Generating an Arabic full-form lexicon for bidirectional morphology lookup

Abdelhadi Soudi

CLC and CS department
Ecole Nationale
de L'Industrie Minérale
Rabat, Morocco
asoudi@enim.ac.ma

Andreas Eisele

Computational Linguistics Department
Saarland University
P.O.Box 151150
D-66041 Saarbrücken, Germany
eisele@coli.uni-sb.de

Abstract

We describe the generation of an Arabic full-form lexicon and its conversion into a two-level Finite State Transducer (FST) for morphology analysis and generation. The implementation of morphological lookup is based on a representation of the relevant data in the form of a FST, for which generic implementations exist that facilitate the integration into larger software systems for natural language processing. We show the feasibility of our encoding and the analysis of both vowelled and unvowelled Arabic words.

Introduction

Arabic morphology represents a special type of morphological systems. It is generally considered to be a non-concatenative system that depends on manipulating root letters in a non-concatenative manner, using different operations such as gemination and infixation. An Arabic verb can be conjugated according to one of the traditionally recognized patterns. There are 15 triliteral forms, of which at least 9 are in common. They represent very subtle differences. Within each conjugation/pattern, an entire paradigm is found: two voices (perfect and imperfect), two voices (active and passive) and five moods (indicative, subjunctive, jussive, imperative and energetic). In addition to prefixation and suffixation, inflectional and derivational processes may cause stems to undergo infixational modification in the presence of different syntactic features as well as certain consonants.

With respect nouns, there are three number categories for Arabic nouns (including adjectives): singular (mufrad), dual (mu#anGaY), and plural (jam`). The plural is further divided into sound (Al-jam`u Al-sGaAlim-u), the use of which is practically confined to (at least in the masculine) to participles and nouns indicating profession and habit, and broken (al-jam`u l-mukasGaru) types. Broken plurals are then divided into plurals of paucity (jam`u Al-qilGa0i), denoting three to ten items, and plurals of multiplicity (jam`u Al-ka#ra0i), denoting more than ten items. There are four forms of the plural of paucity and at least 23 forms of the plural of multiplicity. Several singulars have more than one plural form. There are also underived nouns with plural or collective sense (usually indicating a group of animals or plants). These are treated as singular, but may form a 'singulative' (çismu Al-waHda0i) indicating an individual of the group, by attachment of the suffix 0. [1]

Most of the computational attempts to model Arabic, such as Kay's (1987) non-concatenative Finite State model, Kiraz's (1994a, 1994b) Multi-tape two-level model and Beesley's Finite State model (1990), reflect the separation of levels advanced by McCarthy (1981, 1990). In McCarthy's proposal, Arabic stems are formed by a derivational combination of a root and a vowel melody. The two are arranged according to canonical patterns.

A detailed description and evaluation of the above-cited computational models is provided in Soudi (2002).

In this paper, we describe the generation of an Arabic full-form lexicon for bi-directional morphology lookup on the basis of an Arabic morphology generator (Soudi & al., 2002, 2001; Cavalli-Sforza & al., 2000 ; Leavitt, 1994).

1. The Construction of the Lexicon

We have taken advantage of an Arabic morphology generator based on Morphe (leavitt, 1994), a tool for compiling morphological rules into a word generation program.[2] In Morphe, the generation of Arabic inflected forms takes place in two steps, the first to generate the required stem, the second to generate the appropriate inflectional prefixes and suffixes. The generation process is based on the Lexeme-based Morphology framework (Aronoff, 1994; Beard, 1995) which focuses on stems in contrast to the root+pattern+vocalism approaches followed by other researchers.[3]

The lexicon is populated with full-forms by using a lisp function that takes as its argument a word or a list of words and provides all their inflectional paradigms/declensions. The output is stored in a lexicon file. In building the lexicon, we have avoided duplication of morphosyntactic information by separating the static features (part of speech and any other idiosyncratic information) from the dynamic features, such as voice, tense, mood, number, person, and gender, in the case of verbs, and case, number and definiteness in the case of nouns, which realize the different surface forms. That is, all the surface forms generated according to a combination of the dynamic features share or inherit the same static features. This advantage, coupled with the brevity of the lexicon, saves a significant amount of space. The lexicon is growing at a fast pace.

[1] The symbols we use in the transliteration are provided at the end of the paper.

[2] Leavitt also says that MORPHE can be used for analysis, but it has never been tested for it.

[3] A detailed description of the generation of Arabic verbal and noun morphology is provided in (Cavalli-Sforza & Soudi, 2003; Soudi & al., 2002, 2001; Cavalli-Sforza & al., 2000)

Interestingly, the full-form lexicon encodes morphosyntactic information that is missing in some existing systems, such as xerox' and the Arabic Tree Bank's (ATB) corpus which is analyzed by Buckwalter's morphological analyzer. Examples of such information include:

- linking a broken plural noun with its corresponding singular. By way of example, our lexicon encodes the following information for the noun *rijaAlun* "men" : (baseform rajul) (number plural) (case nominative) (definiteness -) (gender masc). The ATB corpus, however, does not provide information on the number feature; nor does it link the broken plural form to the singular noun. In the ATB corpus, number and gender are only determined by the suffix, especially in the case of sound plural nouns;
- linking the passive and the active participle to the corresponding verb;and
- linking the verbal noun to the corresponding noun.

2. FSTs and Morphological Lookup

The implementation of morphological lookup is based on a representation of the relevant data in the form of a finite-state transducer (FST) in the style of Kaplan and Kay (1994) and Karttunen (1994)[4]. FSTs describe regular relations between strings in a declarative way. Their mathematical properties are simple and well understood, and they can be used for generation of surface forms as easily as for morphological analysis. They generalize finite-state acceptors to multiple tapes, where transitions between states simultaneously affect several levels of string representation. Applications for morphological lookup use one of these tapes for the surface string, and another for the underlying linguistic analysis. Since FSTs factor out independent sources of variation, exponential numbers of forms with all morphological readings can be represented very compactly. The clear separation between compilation of the FST and transduction of strings nicely accommodates different requirements for off-line and on-line processing of the linguistic specification and makes it easy to embed morphological processors for different natural languages into larger systems.

2.1. Requirements of off-line and on-line processing

At first sight it may look somewhat roundabout to expand a compact and linguistically adequate morphological description into a full-form lexicon and then transform the data again into a different compact representation. It is important to note that these representations serve completely different purposes.

The source form of the morphological resource needs to be easy to read and write for linguists who build and maintain the specification, in order to facilitate frequent activities such as inspecting and updating the lexicon. But when morphology is applied, it is typically a submodule within packages for syntactic analysis or generation, which are themselves part of larger systems. Hence, the representations and algorithms used at run-time need to facilitate integration with other software. An implementation should impose the smallest possible number of constraints onto its clients and should offer APIs that can be used from multiple programming languages. Algorithms that can treat different natural languages in a uniform way are very helpful when building larger multi-lingual applications.

The existence of generic implementations of FST construction and lookup, developed for morphological processing of other languages, made it particularly attractive to try the same approach on Arabic data.

2.2. Off-line processing

The construction of FSTs for Arabic morphology involves several steps (see also Beesley 1990, 1996, 1998). Pairs consisting of a surface form and a lexical analysis are transformed into a sequence of pair labels at the granularity of individual characters and attribute-value pairs. For example, the pair:

```
AlomudarGisu
mudarGis+sp=Un+cat=n+def=def+number=sg+case=nom
```

is transformed to a sequence

```
A:ε l:ε o:ε m u d a r G i s u:ε ε:+sp=Un ε:+cat=n
ε:+def=def ε:+number=sg ε:+case=nom
```

The complete enumeration of all sequences of pairs derived from the lexicon is given to a generic and highly scalable algorithm that generates a minimal finite-state representation of the data. The resulting FST is written out in a format that requires only somewhat more than 4 Bytes per transition. This algorithm has been used before to transform MMorph databases (Petitpierre & Russel, 1994; Bouillon E.A., 1998) for 5 European languages into FSTs, where it has shown impressive capabilities for speed and compression. It has encoded more than 830000 German full forms (with over 6.5 million different analyses) in only 1.2 MB. For Arabic, we can optionally strip all vowels from the surface part of the pairs and use the algorithm to build an FST that relates unvowelled words with their (vowelled) analyses.

2.3. On-line processing

The compiled FST representations can then be interpreted by existing implementations of exact and error-tolerant finite-state lookup implemented at the German Research Center for Artificial Intelligence (DFKI). In order to accommodate different requirements for functionality, speed and compatibility, there are two implementations of the lookup routine that can work with the same binary representations. The Java implementation focuses on simplicity and reliability and can be included in multi-threaded applications. The C implementation[5] is significantly faster and allows for the search of a set of most similar strings under a given distance metric. The error-tolerant lookup follows roughly the approach of Oflazer (1996) and furthermore allows specifying the likelihood of deviations (such as typing/spelling/OCR errors or phonetic similarity) in an application-specific error metric. Programming interfaces to several host

[4] See also (Beesley and Karttunen 2003) for an extensive introduction

[5] Sometimes called SILO for "similarity-based lookup".

languages (Python, Perl, Prolog, LISP, Java via JNI[6]) and to XML/RPC exist, as sketched in Figure 1. Using this generic framework, robust morphological analysis and generation can be embedded quite flexibly into various platforms for NLP, including deep syntactic analysis and generation in the framework of HPSG, based on LKB (Copestake, 2002) or PET (Callmeier, 2000). However, the simplest way to use the toolkit is a command-line lookup program that enumerates all possible analyses for a given full form or vice versa.

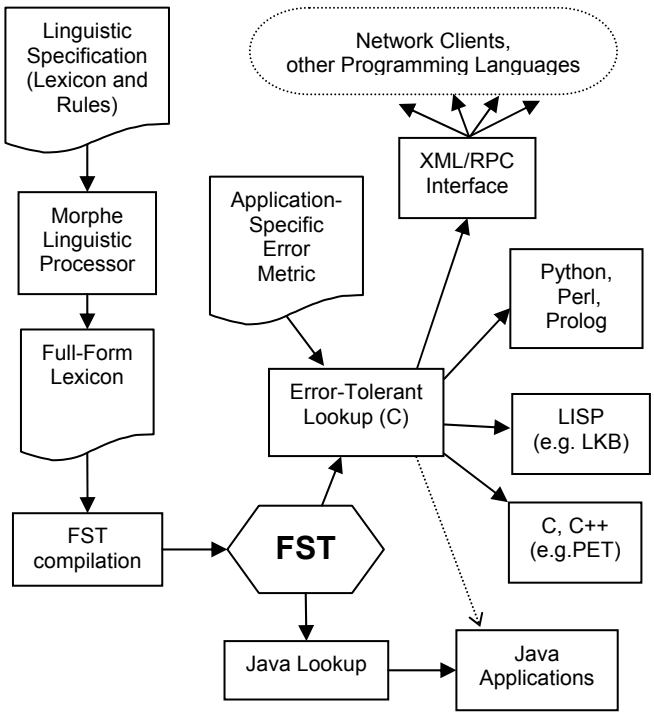

Figure 1: Software Architecture

Using the lookup tool, the example given above comes out as follows:

```
> java –jar fst.jar morph-ar.fst AlomudarGisu
---- 1 result(s) for AlomudarGisu:
mudarGis+sp=Un+cat=n+def=def+number=sg+case=nom
> java –jar fst.jar morph-ar.fst –g\
mudarGis+sp=Un+cat=n+def=def+number=sg+case=nom
---- 1 result(s) for mudarGis+sp=Un+cat=n+def=…:
AlomudarGisu
```

The same lookup routine, used with the "unvowelled" FST, will enumerate analyses of all ways how vowels can be inserted, as is shown below:

```
> java –jar fst.jar morph-ar-u.fst Almdrs
---- 3 result(s) for Almdrs:
mudarGis+sp=Un+cat=n+def=def+number=sg+case=nom
mudarGis+sp=Un+cat=n+def=def+number=sg+case=gen
mudarGis+sp=Un+cat=n+def=def+number=sg+case=acc
```

For nouns that follow the broken plural pattern, the analysis displays also the singular stem:

```
> java –jar fst.jar morph-ar-u.fst "Al>jwbO"
---- 3 result(s) for Al>jwbO:
jawaAb+bpstem=>ajowibaO+cat=n+def=def+number=pl+case=nom
jawaAb+bpstem=>ajowibaO+cat=n+def=def+number=pl+case=gen
jawaAb+bpstem=>ajowibaO+cat=n+def=def+number=pl+case=acc
```

[6] One interesting application of the JNI interface to SILO was in a system for trademark search based on phonetic similarity.

3. Evaluation

We have studied the feasibility of our encoding and the analysis of both unvowelled and vowelled Arabic word forms, using both our full-form lexicon - which contains currently 65 000 analyses but is rapidly growing - and a lexicon of 34 590 analyzed word forms which we have extracted from the Arabic Treebank. Although the lookup involves searching for a consistent instantiation of the vowels, the current lookup speed of 4000 words per second is fast enough for many important types of applications.

So far, our focus in the building of the lexicon has been concentrating on encoding all the necessary features to avoid duplicating the development process in case we realize later that there are missing features. In so doing, we have gained insight from our evaluation of the morphosyntactic description in the ATB corpus and Xerox' morphological analyzer with respect to the missing features in these resources, such as those indicated in Section 1.

Since our focus has been so far on the design, the current lexicon contains only 65 000 full-forms – which is very little for a language with a rich morphology – but it is growing at a fast pace.

Conclusion

We have described the generation of an Arabic full-form lexicon based on an adequate morphosyntactic description and its conversion into a two-level Finite State Transducer (FST) for morphology analysis and generation. We have also shown the feasibility of our encoding and the analysis of both unvowelled and vowelled Arabic words.

Acknowledgements

The integration of FST-based morphological lookup into HPSG parsers is being done in the framework of the EU project DeepThought, Contract N° IST-2001-37836.
Special thanks must go to Tim vor der Brück and Marc Schröder for their help with the Java lookup code.

References

Aronoff, M. (1994). *Morphology by Itself: Stems and Inflectional Classes*. MIT Press, Cambridge,Mass.

Beard, R. (1995). *Lexeme-Morpheme Base Morphology: A General Theory of Inflection and Word Formation*. State University of New York Press.

Beesley, K. (1998). Consonant Spreading in Arabic Stems. In *Proceedings of COLING'98*.

Beesley, K. (1990). Finite-State Description of Arabic Morphology. In *Proceedings of the Second Cambridge Conference: Bilingual Computing in Arabic and English*.

Beesley, K (1996). Arabic Finite-State Morphological Analysis and Generation. In Proceedings of the 16th Intl. Conference on Computational Linguistics, vol. 1, pp. 89-94, Copenhagen, August 1996.

Beesley, K. & Karttunen, L. (2003). *Finite State Morphology*. CSLI Publications, University of Stanford, CA

Bouillon, P., Lehmann, S., Manzi, S. & Petitpierre, D. (1998). Développement de lexiques à grande échelle. In *Actes du Colloque des journées LTT de TUNIS*, (pp. 71-80) ftp://issco-ftp.unige.ch/pub/publications/ltt.ps.gz

Buckwalter, T (2002). http://www.qamus.org.

Callmeier, U. (2000). PET -- a platform for experimentation with efficient HPSG processing techniques. In *Journal of Natural Language Engineering*, 6(1) (Special Issue on Efficient Processing with HPSG):99-108.

Cavalli-Sforza, V. & Soudi, A. (2003). Enhancements to a Morphological Generator Motivated by English-to-Arabic MT. In *Proceedings of the Eight International Symposium on Social Communication*, Center for Applied Linguistics, SANTIAGO DE CUBA, January 20-24 2003.

Cavalli-Sforza, V., Soudi, A. & Mitamura, T. (2000): Arabic Morphology Generation Using a Concatenative Strategy. In *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics* (NAACL 2000), Seattle, April 29-May 3, pp. 86-93.

Copestake, A. (2002). Implementing Typed Feature Structure Grammars. CSLI Publications, University of Stanford, CA.

Kaplan, R. M. & Kay, M. (1994). Regular Models of Phonological Rule Systems. In *Computational Linguistics*. 20:3, pp. 331-378.

Karttunen, L. (1994). Constructing Lexical Transducers. In *Proceedings of the Fifteenth International Conference on Computational Linguistics*. Coling 94, I, pages 406-411, Kyoto, Japan

Leavitt J.R. (1994). MORPHE: A Morphological Rule Compiler. Technical Report, CMU-CMT-94-MEMO.

Oflazer, K. (1996). Error-tolerant Finite State Recognition with Applications to Morphological Analysis and Spelling Correction, In Computational Linguistics Vol.22:1.

Petitpierre, D. and Russell G. (1994). MMorph - the multext morphology program. Technical report, ISSCO, University of Geneva.

Soudi, A. & Cavalli-Sforza,V. (2002c). Arabic Morphology Generation: A two-step strategy. In *Proceedings of the Arabic and Information Technology International Conference,* organized by "Le Haut Conseil de la Langue Arabe", Algiers, Algeria, 28-29 December 2002.

Soudi, A., Cavalli-Sforza,V. and Jamari, A. (2002a): Arabic Noun System Generation. In *Proceedings of The International Conference on the Processing of Arabic*, Lamanouba University, Tunisia, April 2002, pp. 69-87.

Soudi, A (2002). A Computational Lexeme-based Treatment of Arabic Morphology. Doctorat d'Etat Thesis. Jointly supervised by Mohammed V University (Rabat , Morocco) and Carnegie Mellon University (Pittsburgh, USA).

Soudi, A., Cavalli-Sforza,V. & Jamari, A. (2001): A Computational Lexeme-Based Treatment of Arabic Morphology. In *Proceedings of the Association for Computational Linguistics, Arabic Processing Workshop*, Toulouse, France, July 2001, pp.155-162.

Transliteration

The transliteration we use is for the sake of implementation and portability. This is not a phonetic system.

ا	(^alif)	A
ب		b
ت ة		t
ث		#
ج		j
ح		H
خ		x
د		d
ذ		›
ر		r
ز		z
س		s
ش		š
ص		S
ض		D
ط		T
ظ		Z
ع		`
غ		<
ف		f
ق		q
ك		k
ل		l
م		m
ن		n
ه		h
و		w
ي		y
ى		Y
ٰ		0
ـَ	(fatHa0)	a
ـِ	(kasra0)	i
ـً	(fatHtaAn)	F
ـٍ	(kasrataAn)	K
ـُ	(DamGa)	u
	(DamGataAn)	M
ـْ	(sukuwun)	o
ـّ	(šadGa)	G
أ	(hamza on ^alif)	^
إ	(hamza under ^alif)	ç
آ	(waSla0 on ^alif)	~
ؤ	(hamza on w)	V
ئ	(hamza on line)	@
ئ	(hamza on y)	v