Finite-state tokenization for a deep Wolof LFG grammar

Cheikh M. Bamba Dione

Abstract. This paper presents a finite-state transducer (FST) for tokenizing and normalizing natural texts that are input to a large-scale LFG grammar for Wolof. In the early stage of grammar development, a language-independent tokenizer was used to split the input stream into a unique sequence of tokens. This simple transducer took into account general character classes, without using any language-specific information. However, at a later stage of grammar development, uncovered and non-trivial tokenization issues arose, including issues related to multi-word expressions (MWEs), clitics and text normalization. As a consequence, the tokenizer was extended by integrating FST components. This extension was crucial for scaling the hand-written grammar to free text and for enhancing the performance of the parser.

1 Introduction

This paper presents a finite-state transducer (FST) (Beesley and Karttunen 2003) that acts as a tokenizer and a normalizer for Wolof[1] natural texts. Tokenization constitutes an important prior task for various language processing applications, e.g. part-of-speech tagging, parsing, information retrieval, information extraction, and machine translation. All these language processing systems need input texts with definite word boundaries. This task can be performed using various techniques, including e.g. rule-based techniques (Kaplan 2005), statistical techniques (Yang and Li 2005) and lexical techniques (Wu and Fung 1994). The tokenization approach proposed in this paper is based on the use of finite-state rules to break up a stream of Wolof texts into individual tokens. The tokenizer is designed using the Xerox finite-state tool *fst* (Beesley and Karttunen 2003).

The tokenization system is built within the broader context of an ongoing process of creating language resources and tools for Wolof. This process is part of the Parallel Grammar (ParGram) project (Butt et al. 2002) which is couched within the Lexical Functional Grammar (LFG) framework. In related work, a Wolof Morphological Analyzer (WoMA) (Dione 2012), a large-scale LFG grammar and a treebank for Wolof

1 Wolof belongs to the Senegambian branch of the Niger–Congo language family mainly spoken in Senegal, Gambia and Mauritania.

The very model of a modern linguist. Edited by Victoria Rosén and Koenraad De Smedt. BeLLS Vol. 8 (2017), DOI 10.15845/bells.v8i1.1340. Copyright © by the authors. Open Access publication under the terms of CC-BY-NC-4.0.

have been constructed (Dione 2014a). Other related work describes parse disambiguation techniques used for Wolof (Dione 2014b), including the integration of Constraint Grammar (CG) models (Karlsson 1990) into probabilistic context-free grammar approaches to disambiguation (Dione 2014c).

The tokenizer is used as part of a finite-state transducer cascade (Kaplan et al. 2004) that preprocesses the input sentences before they are parsed by the Wolof grammar.

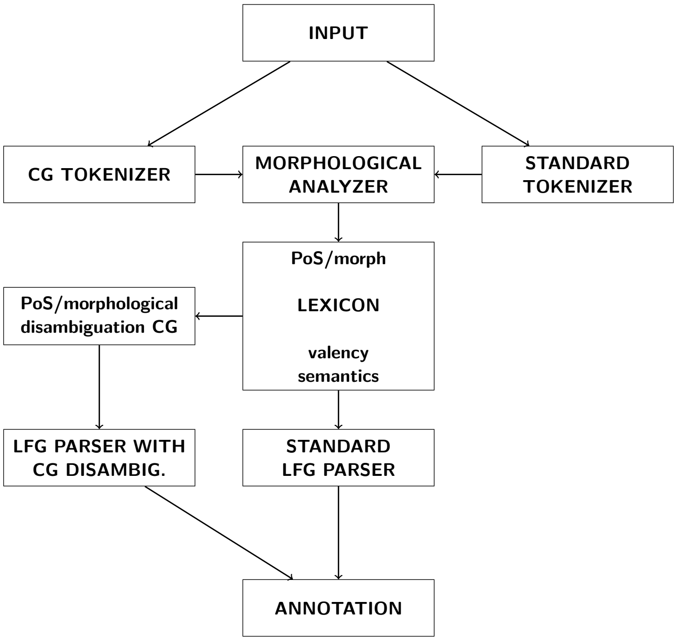

Figure 1: Anatomy of the Wolof parsing system

The parsing workflow is depicted in Figure 1. First, the input is tokenized and normalized either by a deterministic tokenizer when the LFG parsing is combined with CG disambiguation (Dione 2014c), or by a deterministic tokenizer when the syntactic analysis is performed without CG disambiguation. The former tokenizer is referred to as the CG tokenizer, while the latter is called the standard tokenizer. The only difference between the two tokenizers is their determinism. Next, morphological analysis is carried out. The output of the morphology is either disambiguated prior to syntactic analysis or directly fed into the standard LFG parser (i.e. without CG disambiguation). Finally, the morpho-syntactic annotation is produced.

In the early stage of grammar development, a language-independent tokenizer was used by the Wolof LFG system. However, as the development of the grammar progressed, the parsing system encountered various issues due to inappropriate tokenization. For instance, a significant number of sentences with MWEs could not be parsed or were not handled correctly. Likewise, the time needed to process sentences including words with clitics was growing due to parsing complexity. Hence, the need to integrate language-specific information into the tokenization process arose naturally with the aim to enhance coverage and quality as well as efficiency of the parser.

The remainder of this paper is organized as follows: Section 2 discusses the general concept of tokens in Wolof and non-trivial tokenization issues found in this language. Section 3 discusses the development of the tokenizer using finite-state technology. It presents the language-independent tokenizer used at the early stage of grammar development and describes the final transducer which integrates language-specific approaches to multi-word expressions and clitics into the tokenization process. Section 4 discusses issues related to text normalization. Section 5 reports on results of experimental evaluation of the tokenizer. Section 6 concludes the discussion.

2 Wolof tokens

Tokenization can be defined as the process of breaking a stream of texts up into words, symbols, or other meaningful elements called tokens. Accordingly, the process is assumed to typically occur at the word / token level. However, in some cases, it may be difficult to exactly define what is meant by a ‘word’ or ‘token’. This is particularly true for an agglutinative language like Wolof.

Similar to Turkish (Oflazer et al. 2004), many derivational phenomena in Wolof take place within a word form, but there are other complex derivations involving compounds and reduplications (Ka 1994). Wolof word forms consist of morphemes concatenated to a root morpheme or to other morphemes. The language is almost exclusively suffixing. In many contexts, the surface realizations of the morphemes are conditioned by various morphophonemic processes such as vowel harmony, vowel and consonant elisions, gemination, degemination, vowel coalescence, glide insertion, prenasalization, etc. (Ka 1994).

The morphotactics of Wolof word forms can be quite complex when multiple derivations and inflections are involved. For instance, the verb *gënoonatee* in (1) which consists of different derivational and inflectional morphemes can be represented as in (2).

(1)	*Li*	*gën-oon-ati-a*	*metti*
	What	SURPASS-PST-Iter-Cinf	be.painful

‘Particularly painful was … ’

(2) gënoonatee ⇔ gën+Verb+Modal+Past+Iter+A

This word starts out with a root *gën* ‘be better/worse’ followed by the past tense marker *-oon*, the iterative suffix *-ati* and the infinitival complementizer *a* which surfaces as a clitic. The ending *-atee* results from a vowel coalescence process: the final vowel of the suffix *-ati* is collapsed with the clitic *a*. Without derivation and inflection, the contraction of the verb in (2) with the the infinitival complementizer (Cinf) *a* can be tokenized at least in two different ways: either by handling the clitic as a normal affix integrated into the verbal stem (i.e. *gëna*) or by demarcating it from the stem (i.e. *gën a*). Accordingly, a precise definition of the *token* concept in Wolof is required, before an accurate tokenization system can be built for this language.

Throughout this research work, the definition of a token follows the one given for Arabic by Attia (2007, p. 66): “the minimal syntactic unit; it can be a word, a part of a word (or a clitic), a multiword expression, or a punctuation mark”. Accordingly, two categories of tokens can be distinguished for Wolof: *main tokens* vs. *sub-tokens. Main tokens* refer to stems with or without clitics, as well as numbers, which are typically separated by white spaces and punctuation marks as delimiters or word boundaries. Also, single character symbols like quotation marks and punctuation used in Wolof, such as the period, comma, question mark, semicolon, etc., are treated as individual tokens. In contrast, in some other cases (see Table 1), a stem may be suffixed with a clitic, both represented as *sub-tokens.*

2.1 Wolof clitics

As discussed in the previous section, a challenging tokenization issue is cliticization. Like Arabic (Attia 2007), Wolof morphotactics allows words to be suffixed with clitics. Clitics themselves can be concatenated one after the other. Furthermore, clitics undergo assimilation with word stems and with each other, making it difficult to recognize and handle them properly. Examples of full form words consisting of stems with clitics are shown in Table 1. Assimilation can be observed in some of these examples. The first row of the table is to be read as follows: the preposition *ak* ‘with’ may encliticize to the verbal stem *daje* ‘meet’, yielding the surface form *dajeek.*[2] The other surface forms involve different grammatical categories (determiners, conjunctions, pronouns, etc.) and occur in a similar manner.

2.2 The use of multiword expressions

Another relevant tokenization issue is the use of multiword expressions (MWEs). Formally, MWEs can be defined as “idiosyncratic interpretations that cross word boundaries (or spaces)” (Sag et al. 2002, p. 90). More specifically, MWEs are “two or more words that behave like a single word syntactically and semantically” (Attia 2007, p. 68). MWEs can be of different types, including idioms, prepositional verbs, verbs with particles, collocations, etc. Following Attia (2007), Oflazer et al. (2004), and Sag et al. (2002), Wolof MWEs are classified into four types: named entities, fixed expressions, semi-fixed expressions and syntactically flexible expressions.

1. Multi-word named entities refer to proper nouns for persons, organizations, places, etc., as illustrated in (3).

2 The long vowel *ee* in the surface form *dajeek* results from a vowel coalescence: the final vowel of the verbal stem *-e* coalesces with the stem-initial vowel of the preposition, i.e. *-a.*

Stem	*Clitic category*	*Example*	*Word form*	*Literal translation*
Verb	PREP	*daje* 'meat' + *ak* 'with'	*dajeek*	'met with'
	DET	*joxe* 'give' + ay 'some'	*joxeey*	'give some'
	Inf. COMP	*soog* 'start' + *a*	*sooga*	'start to V'
Determiner	PREP	*ba* 'the' + *ak* 'with'	*baak*	'the with'
	CONJ	*bi* 'the' + *ak* 'and'	*beek*	'the and'
Preposition	DET	*ci* 'in' + *ab* 'a'	*cib*	'in a'
	PREP	*ca* 'about' + *ak* 'with'	*caak*	'about with'
Noun	CONJ	*ndox* 'water' + *ak* 'and'	*ndoxak*	'water and'
Name	CONJ	*Ali* 'Ali' + *ak* 'and'	*Aleek*	'Ali and ...'
Adverb	PRON	*fu* 'where' + *nga* 'you'	*foo*	'where you ...'
Complementizer	PRON	*bu* 'if' + *nga* 'you'	*boo*	'if you ...'
Pronoun	CONJ	*moom* 'him' + *ak* 'and'	*mook*	'... and him'
Object pronoun	Inf. COMP	*ko* 'him/her' + *a*	*koo*	'him/her' + inf. V
Conjunction	AUX	*te* 'and' + *di* imperf.	*tey*	'and' + imperf.
	DET	*mbaa* 'or' + *ay* 'some'	*mbaay*	'or some'

Table 1: Examples of Wolof stems from different grammatical categories with clitic sub-tokens

(3) *Daara* *ju* *Kowe* *ji*
school REL high DET
'The University' (Lit. 'The school which is high')

2. Fixed expressions denote *collocations* where all components of the collocation are lexically, syntactically and morphologically rigid, as in (4). Other Wolof fixed MWEs include adverbials *saa su ne* 'every time', and quantifying expressions *bu baax* 'very well', etc. None of these MWEs can be reordered or separated by external elements.

(4) Mag ak rakk 'siblings'

3. Semi-fixed expressions refer to collocations where some components of the collocation are fixed and some can vary. The variation can be of morphological (e.g. inflectional or derivational) or lexical type (where one word can be replaced by another). Wolof inflectional subject markers like *maa ngi* in (5) and *noo ngi* in (6) are instances of semi-fixed expressions. They vary according to person and number of the subject as well as the aspect of the verb, as illustrated in (5)–(6), the optional attachment of the imperfective marker *-y* indicates the collocation

being inflected for aspect. In contrast, (6) exemplifies a case of lexical variation, where the word *maa* in (5) is replaced by *noo.*

(5)

Maa ngi(-y)	*dem*
1sg.PROG-IPF	go

‘I am leaving (here and now)’

(6)

Noo ngi	*dem*
1pl.PROG	go

‘We are leaving’

4. Syntactically flexible expressions are non-lexicalized expressions that can undergo reordering or allow external elements to intervene between the components of the collocation. For example, the MWE in (7) is disrupted in (8) by the agreement marker *na* and the clitic pronoun *ko.*

(7) fas yéene ‘decide’

(8) fas na ko yéene ‘He has decided it’

Wolof multiword tokens can be of different grammatical categories: inflectional elements (5); adverbial expressions like *bu baax* ‘very well’ (10) and *saa su ne* ‘every time’ (9); prepositions like *ci biir* ‘inside’; pronouns such as *yoo xam ne* ‘that/which’; nouns such as *mag ak rakk* ‘brothers’; quantifiers like *ku ne* ‘every one’; reduplicated words like *jékki jékki* ‘suddenly’, and other units. Some Wolof examples including multiword expressions are given in (9) and (10).

(9)

Saa su ne,	***noo ngi***	*nekk*	***ci biir***	*kër*	*gi.*
Every time	1pl.PROG	be	inside	house	the

‘Every time, we are inside the house.’

(10)

Ku ne	***jékki jékki***	*xàqtaay*	***bu baax.***
Every one	suddenly	laugh.out	very well.

‘Suddenly, every one laughs out loud.’

3 The Wolof tokenizer

This section describes the development of the Wolof tokenizer in finite-state technology. Section 3.1 presents the language-independent tokenizer used in the early stage of grammar development. Section 3.2 discusses the integration of language-specific information into the tokenizer.

3.1 Language-independent tokenization

As noted above, in the early stage of grammar development, tokenization was carried out by a language-independent FST. Typically, a language-independent tokenizer is a simple kind of deterministic tokenizer, i.e. an unambiguous finite-state transducer which relies on simple heuristics and takes into account some general character classes. For instance, it assumes that contiguous strings of alphabetic characters are part of one token; likewise with numbers. Tokens are separated by whitespace characters (designated by the category `WS`), including e.g. space (`SP`) or line break (`NL`), or by punctuation marks. Accordingly, the `Token` transducer (11) was defined as the union of sequences of alphabetic characters (`WORD`), numbers (`NUM`), punctuation and some other symbols (`SYMB`).

(11)
```
define Token [WORD | SYMB | NUM];
```

In addition, a language-independent tokenizer generally needs to normalize white space. This is because, in natural texts, the use of white space may be uneven and sometimes very inconsistent. For instance, one may find two or more white-space characters, including space, tab, newline characters, etc., instead of a single space. Similarly, spaces might inadvertently be added before or after punctuation marks. Therefore, normalizing tools for eliminating such inconsistencies are needed at a preliminary stage of tokenization. Normalizing the input before processing it allows for the separation of concerns, because the input is assumed to be consistent before operations are performed on it.

With the token definition (11), a language-independent tokenizer that inserts newlines to mark token boundaries (`TB`) can be compiled from the regular expression in (12). It represents the composition of three simple replace terms.

(12)
```
WS+ @-> SP
.o. Token @-> ... NL
.o. [WS]+ & $[NL] @-> TB
```

The first term in (12) reduces strings of whitespace characters into a single blank using longest-match replacement. The second term inserts a newline as a token boundary after the longest matches of letter sequences and other non-whitespace sequences. The third term denotes a rule that removes a set of spaces by replacing it with a token boundary. This set represents the intersection or conjunction of one or more spaces and the set of strings that contains at least one instance of newline somewhere.

When the white space normalizer is fed an input like (13), in which additional spaces are inserted and some spaces are misplaced, it corrects the errors and gives the output in (14).

(13)

Xale	*yi*	*lekk*	*jën*	*wi.*
child	the.pl	eat	fish	the.sg

‘So, the children eat the fish.’

(14) Xale yi lekk jën wi.

When surface strings like (14) are looked up using this model, the output string is the input string plus the multi-character symbols TB, inserted between tokens as in (15).

(15) Xale yi lekk jën wi. ⇒ Xale TB yi TB lekk TB jën TB wi TB . TB

However, this language-independent tokenizer encountered serious problems with respect to contracted words, hyphenated words and multiword expressions, as illustrated by examples (16) and (17). For instance, a MWE like *Saa su ne* in (9) was tokenized as shown in (16). However, in an appropriate tokenization model, it should be tokenized as in (17), neglecting the space between the individual tokens when assigning token boundaries. Conversely, an appropriate model would identify the vowel coalescence process involved in (18) and demarcate the collapsed vowels by inserting a space between them, as shown in (19).

(16) Saa su ne ⇒ Saa TB su TB ne TB

(17) Saa su ne ⇒ Saa su ne TB

(18) gënoonatee ⇒ gënoonatee TB

(19) gënoonatee ⇒ gënoonati TB a TB

Accordingly, more sophisticated tokenization techniques were needed to account for these non-trivial issues.

3.2 Integrating language-specific information into the tokenization process

As a result of the tokenization problems in using a language-independent model, I decided to integrate FST components for handling MWEs and clitics into the tokenization system. Similarly, preprocessing tools for text normalization were added. The final architecture of the Wolof tokenization system is depicted in Figure 2.

As Figure 2 shows, the internal preprocessing workflow consists of a cascade of transducers. Thus, during tokenization, the input is first normalized. Then string-based multi-word identification for named entities, fixed expressions and semi-fixed expressions is performed, allowing morphological variation for the latter MWE group. Next, the input is normalized again in order to remove space and to lowercase the first word in a sentence. After, clitics detection and demarcation are performed either deterministically by a clitic transducer, or indeterministically by a clitic guesser. Finally the tokenized and normalized output is produced in two variants according to whether it will be fed into the standard parsing system or the one based on CG disambiguation.

Note that the clitic transducer proposes analyses for contracted words using very basic morphological information carried out by an internal component of the tokenizer. For the standard parsing system, this step is non-deterministic and carried out by a

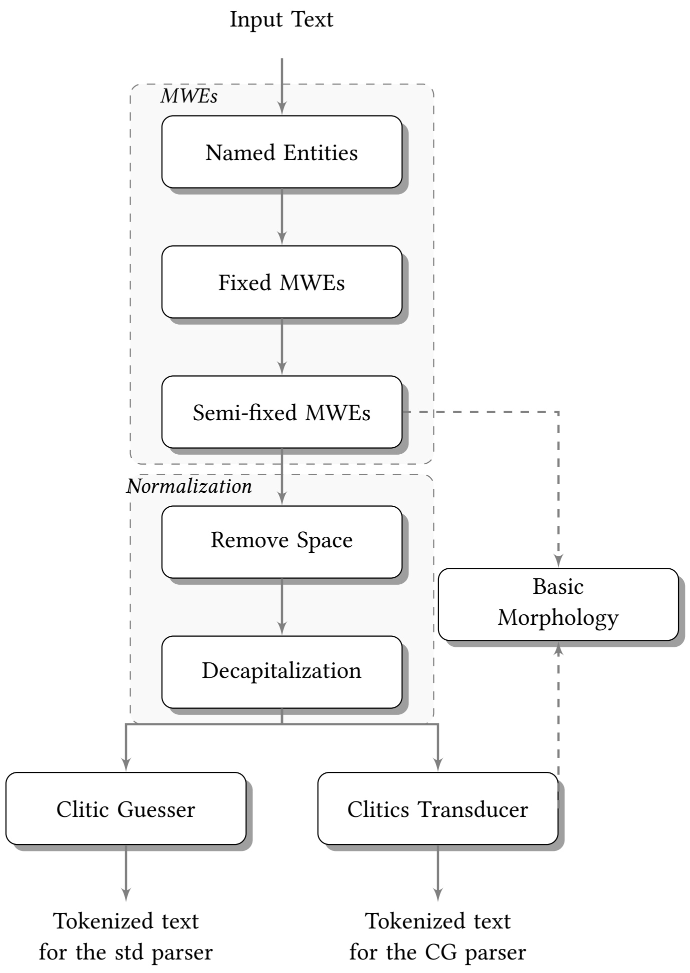

Figure 2: Architecture of the Wolof tokenization system

clitic guesser, since the goal is to allow all possible tokenizations as candidates for syntactic analysis. However, due to the nondeterministic nature of a guesser, there will be increased tokenization ambiguities. In contrast, when parsing is combined with CG disambiguation, this step is deterministic. Therefore, the clitics are not guessed, but rather handled by a transducer carefully designed to produce unambiguous outputs. The individual tokenizing and normalizing components are described in the next sections.

3.2.1 Multiword transducer

Parsing MWEs requires "a deep analysis that starts as early as the normalization and tokenization, and goes through morphological analysis and into syntactic rules" (Attia 2008, p. 70). Handling MWEs at the early preprocessing stage presents some advantages in that it avoids needless analysis of idiosyncratic structures. Additionally, it allows a reduction of ambiguity and parsing time. A precise treatment of MWEs is, however, challenging in that it requires adequate strategies (Beesley and Karttunen 2003). For instance, with a naive model, multi-word tokens may be recognized even when they are just part of a longer alphabetic string, leading to inappropriate tokenization. Therefore, the model used in the present work has been designed such that it will handle them as accurately as possible.

Using regular expressions, a two-sided transducer was created for handling the three following types of MWEs: named entities, fixed expressions and semi-fixed expressions.[3] This transducer was then embedded in the tokenizer as described by Beesley and Karttunen (2003).

In the first stage, finite-state networks including named entities (proper names, locations, organizations, etc.), fixed expressions and semi-fixed expressions were created. The networks represent lists of words separated by space. The lists were created according to the grammatical categories of the MWEs.[4] For instance, for each part of speech such as nouns, adverbs, prepositions, etc., a corresponding finite-state network was built. Unlike named entities and fixed MWEs, the handling of semi-fixed expressions needs some very basic morphological information (see Figure 2) due to morphological variations. Such information was explicitly encoded in the tokenizer as an FST that generates the possible inflectional forms for these MWEs before the final compilation.

In the second stage, a main MWE was built by concatenating all the different finite-state networks created so far. In the third stage, special brackets (e.g. `M1` and `M2`) were inserted around maximally long multi-word expressions, as shown in (20). Subsequently, the main MWE transducer `MWE1` was integrated into the tokenizer, as illustrated in (21). With this rule, the initial *token* concept given in (11) was redefined and augmented with information about MWEs.

(20) `define MWE1 [M1 MWE M2];`

(21) `define Token [WORD | SYMB | NUM | MWE1];`

Finally, the rule in (22) was used to identify multi-word tokens on the basis of information provided by the previous terms. This rule represents a composition of rules. The

3 Note that the MWE transducer was not responsible for the treatment of syntactically flexible expressions, which are handled by the Wolof grammar. As noted above, such expressions are not included in the tokenizer, since their structures allow external (e.g. pronominal) elements to intervene. For more details on how such verbs with particles are handled see e.g. Dione (2014b).

4 The lists included in the MWE transducer are of a moderate size and mostly include multi-word tokens found in the corpus.

first part in (22) considers MWE as those tokens delimited (bounded) by the elements defined in (24). These consist of single character symbols `SINGLE` as defined in (23), or whitespace or the boundary symbol.[5] The second part in (22) then inserts a newline as a token boundary after longest matches of letter sequences and other non-whitespace sequences.

(22)
```
define TOK1 [ MWE @-> M1 ... M2
                || Bound _ Bound
                .o. Token @-> ... NL
                .o. [M1|M2] -> 0
            ];
```

(23)
```
define SINGLE [%"|%«|%»|%.|%,|%;|%:|%!|%?|%(|%)|%[|%]|%{|%}|%–];
```

(24)
```
define Bound [SINGLE|WS|.#.];
```

The third part in (22) removes the special brackets around multiword expressions when they are no longer used. However, as Beesley and Karttunen (2003) pointed out, the brackets also need to be deleted from the sigma alphabet[6] in order to make the input side of the resulting transducer match the universal language. This is accomplished by (25), which explicitly 'absorbs' the auxiliary symbols into the unknown alphabet by calling the Xerox *substitute* commands for each of the brackets used.

(25)
```
substitute symbol ? for "<<"
substitute symbol ? for ">>"
```

With this extended tokenization model, a sentence like (26) will have the phrase structure shown in Figure 3.

(26)

mu	*dem*	*ci*	*Daara ju Kowe ji*
3sg	go	PREP	university

'He went to the university'

3.2.2 Clitic transducer

In the extended model, tokenization is also expected to handle clitic boundaries. For this purpose, a clitic transducer is embedded into the tokenizer, in a similar manner to the MWE transducer. The clitic transducer has the functions of detecting and demarcating contracted morphemes and handling them as separate words. Yet information on what may constitute a clitic is still needed. One possibility is to use a clitic guesser, making assumptions about clitic occurrence and position in Wolof word formation.

5 The boundary symbol `.#.` denotes the beginning of a string in the left context and the end of the string in the right context of a replace expression.

6 The sigma alphabet is the set of individual symbols known to the network.

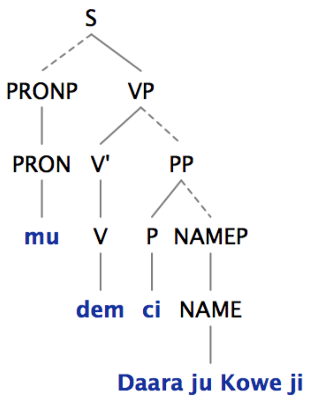

Figure 3: Phrase structure of (26)

For instance, joined words like *baak* (see Table 1) are decomposed into *ba TB ak TB*. The long vowel *aa* is produced by a vowel coalescence rule which collapses the final vowel of the determiner stem *b-* and the initial short vowel of the conjunction *ak* 'and'. This kind of contraction is very common for Wolof determiners, demonstratives, pronouns, etc., which take the noun class index (e.g. *b, g, j, k, l, m, s, w*, etc.).[7] Thus, abstracting from the noun class index, one can formulate a non-deterministic rule like (27) which optionally inserts a token boundary between the collapsed vowels if the morpheme *aak* is found at the end of a word.

(27) `{aak} (->) [a] TB [a k] || _ [.#.|TB]`

Such an operation may be particularly useful when applied to constituents like (28), with the coordinated nouns tokenized as given in Figure (4).

(28)

Petu	*ma-ak*	*yedd*	*ya-ak*	*xuloo*	*ba-ak*	*lépp*
meeting	det-conj	lecture	det-conj	dispute	det-conj	all

'The secret meetings, the lectures, the disputes and all this'

Lit.: 'The secret meetings and the lectures and the disputes and all this'

This solution, however, is based on a guessing mechanism which naturally increases ambiguity due to non-determinism. Furthermore, since the CG disambiguator needs unambiguous input, such rules cannot be included in the CG tokenizer. Also, because

7 Wolof is a noun class language with noun class agreement (McLaughlin 2010). The language has approximately 13 noun classes identified by their index: 8 singular (*b, g, j, k, l, m, s, w*), 2 plural (*y, ñ*), 2 locative (*f, c*), 1 manner (*n*).

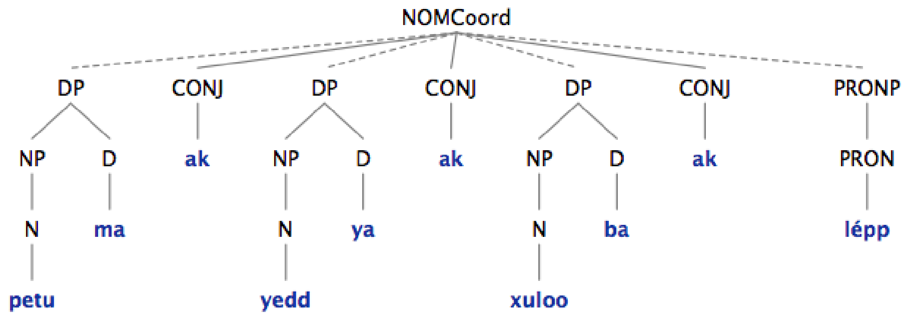

Figure 4: Space insertion using the clitic transducer

cliticization can occur after several morphophonological processes, a proper treatment of clitics needs, for example, morphological information related to verbal inflection and all possible assimilations.

For this reason, a clitic transducer is integrated into the tokenizer. As illustrated in Figure 2, the transducer is associated with an internal FST component that provides basic morphophonological information, e.g. about the forms involved in word contractions as well as the contexts where these occur. For instance, the simplified finite-state network in (29) contains the word stems of modal verbs and other verb operators in Wolof (Church 1981). These stems are not guessed, but taken as a list of actual words extracted from the morphological lexicon.

(29) `define ModalStems [ {soog} | {mën} | {mas} | {bëgg} | {gën} | ... ];`

Such information can be encoded in a finite-state transducer like (30), which represents the composition of the finite-state network `Inflection` with the transducer for vowel coalescence `vowCoal` (both not displayed here).

(30) `define InflModal [ModalStems Inflection] .o. vowCoal;`

Using this information, a clitic transducer like (31) can then detect clitics and insert space between the sub-tokens.

(31)
```
define splitA a -> TB [a] || [.#.|TB]
                       [ModalStems | InflectedModal] _ [.#.|TB]
                       .o.   ...;
```

In (31), a token boundary is unambiguously inserted between the stem of an optionally inflected modal verb and the complementizer clitic *a* found at the end of the verbal string, as illustrated in (32) and (33).

(32) gëna ⇒ gën TB a TB

(33) gënoonatee ⇒ gënoonati TB a TB

4 Text normalization

Besides the integration of language-specific information into the tokenizing transducer, the Wolof FST also includes text and word normalization components. Beesley and Karttunen (2003, p. 440) define word normalization as "the general process of mapping accidental spelling variations to yield normalized forms for analysis". Most commonly needed normalizations in natural-language processing are those that handle initial capitalization (upper-casing) and whole-word capitalization. In Wolof, decapitalization at the beginning of a sentence has proven to be an important issue, as discussed in the next section.

4.1 The initial-capitalization normalizer

Besides the normalization of whitespace shown in section 3.1, decapitalization is a relevant normalization issue for grammar engineering. As Forst and Kaplan (2006, p. 370) noted, "the most important normalization when parsing free text is decapitalization at the beginning of a sentence, **but** also after opening quotes, brackets, colons and hyphens". Accordingly, the tokenizer includes a component for lower-casing accidental spellings, which reflects the orthographical convention of capitalizing the first letter of the first word in a sentence. This kind of normalization is carried out at two different levels: the first word of the sentence is marked by the tokenizer and then lowercased by the morphological analyzer. Note, however, that this normalization form does not apply to proper names, which are considered as a special word category. Proper names are entered in the lexicon with initial capital letter and this spelling will be preserved.

4.1.1 Normalization dependent on the tokenizer

At the tokenization level, the transducer in (34) is used to mark the first word of the sentence. Such a word begins with a capital letter *upper* defined as a network that consists of all the capital letters in Wolof. This word is expected to be found at the beginning of a string (cf. the boundary symbol `.#.`) or after colons followed by one or more token boundaries and optionally symbols occurring before the first word of a sentence (*beforeFirstWord*), e.g. quotes, dashes, parentheses, etc. Accordingly, the transducer inserts the caret symbol before that word.[8] In this particular case, this symbol is used as a hint for the morphological analyzer to identify the word marked as the first one of a sentence.

8 The notation with an initial caret (or hat or circumflex) symbol ˆ follows the convention of encoding feature-like multi-character symbols in Xerox finite-state systems (Beesley and Karttunen 2003).

(34)
```
define mark1stWord [ "\^{}" ... <- upper
                       [.#. | ":" TB* ]
                       beforeFirstWord* TB* _
                     ];
```

4.1.2 Normalization dependent on the morphology

In order to lowercase the first word of a sentence, the Wolof morphological analyzer described in Dione (2012) has been extended with the function in (35). Decapitalization applies then for those words marked by the tokenizer with caret, indicating that they were found at the beginning of the sentence. This function is designed such that it only handles words marked as such, ignoring other words found somewhere else. It also removes the caret symbol after decapitalization is performed by the *downcase* term.

(35)
```
define MarkDowncased(X) [ [\"^"]* .o. X ]
                        | [ X .o. [ 0:"^" ( downcase ) ?* ]];
```

The *downcase* term denotes the inverse of the term *upcase*, as defined in (36).

(36)
```
define upcase \> [ \> A:a|B:b|C:c|D:d|E:e|F:f|G:g|... ];
```

The term in (36) contains a number of pairs and represents the mapping of all uppercase strings to the corresponding lowercase strings. It consists of ordered pairs *<A,a>* of symbols *A:a*, where *A* is the upper-side symbol and *a* is the lower-side symbol. The upper language is the infinite language of uppercase strings, the lower language contains all the lowercase strings, and the term itself is a mapping that preserves the word. Thus, if `upcase` contains *<A, a>*, the inverse relation `upcase.i` contains *<a, A>*.

In a final stage (37), the function `MarkDowncased` is applied to the main Wolof transducer `WolMorph` which represents all those words handled by the Wolof morphology.

(37)
```
read regex MarkDowncased (WolMorph);
```

Using the normalizing transducer, the first word of the sentence in (14) will be lowercased, as illustrated by the tree on the left side of Figure 5. In contrast, the tree on the right side, shows how the spelling of proper names like *Móodu* in the sentence *Móodu dem* 'Móodu left' is preserved.

5 Evaluation

The tokenizer can be evaluated in the context of the standard Wolof LFG parser (Dione 2014a) which makes use of the tokenizer to annotate free text. The performance of the parser was measured on unseen natural text data consisting of 2354 sentences randomly selected from Cissé (1994), Garros (1997) and Ba (2007). The parser was evaluated in terms of coverage and parsing quality. Coverage indicates whether the parser yields

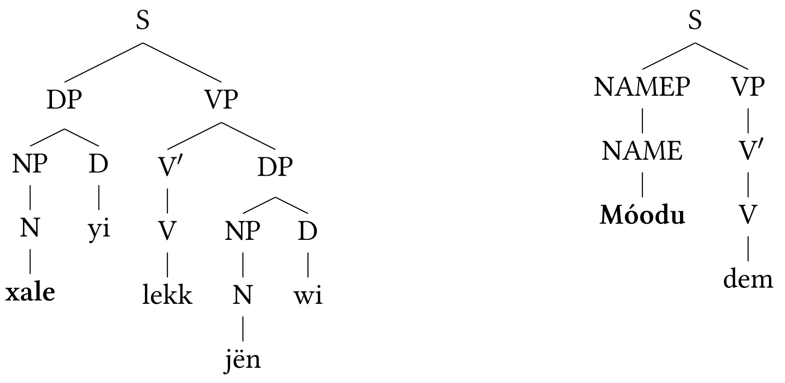

Figure 5: Normalizing the first word of sentence (14)

complete parses or not. Accordingly, the evaluation results given in Dione (2014a) reported that the Wolof parser could find a complete parse for 1712 of the test sentences (i.e. 72.72% coverage for complete parses).

For a direct evaluation of the tokenizer, 350 out of the 642 sentences that could not be parsed were randomly selected to determine whether parsing failure was due to erroneous tokenization. For 330 of them, this was not the case. Among the twenty that failed due to tokenization errors, ten are due to inappropriate treatment of clitics, mostly caused by vowel assimilation or complex derivation (e.g. reduplication); five contain multi-word units, including names; two sentences contain all uppercase strings like *BUKKEEK «PERIGAM» BU XONQ* 'Hyena and its red «wig»' which also involve issues related to clitics and quotes; the rest consists of tokenization errors due to the use of symbols and foreign language material as well as a mixture of the issues discussed so far.

The problem of the multiword units is difficult to address without good lists of person, place, organization and product names. In many cases, the tokenization problems are caused by different issues. For instance, the clitic transducer erroneously inserted a space between *ja* and *ag* in the string (38), considering both as determiners (i.e. *ja* 'the' and *ag* 'a'), which might be correct in some contexts. In (38), however, *Jaag* is a last name which can be treated either as an individual token or as a part of a multiword expression, but not segmented into two strings. Adding this named entity to the list of identified MWEs would help to avoid this kind of problems.

(38) Sàmba Jaag

Likewise, issues related to clitics are complex and need to be addressed in future work on the tokenizer.

6 Conclusion

This paper has presented a finite-state transducer for tokenizing normal Wolof text. It has shown that the design of such a preprocessing tool involves non-trivial issues related to the treatment of clitics, multiword expressions and text normalization. Accordingly, sophisticated techniques covering these issues have been integrated into the tokenization model. Also, the paper has explained how the different preprocessing components interact with each other and how tokenization and normalization are closely connected to and sometimes dependent on morphological analysis.

However, as this paper acknowledges, there are open tokenization issues that need to be addressed in future work. This includes, for instance, better handling of clitics by integrating sophisticated techniques to control ambiguity caused by the guessing mechanisms. Similarly, robust corpus-based approaches to multiword extraction need to be combined with tokenization. Finally, future work on text normalization include issues related to capitalization and haplology (Forst and Kaplan 2006).

7 Acknowledgements

This research has received support from the EC under FP7, Marie Curie Actions SP3-People-ITN 238405 (CLARA). I thank Paul Meurer for helping me integrate the normalizer into the morphological analyzer.

References

Attia, Mohammed A. (2007). "Arabic Tokenization System". In: *Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages: Common Issues and Resources*. Association for Computational Linguistics, pp. 65–72.

– (2008). "Handling Arabic Morphological and Syntactic Ambiguity within the LFG Framework with a View to Machine Translation". PhD thesis. University of Manchester.

Ba, Mariyaama (2007). *Bataaxal bu gudde nii*. Nouvelles Editions Africaines du Sénégal (NEAS), p. 182.

Beesley, Kenneth R. and Lauri Karttunen (2003). *Finite State Morphology*. Stanford, CA: Center for the Study of Language and Information.

Butt, Miriam, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer (2002). "The Parallel Grammar Project". In: *Proceedings of the COLING2002, Workshop on Grammar Engineering and Evaluation*. Vol. 15. Association for Computational Linguistics, pp. 1–7.

Church, Eric D. (1981). *Le système verbal du wolof*. Faculté des Lettres et Sciences Humaines (FLSH), Université de Dakar.

Cissé, Mamadou (1994). *Contes wolof modernes*. Paris: L'Harmattan.

Dione, Cheikh M. Bamba (2012). “A Morphological Analyzer For Wolof Using Finite-State Techniques”. In: *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC’12).* Istanbul, Turkey: ELRA.

– (2014a). “Formal and Computational Aspects of Wolof Morphosyntax in Lexical Functional Grammar”. PhD thesis. University of Bergen, Norway.

– (2014b). “LFG parse disambiguation for Wolof”. In: *Journal of Language Modelling* 2.1, pp. 105–165.

– (2014c). “Pruning the Search Space of the Wolof LFG Grammar Using a Probabilistic and a Constraint Grammar Parser”. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC’14).* Reykjavik, Iceland: European Language Resources Association (ELRA), pp. 2863–2870.

Forst, Martin and Ronald M. Kaplan (2006). “The importance of precise tokenizing for deep grammars”. In: *Proceedings of the Language Resources and Evaluation Conference (LREC’06).* Genoa, Italy.

Garros, Nataali Dominik, ed. (1997). *Bukkeek “perigam” bu xonq: teeñ yi.* Dr. Moren ak mbootayu “xale dimbale xale”. Translated from French to Wolof by Momar Touré. Dakar: SIL / Paris: EDICEF.

Ka, Omar (1994). *Wolof Phonology and Morphology.* Lanham, Maryland: University Press of America.

Kaplan, Ronald M. (2005). “A method for tokenizing text”. In: *Inquiries into Words, Constraints and Contexts. Festschrift for Kimmo Koskenniemi on his 60th Birthday.* Stanford, CA: CSLI Publications.

Kaplan, Ronald M., John T. Maxwell III, Tracy Holloway King, and Richard Crouch (2004). “Integrating Finite-State Technology with Deep LFG Grammars”. In: *Proceedings of the ESSLLI’04 Workshop on Combining Shallow and Deep Processing for NLP.*

Karlsson, Fred (1990). “Constraint Grammar as a Framework for Parsing Running Text”. In: *Proceedings of the 13th Conference on Computational Linguistics.* Vol. 3. ACL. Helsinki, pp. 168–173.

McLaughlin, Fiona (2010). “Noun classification in Wolof: When affixes are not renewed”. In: *Studies in African Linguistics* 26.1, pp. 1–28.

Oflazer, Kemal, Özlem Çetinoğlu, and Bilge Say (2004). “Integrating morphology with multi-word expression processing in Turkish”. In: *Proceedings of the Workshop on Multiword Expressions: Integrating Processing.* Association for Computational Linguistics, pp. 64–71.

Sag, Ivan A., Timothy Baldwin, Francis Bond, Ann Copestake, and Dan Flickinger (2002). “Multiword Expressions: A Pain in the Neck for NLP”. In: *Proceedings of the 3rd International Conference on Computational Linguistics and Intelligent Text Processing (CICLing 2002).* Vol. 2276. Lecture Notes in Computer Science. Springer, pp. 1–15.

Wu, Dekai and Pascale Fung (1994). “Improving Chinese tokenization with linguistic filters on statistical lexical acquisition”. In: *Proceedings of the Fourth Conference on Applied Natural Language Processing*. Association for Computational Linguistics, pp. 180–181.

Yang, Christopher C. and Kar Wing Li (2005). “A heuristic method based on a statistical approach for Chinese text segmentation”. In: *Journal of the American Society for Information Science and Technology* 56.13, pp. 1438–1447.