Arabic Tokenization System

Mohammed A. Attia
School of Informatics / The University of Manchester, PO Box 88, Sackville Street, Manchester M60 1QD, UK
mohammed.attia@postgrad.manchester.ac.uk

Abstract

Tokenization is a necessary and non-trivial step in natural language processing. In the case of Arabic, where a single word can comprise up to four independent tokens, morphological knowledge needs to be incorporated into the tokenizer. In this paper we describe a rule-based tokenizer that handles tokenization as a full-rounded process with a preprocessing stage (white space normalizer), and a post-processing stage (token filter). We also show how it handles multiword expressions, and how ambiguity is resolved.

1 Introduction

Tokenization is a non-trivial problem as it is "closely related to the morphological analysis" (Chanod and Tapanainen 1994). This is even more the case with languages with rich and complex morphology such as Arabic. The function of a tokenizer is to split a running text into tokens, so that they can be fed into a morphological transducer or POS tagger for further processing. The tokenizer is responsible for defining word boundaries, demarcating clitics, multiword expressions, abbreviations and numbers.

Clitics are syntactic units that do not have free forms but are instead attached to other words. Deciding whether a morpheme is an affix or a clitic can be confusing. However, we can generally say that affixes carry morpho-syntactic features (such as tense, person, gender or number), while clitics serve syntactic functions (such as negation, definition, conjunction or preposition) that would otherwise be served by an independent lexical item. Therefore tokenization is a crucial step for a syntactic parser that needs to build a tree from syntactic units. An example of clitics in English is the genitive suffix "'s" in *the student's book.*

Arabic clitics, however, are not as easily recognizable. Clitics use the same alphabet as that of words with no demarcating mark as the English apostrophe, and they can be concatenated one after the other. Without sufficient morphological knowledge, it is impossible to detect and mark clitics. In this paper we will show different levels of implementation of the Arabic tokenizer, according to the levels of linguistic depth involved.

Arabic Tokenization has been described in various researches and implemented in many solutions as it is a required preliminary stage for further processing. These solutions include morphological analysis (Beesley 2001; Buckwalter 2002), diacritization (Nelken and Shieber 2005), Information Retrieval (Larkey and Connell 2002), and POS Tagging (Diab et al 2004; Habash and Rambow 2005). None of these projects, however, show how multiword expressions are treated, or how ambiguity is filtered out.

In our research, tokenization is handled in a rule-based system as an independent process. We show how the tokenizer interacts with other transducers, and how multiword expressions are identified and delimited. We also show how incorrect tokenizations are filtered out, and how undesired tokenizations are marked. All tools in this research are developed in Finite State Technology (Beesley and Karttunen 2003). These tools have been developed to serve an Arabic Lexical Functional Grammar parser using XLE (Xerox Linguistics Environment) platform as part of the ParGram Project (Butt et al 2002).

Proceedings of the 5th Workshop on Important Unresolved Matters, pages 65–72,
Prague, Czech Republic, June 2007.

2 Arabic Tokens

A *token* is the minimal syntactic unit; it can be a word, a part of a word (or a clitic), a multiword expression, or a punctuation mark. A tokenizer needs to know a list of all word boundaries, such as white spaces and punctuation marks, and also information about the token boundaries inside words when a word is composed of a stem and clitics. Throughout this research full form words, i.e. stems with or without clitics, as well as numbers will be termed *main tokens*. All main tokens are delimited either by a white space or a punctuation mark. Full form words can then be divided into *sub-tokens*, where clitics and stems are separated.

2.1 Main Tokens

A tokenizer relies mainly on white spaces and punctuation marks as delimiters of word boundaries (or main tokens). Additional punctuation marks are used in Arabic such as the comma '،', question mark '؟' and semicolon '؛'. Numbers are also considered as main tokens. A few Arab countries use the Arabic numerals as in English, while most Arab countries use the Hindi numerals such as '٢' (2) and '٣' (3). Therefore a list of all punctuation marks and number characters must be fed to the system to allow it to demarcate main tokens in the text.

2.2 Sub-Tokens

Arabic morphotactics allow words to be prefixed or suffixed with clitics (Attia 2006b). Clitics themselves can be concatenated one after the other. Furthermore, clitics undergo assimilation with word stems and with each other, which makes them even harder to handle in any superficial way. A verb can comprise up four sub-tokens (a conjunction, a complementizer, a verb stem and an object pronoun) as illustrated by Figure 1.

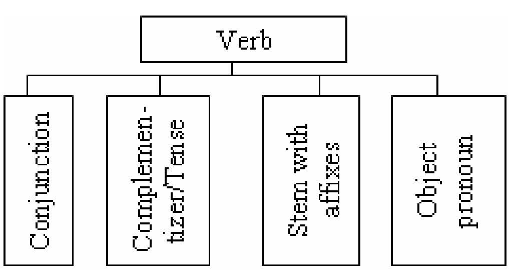

Figure 1: Possible sub-tokens in Arabic verbs

Similarly a noun can comprise up to four sub-tokens. Although Figure 2 shows five sub-tokens but we must note that the definite article and the genitive pronoun are mutually exclusive.

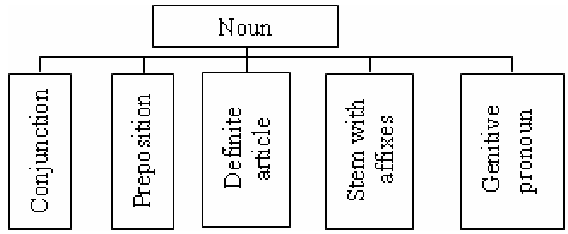

Figure 2: Possible sub-tokens in Arabic nouns

Moreover there are various rules that govern the combination of words with affixes and clitics. These rules are called grammar-lexis specifications (Abbès et al 2004; Dichy 2001; Dichy and Fargaly 2003). An example of these specifications is a rule that states that adjectives and proper nouns do not combine with possessive pronouns.

3 Development in Finite State Technology

Finite state technology has successfully been used in developing morphologies and text processing tools for many natural languages, including Semitic languages. We will explain briefly how finite state technology works, then we will proceed into showing how different tokenization models are implemented.

(1)
```
LEXICON Proclitic
al@U.Def.On@        Root;
                    Root;
LEXICON Root
kitab               Enclitic;
LEXICON Suffix
an                  Enclitic;
                    Enclitic;
LEXICON Enclitic
hi@U.Def.Off@       #;
```

In a standard finite state system, lexical entries along with all possible affixes and clitics are encoded in the lexc language which is a right recursive phrase structure grammar (Beesley and Karttunen 2003). A lexc file contains a number of lexicons connected through what is known as "continuation classes" which determine the path of concatenation. In example (1) above the lexicon *Proclitic* has a lexical form *al*, which is linked to a continuation class named *Root*. This means that the forms in *Root* will be appended to the right of *al*. The lexicon *Proclitic* also has an empty string, which means that *Proclitic* itself is optional and that the path can proceed without it. The bulk of all lexical entries are presumably listed under *Root* in the example.

Sometimes an affix or a clitic requires or forbids the existence of another affix or clitic. This is what is termed "long distance dependencies" (Beesley and Karttunen 2003). So Flag Diacritics are introduced to serve as filters on possible concatenations to a stem. As we want to prevent *Proclitic* and *Enclitic* from co-occurring, for the definite article and the possessive pronoun are mutually excusive, we add a Flag Diacritic to each of them with the same feature name "U.Def", but with different value "On/Off", as shown in (1) above. At the end we have a transducer with a binary relation between two sets of strings: the lower language that contains the surface forms, and the upper language that contains the analysis, as shown in (2) for the noun كتابان kitaban (two books).

(2) Lower Language: كتابان
Upper Language: كتاب+noun+dual+sg

4 Tokenization Solutions

There are different levels at which an Arabic tokenizer can be developed, depending on the depth of the linguistic analysis involved. During our work with the Arabic grammar we developed three different solutions, or three models, for Arabic tokenization. These models vary greatly in their robustness, compliance with the concept of modularity, and the ability to avoid unnecessary ambiguities.

The tokenizer relies on white spaces and punctuation marks to demarcate main tokens. In demarcating sub-tokens, however, the tokenizer needs more morphological information. This information is provided either deterministically by a morphological transducer, or indeterministically by a token guesser. Eventually both main tokens and sub-tokens are marked by the same token boundary, which is the sign '@' throughout this paper. The classification into main and sub-tokens is a conceptual idea that helps in assigning the task of identification to different components.

Identifying main tokens is considered a straightforward process that looks for white spaces and punctuation marks and divides the text accordingly. No further details of main tokens are given beyond this point. The three models described below are different ways to identify and divide sub-tokens, or clitics and stems within a full form word.

4.1 Model 1: Tokenization Combined with Morphological Analysis

In this implementation the tokenizer and the morphological analyzer are one and the same. A single transducer provides both morphological analysis and tokenization. Examples of the tokenizer/analyser output are shown in (3). The '+' sign precedes morphological features, while the '@' sign indicates token boundaries.

(3) وليشكر (waliyashkur: and to thank)
و+conj@ل+comp@+verb+pres+sgشكر@

This sort of implementation is the most linguistically motivated. This is also the most common form of implementation for Arabic tokenization (Habash and Rambow 2005). However, it violates the design concept of modularity which requires systems to have separate modules for undertaking separate tasks. For a syntactic parser that requires the existence of a tokenizer besides a morphological analyzer, this implementation is not workable, and either Model 2 or Model 3 is used instead.

4.2 Model 2: Tokenization Guesser

In this model tokenization is separated from morphological analysis. The tokenizer only detects and demarcates clitic boundaries. Yet information on what may constitute a clitic is still needed. This is why two additional components are required: a clitics guesser to be integrated with the tokenizer, and a clitics transducer to be integrated with the morphological transducer.

Clitics Guesser. We developed a guesser for Arabic words with all possible clitics and all possible assimilations. Please refer to (Beesley and Karttunen 2003) on how to create a basic guesser. The core idea of a guesser is to assume that a stem is composed of any arbitrary sequence of Arabic alphabets, and this stem can be prefixed or/and suffixed with a limited set of tokens. This guesser is then used by the tokenizer to mark clitic boundaries. Due to the nondeterministic nature of a guesser, there will be increased tokenization ambiguities.

(4) وللرجل (and to the man)
و@ل@ال@رجل@
و@ل@الرجل@
و@للرجل@
وللرجل@

Clitics Transducer. We must note that Arabic clitics do not occur individually in natural texts. They are always attached to words. Therefore a specialized small-scale morphological transducer is needed to handle these newly separated forms. We developed a lexc transducer for clitics only, treating them as separate words. The purpose of this transducer is to provide analysis for morphemes that do not occur independently.

(5) و+conj
ل+prep
ال+art+def

This small-scale specialized transducer is then unioned (or integrated) with the main morphological transducer. Before making the union it is necessary to remove all paths that contain any clitics in the main morphological transducer to eliminate redundancies.

In our opinion this is the best model, the advantages are robustness as it is able to deal with any words whether they are known to the morphological transducer or not, and abiding by the concept of modularity as it separates the process of tokenization from morphological analysis.

There are disadvantages, however, for this model, and among them is that the morphological analyzer and the syntactic parser have to deal with increased tokenization ambiguities. The tokenizer is highly non-deterministic as it depends on a guesser which, by definition, is non-deterministic. For a simple sentence of three words, we are faced with eight different tokenization solutions. Nonetheless, this can be handled as explained in subsection 5.1 on discarding spurious ambiguities.

4.3 Model 3: Tokenization Dependent on the Morphological Analyser

In the above solution, the tokenizer defines the possible Arabic stem as any arbitrary sequence of Arabic letters. In this solution, however, the word stem is not guessed, but taken as a list of actual words. A possible word in the tokenizer in this model is any word found in the morphological transducer. The morphological transducer here is the same as the one described in subsection 4.1 but with one difference, that is the output does not include any morphological features, but only token boundaries between clitics and stems.

This is a relatively deterministic tokenizer that handles clitics properly. The main downfall is that only words found in the morphological transducer are tokenized. It is not robust, yet it may be more convenient during grammar debugging, as it provides much fewer analyses than model 2. Here spurious ambiguities are successfully avoided.

(6) وللرجل (and to the man)
و@ل@ال@رجل@

One advantage of this implementation is that the tool becomes more deterministic and more manageable in debugging. Its lack of robustness, however, makes it mostly inapplicable as no single morphological transducer can claim to comprise all the words in a language. In our XLE grammar, this model is only 0.05% faster than Model 2. This is not statistically significant advantage compared to its limitations.

4.4 Tokenizing Multiword Expressions

Multiword Expressions (MWEs) are two or more words that behave like a single word syntactically and semantically. They are defined, more formally, as “idiosyncratic interpretations that cross word boundaries” (Sag et al 2001). MWEs cover expressions that are traditionally classified as idioms (e.g. *down the drain*), prepositional verbs (e.g. *rely on*), verbs with particles (e.g. *give up*), compound nouns (e.g. *traffic lights*) and collocations (e.g. *do a favour*).

With regard to syntactic and morphological flexibility, MWEs are classified into three types: fixed, semi-fixed and syntactically flexible expressions (Baldwin 2004; Oflazer et al 2004; Sag et al 2001).

a. Fixed Expressions. These expressions are lexically, syntactically and morphologically rigid. An expression of this type is considered as a word with spaces (a single word that happens to contain spaces), such as الشرق الأوسط al-sharq al-awsat (the Middle East) and بيت لحم bait lahem (Bethlehem).

b. Semi-Fixed Expressions. These expressions can undergo variations, but still the components of the expression are adjacent. The variations are of two types, morphological variations where lexical items can express person, number, tense, gender, etc., such as the examples in (7), and lexical variations, where one word can be replaced by another as in (8).

(7.a) فترة انتقالية
fatratah intiqaliyyah
translational.sg.fem period.sg.fem

(7.b) فترتان انتقاليتان
fatratan intiqaliyyatan
translational.dual.fem period.dual.fem

(8) على ظهر/وجه الأرض/البسيطة
ala zahr/wajh al-ard/al-basitah
on the face/surface of the land/earth
(on the face of the earth)

c. Syntactically Flexible Expressions. These are the expressions that can either undergo reordering, such as passivization (e.g. *the cat was let out of the bag*), or allow external elements to intervene between the components such as (9.b), where the adjacency of the MWE is disrupted.

(9.a) دراجة نارية
darrajah nariyyah
bike fiery (motorbike)

(9.b) دراجة الولد النارية
darrajat al-walad al-nariyyah
the-bike the-boy the-fiery (the boy's motorbike)

Fixed and semi-fixed expressions are identified and marked by the tokenizer, while syntactically flexible expressions can only be handled by a syntactic parser (Attia 2006a).

The tokenizer is responsible for treating MWEs in a special way. They should be marked as single tokens with the inner space(s) preserved. For this purpose, as well as for the purpose of morphological analysis, a specialized transducer is developed for MWEs that lists all variations of MWEs and provides analyses for them (Attia 2006a).

One way to allow the tokenizer to handle MWEs is to embed the MWEs in the Tokenizer (Beesley and Karttunen 2003). Yet a better approach, described by (Karttunen et al 1996), is to develop one or several multiword transducers or "staplers" that are composed on the tokenizer. We will explain here how this is implemented in our solution, where the list of MWEs is extracted from the MWE transducer and composed on the tokenizer. Let's look at the composition regular expression:

```
(10) 1   singleTokens.i
     2   .o. ?* 0:"[[[" (MweTokens.l) 0:"]]]" ?*
     3   .o. "@" -> " " || "[[[" [Alphabet* | "@"*] _
     4   .o. "[[[" -> [] .o. "]]]" -> []].i;
```

Single words separated by the '@' sign are defined in the variable *singleTokens* and the MWE transducer is defined in *MweTokens*. In the MWE transducer all spaces in the lower language are replaced by "@" so that the lower language can be matched against *singleTokens*. In line 1 the *singleTokens* is inverted (the upper language is shifted down) by the operator ".i" so that composition goes on the side that contains the relevant strings. From the MWE transducer we take only the lower language (or the surface form) by the operator ".l" in line 2. Single words are searched and if they contain any MWEs, the expressions will (optionally) be enclosed by three brackets on either side. Line 3 replaces all "@" signs with spaces in side MWEs only. The two compositions in line 4 remove the intermediary brackets.

Let's now show this with a working example. For the phrase in (11), the tokenizer first gives the output in (12). Then after the MWEs are composed with the tokenizer, we obtain the result in (13) with the MWE identified as a single token.

(11) ولوزير خارجيتها
wa-liwazir kharijiyatiha
and-to-foreign minister-its
(and to its foreign minister)

(12) و@ل@وزير@خارجية@ها@
(approx. and@to@foreign@minister@its@)

(13) و@ل@وزير خارجية@ها
(approx. and@to@foreign minister@its@)

4.5 Normalizing White Spaces

White space normalization is a preliminary stage to tokenization where redundant and misplaced white spaces are corrected, to enable the tokenizer to work on a clean and predictable text.

In real-life data spaces may not be as regularly and consistently used as expected. There may be two or more spaces, or even tabs, instead of a single space. Spaces might even be added before or after punctuation marks in the wrong manner. Therefore, there is a need for a tool that eliminates inconsistency in using white spaces, so that when the text is fed into a tokenizer or morphological analyzer, words and expressions can be correctly identified and analyzed. Table 1 shows where spaces are not expected before or after some punctuation marks.

No Space Before	No Space After
)	(
}	{
]	[
”	“

Table 1. Space distribution with some punctuation marks

We have developed a white space normalizer whose function is to go through real-life texts and correct mistakes related to the placement of white spaces. When it is fed an input such as the one in (14.a) in which additional spaces are inserted and some spaces are misplaced, it corrects the errors and gives the output in (14.b):

(14.a) نشر (الديمقراطية)سيقود إلى   السلام .

(14.b) نشر (الديمقراطية) سيقود إلى السلام.

5 Resolving Ambiguity

There are different types of ambiguity. There are spurious ambiguities created by the guesser. There are also ambiguities which do not exist in the text before tokenization but are only created during the tokenization process. Finally there are real ambiguities, where a form can be read as a single word or two sub-tokens, or where an MWE has a compositional reading. These three types are treated by the following three subsections respectively.

5.1 Discarding Spurious Ambiguities

Tokenization Model 2 discussed above in subsection 4.2 is chosen as the optimal implementation due to its efficiency and robustness, yet it is highly nondeterministic and produces a large number of spurious ambiguities. Therefore, a morphological transducer is needed to filter out the tokenization paths that contain incorrect sub-tokens. Recall example (4) which contained the output of the nondeterministic tokenizer. In (15) below, after the output is fed into a morphological transducer, only one solution is accepted and the rest are discarded, as underlined words do not constitute valid stems.

(15) وللرجل (and to the man)
و@ل@ال@رجل@ - Passed.
و@ل@الرجل@ - Discarded.
و@للرجل@ - Discarded.
وللرجل@ - Discarded.

5.2 Handling Tokenization Ambiguities

Among the function of a tokenizer is separate clitics from stems. Some clitics, however, when separated, become ambiguous with other clitics and also with other free forms. For example the word كتابهم kitabahum has only one morphological reading (meaning *their book*), but after tokenization كتاب@هم there are three different readings, as the second token هم can either be a clitic genitive pronoun (the intended reading) or a free pronoun *they* (*a book, they*) or a noun meaning *worry* (forming the compound *book of worry*).

This problem is solved by inserting a mark that precedes enclitics and follows proclitics to distinguish them from each other as well as from free forms (Ron M. Kaplan and Martin Forst, personal communications, Oxford, UK, 20 September 2006). The mark we choose is the Arabic elongation short line called *cashida* which is originally used for graphical decorative purposes and looks natural with most clitics. To illustrate the usage, a two-word string (16.a) will be rendered without cashidas as in (16.b), and a single-word string that contains clitics (17.a) will be rendered with a distinctive cashida before the enclitic pronoun as in (17.b). This indicates that the pronoun is attached to the preceding word and not standing alone.

(16.a) كتاب هم
kitab hum/hamm (book of worry/a book, they)

(16.b) كتاب@هم

(17.a) كتابهم kitabuhum (their book)

(17.b) كتاب@ـهم

This implementation will also resolve a similar ambiguity, that is ambiguity arising between proclitics and enclitics. The proclitic preposition ك ka (as) always occurs initially. There is a homographic enclitic object pronoun ك ka (you) that always occurs in the final position. This can create ambiguity in instances such as the made-up sentence in (18.a). The sentence has the initial tokenization of (18.b) without a cashida, and therefore the central token becomes ambiguous as it can now be attached either to the preceding or following word leading either to the readings in (18.a) or (18.c). The cashida placement, however, resolves this ambiguity as in (18.d). The cashida is added after the token, indicating that it is attached to the following word and now only the reading in (18.a) is possible.

(18.a) أعطيت كالأمير
a'taitu ka-lamir (I gave like a prince)
(18.b) أعطيت@ك@الأمير
(18.c) أعطيتك الأمير
a'taitu-ka alamir (I gave you the prince)
(18.d) أعطيت@كـ@الأمير

5.3 Handling Real Ambiguities

Some tokenization readings are legal, yet highly infrequent and undesired in real-life data. These undesired readings create onerous ambiguities, as they are confused with more common and more acceptable forms. For example the Arabic preposition بعد ba'd (after) has the possible remote reading of being split into two tokens بـ@عد, which is made of two elements: بـ bi (with) and عد 'add (counting). Similarly بين baina (between) has the possible remote reading بـ@ين, which is made of two tokens as well: بـ bi (with) and ين yin (Yen).

The same problem occurs with MWEs. The optimal handling of MWEs is to treat them as single tokens and leave internal spaces intact. Yet a non-deterministic tokenizer allows MWEs to be analysed compositionally as individual words. So the MWE حظر التجول hazr al-tajawwul (curfew) has two analyses, as in (19), although the compositional reading in (19.b) is undesired.

(19.a) حظر التجول@ hazr al-tajawwul (curfew)
(19.b) حظر@التجول
hazr (forbidding) al-tajawwul (walking)

The solution to this problem is to mark the undesired readings. This is implemented by developing a filter, or a finite state transducer that contains all possible undesired tokenization possibilities and attaches the "+undesired" tag to each one of them.

Undesired tokens, such as بـ@ين and بـ@عد, explained above, can be included in a custom list in the token filter. As for MWEs, the token filter imports a list from the MWE transducer and replaces the spaces with the token delimiter '@' to denote the undesired tokenization solutions. The token filter then matches the lists against the output of the tokenizer. If the output contains a matching string a mark is added, giving the output in (20). Notice how (20.b) is marked with the "+undesired" tag.

(20.a) حظر التجول@ [hazr al-tajawwul (curfew)]
(20.b) حظر@التجول+undesired

This transducer or filter is composed on top of the core tokenizer. The overall design of the tokenizer and its interaction with other finite state components is shown in Figure 3. WE must note that the tokenizer, in its interaction with the morphological transducer and the MWE transducer, does not seek morpho-syntactic information, but it queries for lists and possible combinations.

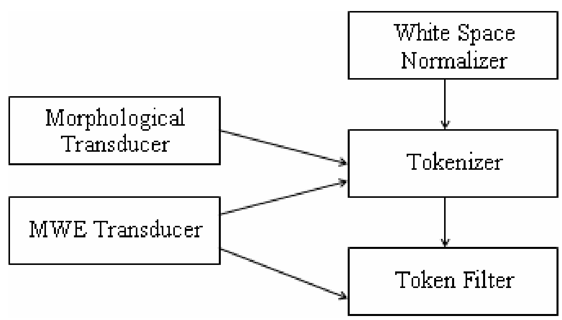

Figure 3: Design of the Arabic Tokenizer

6 Conclusion

Tokenization is a process that is closely connected to and dependent on morphological analysis. In our research we show how different models of tokenization are implemented at different levels of linguistic depth. We also explain how the tokenizer interacts with other components[1], and how it resolves complexity and filters ambiguity. By applying token filters we gain control over the tokenization output.

References

Abbès R, Dichy J, Hassoun M (2004): The Architecture of a Standard Arabic lexical database: some figures, ratios and categories from the DIINAR.1 source program, The Workshop on Computational Approaches to Arabic Script-based Languages, COLING 2004. Geneva, Switzerland.

Attia M (2006a): Accommodating Multiword Expressions in an Arabic LFG Grammar. In Salakoski T, Ginter F, Pyysalo S, Pahikkala T (eds), Advances in Natural Language Processing, 5th International Conference on NLP, FinTAL 2006, Turku, Finland, Vol 4139. Turku, Finland: Springer-Verlag Berlin Heidelberg, pp 87-98.

Attia M (2006b): An Ambiguity-Controlled Morphological Analyzer for Modern Standard Arabic Modelling Finite State Networks, The Challenge of Arabic for NLP/MT Conference. The British Computer Society, London, UK.

Baldwin T (2004): Multiword Expressions, an Advanced Course, The Australasian Language Technology Summer School (ALTSS 2004). Sydney, Australia.

Beesley KR (2001): Finite-State Morphological Analysis and Generation of Arabic at Xerox Research: Status and Plans in 2001, Proceedings of the Arabic Language Processing: Status and Prospect--39th Annual Meeting of the Association for Computational Linguistics. Toulouse, France.

Beesley KR, Karttunen L (2003): Finite State Morphology. Stanford, Calif.: CSLI.

Buckwalter T (2002): Buckwalter Arabic Morphological Analyzer Version 1.0., Linguistic Data Consortium. Catalog number LDC2002L49, and ISBN 1-58563-257-0.

Butt M, Dyvik H, King TH, Masuichi H, Rohrer C (2002): The Parallel Grammar Project, COLING-2002 Workshop on Grammar Engineering and Evaluation. Taipei, Taiwan.

Chanod J-P, Tapanainen P (1994): A Non-Deterministic Tokenizer for Finite-State Parsing, ECAI'96. Budapest, Hungary.

Diab M, Hacioglu K, Jurafsky D (2004): Automatic Tagging of Arabic Text: From Raw Text to Base Phrase Chunks, Proceedings of NAACL-HLT 2004. Boston.

Dichy J (2001): On lemmatization in Arabic. A formal definition of the Arabic entries of multilingual lexical databases, ACL 39th Annual Meeting. Workshop on Arabic Language Processing; Status and Prospect. Toulouse, pp 23-30.

Dichy J, Fargaly A (2003): Roots & Patterns vs. Stems plus Grammar-Lexis Specifications: on what basis should a multilingual lexical database centred on Arabic be built?, Proceedings of the MT-Summit IX workshop on Machine Translation for Semitic Languages. New-Orleans.

Habash N, Rambow O (2005): Arabic Tokenization, Part-of-Speech Tagging and Morphological Disambiguation in One Fell Swoop, Proceedings of ACL 2005. Michigan.

Karttunen L, Chanod J-P, Grefenstette G, Schiller A (1996): Regular expressions for language engineering. Natural Language Engineering 2:305-328.

Larkey LS, Connell ME (2002): Arabic Information Retrieval at UMass. In Voorhees EM, Harman DK (eds), The Tenth Text Retrieval Conference, TREC 2001. Maryland: NIST Special Publication, pp 562-570.

Nelken R, Shieber SM (2005): Arabic Diacritization Using Weighted Finite-State Transducers, Proceedings of the 2005 ACL Workshop on Computational Approaches to Semitic Languages. Michigan.

Oflazer K, Uglu ÖÇ, Say B (2004): Integrating Morphology with Multi-word Expression Processing in Turkish, Second ACL Workshop on Multiword Expressions: Integrating Processing. Spain, pp 64-71.

Sag IA, Baldwin T, Bond F, Copestake A, Flickinger D (2001): Multi-word Expressions: A Pain in the Neck for NLP, LinGO Working Papers. Stanford University, CA.

[1] The tokenizer along with a number of other Arabic finite state tools are made available for evaluation on the website: www.attiapace.com