	Category	Min	Max	Chosen
1	Requirement analysis and design	0	20	0
2	Theoretical analysis	0	25	25
3	Experiment design and execution	0	20	15
4	System development and implementation	0	15	5
5	Results, findings and conclusions	10	20	15
6	Aim formulation and background work	10	15	10
7	Quality of report writing and presentation		10	5
8	Adherence to the project proposal and quality of work		10	5
9	Overall general project evaluation			
Total marks:		**80**		

University of Cape Town

Computer Science Honours 2015

AfriSpel final honours report

Author:
Victor Kabine

Supervisors:
Dr. Maria Keet and
Prof. Hussein Suleman

November 9, 2015

AfriSpel: A rule based spell checker for the isiZulu language

Victor Kabine
University of Cape Town
Rondebosch
Cape Town 7700, South Africa
victorkabini@yahoo.com

ABSTRACT

African languages that are in the Bantu language family belong to the less resourced languages in the world. There has not been much research on these languages and this is evident based on the lack of digital tools that are available to the language.

In this paper we design and implement a morphological analyzer to be used for the spell checking of the isiZulu language. The morphological analyzer is developed using the regular expressions approach and regular language. Using regular expressions we are able to specify patterns that words have to follow in order for them to be recognized as valid isiZulu words.

This spell checker has an accuracy rating of 80 percent. We conducted the experiments using the Ukwabelana corpus, a 2ml token corpus developed at the University of KwaZulu Nata by Prof Langa Khumalo and words added by us in order to test extreme cases such as words numbers and special words accepted as valid isiZulu words.

CCS Concepts

•**Regular languages** → Regular expressions ; •**Morphological analyzer** → Spell checker;

Keywords

Morphological analyzer; Spell checker; regular expressions; Bantu language family; isiZulu language

1. INTRODUCTION

A spell checker is a computer program that performs the act of spell checking. Spell checking is the process of detecting incorrectly spelled words for a given language [1]. The given language in this project is the isiZulu language. The spell checking process also sometimes gives suggestions to the incorrectly spelled words for the isiZulu language [22] . Spell checking is also a sub-field of Natural Language Processing (NLP) [1]. Natural Language Processing is also commonly referred to as Computational linguistics.

Computational linguistics is a relatively new field in the field of language and computer science, with the expression only being coined in the 1950s [18]. Most of the research from the field of computational linguistics goes to languages from developed areas in the world, which are languages such as English. Languages from the developing parts of the world like Africa belong to the lesser studied languages of the world,they are referred to as Bantu languages [26].

Research is vital to the advancement of knowledge in any subject, there are more things that were discovered through research than through trial and error. [9]. This then raises a major issue in that because of there being a lack of research on the isiZulu language, this results in a lack of the linguistic resources available for the isiZulu language [20], this includes not only tools such as spell checkers but also other computational tools that produce and consume digital information for the isiZulu language.

This problem is without a doubt a very important problem that must be explored. There are 11 million people who speak isiZulu in South Africa [32]. With the little research that has been done so far and the little resources that are available for the language [26], it then stands to reason that more emphasis has to be focused on this language.

With the emergence of the world wide web and how the internet has become part of our daily lives [14], the need for these computational tools is is growing larger. With more people looking to internet to find information, the need for documents and other resources to be available digitally is also growing larger. There is also a need to close down the gap between the developed language and the developing languages, research on spell checkers started in the late 1950s however the first spell checker for an African language was only designed in 1992 [15].

This paper focuses on the development of a spell checker for the isiZulu language using a rule based approach. This approach is essentially a theory driven approach involving the study of the morphological structure of the isiZulu language. A morphological structure involves looking at morphemes, a morpheme is the smallest unit of a word [3]. This paper focused on the modelling of morphotactics (rules on how isiZulu words should be formed) as well as morphological alternations (being able to identify the right form of a morpheme) [3]

This is achieved through the use of a morphological analyzer. This morphological analyzer can then be used to perform spell checking functions such as lexical and error

ACM ISBN .

DOI:

recall. Lexical recall is the spell checker's ability to recognize correctly spelled words, while error recall refers to the spell checkers ability to recognize incorrectly spelled words [4]. We also use the confusion matrix approach in order to find out the accuracy of the spell checker.

1.1 Background

The idea of using a morphological analyzer for spell checking the isiZulu language has been carried out before. S. Bosch and L. Pretorius [3] proposed a morphological analyzer for the isiZulu language that is built using xerox finite state tools. The morphological analyzer is used for grammatical tagging (parts of speech tagging) of a corpus. S. Bosch and R. Eiselen [4] describe how a morphological analyzer can be used to develop a spell checker for an agglutinative language like isiZulu.

The focus of the spell checker was on enhancing the lexical and error recall, they used regular expressions to construct the morphological analyzer. S. Spiegler, A. van der Spuy and P. A. Flach[32] introduces a zulu corpus that was developed with the use of a semi automatic morphological analyzer and parts of speech tagger. S. Spiegler, B. Golenia et al [31] introduce different types of supervision that can be used to develop a morphological analyzer in the future.

There are also a few open source spell checkers available for the isiZulu language. The first is by Apache OpenOffice and it was created in the year 2009. There has only been one update, it was released in the following year. There is also an add-on by mozilla thunderbird. it was released in the year 2011.

2. MORPHOLOGY OF ISIZULU

We can think of a language as a set of words and words as a set of morphemes, words can be formed through one or more morpheme(s) [24]. Often times, there are morphemes that are dependent on other morphemes in order to form a sensible or valid word.

In this case, these morphemes are called free morphemes or root [3]. These morphemes cannot stand on their own thus do not make sense on their own. An example of an affix for the English language would be 'ing'. An example of an affix for the isiZulu language would be 'im'.

Other times, there are morphemes that are not dependent on any other morphemes and make a valid word and can stand on their own. These morphemes are called bound morphemes or affixes [24]. An example of such a morpheme is the English word take.

Depending on the type of language, root morphemes can often have meaning by themselves and count as valid words. Fusional languages have root morphemes that can have multiple meanings [25]. The language may have one or more morphemes and the boundary between the morphemes is not always that easy to distinguish this is because the affixes can be fused together with the root.

English is one of the popular language that is fusional [11]. One example from the English language is the word 'spoke', this word denotes both 'to speak' which is the root word and also the past tense. Most of the European languages are somewhat fusional [24].

The other type of language in known as the agglutinative language. Languages that belong to this category have words that are composed of multiple morphemes [2]. The morphemes are put together in way that Agglutinative languages can have one word has the same amount of information as a sentence in an English sentence.

Agglutinative languages have multiple language families and one language family in Africa is the Bantu language family [5]. The Bantu language family has approximately 400 African languages, this includes the Nguni languages. The isiZulu language belongs to the Bantu language family. The Bantu language family belongs to the lesser studied languages in the world [5].

For these reasons, developing a spell checker for the isiZulu will provide a different challenge as opposed to developing a spell checker for the English language. The isiZulu language belongs to a different morphological type of language than English thus research and methods on developing spell checkers for the English cannot necessarily be used to model the isiZulu language. The isiZulu language has little research that has been done from it and out of that research some of it might not be very useful to us.

Inflection is the process of adding morphemes to a word in order to specify the mood, the tense, number gender etc. Fusional languages lose their inflection as the years go by and thus end up with little inflection [8]. However in the isiZulu language, the inflection is more common, this is seen in the nominal classification system and the concordial agreement system [4].

The nominal classification system is a type of inflection that is known as inherent inflection and it occurs in the isiZulu language [34]. The inflectional morphemes that occur in this system are added onto nouns. They are in the form of prefixes and each prefix classifies nouns to their respective categories or classes [33].

Each class gives an indication of the type of nouns that belong in that class, also what the nouns identifies in the language. There are 17 classes in the nominal classification system [4]. Please see appendix A1 for the full view of the table with the noun classes as well as the prefixes that are included.

The concordial agreement is a type of inflection known as contextual inflection [34] and it is different for the isiZulu language. Instead of it affecting only the construction of the sentence, it also changes the internal structure of the words in the sentence. Concordial morphemes are added to a word to make sure that the subject is in agreement with the rest of the sentence [4]. This is why unlike in the English language where it is only considered for grammar because it only affects the sentence construction (ordering of words in a sentence) instead of the structure of the word.

Another complex aspect in the isiZulu language derivation. derivation in isiZulu is post-inflectional [34], it occurs as a suffix. The more common examples of the derivative suffixes in the isiZulu language is the 'i' suffix. Appearing in the word umuhambi and derived from the word hamba or the root hamb [4].

However there are some exceptions to this, they occur mostly in locative adverbs. In this case both the prefix and suffix are added. the prefix is the locative prefix and the suffix is the derivative suffix [4].

3. DEVELOPING THE RULES

3.1 Current methodology

There are multiple approaches that be taken to build the morphological analyzer. There is the statistical approach,

this approach is commonly referred to as the corpus based approach and the rule based approach [29]. We are using the rule based approach for the development of the spell checker. We will be using finite state technology for the development of the morphological analyzer, this is a methodology that has been applied before and is gaining more and popularity [17].

Finite state automata is a type of model of computation [17], this model has a finite number of states and arcs. A state determines whether or not the machine accepts the information that is given by the user. A state also enforces the rules that are in the system, it is only if the input of the user has met the conditions of the user that the system can then transition to another state [30]. There can be multiple states however there should always be two states in order for the finite state machine to work, the start state to symbolise the beginning and the final state [30]

The machine can only have one place to start (start state) however in some cases it can have multiple places to finish (finish states). As the name suggests there are only a limited amount of options that can be accepted by the machine, if the given input does not meet any of the conditions, then the input is this case rejected by the machine[30] . There are different types of finite state machines namely deterministic finite state automata and non deterministic finite state automata [30].

The difference between the two is that with deterministic finite state automata, each path has to be unique from the other and in this way the transition between the start state and the following state has already been determined by the system whereas with non deterministic finite state automata a state can even no condition to meet and the system would transition to the next state [30]

Non-deterministic finite state automaton is mostly used for Natural language processing and is the approach that is used in this system as well. This approach has led to the creation of linguistic tools to help facilitate in the creation of a morphological analyzer. There have been various aspects to designing the morphological analyzer and we go through these aspects to see which of these aspects we should focus on

3.2 Finite state tools

Xerox finite state tools implement a finite state transducer which is then used as a morphological analyzer. The finite state tools use regular expressions to implement the finite state transducer. Regular expressions are a type of formal language that is used to match a given pattern [6]. The pattern that is specified is the 'condition' that has to be met in order for the given input to be seen as valid by the system.

The finite state transducer is a nice tool to use because a lot of the things are already done for you however based on the documentation of the software and the availability of it. The syntax of the regular expressions is also different and it is unique and is set apart differently. The usability is especially troublesome for Windows users

There have been other tools that have been developed and most of these tools have the most of the functionality of the xerox finite state tools. This includes tools such as Openfst, Foma etc. Since we could not attain the results we wanted by using the tools, we decided to use other sources that could benefit us more.

3.3 Regular expressions

Regular expressions is a type of formal language that is considered to be the one of the most useful tools in computer science [18]. Regular expressions can be used to specify what pattern the system is supposed to achieve. Regular expressions use formal language theory and in formal language theory, a language is regarded as just a set of strings and the linking of a series of zero or more symbols [17].

We can specify a type of search pattern using regular expressions for the system to follow, it uses the an algorithm of string matching where the regular expression looks for any string or a specific string that matches the pattern that is given [6]. There are a number of regular expression characters that are used and are the basic syntax of regular expressions.

The `.` symbol identifies any character as input. `?` symbol matches zero or one of the characters before it. `*` symbol matches zero or more of the characters before it. `+` symbol matches one or more characters before it. `^` acts as either as an anchor or as negation depending on the context on which is used. `{} or []` matches the range of characters specified in the brackets [18].

The use of regular expressions only for the creation of a morphological analyzer and has also been explored however, it has not been taken that and gives us a chance to build on what is already there. The paper we will be building on is the paper by S. Bosch and R. Eiselen [4]. This paper also specifies use of 'perl-like' expressions to create a morphological analyzer. We had to take precaution when modelling the regular expressions, We first needed to find an informal representation of the language.

3.4 Using Jflap

Instead of just modelling the entire language in one go and leaving a lot of important aspects out, we have decided to use a table which is an informal but structured representation of the language. We then used Jflap to experiment with the language. This is the main use of Jflap, Jflap is a software that is used to experiment with formal languages [28]. We experiment with the language by creating a finite state automata and testing it with random words from the ukwabelana corpus. From this tool, we can generate regular expressions from the finite state automata that has been created.

As stated above, the isiZulu language is an agglutinative language, this means that the morphemes are 'glued' together, this makes it easer to spot the morpheme boundary [2]. This also means that we can model morphemes separately and 'glue' the morphemes together in the end. The morphemes were 'glued' together using an empty state. We started off with the nominal classification system, an example of how a finite state automata looks and how it looks by examining the first iteration of the finite state automata can be found on figure 1 which is on the next page

The finite state automata was then coverted to Jflap regular expressions. The regular expressions were as follows:

```
(isi+ um(^+ u)+ ama+ izi + imi+ ubu + uku)(w + v
   + t + s + r + q+ p + z + y + x + g + f + c + n
   + m + l+ k + j + h) [a-z]([a-z]+[a-z][a-z])
   *(a + e + i + o + u)
```

Note that these regular expressions have been cut off, they

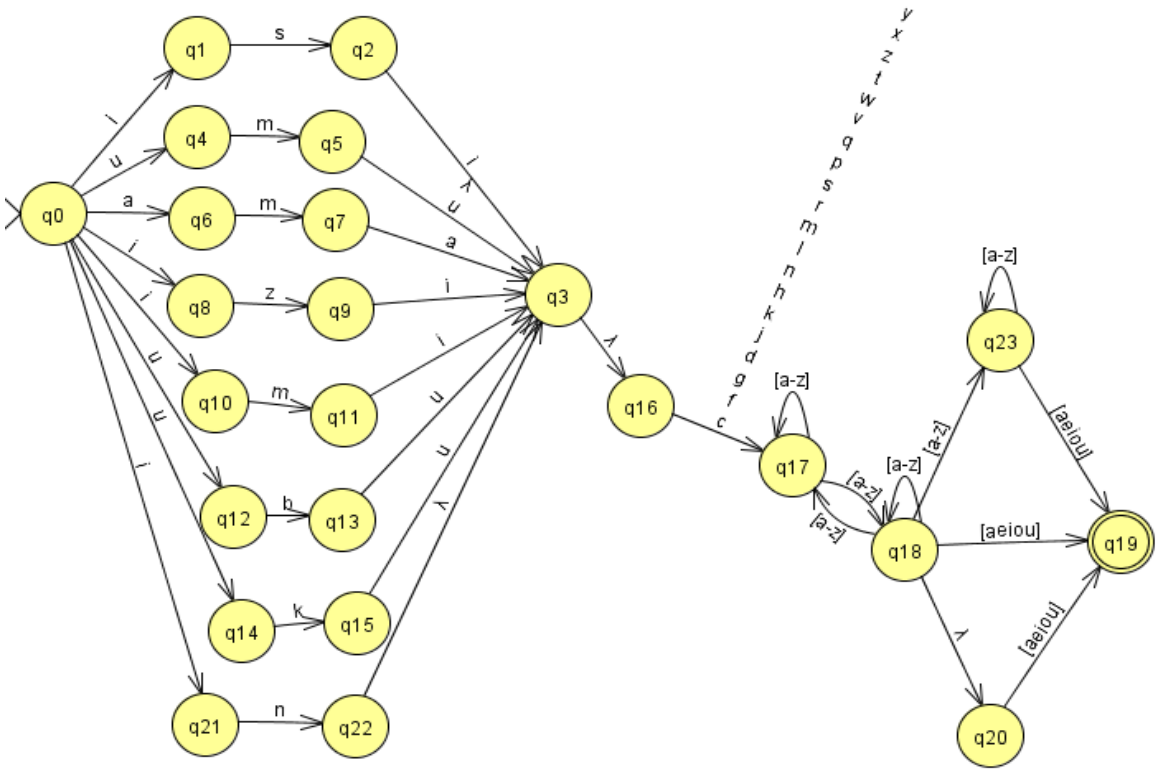

Figure 1: First iteration of the finite state automata for isiZulu nouns using Jflap 7.

reduplicate after this particular sequence was finished. The regular expressions had to be changed to a syntax that is more compatible with the languages that are used in computers. By checking to see which of the following regular expression characters that were used meant something else. We used the Jflap guidelines [28] to find regular expressions that were still written in formal language notation.

The notation was changed to the syntax of the Perl programming language. Unfortunately this was done by hand based on the the formal language notation of Jflap [28] and the syntax of the 'perl-like' regular expressions. Modelling the was beneficial in that we got a more in depth study of the language and to make changes before implementing the regular expressions which would then become the formal representation

There are some of the problems that occur with the change of regular expressions from Jflap to Perl, there are some symbols that are represented in Jflap that are not available in the Perl programming language. This is seen through the empty string shown as a mathematical symbol. Changing the symbol to the regular expressions proved challenging however the method of using solely regular expressions with the help of tools such as Jflap to not only experiment with the formal language gives us a better representation of the isiZulu language. Although it is still incomplete because there are still some rules that have not been modelled, this method is still very effective in that it already modelled most of the language in a form that is not only easy to replicate but also easy to continue and to add onto. The basic starting point for this methodology was found in the paper[4]

3.5 Modelling the morphological analyzer

The morphological analyzer performs the functions of a spell checker in that it can identify when a word is misspelled, or when a word has been spelled correctly. The approach that we used to come up with these computational rules is through first the informal representation of the language [19] in order to get a general idea of how the words were structured and how to determine whether the words were typographically correct. The informal representation also helped us figure out how to use our rules to restrict a lot of the words that are not found in the language. The term informal representation refers to research that has been done on the language that is represented in the form of tables [32] [27].

This regular expression was then changed into Perl programming language without the use of other tools. The regular expression that was done was the once again the following:

```
/^(isi|um(^$|u)|am(a|e)|izi|i(^$|m)|ubu|uku|ab(a|e)
 |in|ye|ezi|olu|u)(^$|[bcdfghjklmnpqrstvwxyz])
 ([a-z]+)([a|e|i|o|u]$)/
```

This approach was then carried out to model other words besides nouns, parts of speech such as verbs, pronouns and other concords as well. In the Perl programming language there is no symbol for empty string, we had to improvise in order to model words that could be seen as exceptions. The empty string modelled in Perl in place of regular language empty string was modelled as `^$`. Putting both anchors together could work as a sign of an empty string.

This however did not model the exception. The fourth iteration involved adding another regular expression without the use of prefixes. the fourth iteration can be seen as the

following:

```
/ ([bcdfghjklmnpqrstvwxyz])([a-z]+)([a|e|i|o|u]$)/
```

With these regular expressions, we were able to design a morphological analyzer that was used as a spell checker which is able to detect incorrectly spelled words. The spell checker was tested using a word list compiled from words from the Ukwabelana corpus and the corpus from the corpus from Prof. Langa Khumalo. We also put nonsensical words in there to test the validity of the spell checker.

An advantage that the Perl programming language regular expressions syntax has over the Jflap regular language syntax is that Perl gives the option to enforce certain rules through the use of anchors, these anchors are either ^ to indicate that the string, pattern or word has to start with a particular pattern or string that has been specified or $ which indicates that the string, pattern or word has to end with a particular pattern or string [18].

4. FINITE STATE NETWORK FOR ISIZULU

The most important factor that all finite state machines must have a initial state (start state) and the final state [30]. This theory can be implemented through the use of a formal language know as regular expressions. We know that a formal language refers to a language a set of strings of any kind [17]. The regular expressions are compiled into non deterministic finite state automata. With the use of non deterministic finite state automata, we can model a large array words with a single final state that accepts different kind of words that link up to the finite state network.

4.1 Process of finite state automata development

We modelled three different finite state networks to handle different types of morphological rules within the language. We stated off by modelling one part of the nouns which was the prefixes. Then we modelled the rest of the noun. We do know that the noun starts with a consonant [12]. Although there has been research that advices against the clustering or combination of consonants [23]. We have decided to not to put restrictions to this rule as there are always exceptions to the rule. Please see appendix for all the finite state automata developed

Tighter restrictions may cause words that are correctly spelled isiZulu words to be flagged as incorrectly spelled. We also know that isiZulu words always end with a vowel [23], these discoveries happened iteratively and this is perhaps the reason why this morphological analyzer is a more accurate model. There are still some words that will not be modelled by the morphological analyzer, these are words that start with a vowel instead of a consonant as well as remarks and words that include an apostrophe.

We then modelled the verbs, from the verbs we can tell that the verbs that verbs are structured as subject, verb stem (verb root) and object [16]. Using this, we can model the parts of speech that form the word. These include the subject marker, the object marker, singular marker, negative marker as well as class markers within the word. Once we can model the morphological analyzer to accept words of the isiZulu language. We have also modelled pronouns both personal and possessive. The pronouns do not have prefixes but they do have a common suffix, this suffix is e, i or a [7].

The negative markers in the isiZulu language appear both as prefixes and as suffixes. The prefix is a and the suffix is anga, the infinitive word form appears with the affirmative prefix uku. The who, what, why questions within the isiZulu language all have the suffix phi.

Some of the markers were modelled into one finite state network, they were connected or joined together through the use of what in regular language is known as an empty string or symbol. In the Jflap language, this is referred to as a lamda. This empty string was not only used to link the markers together but also to handle a few of the isiZulu words that do not meet a certain aspect of the rule that has been specified.

We also did not model infixes because with the structure that has been made by the finite state network, the morphological analyzer will be able to accept those type of words. Q0 signifies the start state. The arrows signify the transition of one state to another. When the arrow is pointed towards the current state, the input needs to be inputed more than once. When there are two states arrows alternating between each other, this allows an odd number of the input to be accepted by the finite state network. The range of alphabets from a to z is the input that can be accepted by the state. The accepting state illustrated as a state within a state.

From the development of the non deterministic finite state automata, we transform the deterministic finite state automata. The deterministic finite state automata can be compiled to regular expressions using the Jflap Software tool

4.2 Changing to regular expressions

The deterministic finite state automata is then converted to regular expressions. The regular expressions were written in regular language. The syntax for the regular expressions in the regular language differs from the regular expressions of the programming languages in computing.

The programming language that has been used in this project is Perl. The output that has been generated by the Jflap software tool, had to be modified to meet the syntax of the Perl programming language. The regular expressions also had to be changed. The regular expressions that have been generated by the Jflap tool for the isiZulu noun finite state network looks like the following:

```
(isi+ um(^+ u)+ ama+ izi + imi+ ubu + uku)(w + v
 + t + s + r + q+ p + z + y + x + g + f + c + n
 + m + l+ k + j + h) [a-z]([a-z]+[a-z][a-z])
 *(a + e + i + o + u)
```

The regular expression continues in this fashion and becomes incredibly redundant, repeating the same sequence and this exact pattern a number of times. This has made it very hard to translate the language into Perl syntax for a number of reasons. The most crucial reason being that the notations used have slightly different meaning. In regular language syntax, the logical or function that depicts that the state will either accept input shown on the left hand side or on the right hand side is represented with the notation +. In the Perl programming language, this notation indicates a range of one or more inputs that can be entered by the user.

Another difference in syntax is the use of the ^ notation. This notation in regular language represents an empty string and is referred to as lamda. This notation in perl however, is known as an anchor, this symbol makes sure to check whether or not the pattern starts with the input that has been specified. Knowing these facts, we have changed the

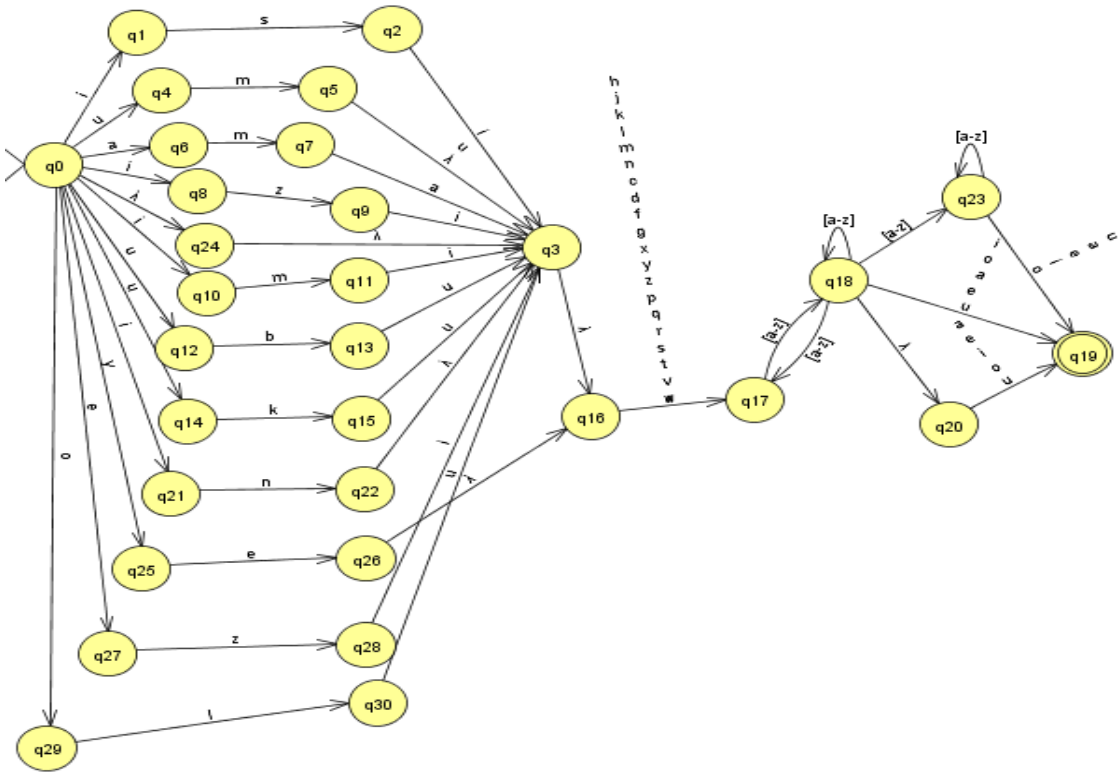

Figure 2: Final iteration of the finite state automata for isiZulu nouns using Jflap 7.

regular expressions to the following:

```
^(isi|um(^$|u)|am(a|e)|izi|i(^$|m)|ubu|uku|ab(a|e)
 |in|ye|ezi|olu|u)(^$|[bcdfghjklmnpqrstvwxyz])
 ([a-z]+)([a|e|i|o|u]$)
```

This particular regular expressions is the final product of the noun model representation. Through this regular expression, we can see that all of the 17 noun prefixes are handled and we ensure that the following input is a consonant then the morphological analyzer can accept a number of inputs that the user has entered afterwards and there is a range of inputs that must be entered by the user at the end of the pattern in order for the string to be accepted as a valid word in isiZulu.

These anchors bring us closer to a more accurate representation of the language. The regular expression above does not only model nouns, it also models other concords such as the adjective concord. Since we are not focusing on grammar, our objective is to model as many linguistic rules of the language as we can. We decided to group some of the concords together but not all.

We did not group all of the concords and words into one finite state network because the regular expressions would have been too complex to comprehend. Another aim of this project is to make sure the results of the project are readable and reproducible. This lead to the development of three different finite state networks and regular expressions all modelling different types of words

The verb finite state network in Jflap produced the following regular expression:

```
(u + i + a + ngi + ni+ ba + ku + si zi + li + lu
+((u + i + a + ngi + ni + ba)y+ kuy + siy +
+ziy + liy + luy)a[a-z]([a-z][a-z])*a +
((u + i + a + ngi + ni + ba)[a-z]+
+ ku [a-z] + si [a-z] + zi [a-z] li [a-z]
+ lu[a-z])*ile
```

Once again, we can see that there are some redundancies in the regular expressions that do not make it very accurate. The regular expression was changed again to match the regular expression syntax in the Perl programming language. The Perl programming language is the following:

```
^(u|i|a|ng|li|si|ni|ba|ku|zi|lu)ya
([bcdfghjklmnpqrstvwxyz])([a-z]+)
(a$)|(u|i|a|ng|si|li|lu|ku|zi|ni|ba)([a-z]+)
(e|ile$)|(t|k)h(o|i)na$|(b|w|l|y|z|s)onke$
```

This is the Perl regular expression for isiZulu verb pattern matching, it followes the subject, verb stem, object and final vowel word structure. Although it was meant to only handle verbs, it has been expanded to include possessive pronouns, past tense as well as future tense. However it does not words that are in the passive form. The make up of nouns and verbs is very complex and differ in a lot of ways. The structure being one of the most important, this has made it very difficult for us to model them together. There are as is the case with many natural languages, exceptions to the rules, these exceptions were not modelled.

By modelling each sets of concords separately, we were able to model the verbs. With the help of the structure guidelines of isiZulu words [7]. The next regular expression

that was generated was targeted at negative markers and concords. We used the Jflap tool to convert the finite state network to a regular expression. The regular expression for negation on the Jflap tool is shown below:

```
(aka+aba+awu+ayi+azi+alu+a(k+b)u+
(aka+aba+awu+ayi+azi+al(u+i)+a(k+b)
u(yi+si)(r+s+p+q+v+w+y+z+b+c+f+g
j+k+h+n+m+l[a-z]([a-z]+[a-z][a-z])*(i+o)
```

From this we are able to change these regular expressions to the syntax the Perl programming languages. The regular expression for perl is as follows:

```
/^(e|aka|aba|aw(a|u)|ayi|azi|a|u|l)(^$|si|yi)/
/([bdcfghjklmnpqrstvwxyz])([a-z]+)(a|o|i|anga|phi$)/
```

There are steps that were taken in the changing of the Jflap regular expressions to the perl programming language regular expressions. The notations mean the following: `[a-z]` specifies the range of alphabets from a to z. This is only specific to lower case alphabets. `[A-Z]` is specific to upper case alphabets, when specifying ranges we use the symbols `[]`, `()` or `{}`.

The * symbol specifies that the user can enter the characters shown zero or more times. The + symbol specifies that the user can enter the inputs shown once or more times. The | symbol specifies the logical or function, this function shows that the system can accept either of the characters specified.

The * symbol is known as a greedy operator, it matches the described pattern as much as it possibly can. This means that the symbol could match a pattern that is as close to the word as it can be. this leads to a lot of false positives and inaccurate words being matched correctly. This symbol has been changed to the + symbol.

5. METHODOLOGY

The question we are asking in this study is this, just how accurate is this spell checker when compared to a spell checker designed using a similar approach by S. Bosch and R. Eiselen [4]. The paper mentioned also made use of perl regular expressions and obtained an accuracy rate of 82.

5.1 Hypothesis

Our hypothesis is as follows, this spell checker will have a higher accuracy rate than that of the paper by S. Bosch because it goes into more depth concerning the language.

5.2 List of Materials

We made use of 2 corpora, the first from the Ukwabela corpus and the second from the University of Kwa-Zulu Natal. On the grounds that we do not have expert level knowledge of the language to be able to discern between false positives and true negatives, so to give a more accurate look at our spell checker we have added a mini word list that is short and tests the extreme cases for example a number found within a string. 50 words will be taken from each of the corpora and put together to form one word list. The system will then test the word list on the basis of 4 important questions, these questions are derived from the confusion matrix [27].

The four questions that are derived from the confusion matrix are the following: how many correctly spelled words appeared as incorrectly spelled words to the spell checker, this term is known as a false negative. Secondly, how many incorrectly spelled words are flagged as correctly spelled words by the spell checker. This term known as a false positive [10]. Thirdly, how many correctly spelled are recognized as correctly spelled words by the spell checker. This term is known as True positive [10].

Lastly, how many incorrectly spelled words are recognized as incorrectly spelled words by the spell checker. This term is known as true negative.

One computer will be used in conducting these experiments, it contains all of the materials needed for the experiment and there is no means any need for another. We used the following software for the development of this morphological analyzer which will be used for spell checking and is now about to be evaluated. The first software tool is Jflap, this tool is used to experiment with formal languages and also works for natural language processing. The second software tool is Perl. It is with this software tool that the morphological analyzer was implemented. It is that same morphological analyzer that we will be evaluating.

5.3 Procedures

The manipulated variable in this experiment is the word list, this is because the size of the word list can be manipulated at any time. This variable is also known as the independent variable. The responding variable in this experiment is the accuracy to be measured, this is because of the value of the output cannot be changed in any way.

The experimental control in this experiment is the accuracy of the spell checker suggested in the paper by S. Bosch and R. Eiselen. That spell checker will be our standard for comparison and in the end we will compare the two in terms of accuracy, they however have calculated the accuracy of their spell checker using only lexical recall. Thanks to the confusion matrix, we can also know what the lexical recall of the spell checker is.

The controlled variables in this experiment are the variables that we think will stay constant and never changing, these variables would also be the results obtained. These results will always remain constant. The validity measures that have been taken to ensure that the results stay the same and are reliable are that we have recorded the data. The words that have been used to test the spell checker and to evaluate it have been stored in a word list.

The results have been stored and tabulated into a table of confusion [10]. So if the trials were to be repeated again, the results could be cross checked for proof.

6. RESULTS

The hypothesis specified above stated again that we believe that our spell checker would do better and be more accurate than the spell checker proposed by the paper [4]. Our results are in two parts, the accuracy level for the Ukwabelana corpus and the accuracy rate for the 2ml token corpus we have received from the University of Kwa-Zulu Natal. In the end, the words will put together onto one for a full view of our system.

The abbreviated terms mean the following: TN: True negatives, TP: True positives, FP: False positives and FN: False negatives. The accuracy of the spell checker using the Ukwabelana corpus registered at 86 percent. The reason why we were able to achieve this results is because theoretically

Table 1: Confusion matrix

Corpus	TN	TP	FP	FN
Ukwabelana		43		7
2ml token	1	46	1	2
Self made mini word-list	1	7	4	

the morphological analyzer that has been created by using a step by step procedure allows us to get a more in depth look at the language in order to provide a spell checker that gives a more accurate representation of the isiZulu language.

The accuracy of the spell checker when tested using the 2ml corpus, the accuracy rate of this is at 92 percent. This is very surprising because this corpus has strings of very old isiZulu language and the dialect is a little off and is very old. This is a surprising turn of events. The regular expression from the paper was: `/^(u)(.*)(a)$/`
While this regular expression may handle many isiZulu words, there are many false patterns that will be matched as correct isiZulu words. The use of the . symbol in regular expressions means any character (numbers and other symbols within the string included) will be allowed in the string

An example of the words that can be accepted by the regular expressions are ubaba and uyabaleka, in short, this regular expression can accept isiZulu words that start with u and end with a. However input entered in the middle of the string will also be accepted. Words such as ub223ba or uya332233bale33[]ka would also be accepted.

This is why

From this we could see that tighter restrictions were needed in order to make sure the spell checker would be an accurate representation of the language. The third iteration was the following regular expression:

```
/^isi(A-Z)(a-z)*([aeiou])$/
/^ama(A-Z)(a-z)*([aeiou])$/
/^izi(A-Z)(a-z)*([aeiou])$/
/^in(A-Z)(a-z)*([aeiou])$/
/^u(A-Z)(a-z)*([aeiou])$/
/^i(A-Z)(a-z)*([aeiou])$/
/^aba(A-Z)(a-z)*([aeiou])$/
/^uku(A-Z)(a-z)*([aeoiu])$/
```

Although these regular expressions have gone a step closer to solving the problem of shown above, there were still a number of discrepancies that were found with this set of regular expressions. They did not handle all of the rules of the isiZulu language and not most of the rules of the isiZulu language.

isiZulu words always start with a consonant after the prefix and always end with a vowel. There was also a problem of the use of the greedy operators such as * which would undoubtedly match incorrect patterns including those with numbers

The third iteration involved starting over and introducing the Jflap software tool. This tool gave us a correct indication of how the language can be modelled. This software allows us to experiment with the formal language, it also allows us to test the formal language with a string of input and from that generate regular languages using the Jflap software tool. The regular expression that was generated was once again was the following:

```
(isi+ um(^+ u)+ ama+ izi + imi+ ubu + uku)(w + v
+ t + s + r + q+ p + z + y + x + g + f + c + n
+ m + l+ k + j + h) [a-z]([a-z]+[a-z][a-z])
*(a + e + i + o + u)
```

7. DISCUSSIONS

There have been many stumbling blocks with this project and there are still many parts that are meant to looked with this problem. We have not been able to model all of the rules of the language and thus there still some improvements that are meant to take place in order for a more complete and accurate spell checker.

For future works, we want to fuse multiple approaches together so one approach can cater for the short comings of another. There are certain features of the regular expressions that the regular expression cannot measure. Not to mention the size of the regular expressions makes it very hard to find a point of failure and to fix it.

We would also like to gain further knowledge of the language so that we can be able to model the exceptions to certain without at the same time, damaging the integrity of the spell checker by having it accept more nonsensical words than the exceptions that it was meant to cater for

8. CONCLUSIONS

In this article, we discuss how to build a morphological analyzer that can be used as a spell checker. We discuss how we can use the morphological structure of the language to model a spell checker.

We also discuss the other methodologies that have been tried and that have and how we used other resources as a springing board to modelling the language.

9. ACKNOWLEDGMENTS

We would like to thank the University of Cape Town and the Department of Computer Science for this wonderful opportunity to be able to undertake in this project. This opportunity has granted me the chance to learn more about ourselves in a more academic standpoint.

We thank our supervisors for the support that they have provided us. We would have never been able to complete this journey without their support. We would also like to thank the support of the Honours conveiner Prof. Michelle Kuttel for all her support with the honours year.

We would also like to thank Prof. Langa Khumalo and the University of Kwa Zulu Natal for the help that has been offered to us. We would also like to thank NRF for funding us during the development of this project

10. REFERENCES

[1] Y. Bassil and M. Alwani. Context-sensitive spelling correction using google web 1t 5-gram information. *Computer and information science*, 5(3):37, May 2012.

[2] A. Bergmann, K. Currie, and S. M. Ross. *Language files: Materials for an introduction to language and linguistics*. Ohio State University Press., Columbus, USA, 10th edition, 2007.

[3] S. Bosch and L. Pretorius. The significance of computational morphological analysis for zulu lexicography. *International Journal of Computer Trends and Technology (IJCTT)*, 4(3):372–374, June 2002.

[4] S. E. Bosch and R. Eiselen. The effectiveness of morphological rules for an isizulu spelling checker. *South African Journal of African Languages*, 25(1):25–36, 2005.

[5] S. E. Bosch, A. Fleisch, and L. Pretorius. Experimental bootstrapping of morphological analyzers for nguni languages. *Nordic Journal of African Studies*, 17(2):66–88, November 2008.

[6] A. Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2):197–213, 1993.

[7] M. de Dreu. The internal structure of the zulu DP. Master's thesis, Leiden University, Netherlands, 2008.

[8] G. Deutscher. *The unfolding of language: An evolutionary tour of mankind's greatest invention.* Holt paperbacks, New York, USA, 2006.

[9] R. Farr and P. Timm. *Business Research: An Informal Guide*. Fifty-Minute series. Crisp Publications, 1994.

[10] T. Fawcett. An introduction to roc analysis. *Pattern Recognition letters*, 27(8):291–296, 2006.

[11] L. B. Feldman, R. Frost, and T. Pnini. Decomposing words into their constituent morphemes: Evidence from english and hebrew. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 21:947–960, 1995.

[12] L. Grout. *The Isizulu: A Grammar of the Zulu Language*. Trübner & Company, London, United Kingdom, 1st edition, 1859.

[13] A. M. Harrison. *Introduction to formal language theory*. Addison-Wesley Longman Publishing Co., Inc., Boston MA, USA, 1st edition, 1978.

[14] B. A. Huberman and L. A. Adamic. Internet: growth dynamics of the world-wide web. *Nature*, 401(6749):131–131, 1999.

[15] A. Hurskainen. A two-level computer formalism for the analysis of bantu morphology. *Nordic Journal of African studies*, 1(1):87–119, 1992.

[16] M. Kafrika and S. Zerbian. Quantification across bantu language. *Quantification: a cross linguistic perspective, North Holland Linguistic Series: Linguistic Variations*, 64(1):383–414, 2008.

[17] L. Karttunen, J.-P. Chanod, G. Grefenstette, and A. Schille. Regular expressions for language engineering. *Natural Language Engineering*, 2(4):305–328, 1996.

[18] G. Kaur. Usage of regular expressions in nlp. *International Journal of Research in Engineering and Technology (IJRET)*, 3(1):168–174, January 2014.

[19] C. M. Keet and L. Khumalo. Basics for a grammar engine to verbalize logical theories in isizulu. In *Rules on the Web. From Theory to Applications*, pages 216–225. Springer, 2014.

[20] M. Koleva. Towards adaptation of NLP tools for closely-related Bantu languages: Building a Parts-of-Speech tagger for Zulu. Master's thesis, Saarland University, Germany, 2013.

[21] K. Kukich. Technique for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, December 1992.

[22] R. Mishra and N. Kaur. A survey of spelling error detection and correction techniques. *International Journal of Computer Trends and Technology (IJCTT)*, 4(3):372–374, September 2013.

[23] Y. Naidoo, A. van der Merwe, E. Groenewald, and E. Naude. Development of speech sounds and syllable structure of words in zulu speaking children. *Southern African Linguistic and Applied Language studies*, 23(1):59–79, November 2009.

[24] T. E. Payne. *Describing morphosyntax: A guide for field linguistics*. Cambridge University Press., New York, USA, 1997.

[25] A. Pirkola. Morphological typology of languages for ir. *Journal of Documentation*, 57(3):330–348, 2001.

[26] L. Pretorius and S. E. Bosch. Finite-state computational morphology: An analyzer prototype for zulu. *Machine Translation*, 18(3):195–216, 2003.

[27] F. J. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *ICML*, volume 98, pages 445–453, 1998.

[28] S. H. Rodger and T. W. Finley. *JFLAP: an interactive formal languages and automata package*. Jones and Bartlett Learning, California, USA, 2006.

[29] K. Shaalan. Rule-based approach in arabic natural language processing. *The International Journal on Information and communication Technologies*, 3:11–19, 2010.

[30] S. Shandilya and R. Yadav. Determinism and non determinism of finite automata. *International Journal of Research in Information Technology (IJRIT)*, 2(8):291–296, 2014.

[31] S. Spiegler, B. Golénia, K. Shalonova, P. Flach, and R. Tucker. Learning the morphology of zulu with different degrees of supervision. *Spoken Language Technology Workshop (SLT)*, 25(1):9–12, 2008.

[32] S. Spiegler, A. Van Der Spuy, and P. A. Flach. Ukwabelana: an open-source morphological zulu corpus. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1020–1028. Association for Computational Linguistics, 2010.

[33] E. K. Twala. The noun class system of isiZulu. Master's thesis, University of Johannesburg, South Africa, 1992.

[34] A. van der Spuy. Post-inflectional derivation in zulu: further evidence against the split morphology hypothesis. *Language matters: Studies in the languages of Africa*, 44:78–93, 2013.

APPENDIX

NC	AU	PRE	Stem (example)	Meaning	Example	
1	u-	m(u)-	-fana	humans and other	umfana	boy
2	a-	ba-	-fana	animates	abafana	boys
1a	u-	-	-baba	kinship terms and proper	ubaba	father
2a	o-	-	-baba	names	obaba	fathers
3a	u-	-	-shizi	nonhuman	ushizi	cheese
(2a)	o-	-	-shizi		oshizi	cheeses
3	u-	m(u)-	-fula	trees, plants, non-paired	umfula	river
4	i-	mi-	-fula	body parts	imifula	rivers
5	i-	(li)-	-gama	fruits, paired body parts,	igama	name
6	a-	ma-	-gama	and natural phenomena	amagama	names
7	i-	si-	-hlalo	inanimates and manner/	isihlalo	chair
8	i-	zi-	-hlalo	style	izihlalo	chairs
9a	i-	-	-rabha	nonhuman	irabha	rubber
(6)	a-	ma-	-rabha		amarabha	rubbers
9	i(n)-	-	-ja	animals	inja	dog
10	i-	zi(n)-	-ja		izinja	dogs
11	u-	(lu)-	-thi	inanimates and long thin	uthi	stick
(10)	i-	zi(n)-	-thi	objects	izinthi	sticks
14	u-	bu-	-hle	abstract nouns	ubuhle	beauty
15	u-	ku-	-cula	infinitives	ukucula	to sing
17		ku-		locatives, remote/ general		locative

Figure 3: Nominal classification system for isiZulu [19]

NC	QC (all)		NEG SC	PRON	RC	QC_{dwa}	EC
	$QC_{oral+onke}$	QC_{nke}					
1	u-onke → wonke	wo-	aka-	yena	o-	ye-	mu-
2	ba-onke → bonke	bo-	aba-	bona	aba-	bo-	ba-
1a	u-onke → wonke	wo-	aka-	yena	o-	ye-	mu-
2a	ba-onke → bonke	bo-	aba-	bona	aba-	bo-	ba-
3a	u-onke → wonke	wo-	aka-	wona	o-	ye-	mu-
(2a)	ba-onke → bonke	bo-	aba-	bona	aba-	bo-	ba-
3	u-onke → wonke	wo-	awu-	wona	o-	wo-	mu-
4	i-onke → yonke	yo-	ayi-	yona	e-	yo-	mi-
5	li-onke → lonke	lo-	ali-	lona	eli-	lo-	li-
6	a-onke → onke	o-	awa-	wona	a-	wo-	ma-
7	si-onke → sonke	so-	asi-	sona	esi-	so-	si-
8	zi-onke → zonke	zo-	azi-	zona	ezi	zo-	zi-
9a	i-onke → yonke	yo-	ayi-	yona	e-	yo-	yi-
(6)	a-onke → onke	o-	awa-	wona	a-	wo-	ma-
9	i-onke → yonke	yo-	ayi-	yona	e-	yo-	yi-
10	zi-onke → zonke	zo-	azi-	zona	ezi-	zo-	zi-
11	lu-onke → lonke	lo-	alu-	lona	olu-	lo-	lu-
(10)	zi-onke → zonke	zo-	azi-	zona	ezi-	zo-	zi-
14	ba-onke → bonke	bo-	abu-	bona	obu-	bo-	bu-
15	ku-onke → konke	zo-	aku-	khona	oku-	zo-	ku-

Figure 4: Other concords that appear in isiZulu [19]

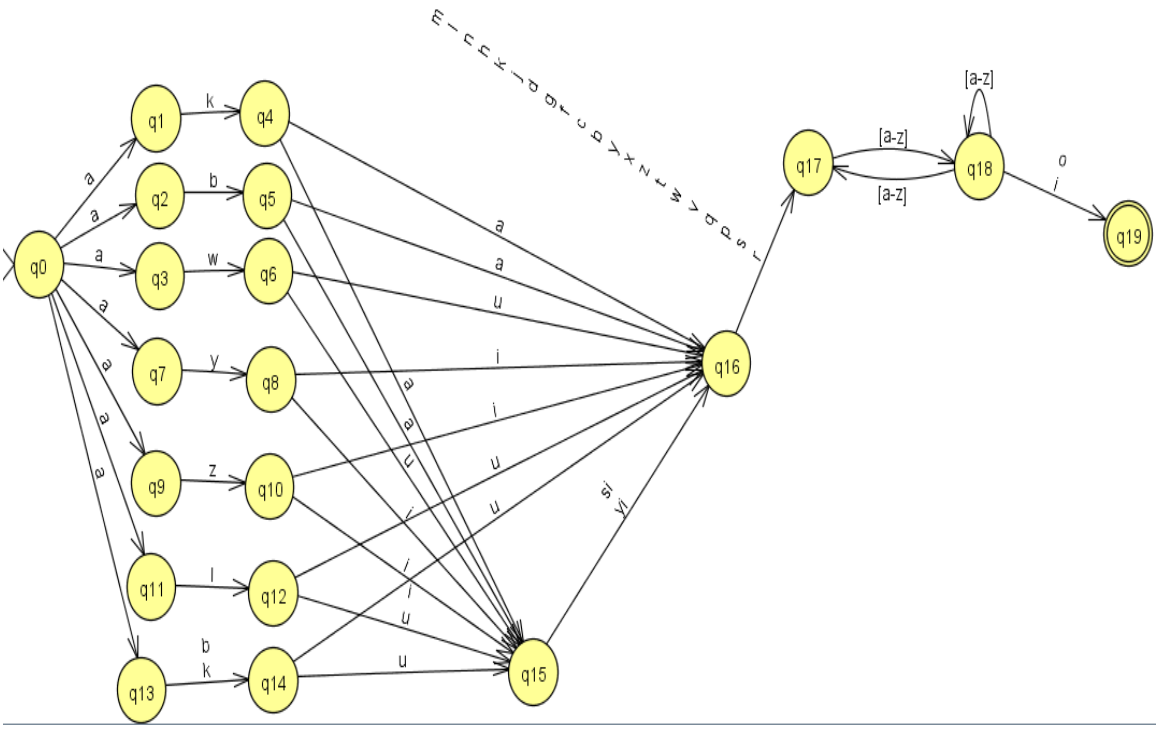

Figure 5: Negation in isiZulu