Nordic Journal of African Studies 10(3): 399-414 (2001)

Mapping between Disjoining and Conjoining Writing Systems in Bantu Languages: Implementation on Kwanyama

HURSKAINEN Arvi and HALME Riikka[1]
University of Helsinki, Finland

ABSTRACT

Several Bantu languages have adopted a disjoining writing system, which forms a special challenge for automatic analysis of written text. In those systems, part of bound morphemes is treated as independent words, while other languages treat equivalent morphemes as affixes of a head morpheme. The concept of word is blurred in disjoining writing systems, because there is no systematic rule system for writing conventions. Not only are bound morphemes written as separate words, also independent words are sometimes written together as one string. The paper makes a claim that disjoining writing systems require a special treatment, before they can be analyzed successfully. This can be done either by pre-processing the text first into a conjoining format, so that the analysis program would get the input in a form that conforms to the linguistically more motivated writing system. Another possibility is to construct the morphological analyzer so that it identifies bound morphemes although they are written as separate words. This paper applies the first choice and describes a system, which maps between a disjoining and conjoining writing system. The implementation was made on Kwanyama, a Bantu language spoken in Southern Angola and Northern Namibia. The performance of the system is evaluated.

Keywords: writing system, language technology, Kwanyama, Bantu language

INTRODUCTION

Kwanyama is a Bantu language spoken by more than half a million speakers in southern Angola and northern Namibia (Zimmermann & Hasheela 1998: 1). According to the classification of Malcolm Guthrie (1971: 60-61) it belongs to the group R21. Along with Oshindonga it belongs to the languages which from the early times have been written by using a disjunctive writing system.[2] Kwanyama is a tone language, but tone is not normally marked in writing.

---

1 The work for this paper was divided so that Riikka Halme provided the data on Kwanyama language and Arvi Hurskainen did the programming, as well as wrote the text.

2 Examples of other languages written with the disjoining writing system are Siswati, Venda (Poulos 1990) and Northern Sotho (Poulos & Louwrens 1994; Makgamatha 1999).

From the viewpoint of automatic language description, both the disjunctive writing system and the absence of tone marking are problems. The morphological analysis of a conjoining writing system, such as the one in used in Swahili, leaves the interpretation of about half of the words ambiguous (Hurskainen 1996: 569). This means that they have more than one correct interpretation as an isolated word. The disjunctive writing system is likely to leave even more ambiguity in readings. What is even more problematic is that the additional ambiguity occurs in short and very frequently occurring 'words'[3], which, however, are not real words but rather bound morphemes. There is ambiguity in regard to their interpretation as bound morphemes[4], as well as in regard to similar strings that are independent morphemes. A further source of ambiguity is the absence of tone marking in text. The treatment of this subject is complex and it will not be dealt with in this paper. The automatic analysis of text written with a disjoining writing system, with the assumption that each string of characters separated by word boundary markers is a word, is possible, but the result of the analysis will contain so much unnecessary ambiguity that the subsequent disambiguation becomes unusually heavy and complex.

There is, therefore, need to isolate this problem from the normal analysis procedure and handle it separately. This can be done in two ways.

In one method, the morphological analyzer is constructed in such a way that bound morphemes, although written as separate words in text, are handled as part of a larger word in analysis. This requires that the analyzer is able to make a difference between a word boundary proper and a quasi-boundary between bound morphemes, although in text both types of boundaries are precisely the same. The facility for crossing quasi-boundaries in text and for keeping them separate from real boundaries is found in the Xerox package of Finate State Tools[5].

In another method, the text is first pre-processed so that it meets the generally accepted standards of word-formation. In other words, the text written in a disjoining writing system is first converted into a conjoining system, and then the text is given as input for the analyzer. In the same process, other text normalization operations needed for the morphological analysis can be done.

---

3 In text these short 'words' occur as strings of characters separated from other strings by an empty space, which is conventionally the separator between words. Such strings are normally handled as words in analysis, although they are linguistically bound morphemes.

4 The formally same verbal affix may occur in two or even three different morpheme slots of the verb structure, and the interpretation of the affix differs according to the position in the sequence of morphemes.

5 The Xerox tool package (Karttunen 2000 ; Beesley and Karttunen 2000) has a facility to write multi-word units, with empty spaces in between, as single lexical strings, while other empty spaces in text are treated as real word boundaries. While lexc, the environment for developing lexical descriptions of language, simply maps between two levels (upper and lower levels) of the language, without defining either of them as lexical or surface levels, it is fairly straightforward, although not necessarily easy in practice, to write descriptions of mapping between a disjoining and conjoining writing system of a language.

Below we discuss the problems involved in the second method. We also present one implementation and give an evaluation of the system that performs the mapping between the disjoining and conjoining writing systems in Kwanyama. The operation, through which bound morphemes are glued together as one word, we call concatenation. We also limit our discussion to concern verbs only. The treatment of tone will not be dealt with here.

1. WHAT SHOULD BE CONCATENATED?

The disjoining writing system in Kwanyama does not mean that all morphemes of the language are written as separate words. In fact it is a mixture of opposite principles. Because this is not a problem from the viewpoint of automatic analysis, we shall leave such morphemes as they are.

The clearest example of where morphemes should be concatenated are the verbal affixes, most of which, but not all, are written as separate words. The verb structure of Kwanyama, with a total of 15 morpheme slots, is schematized in (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	FS	PL	1SGOC	LOC
	O-	ha-	tu				ke	shi	li-	talel-a				ko

'We will look it for ourselves'

Note that this schema shows only the relative location of morphemes, and only part of the morpheme slots can be filled in each individual case. Furthermore, it does not show exactly which of the morphemes are already written together in the present orthography. For example, COM+IV+TAM+SC are written together. On the other hand, GO, OC1 and OC2 are always written separately, except for the reflexive object marker *li,* which is written as part of the verb stem.

Further examples of possible combinations of morpheme slots are in (2-5):

(2)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
COM	IV	TAM	SC				IT			VB	FS			LOC
n-	o-	ta-	mu				ka			mon-	a			po

'and you will see there'

(3)

COM	IV	INF	OC1	OC2+REFL	VB	FS	LOC
n-	o-	ku	ku	shi	p-	a	po

'and to give it to you'

(4)

COM	IV	SC+TAM	VB	FS	LOC
n-	o-	kwa	mon-	a	po

'and s/he saw there'

(5)

	TAM	SC	VB	FS	1SGOC	LOC
	ta		yandj-e		nge	po

's/he gives me away'

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

It is evident that whatever comes to the left of the verb base should be concatenated, because it is part of verbal inflection. On the other hand, it is not clear whether the locative suffixes located after the verb should be joined to the verb. They could also be considered as locative particles and written as separate words. Their location immediately after the verb[6] makes their treatment in analysis rather easy in any case. In the implementation discussed here locative markers are joined to the verb.

[6] Nothing, except for the plural addressee (slot 13) and 1SGOC (slot 14), can come between the verb and the locative marker.

2. PHASES IN IMPLEMENTATION

Below we describe how we have implemented the concatenation system. It consists of a series of programs to be run in a certain order. The algorithm is briefly the following:

1. Normalize the text with a pre-processing program.
2. Identify and mark the verbs first.
3. Concatenate the verbal prefixes and attach them to the verb concerned.
4. Attach the locative marker, and possibly also the marker for the plural addressee and/or 1SGOC, to the end of the verb.
5. Treat ambiguous cases separately and resolve those, which can be safely resolved.
6. Remove remaining excessive verb marks, except for such problematic cases, which should be manually checked.
7. Return the text to normal writing, with capital letters, punctuation marks, diacritics, etc. in original places.
8. Do the final manual checking of unresolved ambiguous cases.

Below we shall describe the process in more detail. However, before that we should say something about the programming language that we have mostly used. We have made use of the utilities of Unix and made the basic programming with Beta, a rewriting language designed originally by Benny Brodda and implemented later also by Fred Karlsson and Kimmo Koskenniemi (1990) by different programming languages. This little known language has made it possible to program almost all of the algorithm. This choice was made, not because we particularly favor it, but because we know for sure that this language can handle such complex rule structures and huge masses of rules, which the task required.

The major disadvantage of Beta in this work was that, because it does not support regular expressions, each rule had to be written on a concrete level. We made tests also with Perl 5.0 and utilized extensively regular expressions supported by it. The number of rules, a total of 187,000 concrete rules in Beta, was reduced to less than 50 rules in Perl. This shows the effectiveness of regular expressions, which allow the combination of a large number of individual rules into one rule with alternative string combinations. Because the implementation with Perl has not yet been completed, below we shall discuss the Beta implementation.

Before we start discussing the Beta rules, it is appropriate to introduce briefly the Beta rewriting language, which facilitates the writing of such rules. Beta rewrites strings according to rules, the application of which can be conditioned through the left and right context, and also by establishing state sets with individual states. Rule application can be further controlled through

directing the cursor position after the application of a rule. Rules are parallel. It means that they all are checked each time for their possible application. Rule ordering can be imitated, however, by forcing the program to perform more than one round for the same input material, whereby on each round a certain set of rules will be tested for application.

In order to illustrate how the rewriting rules operate, look at the following:

```
! X                    Y                lc   rc   sc      rs  mv md
! Rule 1
okwe ke tu shi @; okweketushi; B   C   Begin  2   5    1
! Rule 2
okwe ke tu @; okweketu;          B   C   Begin  2   5    1
```

(Key: X = string to be replaced, Y = replaced string, lc = left context, rc = right context, sc = state condition, rs = resulting state, mv = move of cursor after applying the rule, md = mode of operation)

Rule 1 states: rewrite the string *okwe ke tu shi @* as *okweketushi*, if there is blank (B) on the left side and a character (C) on the right side. The state condition is Begin, which means the initial state. The resulting state is 2, which allows the application of such rules that have this state as condition. The cursor moves (5) to the character following the string to be replaced, i.e. after @, which is a marker of the beginning of the verb root. The role of @ in the beginning of the verb stem is to ensure that the rule applies only to the prefixes in front of the verb stem and not to similar strings elsewhere. Note also that the rule concatenates the morphemes and deletes @, because it is not any more needed.

Rule 2, on the other hand, applies when the OC2 *shi* does not occur in the string.

2.1. PRE-PROCESSING

The text is first run through a pre-processing program which, among other things, returns upper case letters temporarily to lower case while retaining info on their real status, and makes the text into a sentence-per-line format. It depends on the desired end result what kind of pre-processing is needed. If the aim is just to convert the disjoining writing into a conjoining one, only minor modifications are needed. In case the aim is to prepare such a version of text that is ideal for morphological analysis, a much heavier pre-processing is needed.[7]

[7] Various phases of processing of a piece of Kwanyama text are illustrated in APPENDIX.

2.2. IDENTIFICATION AND MARKING OF VERBS

If manual marking is excluded, the only available way of marking verbs is through comparing verbs in the lexicon and the strings found in text. Since verbs are normally listed in the lexicon only in the base form, leaving out the extensions, sometimes quite complicated ones, the marking system had to be constructed in such a way that it allows also the extended forms to be marked as verbs. Since in Kwanyama verbal clitics are attached to the verb root, whereby the verb-final vowel is pushed to the end of the extended verb form, the non-extended verb root, without the final vowel, was considered the string, on the basis of which verbs could be identified in text.

A number of problems were identified in implementation, however. First, most monosyllabic, and some disyllabic, verb roots had so few distinctive features that they mixed with non-verbs, even with the majority of the 'prefixes' of the verbs. However, the risk of over-marking was taken. Second, monosyllabic verbs as well as some disyllabic verbs were treated more carefully, with the whole verb-form as a string to be matched, so as to reduce the likelihood of being mixed with non-verbs. Third, there are several defective verbs with a verbal structure different from the ordinary verbs. Those had to be treated separately. Fourth, although the prefixes of verbs are normally written as separate words, this is not always the case. For example, the reflexive prefix *li* is written as part of the verb. Also the infinitive marker *oku*, as well as the genitive connector with several forms (depending on the noun class), are written together with the head word.

All this was taken into consideration in implementing the rule system. As a result, the program marks the verbs in text and also a number of such non-verbs, which fulfill the criteria defined in the rules. Tests show that there is about 32 % over-marking in the average. Although the percentage is fairly high, this does not matter in cases where the word wrongly marked as a verb is not preceded by such strings that can be considered as verbal prefixes. In an ambiguous case, where there is a shorter and longer sequence of strings, both of which fulfill the conditions of some rule, the longer rule wins. This feature of Beta rules reduces the risk of wrong concatenations.

In brief, verbs are marked in text on the basis of the verb root. Through this method, all real verbs will be marked, provided that they are included into the list of verbs. The fact is that it is very difficult, perhaps impossible, to collect a full list of verb stems of a language. If a verb is not found in the lexicon, it will not be marked as a verb in text and as a result the concatenation of the verbal prefixes (and suffixes) will fail. Also a number of other words, including a large part of the verbal prefixes, will be marked as verbs[8]. Normal verbs are marked

[8] The verbal prefixes marked wrongly as verbs are : *di, hadi, i, kave, ke, lu, na, oka, oke, ola, oli, onda, ota, oya, pe, she, shi, shihe, tali, tu, twe, u, va,* and *we*. In addition, the prefix *li* mixes with the defective verb *li*.

by placing @ at the beginning of the verb root and + at the end of the verb-form. Defective verbs are marked by a double @@ for keeping them separate from the normal verbs. These arbitrarily chosen markers serve as triggers for rules to apply. A sequence of strings can be considered as a set of prefixes of a normal verb only if it is followed by an @ sign. In order to be accepted as prefixes of the defective verbs the set of strings requires a double @@ to follow. The + sign marks the end of the verb stem[9]. It is used as a trigger in the rules for the subsequent locative and other suffixes to be attached to the verb stem.

2.3. CONCATENATING VERBAL PREFIXES AND ATTACHING THEM TO THE VERB STEM

Since normal verbs and defective verbs behave differently, these two categories of verbs have to be treated separately. However, all rules may be placed into the same rule system, because the different verb markers, @ for normal and @@ for defective verbs, take care that the application of a rule is allowed only in the appropriate context. Rules have been so constructed that when a rule applies it performs the concatenation but also deletes the verb marker, which has now become obsolete.

In the present application, written in Beta, all rules were written as separate and concrete string replacement rules, because the system does not allow the use of regular expressions for generalizing rules. On the other hand, although the number of rules became fairly large, a total of about 187,000 rules in this program, such concrete rules are fairly easy to maintain and correct.

2.4. ATTACHING THE LOCATIVE MARKERS TO VERBS

If a locative marker is found immediately after the verb, it is attached to the previous verb. This is performed as part of the program, which concatenates the verbal prefixes. The end mark + of the verb is deleted in the process.

9 Note that the verb root is different than the verb stem. The verb root (without the extensions and final suffix) was used for marking the beginning of the prospective verb stems. The end of the verb stem is identified and marked with a + sign by scanning the found verb root until the end of the verb-form is found.

2.5. ORDER OF RULE APPLICATION

Since replacement rules in Beta are not ordered, any of the rules matching the string in text may apply. This would be a source of major confusion if there were not a specific principle in rule application. A rule with a longer X (i.e. the string to be replaced) part will be applied before the one with a shorter X part. Therefore, the longest possible rule will have precedence in application. This prevents the application of other rules that would match with part of X as well. Therefore, for example for the string-set *@ve+ @ke+ mu @talela+ po*[10] the rule

(6) @ve+ @ke+ mu @; vekemu;

will apply, and not the rule

(7) mu @; mu;

Particularly difficult are those cases where a string may be a real verb in one context and a prefix in another, especially if these two possibilities occur in a single sequence of strings.

(8) oku @ya+ @dilila+

There are two rules applicable to this string. One would concatenate wrongly as *okuya*, and another rule would have the result *yadilila*. Example (9) has four strings where three rules compete for application.

(9) kwa @@li+ @wa+ @hanga+

The correct concatenation is: *kwali wahanga*

One rule would interpret *@wa+* as a verb and *@@li+* as a prefix, and the result would be wrongly: *kwa liwa hanga*.

Which interpretation of these ambiguous strings marked as verbs is correct depends on features that cannot be generalized. In other words, a general rule for resolving this kind of ambiguity cannot be constructed. One safe way is to leave the ambiguity resolution to manual choice. Another possibility is to construct a post-processor where rules handle ambiguous string sets separately and con-

[10] Note that the program for marking verbs has marked also part of verbal prefixes as verbs. Such marking is deleted by the rules, as example (6) shows.

catenate them as needed. Part of ambiguity can be resolved in this way. The rest may be left to manual choice, or a heuristic guesser may be constructed for doing the choice.

2.6. Removing temporary marking of verbs and restoring the normal writing system

As stated above, when a rule applies it deletes also the verb marker. The end mark + of the verb will be deleted if a locative suffix or an object marker of the first person singular is appended to the end of the verb. There are, however, words marked as verbs, although they belong to other word categories. Because no rule has applied to them they still retain the verb marking. This marking is finally removed with a program which also restores the ordinary writing system, with capital letters in the beginning of sentences and proper names, with punctuation marks and diacritics returned to their original places, etc.

3. Performance

3.1. Recall and precision

The performance of the program can be measured according to two criteria. The recall of the system is 100 % if all relevant cases of the text are identified and processed. The degree of precision, on the other hand, tells how correctly the system treats the found cases. It is seldom, if ever, full 100 % with unrestricted text. The degree of precision may be less than 100 % because of wrong processing results of those strings, which it should process, but also because the system processes strings, which it should leave un-processed. The danger of over-processing is imminent in a system such as this, where certain strings from a large mass of strings are selected for processing on the basis of criteria, which result in over-marking. Although (almost) all verbs are marked as verbs with one program before running the concatenating program, also some number of non-verbs will be marked. The over-marking as verbs is not, however, fatal from the viewpoint of the final result, as far as there are no such strings in front of the wrongly marked 'verbs', which would match with some rule in the program.

In the test text of 30,835 words (Test text I), there were a total of 6,584 string-sets to be concatenated. Out of these a total of 4,113 were unique string-sets. The recall was 98.6 %. Failing concatenations were mainly due to the fact that not all verbs were marked as verbs. The precision was tested with a text of 5,986 words (Test text II). Only 0.2 % of string-sets were wrongly concatenated. Therefore, the precision of the system was 99,8 %.

3.2. SPEED

The speed of the system was tested in Unix. All phases, text normalization, marking of verbs, concatenating prefixes and suffixes, and pruning the output, were run as one sequence. A test text had 30,835 words, and after running the concatenation program there were 24.399 words left. The speed was 770 words per second, and it took 40 seconds to process the whole text.

4. ALTERNATIVE WAYS OF IMPLEMENTATION

Above we have explained the method where verbs are marked first with the help of the verb dictionary. This method is not fully accurate, because only the verb root is used as a string to be matched in text. In addition to verbs, it allows also part of other words to be marked as verbs. In a test corpus of 5,986 words (Test text II) there were 1,909 words marked as finite verbs after running the program. Among those, the number of real finite verbs was 1307, and there were also 602 (32 %) other words marked as verb. In spite of the rather high rate of over-marking, tests showed that the likelihood of concatenation rules to be applied in front of wrongly marked 'verbs' was minimal. Therefore, we consider this method the next-to-best available and suitable to the task.

The best method would be the one where all verb-forms, also derived verbs, are available from the dictionary. This would reduce the rate of wrong marking to a minimum. It is not known to what extent the full listing of verbs would improve the overall performance. One could envisage that with the help of a morphological analyzer and sufficiently large corpus such a dictionary of verbs with derived forms could be constructed. This is not possible now, however, because neither the analyzer nor corpus is available for Kwanyama.

We assumed that a concatenation program could be constructed also without previous marking of verbs. In this approach the algorithm is very simple:

> Concatenate all such sequences of strings, which fulfill the criteria of verbal prefixes, and attach them to the following word, which is assumed to be a verb.

We implemented this algorithm as a program and made a test with a corpus text that contained 1307 such sequences of strings that had to be concatenated. The replacement rules were basically the same (187,000 rules) as in the system described above, modified to meet the absence of verb marking. The concatenation failed to be correct in 107 cases. This means that the accuracy of

the program was 92 %. Although the result is not bad it cannot be considered accurate enough as a basis for further processing.

This version of the program tended to be too permissive, because any string following the set of strings, assumed to be a set of verbal prefixes, was a verb candidate. And this is not the case, of course. It seems clear, therefore, that some kind of previous marking of the verbs improves the performance considerably.

5. MODIFYING THE OUTPUT

The output of the concatenation program can be modified according to need. If the aim is to produce normal text meant for reading by humans, then the output should be simple concatenation of morphemes, so that the result is a single string of characters. This is in fact a useful facility in a situation, where both a disjunctive and conjunctive writing system are in use simultaneously. A text can be made available in disjoining and conjoining writing systems to meet both tastes.

The output may be modified also in such a way that the morpheme boundaries will be marked, for example with an underscore _. If the output will be given as input for a morphological parsing program, this is even desirable, because in it the morphemes are already clearly identified. When using such a text version as input, the result of the morphological analysis will contain less ambiguity and consequently there will be less work in disambiguating the morphological analysis.

6. MAPPING TO BOTH DIRECTIONS

This paper concerns methods of mapping between two writing systems. In other words, the system should be able to convert one writing system into another, and vice versa. So far we have been discussing only one direction of conversion, i.e. from disjoining to conjoining writing. We have also implemented the conversion from conjoining to disjoining writing. The rule system of the first type of conversion was used as a basis for constructing the rule system for the opposite conversion. Rules were converted so that what was the X-part in the former, became the Y-part in the latter, and what was the Y-part in the former became the X-part in the latter. Context constraints as well as post-processing were modified accordingly. The result was a sequence of programs that perform the operation to the opposite direction, i.e. from the conjoining to the disjoining writing system. Therefore, the user of the system may choose the direction of conversion according to need. This facilitates real mapping between two writing systems.

7. OPEN QUESTIONS

In the implementation of the concatenation program of Kwanyama some questions have been left open. First, although Kwanyama is a tone language, the marking of tone has not been taken into account at all. The reason for this omission is that Kwanyama is normally written without tone marks. Although tone has a central significance in spoken language, it does not feature in writing. The absence of tone marking has also simplified the concatenation program, which even as such has a massive amount of rules. On the other hand, the absence of tone marking in text will increase morphological ambiguity, which then has to be resolved in a later stage of analysis.

REFERENCES

Beesley, Kenneth R. and Karttunen, Lauri, 2000.
Finite-State Non-Concatenative Morphotactics. In: *SIGPHON-2000. Proceedincs of the Fifth ACL Special Interest Group in Computational Phonology*. p. 1-12. Aug. 6, 2000. Luxembourg.

Hurskainen, Arvi, 1996.
Disambiguation of morphological analysis in Bantu languages. In: *COLING-96, Proceedings of the 16th International Conference on Computational Linguistics*, Copenhagen, August 5-9, 1996. Pp. 568-573.

Karlsson, Fred & Kimmo Koskenniemi, 1990.
Beta-ohjelma kielentutkijan apuvälineenä. Helsinki: Yliopistopaino.

Karttunen, Lauri, 2000.
Applications of Finite-State Transducers in Natural Language Processing. In: *Proceedings of CIAA-2000. Lecture Notes in Computer Science.* Springer Verlag.

Makgamatha, Phaka M., 1999.
Narration as Art in the Northern Sotho Narative: From Oral to Written. In: Rosalie Finlayson (ed.), *African Mosaic. Festschrift for J. A. Louw*. Pretoria: Unisa Press. Pp. 1-13.

Poulos, George, 1990.
A Linguistic Analysis of Venda. Pretoria: Via Afrika Ltd.

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

Zimmermann Wolfgang and Hasheela Paavo 1998.
Oshikwanyama Grammar. Windhoek: Gamsberg.

APPENDIX

Below is an extract from a Kwanyama text in various phases of processing.

Original text with a disjoining writing system:

"Hambili, nghi lele po," Shomupu ta nyamukula.
"Oike wani?" Mutumbo ta pula.
"Ondi na oilya yange ya lika po keengobe deni onghela; onda hala u uye u ke i tale, eshi ya lika." Shomupu ta nyamukula. "Ai, vakwetu eengobe detu nalye vali?" Mutumbo ta pula ongaao a kumwa, e litilifa oshimbwela.
Odeni naShalongo.
"Shalongo ye okwe i mona nale, ile aame andike handi ke i tala?"
"Shalongo mbela hato mu tumine okanona."
"Ongeenge nee nda ka tala oilya ile?"
"Aaye, opaife ngoo."
"Paife ohandi uya ndi mu lombwele shike ame oilya inandi i mona?"
"Otamu ke i tala amushe."

Free translation of the text:

"My crops were eaten up by your cows yesterday. I want you to come to see it and to observe the damages." Shomupu answered. "Oh my God, it is our cows and whose else?" Mutumbo asked as someone who is very much surprised. "Yours and Shalongo's."
"Has Shalongo seen it already, or is it only me who is going to see it?"
"Well, why don't you send a child to tell the news to Shalongo?"
"Would that be after having visited the site?"
"No, better do it now."
"What shall I tell him, having not even seen the crops myself?"
"The two of you will go and see it together."

Spaces marked where concatenation should occur:

"Hambili, nghi_lele_po," Shomupu ta_nyamukula.
"Oike wani?" Mutumbo ta_pula.
"Ondi_na oilya yange ya_lika_po keengobe deni onghela; onda_hala u_uye u_ke_i_tale, eshi_ya_lika." Shomupu ta_nyamukula. "Ai, vakwetu eengobe detu nalye vali?" Mutumbo ta_pula ongaao a_kumwa, e_litilifa oshimbwela.
Odeni naShalongo.
"Shalongo ye okwe_i_mona nale, ile aame andike handi_ke_i_tala?"
"Shalongo mbela hato_mu_tumine okanona."
"Ongeenge nee nda_ka_tala oilya ile?"
"Aaye, opaife ngoo."
"Paife ohandi_uya ndi_mu_lombwele shike ame oilya inandi_i_mona?"
"Otamu_ke_i_tala amushe."

Text after initial preprocessing: *The main feature of this program is that it converts the upper case letters to lower case and retains the info by placing an * in front of the letter.*

"*hambili, nghi lele po," *shomupu ta nyamukula.
"*oike wani?" *mutumbo ta pula.
"*ondi na oilya yange ya lika po keengobe deni onghela; onda hala u uye u ke i tale, eshi ya lika." *shomupu ta nyamukula. "*ai, vakwetu eengobe detu nalye vali?" *mutumbo ta pula ongaao a kumwa, e litilifa oshimbwela.
*odeni na*shalongo.
"*shalongo ye okwe i mona nale, ile aame andike handi ke i tala?"
"*shalongo mbela hato mu tumine okanona."
"*ongeenge nee nda ka tala oilya ile?"
"*aaye, opaife ngoo."
"*paife ohandi uya ndi mu lombwele shike ame oilya inandi i mona?"
"*otamu ke i tala amushe."

Text after marking the prospective verbs: *Note that the program marks also some non-verbs, as well as many bound morphemes of verbs.*

"*@hambili+, nghi @lele+ po," *@shomupu+ ta @nyamukula+.
"*oike @wani?" *mutumbo+ ta @pula+.
"*ondi @@na+ oilya @yange+ @ya+ @lika+ po keengobe deni @onghela; onda+ @hala+ @u+ @uye+ @u+ @ke+ @i+ @tale+, eshi @ya+ @lika+." *@shomupu+ ta @nyamukula+.
"*ai, @vakwetu+ eengobe detu nalye @vali?" *mutumbo+ ta @pula+ @ongaao+ a @kumwa+, e @litilifa+ oshimbwela.
*odeni @@na+*shalongo.
"*shalongo @ye+ okwe @i+ @mona+ nale, @ile+ aame @andike+ handi @ke+ @i+ @tala+?"
"*shalongo mbela hato mu @tumine+ okanona."
"*@ongeenge+ nee nda ka @tala+ oilya @ile+?"
"*aaye, opaife ngoo."
"*paife ohandi @uya+ ndi mu @lombwele+ @shike+ @ame+ oilya inandi @i+ @mona+?"
"*otamu @ke+ @i+ @tala+ @amushe+."

The result of the conjoining program: *The correct concatenations have taken place. An applied rule removes verb marks from the that string. The rest of marking is removed by a post-processing program.*

"*hambili, nghilelepo," *shomupu tanyamukula.
"*oike wani?" *mutumbo tapula.
"*ondina oilya yange yalikapo keengobe deni onghela; ondahala uuye ukeitale, eshiyalika." *shomupu tanyamukula. "*ai, vakwetu eengobe detu nalye vali?" *mutumbo tapula ongaao akumwa, elitilifa oshimbwela.
*odeni na*shalongo.
"*shalongo ye okweimona nale, ile aame andike handikeitala?"
"*shalongo mbela hatomutumine okanona."
"*ongeenge nee ndakatala oilya ile?"
"*aaye, opaife ngoo."
"*paife ohandiuya ndimulombwele shike ame oilya inandiimona?"
"*otamukeitala amushe."

Conjoined text returned to normal writing:

"Hambili, nghilelepo," Shomupu tanyamukula.
"Oike wani?" Mutumbo tapula.
"Ondina oilya yange yalikapo keengobe deni onghela; ondahala uuye ukeitale, eshiyalika." Shomupu tanyamukula. "Ai, vakwetu eengobe detu nalye vali?" Mutumbo tapula ongaao akumwa, elitilifa oshimbwela.
Odeni naShalongo.
"Shalongo ye okweimona nale, ile aame andike handikeitala?"
"Shalongo mbela hatomutumine okanona."
"Ongeenge nee ndakatala oilya ile?"
"Aaye, opaife ngoo."
"Paife ohandiuya ndimulombwele shike ame oilya inandiimona?"
"Otamukeitala amushe."

A version of conjoined text suitable as input for a morphological analyzer*: Note that the morpheme boundaries of verbs are marked and the text is pre-processed so that each word is uniquely identified as a word and separated from punctuation marks and diacritics. Each sentence is on a separate line and in the end of the sentence is a $ sign.*

" *hambili , nghi_lele_po , " *shomupu ta_nyamukula .$
" *oike wani ? " *mutumbo ta_pula .$
" *ondi_na oilya yange ya_lika_po keengobe deni onghela ; onda_hala u_uye u_ke_i_tale , eshi_ya_lika . "$
*shomupu ta_nyamukula .$
" *ai , vakwetu eengobe detu nalye vali ? " *mutumbo ta_pula ongaao a_kumwa , e_litilifa oshimbwela .$
*odeni na*shalongo .$
" *shalongo ye okwe_i_mona nale , ile aame andike handi_ke_i_tala ? "$
" *shalongo mbela hato_mu_tumine okanona . "$
" *ongeenge nee nda_ka_tala oilya ile ? "$
" *aaye, opaife ngoo . "$
" *paife ohandi_uya ndi_mu_lombwele shike ame oilya inandi_i_mona ? "$
" *otamu_ke_i_tala amushe . "$