STEPHEN TAYLOR
Fitchburg State College
Fitchburg, Massachusetts, USA
staylor@fsc.edu

Porting the ARAMORPH arabic morphological system to a relational database

The problem

An important trend in natural language processing is extracting statistical order from manually annotated corpora. The process of annotation is called "tagging." Words may be marked for part-of-speech or a particular word-sense. Although the situation is rapidly improving thanks to the work of the Linguistic Data Consortium and Charles University in Prague, there are as yet only a few annotated Arabic texts, and none of them is freely available. Here we describe some tools we wrote to facilitate tagging process Arabic words morphologically and for meaning.

Historically there have been a number of problems with morphological analysis in Arabic. Arabic vocabulary is very rich. There is a relatively regular derivational morphology for verbs, in which a three or four or five-letter root is transformed by changing the vowels, or adding fixed medial consonants into one of fifteen verb forms with semi-predictable variations in meaning, transitivity, agency. In addition there are many other transforms with less systematic changes of meaning, leading to nouns and adjectives. These changes are almost all internal to a word, unlike the case in English where most derivational morphology is by appending morphemes or combining words. To a lesser extent, inflexional morphology also uses internal changes. The imperfect form of a verb uses different internal vowels than the perfect form; most plurals involve stem-internal transforms; the comparative form of adjectives and the masculine/feminine forms of color adjectives all require internal changes to the stem.

Arabic is conventionally written without short vowels, so two internally-different words which differ only in their short vowels must be distinguished in reading by the surrounding syntax and semantics. In addition, the definite article, possessive pronouns, and single-letter conjunctions and prepositions are affixed to their head-word, increasing ambiguity.

A morphological analysis of Arabic words is useful for information retrieval, queries of Arabic databases, as well as linguistic experiments.

Some related work

Given the importance of word-internal transformations in Arabic morphology, many researchers have approached the Arabic morphological analysis with tools that deal with word-internal changes. (McCarthy 81) describes the derivation of Arabic words in terms of separate morphemes for root, vowel pattern, and added consonant patterns.

(El Sadany 89) describes a morphological analyzer based on traditional Arabic grammar, including pattern recognition. (Taylor 02) describes a morphological analyzer based on patterns, which is rather prone to over-generation of possible analyses. (Freeman 2001) proposes a heuristic analyzer to split word-forms on morpheme boundaries, and (Khoja 2002)

Following up the work of (Koskenniemi 85) which shows that morphological analysis in many languages can be accomplished with regular expressions and finite state automata, (Beesley 89,96,98) and (Kiraz 00) describe work in morphological analysis with such tools. Beesley describes using new operators in regular expressions to describe word-internal transformations, and Kiraz describes constructing parallel finite-state automata using cross-product operations.

In contrast, it is possible to ignore derivational morphology, if an adequate lexicon is available. In this case, an Arabic word is considered as a (possibly compound) prefix, a stem, and a suffix. It turns out that there are only hundreds of possible prefixes and suffixes, and several tens of thousands of common stems. Papers describing analyzers using this approach include (Maloney and Niv 98) (Zajac 01) (Cavalli-Sforza 00) and (Buckwalter 02) which is described in the next section.

Aramorph: Buckwalter's Database and morphological tagger

Description

In 2002 the Linguistic Data Consortium released the ARAMORPH system under the free software GNU public license. The system comprises a program written in the PERL language (Wall 1990) and several megabytes of files containing Arabic stems and affixes. The program accepts words in the Windows 1256 Arabic encoding, and prints possible morphological analyses for each word. It does this by attempting to split the word into prefix-stem-suffix in every possible way. For each attempt, it determines whether the prefix, stem, and suffix are in its dictionary, and whether they are compatible. For example, a second-person imperfect verb prefix is not compatible with a perfect verb stem, and is not compatible with a perfect masculine plural suffix. The program is quite simple; it is the knowledge contained in the files which drives it.

Affixes and stems are grouped together in categories, each of which has similar compatibility with other categories. Thus most feminine nouns with regular plurals will fall into one category, since they all take the same suffixes and prefixes; but feminine nouns beginning with lam form another, because although they take almost the same set of prefixes, they disallow the prefix lam, lam (preposition li, to- followed by Al, definite article.)

For each morpheme, the files provide

- the morpheme with its underlying vowels (usually absent in the written word, but necessary for e.g. speech generation.)
- a part-of-speech tag or tags, if appropriate. This may be a partial tag, like IV3MS for Imperfect Verb 3rd person Masculine Singular prefix.
- a compatibility category
- an English gloss

For each morpheme compatibility category, the files provide information about which other categories are compatible with it.

The Arabic text is stored internally in a transliteration encoding which provides one printable ASCII character for each distinct Arabic character. For example, ghain and thaa, which are often transcribed as gh and th, are given the encodings g and v respectively. The ligature lam-alif, which is a single keystroke on an Arabic keyboard, is stored as the two characters, lam and alif. However, separate characters are used to encode hamza, depending on its carrier; thus the characters > < } & ' encode hamza on alif, hamza under alif, hamza on yaa, hamza on waw, and hamza on the line.

This encoding permits the datafile to be read and modified with legacy software which does not support input or output of Arabic.

The program can be used to provide candidate morphological analyses for files written in the Windows 1256 encoding, by using it as a filter. If you have an Arabic version of Windows installed presumably you can type in the DOS box to the running PERL program. However, on the (non-Arabized) Windows XP system, one can type and display Arabic text, but the supported encoding is Unicode, not Windows 1256, so the program cannot be used interactively.

Various extensions for the tagger: bwmorph

In order to use the program interactively, we modified it to accept input in its internal transliteration form. We also changed it to accept vowelled text, and optionally discard analyses which produce conflicting vowelling, so that kutub (books) would not be an acceptable analysis for input of kataba (he wrote) but both would acceptable for an input of ktb.

These modifications give us an interactive version, without requiring that we run an Arabic version of Windows. As an interactive program, the analyzer also serves as a Arabic-English dictionary, at least for those 38000+ stems in its files.

Other uses for the DB: extracting broken plurals

The data files can also be used to support various enquiries on Arabic text. For example, we wanted to compile lists of the static frequencies of various forms of irregular plural, versus regular plurals. We extracted stems whose glosses ended in -s or -es, then examined the list for non-plurals like 'loss' This excludes most words with irregular plurals in English, but hopefully the distribution of the excluded stems should have characteristics no different than those extracted. Preliminary results suggest that plurals with the pattern faEaAlil, including words like TaraA}iq (manners) TaxArir (clouds) banAdar (seaports) wasAwis (whispers) are the most common. We suspect frequencies of actual use would give a different result.)

Other uses for the DB: listing verb forms

In another example of using the data files to extract static frequencies, we extracted verb stems, and endeavored to count frequencies of various verb forms. (No great surprises here, except that our counting technique counts some words as matching more than one pattern.) In preliminary results, triliteral form I verbs (fvEvl) are twice as common as triliteral form II verbs (fvE~vl) which are about as common a quadriliteral form I verbs (fvElvl)

Other uses for the DB: English-Arabic glossary

The stem file can be sorted by English gloss, to give a English-Arabic dictionary, but the coverage of English is sparse, since gloss words are not chosen with coverage in mind. We contemplate an experiment in which the Wordnet list of English senses -- which can be used to generate lengthy synonym lists for English words -- is used to improve coverage.

Our port of the Buckwalter DataBase

The ARAMORPH program is written in PERL, and our implementation of PERL doesn't support graphics or Arabic output. However, most of the analyzer is actually its data files. The program is only a few pages long. Another problem is that the files are larger than is convenient for editing. The file of stems is almost 4 MB long.

We decided that we would port the program data to a Microsoft ACCESS database file, and rewrite the program in some other language, preferably one which could easily be used to produce Arabic output on Windows.

Aramorph: The adict Electronic Arabic Dictionary

We rewrote the analysis/database lookup portion of the ARAMORPH application in three languages: Visual Basic, Java, and Java Server Pages (JSP) a Java/HTML hybrid designed for writing server-side interactive web pages. adict.jsp is the JSP version of the lookup, and the implementation which gave the best-looking output and the best-feeling interactions. In this implementation, the database operations are carried out in JavaBeans running under the TomCat Web server, and the Arabic output is rendered by a browser displaying dynamic Web pages. We used Internet Explorer as the browser, and it gives good display of Arabic text. The JavaBean ArabicTagger which implements database portion of the Java implementation is reused as separate class in the applications which are described below.

The implementation of adict doesn't contain any of various tricks in ARAMORPH which compensate for sub-standard orthography, which include modifying letters frequently misspelled and retrying the search, except for those required because they are already built into the data. In the latter category is use of alif alone in searching for prefixes, when correct spelling would suggest using one of hamza on alif, hamza under alif, or alif madda.

Although it would have been possible to accept typed input in Arabic Unicode in the adict.jsp application, we followed our earlier modification to ARAMORPH and use the internal Buckwalter transliteration as input. The input is then converted to Arabic script and output to the screen, along with the possible morphological analyses, if any.

A side effect of using JSP for development is that the dictionary can now be made available on and off campus, wherever there is an Internet connection.

The programs mup (markup) and record

With the basic morphological analysis ported, we turned our attention to building a tool to assist manual tagging of Arabic text. Since statistical properties of text from different domains may vary, it makes sense to have some text marked up for any particular area of interest.

Various kinds of annotation are possible, including part-of-speech, choice of a particular sense as the meaning of a word, syntactic status as head-word or modifier. In this case, the database provides part-of-speech and English gloss information, so that is what we tag. We discard some of the meta-information which is available when we access the database, for example, the categories of the morphemes, and produce an unstructured text tag, from which information can later be re-extracted.

The markup application consists of several different web pages, implemented with JSP. It reads a flat file, and presents the user with a web-based form. The form presents the next several Arabic words from the input file, and for each word offers a drop-down selection menu, containing each of the possible morphological analyses, and the choice "None of the above". A few long words and proper nouns have only one possible analysis, and a few more words get no analysis from the software, so that "None of the above" shows as the only choice, although of course they would have an analysis if the database were sufficiently large. Most of these words are proper nouns, but a few are stems not included in the database.

When the user has surveyed the default choices for analyses offered in the form, and changed those for which there seems to be a better choice, the choices are recorded in an Access database, and the next several words are presented.

A separate part of the application provides for browsing the database and reviewing the tags assigned to each word. Since many words do not have automatically generated tags, an additional facility for "None of the above" tags may be invoked at this time. Manual tagging offers the user the opportunity, but not the requirement, to provide some or all of 1) prefix/stem/suffix boundaries 2) English gloss 3) part-of-speech tags. Manual tags are punctuated like automatically generated ones, so that software for extracting information from the text tag will also work on them, but as yet no effort is made to update the database of stems.

Related tools

At present we have no tools to assist editing the vocabulary database, although the prospect of writing such tools is one of the justifications for porting the database from a flat file. MicroSoft Access provides basic tools for making new entries, and they can easily be extended with Access forms and reports, but adding a new stem requires strong understanding of the stem categories and their compatibilities. To some extent we could extract compatibility information from those tags generated manually. A stem which is tagged with a particular prefix and suffix must obviously be compatible with both of them, and this limits the possible categories into which the stem may fall.

We are also going to need tools for extracting structured tag information from punctuated text tags. Some of this will be easy to get; for example, the text tag always precedes part-of-speech tag information with a slash (/). Others are more difficult. For example, it probably makes sense to combine verb tense affixes, or regular plural markers, which appear as parts of speech tags for the prefix and/or suffix, into feature sets for the stem part-of-speech.

More things to do

We have not yet implemented any mechanism for ranking analyses, but doing so should be a good way to speed up the process of tagging. For example, if the same word has been tagged before in the document, then it is quite likely that it will receive the same tag again. If this were made the default, then the person tagging would have an enhanced chance of doing nothing for that word, thus speeding up the tagging process. Similarly, stems that are in words that have been previously tagged are more likely to occur.

References

1. Beesley, Kenneth R., 1996, *Arabic Finite State Analysis and Generation,* Computational Linguistics Conference 1996,
2. Beesley, Kenneth R., 1998, *Arabic Morphology Using Only Finite State Operations,* Computational Linguistics Conference 1998
3. Buckwalter, Tim, 2001. *ARAMORPH Arabic Morphological Analyzer.* Linguistic Data Consortium
4. Cavalli-Sforza, Violetta, and Abdelhadi Soudi and Teruko Mitamura, 2000, *Arabic Morphology Generation Using a Concatenative Strategy*
5. Darwish, Kareem 2002, *An Arabic Morphological Analyzer,* Workshop on Computational Approaches to Semitic Languages, Computational Linguistics Conference
6. El Sadany, T.A and M.A. Hashish 1989. *An Arabic morphological system*. IBM Systems Journal 28(4):600-612,
7. Freeman, Andrew, 2001. *Brill's POS tagger and a Morphology parser for Arabic*. Arabic Language Processing Workshop, Toulouse.
8. Hajik, Jan and Barbora Hladka, 1998,. *Tagging in languages: Prediction of morphological categories for a rich structured tagset*. Computational Linguistics Conference.
9. Khoja, Shereen, 2002 PhD. Thesis at Lancaster University.
10. Kiraz, George A, 2000, *Computation Nonlinear Morphology,* Cambridge University Press
11. Koskenniemi. Kimmo. 1983. *Two-level morphology: A General Computational Model for Word-Form Recognition and Production.* PhD thesis, University of Helsinki.
12. McCarthy, John J., 1981 *A Prosodic Theory of Non-Concatenative Morphology,* Linguistic Inquiry, vol. 12, n. 3, pp 376-418
13. Maloney, John and Michael Niv, 1998. *TAGARAB: A Fast Accurate Arabic Name Recognizer using High-Precision Morphological Analysis,* Workshop on Computational Approaches to Semitic Languages, Computational Linguistics Conference
14. Taylor, Stephen 2002, *Regular expressions and Arabic Lexical Forms*. Arabic Linguistics Symposium, Cambridge University, 2002
15. Wall, Larry and Randal L. Schwartz. Programming PERL. *O'Reilly and Associates*, Sebastopol, CA, USA, 1990
16. Zajac, Remi, Malki, A., Abdelali, A., Cowie, J., Ogden W. C. (2001, July) *Arabic-English NLP at CRL*, Proceedings of the Arabic NLP Workshop ACL, Toulouse (France).