Nordic Journal of African Studies 14(4): 438–451 (2005)

Computational Description of Verbs in Disjoining Writing Systems*

ARVI HURSKAINEN
University of Helsinki
LUIS LOUWRENS
University of South Africa
GEORGE POULOS
University of South Africa

ABSTRACT

In this paper we discuss the problems encountered in the morphological description of verbs in those Bantu languages, which use a disjoining writing system. In one method, verb structures are first identified and concatenated by using a heuristic pre-processor, and the morphological description is based on this new writing system. In another method, the morphological description is performed directly by using the finite state tool package of Xerox. Test languages are Kwanyama and Northern Sotho. Both methods are evaluated and their limitations discussed, including the memory problems encountered in describing non-concatenative processes.

Keywords: writing methods, finite state methods, morphology, Bantu languages, reduplication

INTRODUCTION

Many Bantu languages, especially in Southern Africa, have a writing system, where most verb morphemes preceding the verb stem and some suffixes are written as separate words. These languages have also other writing conventions, which differ from the way they are written in other related languages. These two systems are conventionally called disjoining and conjoining writing systems. Disjoining writing can be considered simply as an under-specified way of writing, but for computational description it is a challenge, especially if the system allows only continuous sequences of characters to be recognised as units of analysis. In order to reduce unnecessary ambiguity, verb morphemes should be isolated from such strings of characters that are real words.

There are at least two approaches for handling disjoining writing. (a) Each continuous string of characters is considered a 'word' and ambiguity is resolved

* We wish to thank Lauri Karttunen for introducing part of the solutions for solving the types of problems discussed here and for testing versions of the Northern Sotho lexicon.

after morphological description (Hurskainen 2004a). (b) Disjoining writing is first converted to conjoining writing, marking the morpheme boundaries, and the morphological description is carried out on the basis of this new writing form (Hurskainen and Halme 2001).

In solution (a), each string of characters is described with all possible interpretations, including non-verbal interpretations. The analysis result will be heavily ambiguous, and the actual description of the verb structure is carried out as a post-morphological operation. In order to make the overall disambiguation process sensible, it would be important to isolate the structure of the verb (i.e. the verb stem together with its all affixes) and handle it as a single unit. For doing this, one could consider the method used for identifying inflecting idioms (Hurskainen 2004a).

Solution (b) requires a complex tokeniser, which first identifies such sequences of strings that together constitute a verb. This makes it possible to optimise the input strings to the analyser, which operates on a word-per-line basis. The use of this kind of tokeniser makes it possible to handle the text as it were written with a conjoining writing system.

Below we shall discuss problems and solutions involved in the latter approach. The possibility of constructing a reliable tokeniser for concatenating verb structures was tested using Kwanyama as a test language. This language was used in tests, because there was available a detailed description of the verb structure, a covering list of verbs, and sufficiently texts for testing. The implementation of the morphological parser of verbs was tested using Northern Sotho. This language was selected, because it has, in addition to a complex verb structure, a number of interesting features, including a set of morphophonological changes in the beginning of the verb stem if preceded by certain prefixes, and non-concatenative processes in the verb structure. Also two of the writers of this article are experts in this language.

1. Heuristic Method for Identifying Verb Structures

The possibility of constructing a reliable tokeniser needed in solution (b) was tested for identifying verb structures in Kwanyama, a Bantu language spoken in Northern Namibia and Southern Angola (Hurskainen and Halme 2001). In this method, verb candidates were first identified and marked on the basis of the verb root in text. This was done with the help of a covering list of verb roots. Any word that fulfilled the criteria of a verb root was marked as a potential verb. Then tests were made to see whether the verb candidate fulfils other criteria of a verb, especially a sequence of prefixes and suffixes. If criteria were fulfilled,

concatenation rules would join the verb prefixes, the stem and the suffixes as a single word. The example in (1) illustrates the verb structure of Kwanyama.[1]

(1)

```
1   2  3   4   5  6   7   8  9   10       11    12  13  14    15
COM IV TAM SC TAM INF NEG IT OC1 OC2+REFL VB    FV  PL  1SGOC LOC
    o- ha- tu             ke shi li-      talel-a             ko
```

'we will look it for ourselves'

We see that not all prefixes are written disjointly. Especially the second object concord, which is written conjointly with the verb stem, complicates the identification of verbs. Assuming that the beginning of the verb root is marked by @, then the rule should perform the process illustrated in (2).

(2)

```
ohatu ke shi li{talela ko > ohatu_ke_shi_litalela_ko
```

Concatenation rules were written with two alternative methods. By using Beta, the rules were written concretely, and as a result more than 200,000 rules were needed. In the second method, Perl was used in writing rules. In it, using regular expression notation, a total of 153 rules were needed for handling the verb structures.[2]

Tests showed that the method of rule writing did not have significant effect on processing speed. The average speed, using a PC with the processor of 1 GHz and 512 MB RAM, was more than 1,000 words per second, which is sufficient for most applications.

2. Evaluation of the Heuristic Tokeniser

The method described above has two basic problems. First, because the identification of verbs in text is based on verb roots instead of full verb stems, over-marking is considerable (32% in tests). Also part of verb prefixes will be

[1] Key: COM = comitative affix; IV = initial vowel; TAM = tense, aspect and mood marker; also including negative markers for finate forms; SC = subject prefix indicating the person or noun class of the subject INF = infinitive marker; IT = itive marker ("go and ..."); OC1 = object prefix of the first object of a ditransitive verb; OC2 = object prefix of the second object of a ditransitive verb; REFL = reflexive object marker, alternative for OC2; VB = verb base, including also extended verb forms; FV = verb-final vowel; PL = plural addressee; 1SGOC = object marker of the first person singular; LOC = locative marker; NEG = negative marker of infinitives; FS = final suffix, including the final vowel and the remote past tense marker.

[2] In order to ensure the long-first principle in rule application, the rules were arranged in length order. For the same reason, combined rules with alternative lengths were avoided, so that a short variant in a combined rule would not 'steal' the string from a longer rule, which had not yet been tested.

marked as verbs, because they have the same form as some monosyllabic verbs. However, over-marking is not a serious problem, because, if other criteria for a verb are not fulfilled, concatenation will not be performed.

The second problem is that the rules may concatenate also such sequences of morphemes, which formally can be a single verb but in the given context are two separate words. An example of such a problem is in (2).

(3)
kwa {li {wa {hanga > kwa liwa hanga (wrong)
kwa {li {wa {hanga > kwali wahanga (correct)

Note that some prefixes are marked as verbs, because they fulfil the criteria of a monosyllabic verb. The problem of wrong concatenation can be alleviated with a post-processing program that separates sequences that were wrongly concatenated.

If full verb stems could be used as keys for marking verbs, there would be considerably less over-marking. Because, due to the large number of verbal extensions, including sometimes several extension suffixes, the construction of such a full list is not practical, the restriction of the identification key to the verb root only is a good compromise solution. Obviously by studying carefully, and with a sufficiently large text corpus, problematic cases, wrong concatenations could be handled separately. This applies particularly to monosyllabic verbs and other short verb roots that are sources of wrong concatenation.

The basic principle should be, however, that we make sure that all verbs are marked, even with the risk of over-marking. Wrong concatenations can be handled with a post-processing program.

The verb tokeniser of Kwanyama was tested with a text of 30,835 words, which included 6,584 string-sets to be concatenated. Of these a total of 4,113 were unique string-sets. The recall, i.e. the ability of the program to find verbs, was 98.6 %. Failing concatenations were mainly due to the fact that not all verbs were marked. The precision, in other words the ability of the system to make correct marking and concatenation, was tested with a text of 5,986 words. It contained 1,307 finite verbs, and only 0,2 % of the string-sets were wrongly concatenated. Therefore, the precision of the system in this test was 99,8 %.

We also tested how well verb structures can be identified using the verb prefixes alone as marking criteria without previous marking of verb candidates. In this test a total of 107 concatenations out of 1,307 were wrong. This means that the recall no doubt is 100 %, but the precision only 92 %. Although the result is not bad, the method cannot be considered reliable enough for any serious implementation of a verb tokeniser.

3. Description of Verb Structures

When a verb structure has been identified using a tokeniser, it can be given as a single input string to the morphological analyser. The analyser can be constructed on the basis of this form, whatever it is. It can be a fully concatenated verb-form, with or without morpheme boundary marking, or it can be also in its original disjoint writing form. The only thing that is important is that the full verb-form is correctly identified with all its affixes. Below we shall discuss the construction of a morphological parser of Northern Sotho verbs (Poulos and Louwrens 1994) based on disjoint writing form. The aim is to construct a complete implementation that includes all verb structures and all verb stems.

In addition to disjoint writing, the description of the verb involves also such non-concatenative features as reduplication[3] of the verb stem and the constraining of the co-occurrence of such verb morphemes that are on different sides of the verb stem. All these phenomena are handled in the following implementation, which makes use of the finite state methods developed by Xerox (Beesley and Karttunen 2003). To start with, the verb structure ba ka se sa bonana (they shall no longer see one another) is described in (4).

(4)

```
Multichar_Symbols
  ^[ ^]
LEXICON Root
  SubjPref;
LEXICON SubjPref
  ba+:ba%  NegPref;
LEXICON NegPref
  ka+se+:ka% se%  ProgPref;
LEXICON ProgPref
  sa+:sa%  VStart;
LEXICON VStart
  0:^[{ VStem;
LEXICON VStem
  bon VSuff;
LEXICON VSuff
  VSuffRec;
LEXICON VSuffRec
  +an:an VFinV ;
LEXICON VFinV
  +a:A VEnd;
  +a=Redup:A VEndRedup;
```

[3] In SALAMA (Swahili Language Manager), reduplication was described in the lexicon with the basic finite state method (Koskenniemi 1983), and the reduplicated forms frequently used were found from a corpus of 12 million words (Hurskainen 2004b).

```
LEXICON VEnd
  0:}^1^] #;
LEXICON VEndRedup
  0:^2^] #;
```

We see that in the lower language the blank after the verb prefix is described as a literal space by inserting the percent character '%' to denote the literal interpretation of the following character, that is the space. Note that any additional blanks following the first blank are not interpreted as a space character. Therefore, there has to be at least two blanks after the morpheme before the continuation class name, one for the literal blank character and another for separating the lexical string from the continuation class name.

When marking the lexical blanks in this way, it is assumed that the tokeniser has converted the text into a word-per-line format in the sense that each multi-word construction, such as full verb-forms with disjoint parts, is on a line of its own. If we want to construct a system which recognises also the line feed in its different varieties and the tab character, we can, in stead of directly marking the lexical blank in the lexicon, mark it with an arbitrary symbol, e.g. with a literal underscore '_', and then in rules interpret it to mean all the positions, where the morpheme can occur in text. An example of such a rule is in (5).

(5)

```
   %_ -> [" " | "\t" | "\n" | "\r"]+ ;
```

The lexicon in (4) is constructed so that it handles also the reduplication. In order to make the upper language more readable and to show clearly the morpheme boundaries, the morphological tags have been excluded. The lower language is not yet the surface representation of Northern Sotho but rather a meta-language, which contains a sequence of characters that are in the form of a regular expression. This is needed for reduplication to operate. The form of the upper language and meta-language is illustrated in (6).

(6)

```
xfst[0]: read lexc < NSotho.lex
Reading from 'NSotho.lex'
Root...1, SubjPref...1, NegPref...1, ProgPref...1, VStart...1,
VStem...1, VSuff...1, VSuffRec...1, VFinV...2, VEnd...1,
VEndRedup...1
Building lexicon...Minimizing...Done!
2.3 Kb. 41 states, 41 arcs, 2 paths.
Closing 'NSotho.lex'
xfst[1]: print upper-words
ba+ka+se+sa+bon+an+a
ba+ka+se+sa+bon+an+a{Redup}
xfst[1]: print lower-words
ba ka se sa ^[{bonanA}^1^]
ba ka se sa ^[{bonanA}^2^]
```

Particularly important in the lower-side language is the section of the string that is subject to reduplication. This section is delimited with special multi-character symbols ^[and ^]. Whatever is between these symbols is a regular expression that can be repeated n-times. We also see that the actual string to be defined as a regular expression is enclosed with curly brackets '{' and '}' for making sure that the string is interpreted as a regular expression. The multi-character symbol '^2' in the lower string stands for repeating twice the preceding regular expression enclosed between curly brackets '{' and '}'.

The Xerox tool package contains a compile-replace algorithm, which makes it possible to include finite state operations other than concatenation into the morphotactic description (Beesley and Karttunen 2003: 379-380). In this method of describing non-concatenative phenomena, the initial lexical description is made by concatenating partial strings, usually morphemes, into well-formed words through a finite state lexicon structure. This partly abstract lexical description is mapped to the surface strings by applying morphophonological alternation rules. While in the usual description the lower language represents the orthographically correct word forms, in the compile-replace algorithm the initial network (i.e. the composition of the lexicon and the rules) is left abstract for including meta-morphotactic descriptions of non-concatenative phenomena.

When processing this kind of description, the morphophonological rules and lexicon, which are in the form of regular expressions, are first read and composed into a network. This network contains strings which also include meta-morphotactic descriptions in the form of regular expressions. The compile-replace command is applied to the lower side of the initial network, where it finds the meta-morphotactic descriptions, compiles them as regular expressions and replaces them in the lexicon network with the new network resulting from the compilation (Beesley and Karttunen 2003: 381-382). The compilation of the final network and its operation is demonstrated in (7).

(7)

```
xfst[0]: read regex < NSotho.rul
Opening file NSotho.rul...
3.6 Kb. 27 states, 249 arcs, Circular.
Closing file NSotho.rul...
xfst[1]: read lexc < NSotho.lex
Reading from 'NSotho.lex'
Root...1, SubjPref...1, NegPref...1, ProgPref...1, VStart...1,
VStem...1, VSuff...2, VSuffRec...1, VFinV...2, VEnd...1,
VEndRedup...1
Building lexicon...Minimizing...Done!
2.3 Kb. 41 states, 42 arcs, 4 paths.
Closing 'NSotho.lex'
xfst[2]: compose
2.3 Kb. 41 states, 42 arcs, 4 paths.
xfst[1]: compile-replace lower
4 regular expressions compiled successfully. No errors.
```

```
2.4 Kb. 43 states, 45 arcs, 4 paths.
xfst[1]: up ba ka se sa bona
ba+ka+se+sa+bon+a
xfst[1]: up ba ka se sa bonana
ba+ka+se+sa+bon+an+a
xfst[1]: down ba+ka+se+sa+bon+an+a
ba ka se sa bonana
xfst[1]: up ba ka se sa bonanabonana
ba+ka+se+sa+bon+an+a{Redup}
xfst[1]: down ba+ka+se+sa+bon+an+a{Redup}
ba ka se sa bonanabonana
```

If we enter a string where the reduplicated part is composed of two different, alone legal, verb forms, the test fails.

```
xfst[1]: up ba ka se sa bonanabona
xfst[1]: up ba ka se sa bonabonana
```

4. Constraining the Co-occurrence of Morphemes

The full description of the verb in Northern Sotho contains a number of structures, where the verb-final vowel, or a suffix, constrains the co-occurrence of certain verb prefixes. An example of such a case is in (8).

(8)

	1SG	FUT	write	VF		1SG	write	VF
(a)	ke	tla	bon-	a	(b)	ke	bon-	E
	‘I may see in future’					‘I may see’		
(c)	ke		bon-	e	(d)	ke	bon-	ilE
	‘I see usually’					‘I have seen’		

Here we have four cases, where the marker of the correct word form is the verb-final vowel or suffix. Because the marker is after the verb stem, it is not practical to construct the finite state lexicon separately for each case.

The Xerox tools offer a method for handling such cases. A set of flag diacritics can be used in the lexicon for controlling the co-occurrence of certain morphemes. In this implementation, we have used a pair of the P-type and R-type flag diacritics for controlling the morpheme sequences (Beesley and Karttunen 2003: 353-355). With a P-type (positive) flag diacritic, which is placed in the string of characters on the left side of the verb stem, the named feature is set to a given value. The corresponding R-type (require) flag diacritic is placed in the morpheme which functions as the constraining trigger. If the R-type flag diacritic finds in the same string a flag diacritic that has the same feature and the same value as it itself, the test succeeds and the string is accepted. An example of the use of this method is in (9).

(9)

```
Multichar_Symbols
  ^[ ^]
  @P.PAST.ilE@  @R.PAST.ilE@
  @P.SBJN.a@    @R.SBJN.a@
  @P.SBJN.E@    @R.SBJN.E@
  @P.HABIT.e@   @R.HABIT.e@
  @P.NORM.a@    @R.NORM.a@
LEXICON Root
  SubjPref;
LEXICON SubjPref
  ke=Sbjn+@P.NORM.a@:@P.NORM.a@ke%  FutPref;
  ke=Sbjn+@P.SBJN.E@:@P.SBJN.E@ke%  VStart;
  ke=Habit+@P.HABIT.e@:@P.HABIT.e@ke%  VStart;
  ke=Perf+@P.PAST.ilE@:@P.PAST.ilE@ke%  VStart;
LEXICON FutPref
  tla=Fut+:tla%  VStart;
LEXICON VStart
  0:^[{ VStem;
LEXICON VStem
  bon VFinV;
LEXICON VFinV
  +a@R.NORM.a@:@R.NORM.a@A VEnd;
  +a=Redup@R.NORM.a@:@R.NORM.a@A VEndRedup;
  +E@R.SBJN.E@:@R.SBJN.E@E VEnd;
  +E=Redup@R.SBJN.E@:@R.SBJN.E@E VEndRedup;
  +e@R.HABIT.e@:@R.HABIT.e@e VEnd;
  +e=Redup@R.HABIT.e@:@R.HABIT.e@e VEndRedup;
  +ilE@R.PAST.ilE@:@R.PAST.ilE@ilE VEnd;
  +ilE=Redup@R.PAST.ilE@:@R.PAST.ilE@ilE VEndRedup;
LEXICON VEnd
  0:}^1^] #;
LEXICON VEndRedup
  0:}^2^] #;
```

Flag diacritics have to be defined in the section Multichar_Symbols and they are written so that they are visible on the upper and lower language in order to operate correctly in applying to both directions. In (10) are examples of how the constraints function when the meta-language is a lower language.

(10)

```
xfst[1]: print upper-words
ke=Sbjn+tla=Fut+bon+a=Redup
ke=Sbjn+tla=Fut+bon+a
ke=Sbjn+bon+E=Redup
ke=Sbjn+bon+E
ke=Habit+bon+e=Redup
ke=Habit+bon+e
ke=Perf+bon+ilE=Redup
ke=Perf+bon+ilE
```

```
xfst[1]: print lower-words
ke tla ^[{bonA}^2^]
ke tla ^[{bonA}^1^]
ke ^[{bonE}^2^]
ke ^[{bonE}^1^]
ke ^[{bone}^2^]
ke ^[{bone}^1^]
ke ^[{bonilE}^2^]
ke ^[{bonilE}^1^]
xfst[1]: up ke tla ^[{bonA}^2^]
ke=Sbjn+tla=Fut+bon+a=Redup
xfst[1]: up ke ^[{bonE}^1^]
ke=Sbjn+bon+E
xfst[1]: down ke=Habit+bon+e=Redup
ke ^[{bone}^2^]
xfst[1]: down ke=Habit+bon+e
ke ^[{bone}^1^]
```

We see that only the correct forms are accepted, although the lexicon without flag diacritics would allow the realisation of all types of verb endings listed in the lexicon.

The tests in (11) show that the constraints work also when the final net for describing the surface language is compiled using the compile-replace lower function.

(11)

```
xfst[1]: compile-replace lower
8 regular expressions compiled successfully. No errors.
3.6 Kb. 84 states, 93 arcs, 32 paths.
xfst[1]: up ke tla bona
ke=Sbjn+tla=Fut+bon+a
xfst[1]: up ke bone
ke=Habit+bon+e
xfst[1]: up ke bonE
ke=Sbjn+bon+E
xfst[1]: up ke bonilE
ke=Perf+bon+ilE
xfst[1]: up ke bonebone
ke=Habit+bon+e=Redup
xfst[1]: up ke tla bonabona
ke=Sbjn+tla=Fut+bon+a=Redup
xfst[1]: down ke=Perf+bon+ilE
ke bonilE
```

5. Implementing the Full Verb System

The full description of the Northern Sotho verb is much more complicated than what is described above. The morpheme slots, which mark agreement for each noun class, have a total of twenty alternative prefixes, including first and second

person singular and plural. Morpheme slots of this type are (a) the subject prefix, which can be repeated after tense-aspect marking in some forms, and (b) the object prefix. Also verb extensions increase the number of possible forms. An example of alternative morphemes in the subject prefix slot is in the extract from the morphological lexicon in (12).

(12)

```
Lexicon SubjPref
SP-1SG+:ke%  NegTensePrefProgPref;
SP-1PL+:re%  NegTensePrefProgPref;
SP-2SG+:o%  NegTensePrefProgPref;
SP-2PL+:le%  NegTensePrefProgPref;
SP-1+:o%  NegTensePrefProgPref;
SP-2+:ba%  NegTensePrefProgPref;
SP-3+:o%  NegTensePrefProgPref;
SP-4+:e%  NegTensePrefProgPref;
SP-5+:le%  NegTensePrefProgPref;
SP-6+:a%  NegTensePrefProgPref;
SP-7+:se%  NegTensePrefProgPref;
SP-8+:di%  NegTensePrefProgPref;
SP-9+:e%  NegTensePrefProgPref;
SP-10+:di%  NegTensePrefProgPref;
SP-14+:bo%  NegTensePrefProgPref;
SP-15+:go%  NegTensePrefProgPref;
SP-16+:go%  NegTensePrefProgPref;
SP-17+:go%  NegTensePrefProgPref;
SP-18+:go%  NegTensePrefProgPref;
```

Some amount of complexity is added also by the object prefix of the first person singular, which is a nasal. It causes several types of sound changes in the first phoneme of the verb stem, depending on the initial sound of the stem. The phonological processes caused by the nasal object prefix are described in (13).

(13)

```
N+b > mp          N+m > mm
N+bj > mpS        N+r > nth
N+f > mph         N+s > ntsh
N+g > nkg         N+S > ntSh
N+h > nkh         N+y > nki
N+j > ntS         N+w > mku
N+l > nt          N+V > nkV
```

These prefixes are written conjunctively, although the rest of the object prefixes (which do not cause sound changes) are written disjunctively. This requires that the object prefixes of the first person singular have to be handled separately, either by rules or directly in the lexicon. In the current implementation they were described in the lexicon. This means that those verb stems that undergo sound changes have to be listed separately in the lexicon system and a route from the

prefix structure, bypassing the normal object prefix sub-lexicon, has to be constructed for them.

There are also a number of structures, where the verb-final A is realised as E (open) or e (closed), depending on the structure concerned. A set of flag diacritics is applied for constraining the co-occurrence of such morphemes that would produce ungrammatical constructions.

The normal compilation of the lexicon of this size and complexity into a transducer is no problem, although the verb structure produces more than 4 billion paths. The memory problem will be encountered when 'compile-replace lower' is applied to this initial network.[4]

It is possible to reduce the number of paths by merging identical morphemes in a morpheme slot into a single entry and return them into the original form in the post-processing phase. By this method, the maximum number of morphemes in a morpheme slot is reduced from twenty to eleven and the number of paths is reduced accordingly. The lexicon in (12) can be written in an under-specified form as shown in (14).

(14)

```
Lexicon SubjPref
SP-1SG+:ke%  NegTensePrefProgPref;
SP-1PL+:re%  NegTensePrefProgPref;
{SP-1}{SP-2SG}{SP-3}+:o%  NegTensePrefProgPref;
{SP-2PL}{SP-5}+:le%  NegTensePrefProgPref;
SP-2+:ba%  NegTensePrefProgPref;
{SP-4}{SP-9}+:e%  NegTensePrefProgPref;
SP-6+:a%  NegTensePrefProgPref;
SP-7+:se%  NegTensePrefProgPref;
{SP-8}{SP-10}+:di%  NegTensePrefProgPref;
SP-14+:bo%  NegTensePrefProgPref;
{SP-15}{SP-16}{SP-17}{SP-18}+:go%  NegTensePrefProgPref;
```

We see that, for example, the prefix go has four interpretations, each of which is described on the upper side, but the description requires only one path for it. The analysis result can then be easily transformed to meet the requirements of further processing, such as disambiguation.

Another, and more efficient, method for handling the memory problem is to cut the lexicon into parts, so that only the section requiring a regular expression notation, i.e. the verb stems, will be compiled with compile-replace lower. Because the verb reduplication concerns the verb stem only, the section of prefixes can be treated as a partial lexicon of its own. Also the part of the lexicon that undergoes the compile-replace operation can be cut into parts. For test purposes, the verb stem lexicon was compiled with the compile-replace operation in two parts. Then the three partial compiled lexicons, i.e. the prefix lexicon and

[4] For discussion on memory problems see Beesley and Karttunen 2003: 418-420.

two verb stem lexicons were pulled together as a single net. An example of the sequence of operations is in (15)

(15)

```
read lexc < VerbsNorm.lex
compile-replace lower
save VerbsNorm.net
clear stack

read lexc < VerbsObj.lex
compile-replace lower
save VerbsObj.net
clear stack

read lexc < Pref.lex
save Pref.net
clear stack

regex @"Pref.net" [ @"VerbsNorm.net" | @"VerbsObj.net"];

eliminate flag SG1
save Sotho.net
```

Using this method, it was possible to compile the full Northern Sotho verb lexicon with more than 4 billion paths. The total compilation time, using a PC with the processor of 1 GHz and 512 MB RAM, was less than two minutes.

CONCLUSION

The experiments with Kwanyama and Northern Sotho suggest that it is possible to describe the morphology of Bantu languages, including reduplication and other types of non-concatenative morphology, with finite state methods. Languages that allow two object prefixes or a relative prefix have a more complicated structure than Northern Sotho. However, using the method of cutting the lexicon into sections for compilation, the memory problems can be solved. Tests also show that it is possible to describe disjoint writing and verb reduplication as well as constrain the co-occurrence of morphemes within the same system. However, a carefully designed tokeniser is a prerequisite for identifying verb structures in text.

BIBLIOGRAPHY

Beesley, Kenneth and Karttunen, Lauri, 2003.
Finite State Morphology. Series: **CSLI Studies in Computational Linguistics.** Stanford: Center for the Study of Language and Information.

Hurskainen, Arvi, 2004a.
Optimizing Disambiguation in Swahili. In *Proceedings of COLING-04, The 20th International Conference on Computational Linguistics*, Geneva 23-27.8. 2004. Pp. 254-260.

Hurskainen, Arvi, 2004b.
Swahili Language Manager: A Storehouse for Developing Multiple Computational Applications. **Nordic Journal of African Studies** 13(3): 363-397. Also in: www.njas.helsinki.fi

Hurskainen, Arvi and Halme, Riikka, 2001.
Mapping between Disjoining and Conjoining Writing Systems in Bantu Languages: Implementation on Kwanyama. **Nordic Journal of African Studies** 10(3): 399-414.

Koskenniemi, Kimmo, 1983.
Two-level morphology: A general computational model for word-form recognition and production. Publications No.11. Department of General Linguistics, University of Helsinki.

Poulos, George and Louwrens, Louis J. 1994.
A Linguistic Analysis of Northern Sotho. Pretoria: Via Afrika Ltd.