(firstsyllable + " [Class 12 Noun_prefix]")
    elif firstsyllable == "tu":
        newState = "Class_12_state"
        #print(firstsyllable + " [Class 12 prefix]")
        self.structure.append(firstsyllable + " [Class 12 Noun_prefix]")
    elif firstsyllable in ("u","hu"):
        newState = "Class_14_state"
        #print(firstsyllable + " [Class 14 prefix]")
        self.structure.append(firstsyllable + " [Class 14 Noun_prefix]")

    elif firstsyllable in ("ku"):
        newState = "Class_15_or_17_state"
        #print(firstsyllable + " [Class 15 or 17 prefix]")
        self.structure.append(firstsyllable + " [Class 15 or 17 Noun_prefix]")

    elif firstsyllable in ("pa"):
        newState = "Class_16_state"
        #print(firstsyllable + " [Class 16 prefix]")
        self.structure.append(firstsyllable + " [Class 16 Noun_prefix]")
    elif firstsyllable in ("svi"):
        newState = "Class_19_state"
        #print(firstsyllable + " [Class 19 prefix]")
        self.structure.append(firstsyllable + " [Class 19 Noun_prefix]")
    elif firstsyllable in ("zi"):
        newState = "Class_21_state"
        #print(firstsyllable + " [Class 21 prefix]")
        self.structure.append(firstsyllable + " [Class 21 Noun_prefix]")
    elif  firstsyllable in noun_prefixes:
```

```
            newState = "noun_prefixes_state"
            ##print(firstsyllable + " [Negation or Mood]")
            self.structure.append(firstsyllable + " [Noun_prefix]")

        elif  firstsyllable in genitive_prefixes:
            newState = "genitive_prefixes_state"     #Otherwise write Genitive Transitions method
            #print(firstsyllable + " [Genitive Prefix]")
            self.structure.append(firstsyllable + " [Genitive_prefix]")
        else:
            txt = orgTxt
            newState = "noun_stem_state"
            self.structure.append( "[]" + " Noun Prefix]  first syllable was " +firstsyllable)
        return (newState, txt)


    def noun_prefix_transitions(self,txt):
        '''
        TODO:
            Have an IF statement per Noun Class
            Also look at nouns that are formed from verbs
        '''
        orgTxt = txt
        if txt in noun_stems:
            newState = "End_State"
            self.structure.append(txt + " [Noun Stem]")
        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun Stem")
            txt=orgTxt
        return(newState,txt)

    def class_1_or_3_transitions(self,txt):
        if txt in self.class_1_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 1 Noun stem]")
            #print(txt + " Class 1 stem")
        elif txt in self.class_3_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 3 Noun stem]")
            #print(txt + " Class 3 stem")
        elif txt in self.class_18_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 18 Noun stem]")
            #print(txt + " Class 18 stem")

        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun]")
            #print(txt +  " has not been assigned a noun Class")
        return(newState, txt)



    def class_2_transitions(self,txt):
        if txt in self.class_1_stems:
```

```
            newState = "End_State"
            self.structure.append(txt + " [Class 1 Noun stem]")
            #print(txt + " Class 2 stem")
        elif  txt in self.class_1a_nouns:
            newState = "End_State"
            self.structure.append(txt + " [Class 1a Noun]")
        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun]")

        return(newState, txt)

    def class_2b_transitions(self, txt):
        orgTxt = txt
        modtxt = "a" + txt
        firstsyllable, txt = self.splitfirstsyllable(orgTxt)
        if firstsyllable == "nya":
                newState = "noun_stem_state"
                txt = orgTxt
        elif txt in self.class_1a_stems:
            newState = "End_State"
            self.structure.append(orgTxt + " [Class 1a Noun stem]")
            #print(txt + " Class 1a stem")
        elif txt in self.class_1_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 1 Noun stem]")
            #print(txt + " Class 1a stem")
        elif modtxt in self.class_1a_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 1a Noun stem]")
        elif  orgTxt in self.class_1a_nouns:
            newState = "End_State"
            self.structure.append(orgTxt + " [Class 1a Noun]")
        else:
            newState = "error_state"
            #print(txt + " [No class 2b stem found]")
            self.structure.append(txt + " [Unknown Noun]")

        return(newState, txt)

    def class_4_transitions(self, txt):
        if txt in self.class_3_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 3 Noun stem]")
            #print(txt + " Class 3 stem")
        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun]")
            #print(txt +  " has not been assigned a noun Class")
        return(newState, txt)

    def class_6_or_10_transitions(self, txt):
        modtxt = "a" + txt
        if txt in self.class_1a_stems:
            newState = "End_State"
```

```
            self.structure.append(txt + " [Class 1a Noun stem]")
            #print(txt + " Class 1a stem")
        elif modtxt in self.class_1a_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 1a Noun stem]")
            #print(txt + " Class 1a stem")
        elif txt in self.class_5_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 5 Noun stem]")
            #print(txt + " Class 5 stem")
        elif txt in self.class_9_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 9 Noun stem]")
            #print(txt + " Class 9 stem")
        elif txt in self.class_11_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 11 Noun stem]")
            #print(txt + " Class 11 stem")

        else:
            newState = "error_state"
            #print(txt + " [No class 6 stem found]")
            self.structure.append(txt + " [Unknown Noun]")

        return(newState, txt)


    def class_7_transitions(self, txt):
        orgTxt = txt
        firstsyllable, txt = self.splitfirstsyllable(txt)
        if orgTxt in self.class_7_stems:
            newState = "End_State"
            self.structure.append(orgTxt + " [Class 7 Noun stem]")
            #print(txt + " Class 7 stem")
        elif firstsyllable == "mu":             #noun_prefixes:
            newState = "Class_1_or_3_state"
            #print(firstsyllable + " [Class 1 or 3]")
            self.structure.append(firstsyllable + " [Class 1 or 3 Noun_prefix]")
        elif firstsyllable == "va":
            nxtsyllable, rst = self.splitfirstsyllable(txt)
            if nxtsyllable == "na":
                newState = "Class_2b_state"
                #print(firstsyllable + nxtsyllable + " [Class 2b]")
                self.structure.append(firstsyllable + " [Class 2b Noun_prefix]")
                txt = rst
            else:
                newState = "Class_2_state"
                #print(firstsyllable + " [Class 2]")
                self.structure.append(firstsyllable + " [Class 2 Noun_prefix]")
        elif firstsyllable == "ma":
            nxtsyllable, rst = self.splitfirstsyllable(txt)
            if nxtsyllable == "dzi":
                newState  = "Class_6_or_10_state"
                #print(firstsyllable + nxtsyllable + " [Class 6]")
                self.structure.append(firstsyllable + " [Class 6 Noun_prefix]")
```

```
            txt = rst
        else:
            newState = "Class_6_or_10_state"
            #print(firstsyllable + " [Class 6 or 10]")
            self.structure.append(firstsyllable + " [Class 6 Noun_prefix]")
    elif firstsyllable == "mi":
        newState = "Class_4_state"
        #print(firstsyllable + " [Class 4]")
        self.structure.append(firstsyllable + " [Class 4 Noun_prefix]")
    elif firstsyllable in ("chi","cha"):
        newState = "Class_7_state"
        #print(firstsyllable + " [Class 7]")
        self.structure.append(firstsyllable + " [Class 7 Noun_prefix]")
    elif firstsyllable in ("zvi","zva","zvu"):
        newState = "Class_7_state"
        #print(firstsyllable + " [Class 8 prefix]")
        self.structure.append(firstsyllable + " [Class 8 Noun_prefix]")
    elif firstsyllable in ("ru","rwa","gwa", "rwe", "gwe", "gu"):
        newState = "Class_11_state"
        #print(firstsyllable + " [Class 11 prefix]")
        self.structure.append(firstsyllable + " [Class 11 Noun_prefix]")
    elif firstsyllable == "ka":
        newState = "Class_12_state"
        #print(firstsyllable + " [Class 12 prefix]")
        self.structure.append(firstsyllable + " [Class 12 Noun_prefix]")
    elif firstsyllable == "tu":
        newState = "Class_12_state"
        #print(firstsyllable + " [Class 12 prefix]")
        self.structure.append(firstsyllable + " [Class 12 Noun_prefix]")
    elif firstsyllable in ("u","hu"):
        newState = "Class_14_state"
        #print(firstsyllable + " [Class 14 prefix]")
        self.structure.append(firstsyllable + " [Class 14 Noun_prefix]")

    elif firstsyllable in ("ku"):
        newState = "Class_15_or_17_state"
        #print(firstsyllable + " [Class 15 or 17 prefix]")
        self.structure.append(firstsyllable + " [Class 15 or 17 Noun_prefix]")

    elif firstsyllable in ("pa"):
        newState = "Class_16_state"
        #print(firstsyllable + " [Class 16 prefix]")
        self.structure.append(firstsyllable + " [Class 16 Noun_prefix]")
    elif firstsyllable in ("svi"):
        newState = "Class_19_state"
        #print(firstsyllable + " [Class 19 prefix]")
        self.structure.append(firstsyllable + " [Class 19 Noun_prefix]")
    elif firstsyllable in ("zi"):
        newState = "Class_21_state"
        #print(firstsyllable + " [Class 21 prefix]")
        self.structure.append(firstsyllable + " [Class 21 Noun_prefix]")
    elif  firstsyllable in noun_prefixes:
        newState = "noun_prefixes_state"
        ##print(firstsyllable + " [Negation or Mood]")
        self.structure.append(firstsyllable + " [Noun_prefix]")
```

```
        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun]")
            #print(txt +  " has not been assigned a noun Class")
        return(newState, txt)

    def class_11_transitions(self, txt):
        if txt in self.class_11_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 11 Noun stem]")
            #print(txt + " Class 11 stem")
        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun]")
            #print(txt +  " has not been assigned a noun Class")
        return(newState, txt)

    def class_12_transitions(self, txt):
        if txt in self.class_12_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 12 Noun stem]")
            #print(txt + " Class 12 stem")
        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun]")
            #print(txt +  " has not been assigned a noun Class")
        return(newState, txt)

    def class_14_transitions(self, txt):
        if txt in self.class_14_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 14 Noun stem]")
            #print(txt + " Class 14 stem")
        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun]")
            #print(txt +  " has not been assigned a noun Class")
        return(newState, txt)

    def class_15_or_17_transitions(self,txt):
        if txt in self.class_15_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 15 Noun stem]")
            #print(txt + " Class 15 stem")
        else:
            nxtsyllable, rst = self.splitfirstsyllable(txt)
            if nxtsyllable == "sa":
                if rst in self.verbStems:
                    newState = "End_State"
                    self.structure.append(txt + " [Negative Noun stem]")
                    #print(txt + " Negative Noun stem")
                else:
                    newState = "error_state"
                    self.structure.append(txt + " [Unknown Noun]")
```

```
            #print(txt +  " has not been assigned a noun Class")
        elif rst in self.verbStems:
            newState = "End_State"
            self.structure.append(txt + " [Negative Noun stem]")
            #print(txt + " Negative Noun stem")
        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun]")
    return(newState, txt)


def  class_16_transitions(self, txt):
    if txt in self.class_16_stems:
        newState = "End_State"
        #print(txt + " Class 16 stem")
        self.structure.append(txt + " [Class 16 Noun stem]")
    else:
        newState = "error_state"
        self.structure.append(txt + " [Unknown Noun]")

    return(newState, txt)


def class_19_transitions(self,txt):
    orgTxt = txt
    firstsyllable, txt = self.splitfirstsyllable(txt)

    if orgTxt in self.class_1_stems:
        newState = "End_State"
        #print(orgTxt + " Class 1 stem")
        self.structure.append(orgTxt + " [Class 1 Noun stem]")
    elif orgTxt in self.class_1a_stems:
        newState = "End_State"
        #print(orgTxt + " Class 1a stem")
        self.structure.append(orgTxt + " [Class 1a Noun stem]")
    elif orgTxt in self.class_3_stems:
        newState = "End_State"
        #print(orgTxt + " Class 3 stem")
        self.structure.append(orgTxt + " [Class 3 Noun stem]")
    elif orgTxt in self.class_5_stems:
        newState = "End_State"
        #print(orgTxt + " Class 5 stem")
        self.structure.append(orgTxt + " [Class 5 Noun stem]")
    elif orgTxt in self.class_7_stems:
        newState = "End_State"
        #print(orgTxt + " Class 7 stem")
        self.structure.append(orgTxt + " [Class 7 Noun stem]")
    elif orgTxt in self.class_9_stems:
        newState = "End_State"
        #print(orgTxt + " Class 9 stem")
        self.structure.append(orgTxt + " [Class 9 Noun stem]")
    elif orgTxt in self.class_11_stems:
        newState = "End_State"
        #print(orgTxt + " Class 11 stem")
        self.structure.append(orgTxt + " [Class 11 Noun stem]")
```

```
elif orgTxt in self.class_14_stems:
   newState = "End_State"
   #print(orgTxt + " Class 14 stem")
   self.structure.append(orgTxt + " [Class 14 Noun stem]")


elif firstsyllable == "mu":            #noun_prefixes:
   newState = "Class_1_or_3_state"
   #print(firstsyllable + " [Class 1 or 3]")
   self.structure.append(firstsyllable + " [Class 1 or 3 Noun_prefix]")
elif firstsyllable == "va":
   nxtsyllable, rst = self.splitfirstsyllable(txt)
   if nxtsyllable == "na":
      newState = "Class_2b_state"
      #print(firstsyllable + nxtsyllable + " [Class 2b]")
      self.structure.append(firstsyllable + " [Class 2b Noun_prefix]")
      txt = rst
   else:
      newState = "Class_2_state"
      #print(firstsyllable + " [Class 2]")
      self.structure.append(firstsyllable + " [Class 2 Noun_prefix]")
elif firstsyllable == "ma":
   nxtsyllable, rst = self.splitfirstsyllable(txt)
   if nxtsyllable == "dzi":
      newState  = "Class_6_or_10_state"
      #print(firstsyllable + nxtsyllable + " [Class 6]")
      self.structure.append(firstsyllable + " [Class 6 Noun_prefix]")
      txt = rst
   else:
      newState = "Class_6_or_10_state"
      #print(firstsyllable + " [Class 6 or 10]")
      self.structure.append(firstsyllable + " [Class 6 Noun_prefix]")
elif firstsyllable == "mi":
   newState = "Class_4_state"
   #print(firstsyllable + " [Class 4]")
   self.structure.append(firstsyllable + " [Class 4 Noun_prefix]")
elif firstsyllable in ("chi","cha"):
   newState = "Class_7_state"
   #print(firstsyllable + " [Class 7]")
   self.structure.append(firstsyllable + " [Class 7 Noun_prefix]")
elif firstsyllable in ("zvi","zva","zvu"):
   newState = "Class_7_state"
   #print(firstsyllable + " [Class 8 prefix]")
   self.structure.append(firstsyllable + " [Class 8 Noun_prefix]")
elif firstsyllable in ("ru","rwa","gwa", "rwe", "gwe", "gu"):
   newState = "Class_11_state"
   #print(firstsyllable + " [Class 11 prefix]")
   self.structure.append(firstsyllable + " [Class 11 Noun_prefix]")
elif firstsyllable == "ka":
   newState = "Class_12_state"
   #print(firstsyllable + " [Class 12 prefix]")
   self.structure.append(firstsyllable + " [Class 12 Noun_prefix]")
elif firstsyllable == "tu":
   newState = "Class_12_state"
   #print(firstsyllable + " [Class 12 prefix]")
```

```
            self.structure.append(firstsyllable + " [Class 12 Noun_prefix]")
        elif firstsyllable in ("u","hu"):
            newState = "Class_14_state"
            #print(firstsyllable + " [Class 14 prefix]")
            self.structure.append(firstsyllable + " [Class 14 Noun_prefix]")

        elif firstsyllable in ("ku"):
            newState = "Class_15_or_17_state"
            #print(firstsyllable + " [Class 15 or 17 prefix]")
            self.structure.append(firstsyllable + " [Class 15 or 17 Noun_prefix]")

        elif firstsyllable in ("pa"):
            newState = "Class_16_state"
            #print(firstsyllable + " [Class 16 prefix]")
            self.structure.append(firstsyllable + " [Class 16 Noun_prefix]")
        elif firstsyllable in ("svi"):
            newState = "Class_19_state"
            #print(firstsyllable + " [Class 19 prefix]")
            self.structure.append(firstsyllable + " [Class 19 Noun_prefix]")
        elif firstsyllable in ("zi"):
            newState = "Class_21_state"
            #print(firstsyllable + " [Class 21 prefix]")
            self.structure.append(firstsyllable + " [Class 21 Noun_prefix]")
        elif  firstsyllable in noun_prefixes:
            newState = "noun_prefixes_state"
            ##print(firstsyllable + " [Negation or Mood]")
            self.structure.append(firstsyllable + " [Noun_prefix]")




        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun]")

        return(newState, txt)

    def class_21_transitions(self,txt):
        orgTxt = txt
        firstsyllable, txt = self.splitfirstsyllable(txt)
        if (orgTxt in self.class_21_stems) or (orgTxt in self.class_1_stems):
            newState = "End_State"
            #print(orgTxt + " Class 21 stem")
            self.structure.append(orgTxt + " [Class 21 Noun stem]")
        elif firstsyllable == "mu":             #noun_prefixes:
            newState = "Class_1_or_3_state"
            #print(firstsyllable + " [Class 1 or 3]")
            self.structure.append(firstsyllable + " [Class 1 or 3 Noun_prefix]")
        elif firstsyllable == "va":
            nxtsyllable, rst = self.splitfirstsyllable(txt)
            if nxtsyllable == "na":
                newState = "Class_2b_state"
                #print(firstsyllable + nxtsyllable + " [Class 2b]")
                self.structure.append(firstsyllable + " [Class 2b Noun_prefix]")
```

```
            txt = rst
        else:
            newState = "Class_2_state"
            #print(firstsyllable + " [Class 2]")
            self.structure.append(firstsyllable + " [Class 2 Noun_prefix]")
    elif firstsyllable == "ma":
        nxtsyllable, rst = self.splitfirstsyllable(txt)
        if nxtsyllable == "dzi":
            newState  = "Class_6_or_10_state"
            #print(firstsyllable + nxtsyllable + " [Class 6]")
            self.structure.append(firstsyllable + " [Class 6 Noun_prefix]")
            txt = rst
        else:
            newState = "Class_6_or_10_state"
            #print(firstsyllable + " [Class 6 or 10]")
            self.structure.append(firstsyllable + " [Class 6 Noun_prefix]")
    elif firstsyllable == "mi":
        newState = "Class_4_state"
        #print(firstsyllable + " [Class 4]")
        self.structure.append(firstsyllable + " [Class 4 Noun_prefix]")
    elif firstsyllable in ("chi","cha"):
        newState = "Class_7_state"
        #print(firstsyllable + " [Class 7]")
        self.structure.append(firstsyllable + " [Class 7 Noun_prefix]")
    elif firstsyllable in ("zvi","zva","zvu"):
        newState = "Class_7_state"
        #print(firstsyllable + " [Class 8 prefix]")
        self.structure.append(firstsyllable + " [Class 8 Noun_prefix]")
    elif firstsyllable in ("ru","rwa","gwa", "rwe", "gwe", "gu"):
        newState = "Class_11_state"
        #print(firstsyllable + " [Class 11 prefix]")
        self.structure.append(firstsyllable + " [Class 11 Noun_prefix]")
    elif firstsyllable == "ka":
        newState = "Class_12_state"
        #print(firstsyllable + " [Class 12 prefix]")
        self.structure.append(firstsyllable + " [Class 12 Noun_prefix]")
    elif firstsyllable == "tu":
        newState = "Class_12_state"
        #print(firstsyllable + " [Class 12 prefix]")
        self.structure.append(firstsyllable + " [Class 12 Noun_prefix]")
    elif firstsyllable in ("u","hu"):
        newState = "Class_14_state"
        #print(firstsyllable + " [Class 14 prefix]")
        self.structure.append(firstsyllable + " [Class 14 Noun_prefix]")

    elif firstsyllable in ("ku"):
        newState = "Class_15_or_17_state"
        #print(firstsyllable + " [Class 15 or 17 prefix]")
        self.structure.append(firstsyllable + " [Class 15 or 17 Noun_prefix]")

    elif firstsyllable in ("pa"):
        newState = "Class_16_state"
        #print(firstsyllable + " [Class 16 prefix]")
        self.structure.append(firstsyllable + " [Class 16 Noun_prefix]")
    elif firstsyllable in ("svi"):
```

```
            newState = "Class_19_state"
            #print(firstsyllable + " [Class 19 prefix]")
            self.structure.append(firstsyllable + " [Class 19 Noun_prefix]")
        elif firstsyllable in ("zi"):
            newState = "Class_21_state"
            #print(firstsyllable + " [Class 21 prefix]")
            self.structure.append(firstsyllable + " [Class 21 Noun_prefix]")
        elif  firstsyllable in noun_prefixes:
            newState = "noun_prefixes_state"
            ##print(firstsyllable + " [Negation or Mood]")
            self.structure.append(firstsyllable + " [Noun_prefix]")
        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun]")

        return(newState, txt)

    def genitive_transitions(self,txt):
        orgTxt = txt
        firstsyllable, txt = self.splitfirstsyllable(txt)
        # #print(firstsyllable+ " [First Syllable]")
        if  firstsyllable == "mu":              #noun_prefixes:
            newState = "Class_1_or_3_state"
            #print(firstsyllable + " [Class 1 or 3]")
            self.structure.append(firstsyllable + " [Class 1 or 3 Noun_prefix]")
        elif firstsyllable == "va":
            nxtsyllable, rst = self.splitfirstsyllable(txt)
            if nxtsyllable == "na":
                newState = "Class_2b_state"
                #print(firstsyllable + nxtsyllable + " [Class 2b]")
                self.structure.append(firstsyllable + " [Class 2b Noun_prefix]")
                txt = rst
            else:
                newState = "Class_2_state"
                #print(firstsyllable + " [Class 2]")
                self.structure.append(firstsyllable + " [Class 2 Noun_prefix]")
        elif firstsyllable == "ma":
            nxtsyllable, rst = self.splitfirstsyllable(txt)
            if nxtsyllable == "dzi":
                newState  = "Class_6_or_10_state"
                #print(firstsyllable + nxtsyllable + " [Class 6]")
                self.structure.append(firstsyllable + " [Class 6 Noun_prefix]")
                txt = rst
            else:
                newState = "Class_6_or_10_state"
                #print(firstsyllable + " [Class 6 or 10]")
                self.structure.append(firstsyllable + " [Class 6 Noun_prefix]")
        elif firstsyllable == "mi":
            newState = "Class_4_state"
            #print(firstsyllable + " [Class 4]")
            self.structure.append(firstsyllable + " [Class 4 Noun_prefix]")
        elif firstsyllable in ("chi","cha"):
            newState = "Class_7_state"
            #print(firstsyllable + " [Class 7]")
            self.structure.append(firstsyllable + " [Class 7 Noun_prefix]")
```

```
elif firstsyllable in ("zvi","zva","zvu"):
    newState = "Class_7_state"
    #print(firstsyllable + " [Class 8 prefix]")
    self.structure.append(firstsyllable + " [Class 8 Noun_prefix]")
elif firstsyllable in ("ru","rwa","gwa", "rwe", "gwe", "gu"):
    newState = "Class_11_state"
    #print(firstsyllable + " [Class 11 prefix]")
    self.structure.append(firstsyllable + " [Class 11 Noun_prefix]")
elif firstsyllable == "ka":
    newState = "Class_12_state"
    #print(firstsyllable + " [Class 12 prefix]")
    self.structure.append(firstsyllable + " [Class 12 Noun_prefix]")
elif firstsyllable == "tu":
    newState = "Class_12_state"
    #print(firstsyllable + " [Class 12 prefix]")
    self.structure.append(firstsyllable + " [Class 12 Noun_prefix]")
elif firstsyllable in ("u","hu"):
    newState = "Class_14_state"
    #print(firstsyllable + " [Class 14 prefix]")
    self.structure.append(firstsyllable + " [Class 14 Noun_prefix]")

elif firstsyllable in ("ku"):
    newState = "Class_15_or_17_state"
    #print(firstsyllable + " [Class 15 or 17 prefix]")
    self.structure.append(firstsyllable + " [Class 15 or 17 Noun_prefix]")

elif firstsyllable in ("pa"):
    newState = "Class_16_state"
    #print(firstsyllable + " [Class 16 prefix]")
    self.structure.append(firstsyllable + " [Class 16 Noun_prefix]")
elif firstsyllable in ("svi"):
    newState = "Class_19_state"
    #print(firstsyllable + " [Class 19 prefix]")
    self.structure.append(firstsyllable + " [Class 19 Noun_prefix]")
elif firstsyllable in ("zi"):
    newState = "Class_21_state"
    #print(firstsyllable + " [Class 21 prefix]")
    self.structure.append(firstsyllable + " [Class 21 Noun_prefix]")
elif  firstsyllable in noun_prefixes:
    newState = "noun_prefixes_state"
    ##print(firstsyllable + " [Negation or Mood]")
    self.structure.append(firstsyllable + " [Noun_prefix]")

elif  firstsyllable in genitive_prefixes:
    newState = "genitive_prefixes_state"     #Otherwise write Genitive Transitions method
    #print(firstsyllable + " [Genitive Prefix]")
    self.structure.append(firstsyllable + " [Genitive_prefix]")


else:
    txt = orgTxt
    newState = "noun_stem_state"
    self.structure.append( "[]" + " Noun Prefix]")

return(newState, txt)
```

```
    def noun_stem_transitions(self,txt):
        if txt in self.class_1_nouns:
            newState = "End_State"
            self.structure.append(txt + " [Class 1 Noun stem]")
        elif txt in self.class_1a_nouns:
            newState = "End_State"
            self.structure.append(txt + " [Class 1 Noun stem]")
        elif txt in self.class_5_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 5 Noun stem]")
        elif txt in self.class_9_stems:
            newState = "End_State"
            self.structure.append(txt + " [Class 9 Noun stem]")
        else:
            newState = "error_state"
            self.structure.append(txt + " [Unknown Noun]")

        return(newState, txt)



    def end_state_transitions(self,package):
        done =1
        #print("Noun Accepted with root ", package[0], " and extensions ",package[1], "and final vowel
",package[2])

        return done

    def spellCheck(self,txt):
        retValue = self.run(txt)
        if retValue:
            retValue = True
        else:
            retValue = False
        return retValue

    def get_class_data():

        nounClasses = pkl.load(open("e:/data/programming/python/Shona NLP/Language
Modelling/Dictionaries/nounClasses.p","rb"))

        class_1_nouns,class_1a_nouns, class_1b_nouns, class_2_nouns, class_2a_nouns,
class_2b_nouns, class_3_nouns, class_4_nouns, class_5_nouns, class_6_nouns, class_7_nouns,
class_8_nouns, class_9_nouns, class_10_nouns, class_11_nouns, class_12_nouns, class_13_nouns,
class_14_nouns, class_15_nouns, class_16_nouns, class_17_nouns, class_17a_nouns,
class_18_nouns, class_19_nouns, class_20_nouns, class_21_nouns = nounClasses

        return nounClasses
```

Listing 4 - Finite State Automata – using Bernd Klein's code

```
"""
Created on Sun Jul  4 19:30:50 2021

@author: Farayi
"""


import tkinter as tk
from tkinter import messagebox
import pygubu
import sys
import Trie as t3
import string
import ShonaVerb as sv
import ShonaNoun as sn
import ctypes


v=sys.version
if "2.7" in v:
    import Tkinter as tk
    from Tkinter import *
elif "3." in v:
    import tkinter as tk
    from tkinter import *
    from  tkinter.filedialog import FileDialog

from tkinter.filedialog import askopenfilename, asksaveasfilename


def get_char_3grams(inword):
    char_3grams = [inword[i:i+3].lower() for i in range(len(inword)-1)]

    return char_3grams


def find_error(test_word, ngram_dict):
    error_list = [w for w in get_char_3grams(test_word) if w not in ngram_dict]
    return error_list


def spell_check(test_word, ngram_dict):
    result = (find_error( test_word,ngram_dict) == [])
    return result


def openDictionary():
    with open('dictionaries/Shona_words.txt') as f: #'DGR_vocab.txt') as f: # mazwi_eduramazwi
        words_dict = f.read().splitlines()
    return words_dict

def shonaTokeniser(mazwi):
```

```
    ch=""
    wrd = ""
    shonaTokens = []
    for i in range(len(mazwi)):
        ch=mazwi[i]
        if (ch.isalpha()) or (ch.isnumeric()) or (ch == """) or (ch == "’"):
            wrd=wrd+ch
        elif ch != " ":
            if wrd != "":
                shonaTokens.append(wrd)
            if ch in string.punctuation:
                if ch in ["", "’"]:
                    wrd = wrd+ ch
                else:
                    shonaTokens.append(ch)
            wrd=""
        else:
            if wrd != "":
                shonaTokens.append(wrd)
            wrd = ""
    return shonaTokens


def open_file():
    """Open a file for editing."""
    filepath = askopenfilename(
        filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")]
    )
    if not filepath:
        return
    txt_edit.delete(1.0, tk.END)
    with open(filepath, "r", encoding='utf-8', errors="ignore") as input_file:
        text = input_file.read()
        txt_edit.insert(tk.END, text)
    root.title(f"Text Editor Application - {filepath}")

def splitfirstsyllable(txt):
    firstSyllable=""
    restOftext=""
    vowels={"a","e","i","o","u"}
    if len(txt) ==1 :
        firstSyllable=txt
    else:
        if len(txt) > 0:
            startChar=txt[0]
            #print("startChar is : " +startChar )
            if startChar in vowels:
                firstSyllable = startChar
                restOftext = txt[1:]
            else:
                i=1
                nextChar=txt[i]
                firstSyllable=startChar
                while (nextChar not in vowels) and ( (i+1) < len(txt)):
                    firstSyllable= firstSyllable+nextChar
```

```
            i+=1
            nextChar=txt[i]
        if (i+1) != len(txt):
            firstSyllable=firstSyllable+nextChar
            restOftext = txt[len(firstSyllable):]
    return firstSyllable, restOftext


def saveas():
    global txt_edit
    t = txt_edit.get("1.0","end-1c")
    savelocation=asksaveasfilename()
    file1=open(savelocation,"w+")
    file1.write(t)
    file1.close()
    root.title(f"Text Editor Application - {savelocation}")

def syllabify(txt):
    syllables = []
    firstsyllable, restoftxt = splitfirstsyllable(txt)
    while len(restoftxt) > 0:
        syllables.append(firstsyllable)
        firstsyllable, restoftxt = splitfirstsyllable(restoftxt)
        #print(firstsyllable)
    if len(firstsyllable) > 0:
        syllables.append(firstsyllable + txt[-1:])
    return syllables

def logical_xor(str1, str2):
    return bool(str1) ^ bool(str2)

def hasValidSyllables(txt):
    itdoes = True
    chktxt = txt.lower()
    syllables = syllabify(chktxt)

    for syllable in syllables:
        if syllable[:-1] in sv.validSyllables:
            #flash('{} is a valid syllable.'.format(syllable))
            itdoes= itdoes
        elif syllable in sv.vowels:
            #flash('{} is a valid syllable.'.format(syllable))
            itdoes= itdoes
        else:
            #flash('{} is not a valid syllable'.format(syllable))
            itdoes= False
    #flash('{} has been checked and the statement of whether it  has valid syllables has been found to
be {}'.format(txt,itdoes))
    return itdoes



def hasValidSubwords(txt):
    word_list = sv.openDictionary()
    found = False
```

```
    restoftxt = txt
    while (not found) and (len(restoftxt) > 2):
        firstsyllable, restoftxt = splitfirstsyllable(restoftxt)
        #flash('Split into <{}> and <{}>.'.format(firstsyllable, restoftxt))
        subword = restoftxt
        if subword in word_list:
            #print('{} is a valid subword'.format(subword))
            found = True
        else:
            found=False
    #flash('{} has been checked and the statement of whether it has valid subwords has been found to
be {}'.format(txt,found))
    return found



def hasValidStem(txt):
    #words_list = sv.openDictionary()
    stem_list = sv.openStemDictionary()
    found = False
    restoftxt = txt
    while (not found) and (len(restoftxt) > 2):
        firstsyllable, restoftxt = splitfirstsyllable(restoftxt)
        #flash('Split into <{}> and <{}>.'.format(firstsyllable, restoftxt))
        stem = restoftxt[:-1]
        stem = "-" + stem + "a"
        if stem in stem_list:
            #flash('{} is a valid stem'.format(stem))
            found = True
        else:
            found=False
    #flash('{} has been checked and the statement of whether it has valid stems has been found to be
{}'.format(txt,found))
    return found


def peretera(txt, words_list):
    shv = sv.ShonaVerb()
    #words_list = shv.openDictionary()
    if txt.isnumeric() or (txt in string.punctuation):
        found = True
    else:
        if txt in words_list:
            print('Word {} found in our our dictionary'.format(txt))
            found = True
        else:
            found = shv.spellCheck(txt)
            if found:
                print('The word {} is not in the dictionary but was found via verb morphological
analysis.'.format(txt))
            else:
                found = shv.spellCheck(txt)
                if found:
                    print('The word {} is not in the dictionary but was found via verb morphological
analysis.'.format(txt))
```

```
            if not found:
                shn = sn.ShonaNoun()
                found = shn.spellCheck(txt)
                if found:
                    print('The word {} is not in the dictionary but was found via noun morphological
analysis.'.format(txt))
                else:
                    print('The word {} is not in the dictionary and could not be re-created via both verb and
noun morphological analysis.'.format(txt))

        #if (hasValidStem(txt)) : #or hasValidSubwords(txt)) : #or hasValidSyllables(txt):
        #    found = True
        #else:
        #    found = False
        #if not found:
        #    print('The word {} is not a known word.'.format(txt))
    return_val = found
    return return_val

def spellcheck():
    global txt_edit
    t = txt_edit.get("1.0",'end-1c')
    contents = shonaTokeniser(t)
    for w in contents:
        is_found = trie_dictionary.query(w.lower())
        if not is_found:
            if not peretera(w.lower(), wd):
                txt_edit.tag_configure("red", foreground="#ff0000")
                txt_edit.highlight_pattern(w, "red")
            #print(w + " is an invalid word.")
            #TODO: Implement Suggestion generator
    MessageBox = ctypes.windll.user32.MessageBoxW
    MessageBox(None, 'Done', 'Spell Checking Progress', 0)


def open_file():
    """Open a file for editing."""
    filepath = askopenfilename(
        filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")]
    )
    if not filepath:
        return
    txt_edit = self.builder.get_variable('OriginalText')
    txt_edit.delete(1.0, tk.END)
    with open(filepath, "r") as input_file:
        text = input_file.read()
        txt_edit.insert(tk.END, text)
    root.title(f"Text Editor Application - {filepath}")


class CompareSpell:

    def __init__(self):

        #1: Create a builder
```

```
        self.builder = builder = pygubu.Builder()

        #2: Load an ui file
        builder.add_from_file('comparespell.ui')

        #3: Create the mainwindow
        self.mainwindow = builder.get_object('mainwindow')

        builder.connect_callbacks(self)


        self.open_button_object = builder.get_object('Openbtn', self.mainwindow)

    def on_button1_clicked(self):

        """Open a file for editing."""
        filepath = askopenfilename(
            filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")]
        )
        if not filepath:
            return
        txt_edit = self.builder.get_object('OriginalText')

        maOutput = self.builder.get_object('MABasedSpellChecker')
        nGramOutput = self.builder.get_object('NGramBased')

        outmsg = self.builder.get_object('ma_msg')
        nGramOutmsg = self.builder.get_object('n_gram_msg')

        unKnownCount = self.builder.get_object('Unknown_count')

        #messagebox.showinfo('Message','SUcces with text_edit')
        txt_edit.delete(1.0, tk.END)
        with open(filepath, "r") as input_file:
            text = input_file.read()
            txt_edit.delete(1.0, tk.END)
            txt_edit.insert(tk.END, text)

            maOutput.delete(1.0, tk.END)
            nGramOutput.delete(1.0, tk.END)

            outmsg.config(text="")
            nGramOutmsg.config(text = "")
            unKnownCount.config(text = "")


    def on_button2_clicked(self):
        #messagebox.showinfo('Message', 'You clicked The Spell Check Button')
        txt_edit = self.builder.get_object('OriginalText')
        outmsg = self.builder.get_object('ma_msg')
        nGramOutmsg = self.builder.get_object('n_gram_msg')
        maOutput = self.builder.get_object('MABasedSpellChecker')
        nGramOutput = self.builder.get_object('NGramBased')

        #maOutput.delete(1.0, tk.END)
```

```
        #nGramOutput.delete(1.0, tk.END)

        unKnownCount = self.builder.get_object('Unknown_count')

        t = txt_edit.get("1.0",'end-1c')
        #maOutput.insert(tk.END, t)
        contents = shonaTokeniser(t)
        num_found = 0
        ngram_found = 0
        not_in_dict = 0
        OovWords = []

        for w in contents:
            is_found = trie_dictionary.query(w.lower())
            if not is_found:
                OovWords.append(w.lower())
                not_in_dict += 1
                if not peretera(w.lower(),wd):

                    txt_edit.tag_configure("red", foreground="#ff0000")
                    #txt_edit.highlight_pattern(w, "red")

                    #maOutput.tag_configure("red", foreground="#ff0000")
                    maOutput.insert(tk.END, w)
                    maOutput.insert(tk.END, "\r\n")
                    num_found +=1
                    #maOutput.highlight_pattern(w, "red")
                #print(w + " is an invalid word.")
                #TODO: Implement Suggestion generator
            if not spell_check(w.lower(), counts):
                nGramOutput.insert(tk.END, w)
                nGramOutput.insert(tk.END, "\r\n")
                ngram_found +=1
        UniqueOoV = len(set(OovWords)     )
        MessageBox = ctypes.windll.user32.MessageBoxW
        MessageBox(None, 'Done', 'Spell Checking Progress', 0)
        msg_txt = "There are " + str(num_found) + " unknown words that were marked as incorrect by
the Morphological Analyser based spell checker."
        nGram_txt = str(ngram_found) +" incorrect words found using character n-gram spell checker."
        OoVmsg = str(UniqueOoV) + " unique Out of Vocabulary words not found a total of " +
str(not_in_dict) + " times in text."

        outmsg.config(text=msg_txt)
        nGramOutmsg.config(text = nGram_txt)
        unKnownCount.config(text = OoVmsg)




    def run(self):
        self.mainwindow.mainloop()
```

```
if __name__ == '__main__':
    wd = openDictionary()
    trie_dictionary=t3.Trie()
    for w in wd:
        trie_dictionary.insert(w.lower())



    corpora = open("dictionaries/Shona_words.txt","r")  #"mazwi_eduramazwi_akapepetwa.txt", "r") #,
encoding="utf8")
    corpora_text = corpora.read().splitlines()
    #char_3grams = [corpora_text[i:i+3] for i in range(len(corpora_text)-1)]
    char_3grams = [w[i:i+3].lower() for w in corpora_text for i in range(len(w)-1)]
    #d = {x:char_3grams.count(x) for x in char_3grams}
    counts = dict()
    for i in char_3grams:
      counts[i] = counts.get(i, 0) + 1


    app = CompareSpell()
    app.run()
```

Appendix 2 – Results of Mini Experiment on limitations of Google Translate

All screenshots retrieved on 5 April 2020 between 12:41 and 13:03 SAST

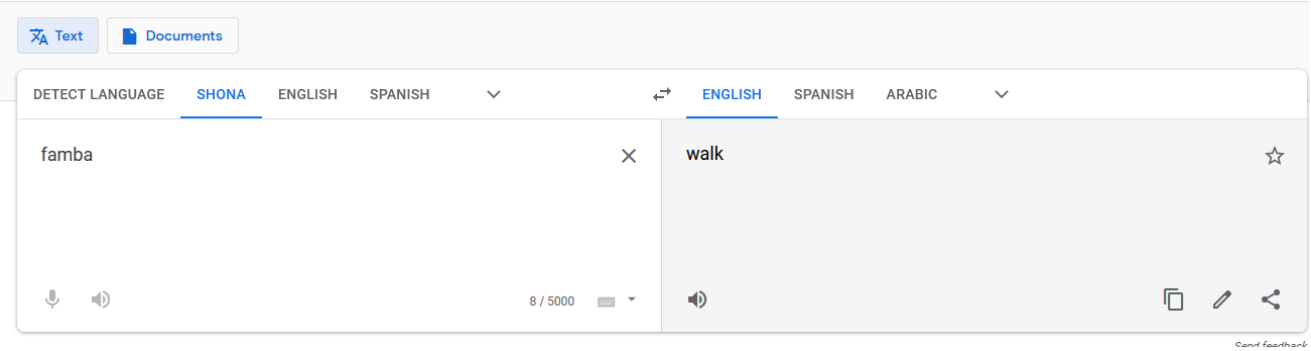


Figure 0-1- Google Translate's translation of the Shona verb "famba" (walk) to English

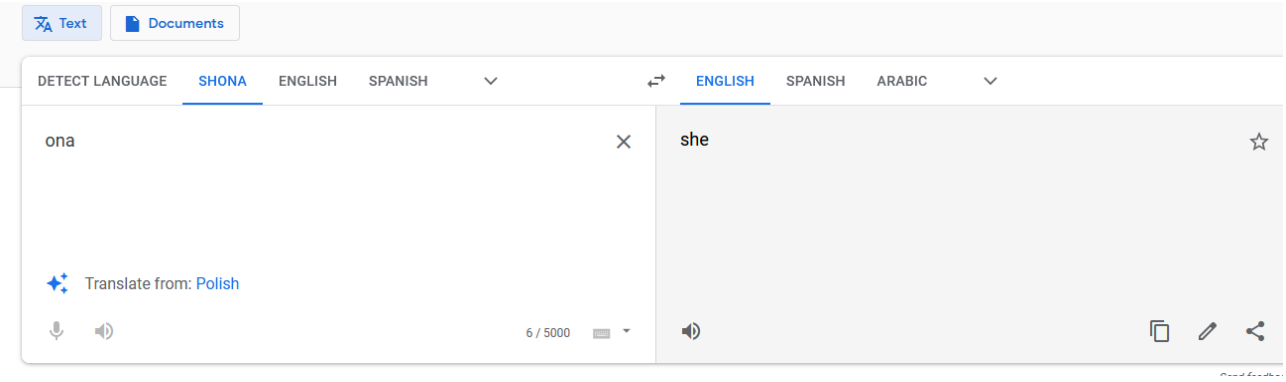


Figure 0-2- Google Translate's translation of the Shona verb "ona"(see) to English

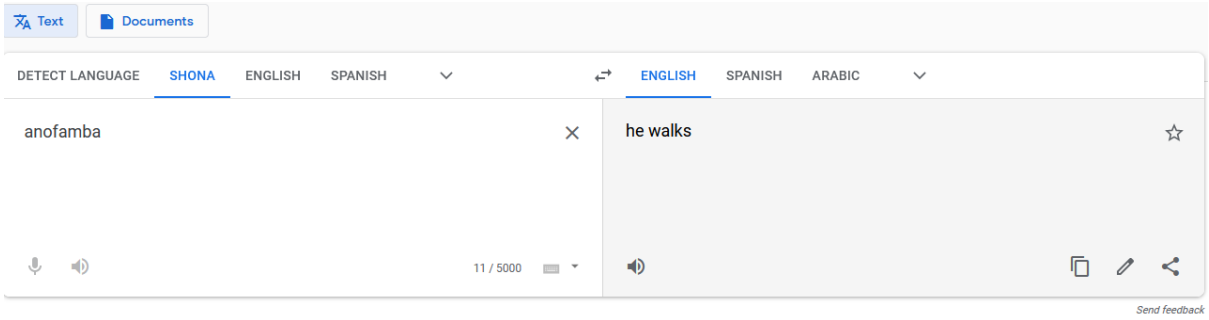


Figure 0-3- Google Translate's translation of the Shona verb "anofamba" (s/he walks) to English

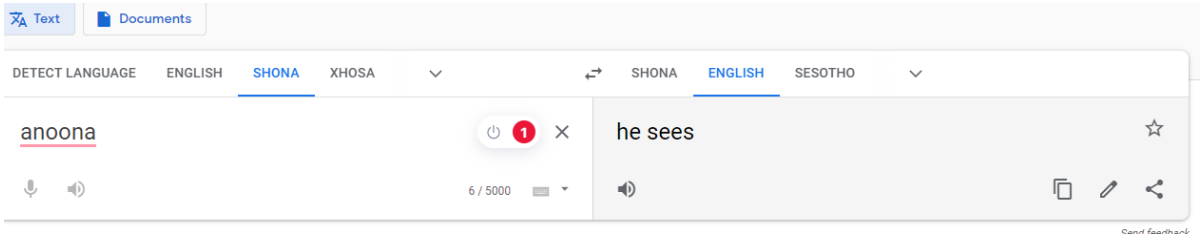


Figure 0-4- Google Translate's translation of the Shona verb "anoona"(he sees) to English

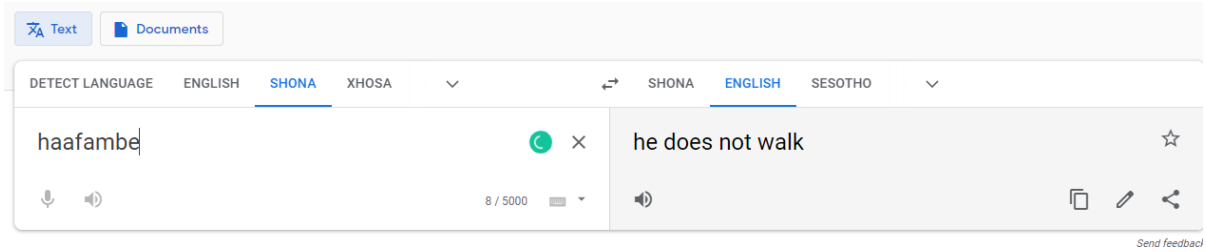


Figure 0-5 - Google Translate's translation of the Shona verb "haafambe" (s/he does not walk) to English

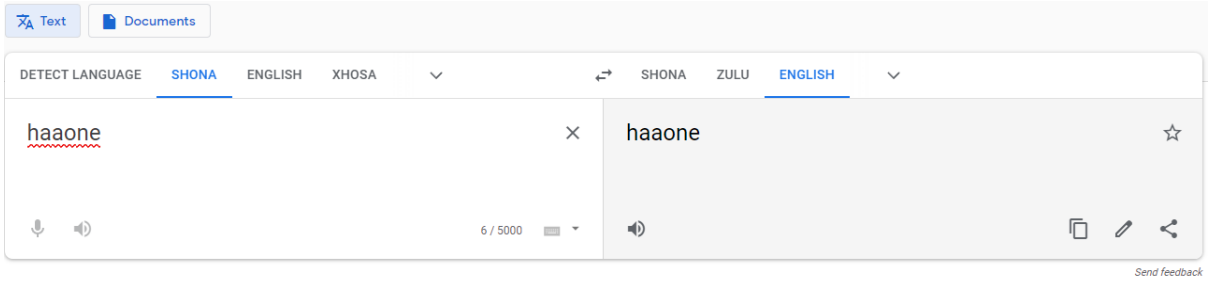


Figure 0-6 - Google Translate's translation of the Shona verb "haaone" (s/he does not see) into English

Appendix 3 – Sample Data – CTLM versus MAShoKO based spell checker results

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
Ndapedza	Valid - Shona word	No	Yes	Yes	Yes
ndokukoromoka	Valid - Shona word	Yes	Yes	Yes	Yes
pasingapihwe	Valid - Shona word	Yes	Yes	Yes	Yes
dzakapwa	Valid - Shona word	Yes	Yes	Yes	Yes
VECHIDIKI	Valid - Shona word	No	Yes	Yes	Yes
naVincent	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
Marwei	Valid - Shona word	No	Yes	Yes	Yes
yemuAfrica	Valid - Shona plus Borrowed Word	Yes	Yes	Yes	No
Kuzvibata	Valid - Shona word	No	Yes	Yes	Yes
dzekunowanikwa	Valid - Shona word	Yes	Yes	Yes	Yes
nyararai	Valid - Shona word	Yes	Yes	Yes	Yes
Emilia	Valid - Borrowed Word	Yes	No	Yes	No
yeNyamatikiti	Valid - Shona word	Yes	Yes	Yes	Yes
eNASH	Valid - Shona plus Borrowed Word	Yes	No	Yes	Yes
inombonyimwe	Valid - Shona word	Yes	Yes	Yes	Yes
mumagwaro	Valid - Shona word	Yes	Yes	Yes	Yes
kutakishopu	Valid - Shona word	Yes	Yes	Yes	Yes
mhosho	Valid - Shona word	No	Yes	Yes	Yes
ndichiriwana	Valid - Shona word	Yes	Yes	Yes	Yes
tinozvinzwira	Valid - Shona word	Yes	Yes	Yes	Yes
yakaisa	Valid - Shona word	Yes	Yes	Yes	Yes
resimbi	Valid - Shona word	No	Yes	Yes	Yes
tozopa	Valid - Shona word	Yes	Yes	Yes	Yes
chakaora	Valid - Shona word	No	Yes	Yes	Yes

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
Enifa	Valid - Shona word	Yes	Yes	Yes	Yes
rwunoyerera	Valid - Shona word	Yes	Yes	Yes	Yes
yandataura	Valid - Shona word	Yes	Yes	Yes	Yes
dzenyaya	Valid - Shona word	Yes	Yes	Yes	Yes
vachibvunzana	Valid - Shona word	No	Yes	Yes	Yes
veChirovarova	Valid - Shona word	Yes	Yes	Yes	Yes
parege	Valid - Shona word	No	Yes	Yes	Yes
tichisanganisira	Valid - Shona word	Yes	Yes	Yes	Yes
zvaimuda	Valid - Shona word	Yes	Yes	Yes	Yes
uchizoguma	Valid - Shona word	Yes	Yes	Yes	Yes
Soocer	Valid - Borrowed Word	Yes	No	Yes	No
nekeratin	Valid - Shona word	Yes	Yes	Yes	Yes
pevanonzi	Valid - Shona word	Yes	Yes	Yes	Yes
chekurembera	Valid - Shona word	Yes	Yes	Yes	Yes
Hazvisati	Valid - Shona word	Yes	Yes	Yes	Yes
full	Valid - Borrowed Word	Yes	No	Yes	No
zvavakazviitira	Valid - Shona word	Yes	Yes	Yes	Yes
pevandinoshanda	Valid - Shona word	Yes	Yes	Yes	Yes
vaimbonamata	Valid - Shona word	Yes	Yes	Yes	Yes
Idzowo	Valid - Shona word	Yes	Yes	Yes	Yes
uchishaika	Valid - Shona word	Yes	Yes	Yes	Yes
Goko	Valid - Shona word	No	Yes	Yes	Yes
zidza	Valid - Shona word	Yes	Yes	Yes	Yes
vanowanzobatwa	Valid - Shona word	Yes	Yes	Yes	Yes
achahwinhwa	Valid - Shona word	Yes	Yes	Yes	Yes
point	Valid - Borrowed Word	Yes	No	Yes	Yes
pachiito	Valid - Shona word	No	Yes	Yes	Yes

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
nevanhukadzi	Valid - Shona word	No	Yes	Yes	Yes
pakamboita	Valid - Shona word	Yes	Yes	Yes	Yes
Mukoki	Valid - Shona word	Yes	Yes	Yes	Yes
panomuka	Valid - Shona word	Yes	Yes	Yes	Yes
mudota	Valid - Shona word	No	Yes	Yes	Yes
achitsotsa	Valid - Shona word	Yes	Yes	Yes	Yes
naSaki	Valid - Shona word	Yes	Yes	Yes	Yes
ikangopinda	Valid - Shona word	Yes	Yes	Yes	Yes
mumivhirinyimo	Valid - Shona word	Yes	Yes	Yes	Yes
cheCelebration	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
eVictoria	Valid - Shona plus Borrowed Word	No	Yes	Yes	No
kanoratidza	Valid - Shona word	No	Yes	Yes	Yes
Achatapurira	Valid - Shona word	Yes	Yes	Yes	Yes
zvamanzi	Valid - Shona word	Yes	Yes	Yes	Yes
rezviuru	Valid - Shona word	Yes	Yes	Yes	Yes
dzekunyora	Valid - Shona word	Yes	Yes	Yes	Yes
regunera	Valid - Shona word	Yes	Yes	Yes	Yes
kutokutadzisa	Valid - Shona word	Yes	Yes	Yes	Yes
ndakaidya	Valid - Shona word	Yes	Yes	Yes	Yes
naMadzibaba	Valid - Shona word	Yes	Yes	Yes	Yes
neBesiktas	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
zvakatarwa	Valid - Shona word	Yes	Yes	Yes	Yes
dzokubudirira	Valid - Shona word	Yes	Yes	Yes	Yes
Tavapa	Valid - Shona word	Yes	Yes	Yes	Yes
ngarwuvewo	Valid - Shona word	Yes	No	Yes	Yes
vakageza	Valid - Shona word	Yes	Yes	Yes	Yes
adoma	Valid - Shona word	Yes	Yes	Yes	Yes

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
chakurumidza	Valid - Shona word	No	Yes	Yes	Yes
vatinoti	Valid - Shona word	Yes	Yes	Yes	Yes
nekusangana	Valid - Shona word	No	Yes	Yes	Yes
rokuberekwa	Valid - Shona word	Yes	Yes	Yes	Yes
vavhiringike	Valid - Shona word	Yes	Yes	Yes	Yes
Parinobuda	Valid - Shona word	Yes	Yes	Yes	Yes
akangopupa	Valid - Shona word	Yes	Yes	Yes	Yes
Mitengo	Valid - Shona word	No	Yes	Yes	Yes
kurwiwa	Valid - Shona word	No	Yes	Yes	Yes
muteereri	Valid - Shona word	No	Yes	Yes	Yes
Muroti	Valid - Shona word	No	Yes	Yes	Yes
description	Valid - Borrowed Word	Yes	No	Yes	No
Vanin'ina	Valid - Shona word	No	Yes	Yes	Yes
wekwaSabhuku	Valid - Shona word	Yes	Yes	Yes	Yes
yaaiziva	Valid - Shona word	Yes	Yes	Yes	Yes
yekuatenga	Valid - Shona word	Yes	Yes	Yes	Yes
takanganwa	Valid - Shona word	Yes	Yes	Yes	Yes
vakaichengeta	Valid - Shona word	Yes	Yes	Yes	Yes
duster	Valid - Borrowed Word	Yes	No	Yes	No
vakamboungana	Valid - Shona word	Yes	Yes	Yes	Yes
nekuongorora	Valid - Shona word	Yes	Yes	Yes	Yes
isasangane	Valid - Shona word	Yes	Yes	Yes	Yes
rakataura	Valid - Shona word	Yes	Yes	Yes	Yes
tikunde	Valid - Shona word	Yes	Yes	Yes	Yes
azvitsvagira	Valid - Shona word	Yes	Yes	Yes	Yes
platinum	Valid - Borrowed Word	Yes	No	Yes	No
pandakarohwa	Valid - Shona word	Yes	Yes	Yes	Yes

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
vakadzinyina	Valid - Shona word	Yes	Yes	Yes	Yes
Mercedes	Valid - Borrowed Word	Yes	No	Yes	No
weSt	Valid - Shona plus Borrowed Word	Yes	No	Yes	Yes
vachingokosora	Valid - Shona word	Yes	Yes	Yes	Yes
achitozvirwarira	Valid - Shona word	Yes	Yes	Yes	Yes
dzakaparwa	Valid - Shona word	Yes	Yes	Yes	Yes
nechijana	Valid - Shona word	No	Yes	Yes	Yes
dzingange	Valid - Shona word	Yes	Yes	Yes	Yes
Manicaland	Valid - Borrowed Word	Yes	No	Yes	No
haandiremekedzewo	Valid - Shona word	Yes	Yes	Yes	Yes
muvadiki	Valid - Shona word	Yes	Yes	Yes	Yes
ndakakundwa	Valid - Shona word	Yes	Yes	Yes	Yes
vozopedzera	Valid - Shona word	Yes	Yes	Yes	Yes
Leeroy	Valid - Borrowed Word	Yes	No	Yes	No
vaitimba	Valid - Shona word	Yes	Yes	Yes	Yes
ukachiona	Valid - Shona word	No	Yes	Yes	Yes
dzinozvitsvakira	Valid - Shona word	Yes	Yes	Yes	Yes
hwekugara	Valid - Shona word	Yes	Yes	Yes	Yes
kuMambas	Valid - Shona plus Borrowed Word	Yes	No	Yes	Yes
wamboshandisa	Valid - Shona word	Yes	Yes	Yes	Yes
mumatumbu	Valid - Shona word	Yes	Yes	Yes	Yes
kwamunositorera	Valid - Shona word	Yes	Yes	Yes	Yes
yemaFungicides	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
sekuZinatha	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
pamukangesi	Valid - Shona word	Yes	Yes	Yes	Yes
kutokwangwayawo	Valid - Shona word	Yes	Yes	Yes	Yes
Start	Valid - Borrowed Word	Yes	No	Yes	No

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
Machinda	Valid - Shona word	No	Yes	Yes	Yes
Jael	Valid - Borrowed Word	Yes	No	Yes	No
ritinakire	Valid - Shona word	Yes	Yes	Yes	Yes
madzimai	Valid - Shona word	No	Yes	Yes	Yes
vakadanana	Valid - Shona word	Yes	Yes	Yes	Yes
paakazodzoka	Valid - Shona word	Yes	Yes	Yes	Yes
Pavanonzi	Valid - Shona word	Yes	Yes	Yes	Yes
Makara	Valid - Shona word	No	Yes	Yes	Yes
kuriziva	Valid - Shona word	Yes	Yes	Yes	Yes
chituko	Valid - Shona word	No	Yes	Yes	Yes
kuida	Valid - Shona word	No	Yes	Yes	Yes
haachisisina	Valid - Shona word	Yes	Yes	Yes	Yes
kambotanga	Valid - Shona word	Yes	Yes	Yes	Yes
kumadambarefu	Valid - Shona word	Yes	Yes	Yes	Yes
maintained	Valid - Borrowed Word	Yes	No	Yes	No
zvichavakwa	Valid - Shona word	Yes	Yes	Yes	Yes
ndichinokecha	Valid - Shona word	Yes	Yes	Yes	Yes
vakafema	Valid - Shona word	Yes	Yes	Yes	Yes
aronge	Valid - Shona word	Yes	Yes	Yes	Yes
nana	Valid - Shona word	No	Yes	Yes	Yes
neEngineering	Valid - Shona word	Yes	Yes	Yes	Yes
Kadora	Valid - Shona word	Yes	Yes	Yes	Yes
chawandipa	Valid - Shona word	Yes	Yes	Yes	Yes
akazobata	Valid - Shona word	Yes	Yes	Yes	Yes
vanobvunzira	Valid - Shona word	Yes	Yes	Yes	Yes
rakarukwa	Valid - Shona word	No	Yes	Yes	Yes
Hunzi	Valid - Shona word	No	Yes	Yes	Yes

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
nezvidza	Valid - Shona word	Yes	Yes	Yes	Yes
ndakatarisira	Valid - Shona word	Yes	Yes	Yes	Yes
thinking	Valid - Borrowed Word	Yes	No	Yes	No
nekumweya	Valid - Shona word	Yes	Yes	Yes	Yes
nepafafitera	Valid - Shona word	Yes	Yes	Yes	Yes
kwaVaChivako	Valid - Shona word	Yes	Yes	Yes	Yes
Kabasa	Valid - Shona word	No	Yes	Yes	Yes
Nyaruvenda	Valid - Shona word	Yes	Yes	Yes	Yes
kubatsirwa	Valid - Shona word	No	Yes	Yes	Yes
wakapfunya	Valid - Shona word	No	Yes	Yes	Yes
dzeboka	Valid - Shona word	Yes	Yes	Yes	Yes
nomuimbi	Valid - Shona word	Yes	Yes	Yes	Yes
ndinoyambira	Valid - Shona word	Yes	Yes	Yes	Yes
Munomirira	Valid - Shona word	Yes	Yes	Yes	Yes
anovamirira	Valid - Shona word	Yes	Yes	Yes	Yes
yezviperengo	Valid - Shona word	No	Yes	Yes	Yes
vaendeswe	Valid - Shona word	Yes	Yes	Yes	Yes
kwakaradzikwawo	Valid - Shona word	Yes	Yes	Yes	Yes
wevapi	Valid - Shona word	Yes	Yes	Yes	Yes
rwakaserera	Valid - Shona word	Yes	Yes	Yes	Yes
ndokubhinya	Valid - Shona word	Yes	Yes	Yes	Yes
pesitendi	Valid - Shona word	Yes	Yes	Yes	Yes
chinovigwa	Valid - Shona word	Yes	Yes	Yes	Yes
apinda	Valid - Shona word	No	Yes	Yes	Yes
Nhoro	Valid - Shona word	No	Yes	Yes	Yes
pazvinotaura	Valid - Shona word	Yes	Yes	Yes	Yes
ngatidzvare	Valid - Shona word	Yes	Yes	Yes	Yes

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
vachimutenda	Valid - Shona word	Yes	Yes	Yes	Yes
Mhishi	Valid - Shona word	No	Yes	Yes	Yes
nemunin’na	Valid - Shona word	Yes	No	Yes	No
anovawanisa	Valid - Shona word	Yes	Yes	Yes	Yes
semutori	Valid - Shona word	Yes	Yes	Yes	Yes
Acid	Valid - Borrowed Word	Yes	No	Yes	No
Kuzvibatsira	Valid - Shona word	No	Yes	Yes	Yes
takabhadharira	Valid - Shona word	Yes	Yes	Yes	Yes
dzekutaridza	Valid - Shona word	Yes	Yes	Yes	Yes
rwavanhu	Valid - Shona word	No	Yes	Yes	Yes
nekatsamba	Valid - Shona word	Yes	Yes	Yes	Yes
zvinoparara	Valid - Shona word	Yes	Yes	Yes	Yes
Chikosoro	Valid - Shona word	No	Yes	Yes	Yes
kutogadza	Valid - Shona word	Yes	Yes	Yes	Yes
tigogamuchira	Valid - Shona word	Yes	Yes	Yes	Yes
achingonyangarika	Valid - Shona word	Yes	Yes	Yes	Yes
siyana	Valid - Shona word	No	Yes	Yes	Yes
vacho	Valid - Shona word	No	Yes	Yes	Yes
Paimhanyidzana	Valid - Shona word	Yes	Yes	Yes	Yes
nenji	Valid - Shona word	No	Yes	Yes	Yes
ndichitarisana	Valid - Shona word	Yes	Yes	Yes	Yes
chemuNou	Valid - Shona word	Yes	Yes	Yes	Yes
vemusha	Valid - Shona word	No	Yes	Yes	Yes
vakandituka	Valid - Shona word	Yes	Yes	Yes	Yes
ndirwoka	Valid - Shona word	Yes	Yes	Yes	Yes
pamashandiro	Valid - Shona word	Yes	Yes	Yes	Yes
minyoro	Valid - Shona word	Yes	Yes	Yes	Yes

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
zvavaishandisa	Valid - Shona word	Yes	Yes	Yes	Yes
dzipindire	Valid - Shona word	Yes	Yes	Yes	Yes
anokahadzika	Valid - Shona word	Yes	Yes	Yes	Yes
yekutamba	Valid - Shona word	Yes	Yes	Yes	Yes
ndaratidzwa	Valid - Shona word	Yes	Yes	Yes	Yes
yevhu	Valid - Shona word	No	Yes	Yes	Yes
randisingazive	Valid - Shona word	Yes	Yes	Yes	Yes
pandakasiyana	Valid - Shona word	Yes	Yes	Yes	Yes
ndokuvaoneka	Valid - Shona word	Yes	Yes	Yes	Yes
pakamira	Valid - Shona word	No	Yes	Yes	Yes
vaizotsvaga	Valid - Shona word	Yes	Yes	Yes	Yes
achibongomora	Valid - Shona word	Yes	Yes	Yes	Yes
naMavis	Valid - Shona word	Yes	Yes	Yes	Yes
rinokufarira	Valid - Shona word	Yes	Yes	Yes	Yes
ungade	Valid - Shona word	Yes	Yes	Yes	Yes
bhosvo	Valid - Shona word	No	Yes	Yes	Yes
yavaeni	Valid - Shona word	Yes	Yes	Yes	Yes
ndichinotanga	Valid - Shona word	Yes	Yes	Yes	Yes
wekumushandira	Valid - Shona word	Yes	Yes	Yes	Yes
azonorichekwa	Valid - Shona word	Yes	Yes	Yes	Yes
Manson	Valid - Borrowed Word	Yes	No	Yes	No
wechikwata	Valid - Shona word	No	Yes	Yes	Yes
Shavanhowe	Valid - Shona word	Yes	Yes	Yes	Yes
zvinonoka	Valid - Shona word	Yes	Yes	Yes	Yes
tichitiburana	Valid - Shona word	Yes	Yes	Yes	Yes
sesabhu	Valid - Shona word	Yes	Yes	Yes	Yes
akaitiswa	Valid - Shona word	Yes	Yes	Yes	Yes

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
paNembudziya	Valid - Shona word	Yes	Yes	Yes	Yes
uchinyanyawo	Valid - Shona word	Yes	Yes	Yes	Yes
ndozvavapo	Valid - Shona word	Yes	Yes	Yes	Yes
zvinowaniswa	Valid - Shona word	Yes	Yes	Yes	Yes
zviitogama	Valid - Shona word	Yes	Yes	Yes	Yes
kutishora	Valid - Shona word	Yes	Yes	Yes	Yes
kunzwisisana	Valid - Shona word	No	Yes	Yes	Yes
dzoramba	Valid - Shona word	Yes	Yes	Yes	Yes
semunhenga	Valid - Shona word	Yes	Yes	Yes	Yes
hwe250	Valid - Shona plus Number	Yes	No	Yes	No
raVaJacob	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
rePremier	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
seIndependence	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
rekugashira	Valid - Shona word	Yes	Yes	Yes	Yes
nembanje	Valid - Shona word	Yes	Yes	Yes	Yes
apare	Valid - Shona word	Yes	Yes	Yes	Yes
nditaurire	Valid - Shona word	Yes	Yes	Yes	Yes
nemakavi	Valid - Shona word	No	Yes	Yes	Yes
anotangira	Valid - Shona word	No	Yes	Yes	Yes
wezvekutengeswa	Valid - Shona word	Yes	Yes	Yes	Yes
zvinogaromunetsa	Valid - Shona word	Yes	Yes	Yes	Yes
pakupfuhwira	Valid - Shona word	Yes	Yes	Yes	Yes
hwekuzvipira	Valid - Shona word	Yes	Yes	Yes	Yes
Yasiyana	Valid - Shona word	Yes	Yes	Yes	Yes
musarudzo	Valid - Shona word	No	Yes	Yes	Yes
nekwaBessem	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
ngaaregedze	Valid - Shona word	Yes	Yes	Yes	Yes

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
ndichekwe	Valid - Shona word	Yes	Yes	Yes	Yes
Hatingadi	Valid - Shona word	Yes	Yes	Yes	Yes
chisarumwe	Valid - Shona word	Yes	Yes	Yes	Yes
nefirimu	Valid - Shona word	Yes	Yes	Yes	Yes
veGolden	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
usingafungiri	Valid - Shona word	No	Yes	Yes	Yes
neronga	Valid - Shona word	Yes	Yes	Yes	Yes
akanyatsonyorwa	Valid - Shona word	Yes	Yes	Yes	Yes
wakanamira	Valid - Shona word	Yes	Yes	Yes	Yes
nemadistricts	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
Ndakazosimuka	Valid - Shona word	Yes	Yes	Yes	Yes
publicity	Valid - Borrowed Word	Yes	No	Yes	No
mechimwe	Valid - Shona word	No	Yes	Yes	Yes
Chikowore	Valid - Shona word	Yes	Yes	Yes	Yes
nemadiploma	Valid - Shona plus Borrowed Word	Yes	No	Yes	No
vanozomiswa	Valid - Shona word	Yes	Yes	Yes	Yes
nekusanzwanana	Valid - Shona word	Yes	Yes	Yes	Yes
ndokukasika	Valid - Shona word	Yes	Yes	Yes	Yes
rekuviga	Valid - Shona word	Yes	Yes	Yes	Yes
zvehutsikamutanda	Valid - Shona word	Yes	Yes	Yes	Yes
vavakawana	Valid - Shona word	Yes	Yes	Yes	Yes
havabhadharise	Valid - Shona word	Yes	Yes	Yes	Yes
zvichazarurira	Valid - Shona word	Yes	Yes	Yes	Yes
wangopfuura	Valid - Shona word	No	Yes	Yes	Yes
mabar	Valid - Shona word	Yes	Yes	Yes	Yes
kutevedza	Valid - Shona word	No	Yes	Yes	Yes
vairumiswa	Valid - Shona word	Yes	Yes	Yes	Yes

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
VanaHamadziripi	Valid - Shona word	Yes	Yes	Yes	Yes
haibatsire	Valid - Shona word	Yes	Yes	Yes	Yes
neveAFZ	Valid - Shona plus Borrowed Word	Yes	No	Yes	Yes
wekutapurira	Valid - Shona word	Yes	Yes	Yes	Yes
emberi	Valid - Shona word	No	Yes	Yes	Yes
ndinoswera	Valid - Shona word	Yes	Yes	Yes	Yes
Inherera	Valid - Shona word	Yes	Yes	Yes	Yes
mare	Valid - Shona word	No	Yes	Yes	Yes
Chihuri	Valid - Shona word	No	Yes	Yes	Yes
nemahapa	Valid - Shona word	Yes	Yes	Yes	Yes
Painetsana	Valid - Shona word	Yes	Yes	Yes	Yes
namakereke	Valid - Shona word	Yes	Yes	Yes	Yes
Notorious	Valid - Borrowed Word	Yes	No	Yes	Yes
vanotoziva	Valid - Shona word	Yes	Yes	Yes	Yes
tisarwe	Valid - Shona word	Yes	Yes	Yes	Yes
agozouya	Valid - Shona word	No	Yes	Yes	Yes
Rushiye	Valid - Shona word	No	Yes	Yes	Yes
himself	Valid - Borrowed Word	Yes	No	Yes	No
ndakazomushevedza	Valid - Shona word	Yes	Yes	Yes	Yes
dzekumatendere	Valid - Shona word	Yes	Yes	Yes	Yes
mandiro	Valid - Shona word	Yes	Yes	Yes	Yes
Madhambu’	Incorrect - tokenisation error	Yes	No	No	No
kakazotanga	Valid - Shona word	Yes	Yes	Yes	Yes
chakatakura	Valid - Shona word	No	Yes	Yes	Yes
dzinotakurwa	Valid - Shona word	Yes	Yes	Yes	Yes
andipewo	Valid - Shona word	Yes	Yes	Yes	Yes
haurape	Valid - Shona word	Yes	Yes	Yes	Yes

Shona 100k Words	Correctness	OOV Status	N-Gram Word Flagged Correct	Word Actually Correct	Mashoko Word Flagged Correct
isagare	Valid - Shona word	Yes	Yes	Yes	Yes
nekunyerekedza	Valid - Shona word	Yes	Yes	Yes	Yes
Nyatanga	Valid - Shona word	Yes	Yes	Yes	Yes
bara	Valid - Shona word	No	Yes	Yes	Yes
uchibatsirikana	Valid - Shona word	Yes	Yes	Yes	Yes
yaisabhadhara	Valid - Shona word	Yes	Yes	Yes	Yes
nechiremerera	Valid - Shona word	No	Yes	Yes	Yes
sekuba	Valid - Shona word	Yes	Yes	Yes	Yes
zvakatongwa	Valid - Shona word	Yes	Yes	Yes	Yes
taitombogara	Valid - Shona word	Yes	Yes	Yes	Yes
chaunicho	Valid - Shona word	Yes	Yes	Yes	Yes
wekurambwa	Valid - Shona word	Yes	Yes	Yes	Yes
vanobatsirika	Valid - Shona word	Yes	Yes	Yes	Yes
Muchakata	Valid - Shona word	No	Yes	Yes	Yes
venhindiri	Valid - Shona word	Yes	Yes	Yes	Yes
Anodzungaira	Valid - Shona word	Yes	Yes	Yes	Yes
wamutsvata	Valid - Shona word	Yes	Yes	Yes	Yes
Yoshifumi	Valid - Shona word	Yes	Yes	Yes	Yes
kwekurudziro	Valid - Shona word	Yes	Yes	Yes	Yes
neDembare	Valid - Shona word	Yes	Yes	Yes	Yes
isvavirire	Valid - Shona word	Yes	Yes	Yes	Yes
rakazokonzeresa	Valid - Shona word	Yes	Yes	Yes	Yes