5

Solutions for Handling Non-concatenative Processes in Bantu Languages

ARVI HURSKAINEN

The paper discusses the description of non-concatenative processes in word formation by using examples from Bantu languages. The focal point is especially the verb, which may have up to fifteen morpheme slots. Because of space restrictions, only reduplication and non-cumulative morpheme concatenation will be discussed and solutions for implementation will be demonstrated. Some solutions require the use of such environments as provided by the Xerox tool package and Koskenniemi's Two-level morphology.

5.1 Introduction

Bantu languages display a number of features that cannot be effectively handled by using the basic finite state processing. Examples of such features include, for example, the full-stem reduplication of verbs, the non-cumulative sequence of verb morphemes, and the disjoining writing system.

Bantu languages exhibit a productive process of verb stem reduplication. Reduplicated forms cannot be described simply by adding reduplicated stems into the dictionary, because it is not the verb root but the full, possibly extended, stem of the verb that is reduplicated. When a verb may have up to 20 different extended stems, and each stem has at least three different surface forms, the listing of all of these forms in the lexicon is not practical.

Verb structures in Bantu languages are complex, comprising up to fifteen morpheme slots. There are a number of rules that restrict the co-occurrence

Inquiries into Words, Constraints and Contexts.
Antti Arppe et al. (Eds.)

of certain morphemes. For the morphemes that precede the verb stem, it is possible to construct separate routes through the morpheme slots, allowing some, and disallowing others, from occurring in the sequence of morphemes. When the constraining morpheme is after the verb root, the problem cannot be solved by constructing separate routes, because this would require also the multiplication, perhaps several times, of the verb stems in the lexicon.

Some Bantu languages have adopted a writing system where verb morphemes, especially those preceding the stem, are written as separate words, while other languages, closely related to these, use a conjoining writing system. The problem in computational processing is how to keep these morphemes separate from similar morphemes which are not part of the verb and constitute grammatical words in themselves. Because of space restrictions, this problem is not discussed here.

The problems of implementing reduplication and the non-cumulative sequence of verb morphemes will be discussed below and solutions to them will be demonstrated. Some of the solutions have been implemented with general-purpose tools, and others require the use of such environments as provided by the Xerox tool package and Koskenniemi's Two-level morphology, licensed by Lingsoft.

Acknowledgements

I wish to thank Kimmo Koskenniemi for his advice in the early phases of developing the Swahili morphological parser in the 1980s, and Lauri Karttunen for introducing the solutions for solving the types of problems discussed here.

5.2 Reduplication

The extensive use of reduplication for grammatical and semantic purposes is a distinctive feature in most African languages. Part of reduplication is fairly easy to describe in the lexicon. Some types of reduplication, however, can hardly be described in a satisfactory way by simply concatenating morphemes. Below I shall describe two methods for handling reduplication.

5.2.1 Using basic finite state methods

In SALAMA, Swahili Language Manager (Hurskainen 2004b), reduplication was implemented by using the basic finite state concatenation. This led to a somewhat strange situation because, although the extended forms of verb stems were described with the help of continuation classes and corresponding sub-lexicons, each reduplicated verb stem had to be written in full. Thus the same basic verb may occur in the lexicon several times in different forms. Reduplicated forms were added to the lexicon if they occurred in texts. The verbs of the corpus of 15 million words have so far been included and new reduplicated forms are added when they are found in new texts. An example

of how the reduplicated forms of the verb `kata` (to cut) are described in this system is in (1).

(1)
```
Lexicon VRoot
katAkat       redup/V "katakat 'cut' SV SVO ";
katikAkatik   redup/V "katakat 'cut' SV STAT ";
katishAkatish redup/V "katakat 'cut' SVO CAUS ";
katizAkatiz   redup/V "katakat 'cut' SVO CAUS ";
katizwAkatizw redup/V "katakat 'cut' SV CAUS PASS ";
katwAkatw     redup/V "katakat 'cut' SV PASS ";
Lexicon Redup/V
A End;
A GenRel;
Lexicon GenRel
ye End;
```

Note that the use of two-level rules reduces the need of listing verb entries, because the rules handle the surface realisation of the verb-final A. Without the use of rules, each entry should be written three times, one for A > a as default, another for A > i in present tense negative, and another for A > e in subjunctive. The rules convert the A in the middle and at the end of the verb as required.

In addition to verbs, reduplication occurs frequently in pronouns and adverbs, and to a limited extent in adjectives. Reduplication has in these contexts mainly a semantic role, which has to be taken into consideration in a bilingual lexicon (Hurskainen 2004a). If reduplicated words, which do not inflect, are written as a single word without a space in between, they are easy to describe in the lexicon. There are, however, reduplicated adjectives with an alternating prefix defined by the noun class, written together as a single word. Some examples of the word `zuri` (good) are in (2).

(2)
```
mzurimzuri   End "zuri Adj 1/2-SG '  good '";
wazuriwazuri End "zuri Adj 1/2-PL '  good '";
mizurimizuri End "zuri Adj 3/4-PL '  good '";
kizurikizuri End "zuri Adj 7/8-SG '  good '";
zurizuri     End "zuri Adj 9/10-SG ' good '";
pazuripazuri End "zuri Adj 16-LOC '  good '";
```

A more elegant way of describing reduplicated adjectives is to formulate them as regular expressions. In this method, each of the basic stems is listed only once. The stem receives its prefixes from the sub-lexicon of prefixes, and the concatenated word is optionally reduplicated. This will be described in more detail in (11-12).

In sum, the method of describing the reduplicated stems directly in the lexicon has disadvantages, but also advantages. One disadvantage is that it is rather tedious to keep track of all real instantiations of reduplicated verbs. The method also requires continuing follow up because it is based on corpus occurrences and not on the grammatical word formation rules. On the positive side we can see the accuracy of the system because no potential verbs are there which do not occur in text. This saves the system from testing unnecessary paths, and also eliminates the risk of unnecessary additional ambiguity.

5.2.2 Solution based on regular expressions

It is well known that if a string or a sequence of strings can be expressed in the form of a regular expression, it can be repeated. Limited reduplication, repetition of strings, can be achieved even with two-level rules, although the power of this method is not sufficient for handling full scale applications such as Swahili and other Bantu languages.

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, following the terminology of Xerox, 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 example in (3) illuminates how the above description is implemented with the Swahili verb `sema` (to say).

(3)
```
Multichar_Symbols
^[  ^]  @U.GENREL.abs@  @U.GENREL.pres@
@U.OBJ.abs@  @U.OBJ.pres@
Lexicon Root
Pref0@;   Pref1;   AdjStart;
```

```
Lexicon Pref0@
@U.GENREL.abs@   Pref0;
Lexicon Pref0
ha=Neg+:haˆ   VStart;
ha=Neg+:haˆ   Pref4;
a=Subjn+:a!   VStart;
a=Sbjn+:a!    Pref4;
Lexicon Pref1
< {a=Sp+}:{a} "@U.GENREL.abs@" >   Pref2@;
a=Sp+:a                            Pref4;
a=Sp+:a                            VStart;
Lexicon Pref2@
@U.GENREL.abs@ Pref2;
Lexicon Pref2
na=Pres+:na   VStart;
na=Pres+:na   Pref3@;
na=Pres+:na   Pref4;
Lexicon Pref3@
"@U.GENREL.abs@"   Pref3;
Lexicon Pref3
ye=1/2-Sg-Rel+:ye   VStart;
ye=1/2-Sg-Rel+:ye   Pref4;
Lexicon Pref4
< {ki=7/8-Sg-Obj+}:{ki} "@U.OBJ.pres@" >   VStart;
Lexicon VStart
@:@ˆ[{   VStem;
Lexicon VStem
sem   VSuff;
Lexicon VSuff
+esh=Caus:Ish   VSuff2;
+esh=Caus:Ish   VFinV ;
VSuff2; VFinV;
Lexicon VSuff2
< {+w=Pass}:{w} "@U.OBJ.abs@" >   VFinV;
Lexicon VFinV
+a:A          EndSimple;
+a=Redup:A    EndRedup;
Lexicon EndSimple
0:}ˆ1ˆ]    End;
0:}ˆ1ˆ]    GenRel;
Lexicon EndRedup
0:}ˆ2ˆ]    #;
0:}ˆ2ˆ]    GenRel;
Lexicon GenRel
< {+ye=1/2-Sg-GenRel}:{ye} "@U.GENREL.pres@" > #;
```

```
Lexicon AdjStart
0:0^[{   AdjPref;
Lexicon AdjPref
m=1/2-SG+:m     AdjRoot;
wa=1/2-PL+:wa   AdjRoot;
mi=3/4-PL+:mi   AdjRoot;
ki=1/2-SG+:ki   AdjRoot;
0=9/10-SG+:0    AdjRoot;
pa=16-LOC+:pa   AdjRoot;
Lexicon AdjRoot
zuri    EndSimple;
zuri    EndRedup;
```

In the lexicon above, the upper-side language is represented in such a way that it contains a sequence of lexical morphemes and their grammatical flags, and morpheme boundaries are shown with a plus sign. The lower-side language is also abstract in that it contains characters in upper case that are subject to alternation rules for producing correct surface forms. 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 manipulated accordingly, in this case, repeated. 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. In (4) is an example of how the surface string `anasemeshasemesha` (he makes to speak) is represented in the lexicon. Note that the influence of alternation rules is here excluded.

(4)
```
upper: a=1/2-Sg-Sp+na=Pr+sem+esh=Caus+a=Redup
lower: a           na@^[{sem Ish      A}^2^]
```

The multi-character symbol `^2` in the lower string stands for repeating, i.e. the preceding regular expression enclosed between curly brackets { and } is repeated. The alternation rules rewrite the I and A as needed in the surface string. In (5) we show in stages how the final network is compiled by using a script file.

(5)
```
xfst -e "read regex < rules.txt"
-e "read lexc < redup.lex"
-e "compose"
-e "compile-replace lower"
-e "substitute symbol 0 for Caret"
-e "substitute symbol 0 for !"
-e "substitute symbol 0 for @"
-e "save redup.fst"
-stop
```

Note that all diacritics needed as triggers in alternation rules are deleted in the final network. They are substituted with the zero symbol. The command sequence in (6) shows how the network operates.

(6)
```
xfst[0]: load redup.fst
Opening 'redup.fst'
Closing 'redup.fst'
xfst[1]: up anasema
a=1/2-Sg-Sp+na=Pr+sem+a
xfst[1]: up anasemasema
a=1/2-Sg-Sp+na=Pr+sem+a=Redup
```

We see that both the simple and reduplicated stems are analysed. The simple stem is analysed when the repetition trigger is set to `^1`. We can also test the network in the other direction, as shown in (7).

(7)
```
xfst[1]: down a=1/2-Sg-Sp+na=Pr+sem+a
anasema
xfst[1]: down a=1/2-Sg-Sp+na=Pr+sem+a=Redup
anasemasema
```

As is shown in (8), adding verb affixes does not affect the correct realisation of the verb stem.

(8)
```
xfst[1]: up anasemesha
a=1/2-Sg-Sp+na=Pr+sem+esh=Caus+a
xfst[1]: up anasemeshasemesha
a=1/2-Sg-Sp+na=Pr+sem+esh=Caus+a=Redup
xfst[1]: up anayesemeshasemesha
a=1/2-Sg-Sp+na=Pr+ye=1/2-Sg-Rel+sem+esh=Caus+a=Redup
xfst[1]: up anakisemeshasemesha
a=1/2-Sg-Sp+na=Pr+ki=7/8-Sg-Obj+sem+esh=Caus+a=Redup
```

The lexicon in (3) shows that the verb stem is not always the last element in the verb. For example, the marker of the general relative is attached to the end of the verb stem, and this suffix is not reduplicated. The formalism also handles such cases, as shown in (9). Ungrammatical concatenations cause a failure.

(9)
```
xfst[1]: up asemaye
a=1/2-Sg-Sp+sem+a+ye=1/2-Sg-GenRel
xfst[1]: up asemasemaye
a=1/2-Sg-Sp+sem+a=Redup+ye=1/2-Sg-GenRel
xfst[1]: up akisemasemaye
a=1/2-Sg-Sp+ki=7/8-Sg-Obj+sem+a=Redup+ye=1/2-Sg-GenRel
xfst[1]: up anasemasemaye
```

The negative present and subjunctive affect the verb-final vowel, and this

is implemented with alternation rules as shown in (10). The analysis fails if the final vowel is not correct.

(10)
```
xfst[1]: up hasemi
ha=Neg1/2-SG-SP+sem+a
xfst[1]: up hasemisemi
ha=Neg1/2-SG-SP+sem+a=Redup
xfst[1]: up aseme
a=Sbjn1/2-SG-SP+sem+a
xfst[1]: up asemeseme
a=Sbjn1/2-SG-SP+sem+a=Redup
xfst[1]: up hasema
xfst[1]: up hasemasema
xfst[1]: up haseme
```

5.2.3 Reduplicated adjectives

We saw in (2) that inflecting reduplicated adjectives require multiple listing in the dictionary if only the basic concatenation method in the system is available. Here we show that this can be avoided by describing the adjective, together with its prefix, as a regular expression. In the example lexicon (3) the solution for adjectives is also demonstrated. The adjective `zuri` (good) is described, together with a sample of alternative prefixes. Test examples in (11) show that both the simple and reduplicated forms are recognised and analysed accordingly.

(11)
```
xfst[1]: up mzuri
m=1/2-SG+zuri
xfst[1]: up mzurimzuri
m=1/2-SG+zuri=Redup
xfst[1]: up wazuriwazuri
wa=1/2-PL+zuri=Redup
xfst[1]: up pazuri
pa=16-LOC+zuri
xfst[1]: up pazuripazuri
pa=16-LOC+zuri=Redup
```

When the upper-side language is applied to the lower-side language, we get the correct surface forms. When a string with different prefixes in the first and second part of the reduplicated stem is entered, the test fails.

(12)
```
xfst[1]: down m=1/2-SG+zuri=Redup
mzurimzuri
xfst[1]: down pa=16-LOC+zuri=Redup
pazuripazuri
xfst[1]: up kizurizuri
xfst[1]: up zurikizuri
xfst[1]: up kizuripazuri
```

5.3 Non-concatenative dependencies

In the lexicon in (3) there are so-called flag diacritics, the purpose of which is to constrain the occurrence of incompatible features in the same string. The relative marker after the verb stem blocks the occurrence of the relative marker in Pref3, and also a number of other prefixes. Another similar case is that the object prefix cannot co-occur with the passive marker. The triggers for both types of constraints are located on different sides of the verb stem, and this calls for the use of flag diacritics (Beesley and Karttunen 2003: 339-373). The unification flag diacritics are used in the lexicon for preventing the co-occurrence of unwanted features in the same string. In the current example lexicon (3), the flag diacritics are made visible on the upper and lower side of the transducer, so that they function correctly in analysis and production.

Another possibility for constraining the unwanted combination of morphemes here would be to use pairs of P-type (positive) and R-type (require) diacritics for defining the correct strings, where both of the types of the same flag with the same value must co-occur. Space does not allow the demonstration of this alternative. Examples in (13) show how the constraints with the U-type (unification) flag diacritics work.

(13)
```
xfst[1]: up anayesema
a=1/2-Sg-Sp+na=Pr+ye=1/2-Sg-Rel+sem+a
xfst[1]: up asemaye
a=1/2-Sg-Sp+sem+a+ye=1/2-Sg-GenRel
xfst[1]: up anayesemasema
a=1/2-Sg-Sp+na=Pr+ye=1/2-Sg-Rel+sem+a=Redup
xfst[1]: up asemasemaye
a=1/2-Sg-Sp+sem+a+Redup+ye=1/2-Sg-GenRel
xfst[1]: up anayesemaye
xfst[1]: up asemayesemaye
```

Note that if the verb stem with the general relative suffix is reduplicated, the analysis fails. Only the verb stem, simple or extended, is reduplicated, and the relative marker is attached to the end of the reduplicated stem.

5.4 Conclusion

We have discussed non-concatenative processes that take place in Bantu languages on the word level and tested methods for solving them. Our conclusion on the basis of the tests is that the environment offered by the Xerox tool package offers elegant solutions to the problems discussed. The reduplication of verbs and adjectives and constraining the co-occurrence of non-contiguous morphemes can be described simultaneously and compiled into a network.

Reduplication can be handled also in the basic finite state lexicon, but this is more labor-intensive than with Xerox tools. The number of various redupli-

cated extended verb stems is in practice much more limited than the number of extended non-duplicated verb stems. The number of extra listings required by reduplicated verbs in Swahili for handling normal text is less than 300. The reduplicated adjectives can be listed as such if they occur in real language.

References

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

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.