Arabic Finite-State Morphological Analysis and Generation

Kenneth R. Beesley
Rank Xerox Research Centre
Grenoble Laboratory
Le Quartz
6, chemin de Maupertuis
38240 MEYLAN
France
ken.beesley@xerox.fr

Abstract

This paper describes a large-scale system that performs morphological analysis and generation of on-line Arabic words represented in the standard orthography, whether fully voweled, partially voweled or unvoweled. Analyses display the root, pattern and all other affixes together with feature tags indicating part of speech, person, number, mood, voice, aspect, etc. The system is based on lexicons and rules from an earlier KIMMO-style two-level morphological system, reworked extensively using Xerox Finite-State Morphology tools. The result is an Arabic Finite-State Lexical Transducer that is applied with the same runtime code used for English, French, German, Spanish, Portuguese, Dutch and Italian lexical transducers.

1 Introduction

1.1 Challenges of Arabic Morphology

Semitic languages like Arabic present unusual challenges to automatic morphological analysis and generation. The first challenge is morphotactic: whereas most languages construct words out of morphemes which are just concatenated one after another, as in *un+fail+ing+ly*, an Arabic **stem** like *daras* (دَرَس)[1] is traditionally analyzed as consisting of a three-consonant **root**, transliterated as *drs* (د ر س), which is interdigitated with a **pattern** *CaCaC*, where *C* represents a slot for a root consonant, sometimes termed a **radical**; various prefixes and suffixes can then concatenate to the stem in the familiar way. See Figure 1.

Similarly, the root *ktb* (ك ت ب) interdigitates with the same pattern to form *katab* (كَتَب); and

Abstract lexical level:

wa+ CaCaC +at
d r s

Abstract intersected level:

wa+daras+at

Figure 1: Abstract *wa+daras+at* ("and she learned/studied")

the root *brj* (ب ر ج) interdigitates with the pattern *taCaC˜aC* to form the stem *tabar˜aj* (تَبَرَّج). There are perhaps 5000 Arabic roots in common usage, and about 400 phonologically distinct patterns, most of which are ambiguous. Each root can legally combine with only a small subset of the phonologically distinct patterns, an average of about seventeen or eighteen, and this decidedly derivational process must be controlled by old-fashioned lexicography.

The second challenge is that standard Arabic surface orthography seldom represents short vowels, distinctive consonant length, and other potentially helpful details. The *wa+daras+at* example could conceivably be written fully voweled as *wadarasat* (وَدَرَسَتْ), but it is much more likely to appear as the unvoweled *wdrst* (ودرست). The resulting incompleteness of the surface orthography makes written text unusually ambiguous, with an average of almost five valid morphological analyses per word. Finally, Arabic orthography displays an idiosyncratic mix of deep morphophonological elements carried to the surface, resulting in silent letters, and more surfacy representations of epenthesis, deletion and assimilation.

1.2 Challenges of Arabic Lexical Lookup

Standard Arabic dictionaries like the Wehr-Cohen are organized by root headwords like *drs* (د ر س) and *ktb* (ك ت ب). In fact the roots by themselves

[1] The Arabic examples in this paper were produced using the ArabTeX package for TEX and LATEX by Professor Klaus Lagally.

are not valid words, nor are they even pronounceable until they are combined with a pattern. Because in orthographical words these root consonants or radicals are usually surrounded, and even split up, by other consonant letters, and because the radicals themselves may be modified by assimilation or even deleted entirely in a written word, root identification and dictionary lookup are significant challenges for learners and native speakers alike.

2 Goals

To be interesting in our applications, the Arabic morphology system had to have the following qualities:

1. It had to deal with real Arabic surface orthography, as represented on-line in standards such as ASMO 449 or the Macintosh Arabic code page (ISO8859-6). While it is possible to devise strict roman transliterations of Arabic orthography that are unambiguously convertible back and forth into real Arabic orthography, most existing romanizations are in fact transcriptions that contain more or less information than the original and so represent different orthographical systems.
2. It had to be able to analyze Arabic words as they appear in real texts. This means that input words may be fully voweled or *diacriticized* (i.e. supplied with full diacritical markings, a style of writing found only in religious texts, poetry, and writings intended for children and other learners), partially diacriticized or undiacriticized, which is the normal case. A single system had to handle undiacriticized words and yet be able to take advantage of any diacritics that might be present.
3. To facilitate lookup of words in printed and on-line dictionaries, and for pedagogical purposes, the system had to return the root as an easily distinguished part of the analysis. An easier to build, but less useful, system would simply deal with complete stems rather than roots and patterns.
4. The system had to be large and open-ended, with each root coded to restrict the patterns with which it can in fact co-occur.
5. It had to be efficient and accurate, successfully analyzing hundreds or thousands of words per second on commonly available workstations and higher-end PCs.
6. It had to perform efficient and accurate generation of valid surface forms when supplied with the component root and relevant feature tags. Analysis and generation had to be straightforward inverse operations.

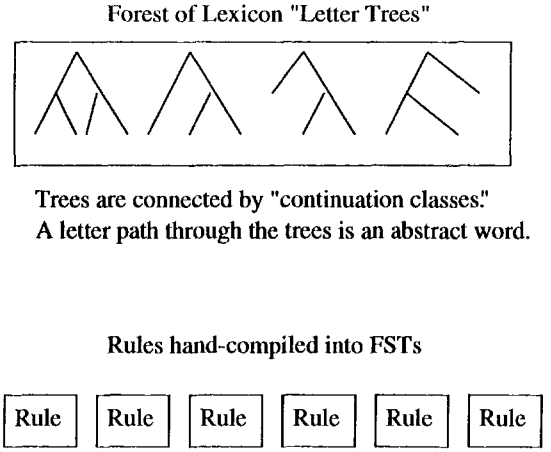

Figure 2: Traditional Kimmo-Style System Architecture

3 History

In 1989 and 1990, with colleagues at ALPNET (Beesley, 1989; Beesley, Buckwalter and Newton, 1989; Buckwalter, 1990; Beesley, 1990), I built a large two-level morphological analyzer for Arabic using a slightly enhanced implementation of KIMMO-style two-level morphology (Koskenniemi, 1983, 1984; Gajek, 1983; Karttunen, 1983). Traditional two-level morphology (see Figure 2), as in the publicly available PC-KIMMO implementation (Antworth, 1990), allows only concatenation of morphemes in the morphotactics. Lexicons are stored and manipulated at runtime as a forest of letter trees, with each tree typically containing a single class of morphemes, with the leaves connected to subsequent morpheme trees via a system of "continuation classes". A letter path through the lexical trees from a legal starting state to a final leaf defines an abstract or "lexical" string. The various two-level rules, which had to be hand-compiled into finite-state transducers, were run in parallel by code that simulated their intersection. The rules allowed and controlled the variations between the lexical strings and the surface strings being analyzed: thus the Arabic surface word *wdrst* (وذرست) could be matched with the lexical string *wa+daras+at*, among others, via appropriate rules.

In the ALPNET Arabic system, roots and patterns were stored in separate trees in the lexical forest, and an algorithm, called Detouring, performed the interdigitation of semitic roots and patterns into stems at runtime. The other chal-

lenges of Arabic morphological variation and orthography, including varying amounts of diacritical marking, all succumbed to rather complex but completely traditional two-level rules. While the resulting system was successfully sold and is also currently being used as the morphological engine of an Arabic project at the University of Maryland, it suffers from many well-known limitations of traditional two-level morphology.

1. As there was no automatic rule compiler available to us, the rules had to be compiled into finite-state transducers by hand, a tedious task that often influences the linguist to simplify the rules by postulating a rather surfacy lexical level. Hand-compilation of a complex rule, which can easily take hours, is a real disincentive to change and experimentation.

2. Because there was no algorithm to intersect the rule transducers, over 100 of them in the ALPNET system, they are stored separately and must each be consulted separately at each step of the analysis. As the time necessary to move a rule transducer to a new state is usually independent of its size, moving 100 transducers at runtime can be 100 times slower than moving a single intersected transducer.

3. Because the lexical letter trees in a traditional Kimmo-style system are decorated with glosses, features and other miscellaneous information on the leaves, they are not pure finite-state machines, cannot be combined into a single fsm, cannot be composed with the rules, and have to be stored and run as separate data structures.

4. Various diacritical features inserted into the lexical strings to insure proper analyses made this and other KIMMO-style systems awkward or impractical for generation.

5. Finally, in the enhanced ALPNET implementation, the storage of almost 5000 roots and hundreds of patterns in separate sublexicons saved memory space, but the Detouring operation that interdigitated them in realtime was inherently inefficient, building and then throwing away many superficially plausible stems that were not sanctioned by the lexicon codings. (Any Arabic root can combine legally with only a small subset of the possible patterns.) With building phantom stems and the unavoidable backtracking caused by the overall deficiency and ambiguity of written Arabic words, the resulting system was rather slow, analyzing about 2 words per second on a small IBM mainframe.

Abstract lexical level:

[drs & CaCaC]

Abstract intersected level:

daras

Figure 3: Intersection of Lexically Consecutive Root and Pattern

Abstract lexical level:

[drs & CVCVC & aa]

Abstract intersected level:

daras

Figure 4: Intersection of Lexically Consecutive Root, CV-Template, and Voweling

4 Reimplementation

Work began in 1995 to convert the analysis to the Xerox fst format. The ALPNET lexicons were first converted into the format of lexc, the lexicon compiler (Karttunen and Beesley, 1992). Although lexc by itself is largely limited to concatenative morphotactics, just like traditional two-level morphology, it was noted that the interdigitation of semitic roots and patterns is nothing more or less than their intersection, an operation supported in the Xerox finite-state calculus. Thus if ? represents any letter, and C represents any radical (consonant), the root *drs* (د ر س) can be interpreted as `?*d?*r?*s?*`.

The intersection of this root with the pattern *CaCaC* yields the stem *daras* (دَرَس). See Figure 3.

In some analyses (e.g. McCarthy, 1981), the voweling of the pattern is also abstracted out, leaving pattern templates like *CVCVC* and a vocalic element that can be formalized as `?*a?*a?*`. If V represents a vowel, then the intersection of the root, template and vocalic elements yields the same result. See Figure 4.

Using standard operations available through the lexc compiler and other finite-state tools, the analysis can be constructed according to the taste and needs of the linguists.

Because the upper-side string is returned as the result of an analysis, it is often more helpful to define the upper-side string as a baseform (here a root) followed by a set of symbol tags designed to represent relevant morphosyntactic features of the analysis. For example, *daras* (دَرَس) happens to be the Form I perfect active stem based on the root *drs* (د ر س, with *CVCVC* being the Form

Abstract lexical level:

drs+FormI+Perfect+Active

Abstract intermediate level:

drs+CVCVC+aa

Abstract intersected level:

daras

Figure 5: Root *drs* with CVCVC Template and Active Voweling

Abstract lexical level:

drs+FormI+Perfect+Passive

Abstract intermediate level:

drs+CVCVC+ui

Abstract intersected level:

duris

Figure 6: Root *drs* with CVCVC Template and Passive Voweling

I pattern and the vocal element *aa* representing active voice. The stem *duris* (دُرِس), using the passive voweling *ui* is the parallel passive example. If +FormI, +Perfect, +Active and +Passive are defined as single symbols, and if +FormI+Perfect maps to *CVCVC*, and if +Active maps to *aa* and +Passive to *ui*, the analyses can be constructed as in Figures 5 and 6.

After composition of the relevant transducers, the intermediate levels disappear, resulting in a direct mapping between the upper and lower levels shown. The resulting single transducer is called the lexicon transducer.

All valid stems, currently about 85,000 of them, are automatically intersected, at compile time, at one level of the analysis. Suitable prefixes and suffixes are also present in the lexicon transducer, added in the normal concatenative ways.

Stems like *daras* (دَرَس) and *duris* (دُرِس), and especially those like *banay* (بَنَى) based on "weak" roots, are still quite abstract and idealized compared to their ultimate surface realizations. Finite-state rules rules map the idealized strings into surface strings, handling all kinds of epentheses, deletions and assimilations. The twolc rule compiler (Karttunen and Beesley, 1992) is able not only to recompile the rules automatically but to intersect them into a single rule fst. This rule fst is then composed on the bottom of the lexicon fst, yielding a single Lexical Transducer. The symbol *.o.* in Figure 7 indicates composition.

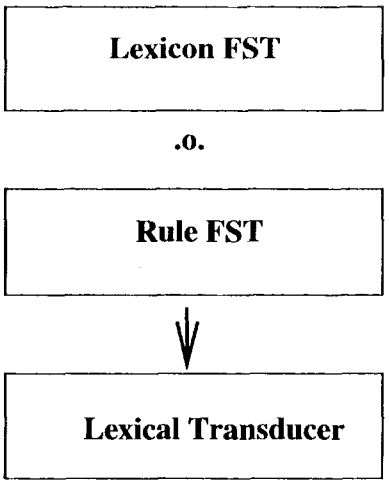

Figure 7: Composition of Lexicon and Rule FSTs into a Single Lexical Transducer

Lexical level:

drs+FormI+Perfect+Active+3P+Fem+Sg

Surface level:

drst

Figure 8: Typical Transduction from Lexical String to Unvoweled Surface String درست

Another transducer is also composed on top of the lexicon fst to map various rule-triggering features, no longer needed, into epsilon and to enforce various long-distance morphotactic restrictions. All intermediate levels disappear in the compositions, and one is left with a single two-level lexical transducer that contains surface strings on the bottom and lexical strings, including roots and tags, on the top. A typical transduction is shown in Figure 8, where the final *t* (ت) is the surface realization of the third-person feminine singular suffix *-at*. Fully voweled, the surface string for this reading would be *darasat* (دَرَسَتْ). Because short vowels are seldom written in surface words, *drst* is also analyzed as the Form I perfect passive third-person singular, which would be fully voweled as *durisat* (دُرِسَتْ), and as several other forms.

At runtime, strings being analyzed are simply matched along paths on the bottom side of the lexical transducer, and the solution strings are read off of the matching top side. Like all finite-state transducers, it also generates as easily as it analyzes, literally by running the transducer "back-

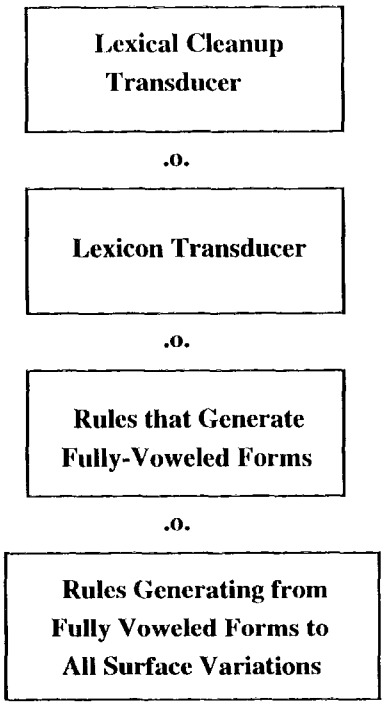

Figure 9: Full System with Two Levels of Rules

wards".

The Arabic system runs in exactly the same way, using the same runtime code, as the lexical transducers for other languages like English, French and Spanish. The Arabic system is, however, substantially slower than the other languages, because the ambiguity of the surface words forces many dead-end analysis paths to be explored and because more valid solutions have to be found and returned. The mismatch between the concatenated root and pattern on the lexical side and the intersected stem on the lower side also creates an Arabic system that is substantially larger than the other languages.

5 Generation

A single underlying Arabic word may be spelled many ways on the surface, depending on how completely the writer specifies the diacritics. Because the system described above recognizes all possible written forms of a word, with varying degrees of diacritical marking, it also generates all the possible surface forms of a word, which may be less than useful in many applications. Typically, a user wants to see only the fully voweled form during generation.

The Arabic rules have now been modified to work in two steps, first to generate the fully vowceled form, and then to generate the various partially voweled forms and the unvoweled form.

Where desired, the lexicon fst can be composed with only the upper set of rules to make a lexical transducer that generates (and recognizes) only fully-voweled surface forms. For general recognition, both sets of rules, as shown in Figure 9, are composed. The result is equivalent to the original lexical transducer described in Figure 7.

6 Conclusion

Arabic morphology, though considerably more difficult than the morphology found in the commonly studied European languages, is fully susceptible to finite-state analysis techniques, either in an enhanced two-level morphology or in the mathematically equivalent but much more computationally efficient Xerox finite-state format. We hope to extend our finite-state techniques to cover Hebrew and other languages with exotic morphology.

References

Antworth, Evan L. 1990. *PC-KIMMO: A Two-level Processor for Morphological Analysis.* Occasional Publications in Academic Computing No. 16. Dallas: Summer Institute of Linguistics.

Beesley, Kenneth R. 1989. Computer Analysis of Arabic Morphology: A Two-Level Approach with Detours. Read at the Third Annual Symposium on Arabic Linguistics, University of Utah, Salt Lake City, Utah, 3-4 March 1989. Published in Bernard Comrie and Mushira Eid (eds.), *Perspectives on Arabic Linguistics III: Papers from the Third Annual Symposium on Arabic Linguistics,* Amsterdam: John Benjamins, pp. 155-172.

Beesley, Kenneth R.; Buckwalter, Tim; and Newton, Stuart N. 1989. Two-Level Finite-State Analysis of Arabic Morphology. In *Proceedings of the Seminar on Bilingual Computing in Arabic and English,* Cambridge, England, 6-7 Sept 1989. No pagination.

Beesley, Kenneth R. 1990. Finite-State Description of Arabic Morphology. In *Proceedings of the Second Cambridge Conference on Bilingual Computing in Arabic and English,* 5-7 Sept 1990. No pagination.

Beeston, A.F.L. 1968. *Written Arabic: an approach to the basic structures.* Cambridge: Cambridge University Press.

Buckwalter, Timothy A. 1990. Lexicographic Notation of Arabic Noun Pattern Morphemes and Their Inflectional Features. In *Proceedings of the Second Cambridge Conference on Bilingual Computing in Arabic and English,* 5-7 Sept 1990. No pagination.

Gajek, Oliver *et al.* 1983. LISP Implementation. *Texas Linguistic Forum 22* ed. by Dalrymple *et al.* Austin: Linguistics Department. University of Texas, pp. 187-202

Kaplan, Ronald M. and Kay, Martin. 1981. Phonological rules and finite-state transducers [Abstract]. *Linguistic Society of America Meeting*

Handbook. Fifty-Sixth Annual Meeting, December 27-30, 1981. New York.

Kaplan, Ronald M. and Kay, Martin. 1994. Regular Models of Phonological Rule Systems. *Computational Linguistics.* 20:3, pp. 331-378.

Karttunen, Lauri. 1983. A General Morphological Processor. *Texas Linguistic Forum 22* ed. by Dalrumple *et al.* Austin: Linguistics Department, University of Texas, pp. 165-186.

Karttunen, Lauri. 1991. Finite-State Constraints. In the *Proceedings of the International Conference on Current Issues in Computational Linguistics.* June 10-14, 1991. Penang:Universiti Sains Malaysia.

Karttunen, Lauri; Kaplan, Ronald M.; and Zaenen, Annie. 1992. Two-Level Morphology with Composition. *COLING 92,* pp. 141-148.

Karttunen, Lauri. 1993. *Finite-State Lexicon Compiler.* Technical Report. ISTL-NLTT-1993-04-02. Xerox Palo Alto Research Center. Palo Alto, California.

Koskenniemi, Kimmo. 1983. *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production.* Publication No. 11. Helsinki: Department of General Linguistics, University of Helsinki.

Koskenniemi, Kimmo. 1984. A General Computational Model for Word-Form Recognition and Production. *COLING 84,* pp. 178-181.

Karttunen, Lauri and Beesley, Kenneth R. 1992. *Two-Level Rule Compiler.* Technical Report. ISTL-1992-2. Xerox Palo Alto Research Center. Palo Alto, California.

McCarthy, J. 1981. A Prosodic Theory of Nonconcatenative Morphology. *Linguistic Inquiry,* 12(3), pp. 373-418.

Wehr, Hans. 1976. *A Dictionary of Modern Written Arabic.* Third edition, ed. by J. Milton Cowan. Ithaca, N.Y.:Spoken Language Services, Inc.