**Conversion of Procedural Morphologies to Finite-State Morphologies: a**

**Case Study of Arabic**

**Mans Hulden** University of the Basque Country IXA Group

**Younes Samih**

Heinrich-Heine-Universita¨t Du¨ sseldorf

[samih@phil.uni-duesseldorf.de](mailto:samih@phil.uni-duesseldorf.de)

IKERBASQUE, Basque Foundation for Science

[mhulden@email.arizona.edu](mailto:mhulden@email.arizona.edu)

**Abstract**

In this paper we describe a conversion of the Buckwalter Morphological Analyzer for Arabic, originally written as a Perl-script, into a pure finite-state morphological ana- lyzer. Representing a morphological ana- lyzer as a finite-state transducer (FST) con- fers many advantages over running a procedu- ral affix-matching algorithm. Apart from ap- plication speed, an FST representation imme- diately offers various possibilities to flexibly modify a grammar. In the case of Arabic, this is illustrated through the addition of the abil- ity to correctly parse partially vocalized forms without overgeneration, something not possi- ble in the original analyzer, as well as to serve both as an analyzer and a generator.

**1 Introduction**

Many lexicon-driven morphological analysis sys- tems rely on a general strategy of breaking down input words into constituent parts by consulting cus- tomized lexicons and rules designed for a particu- lar language. The constraints imposed by the lex- ica designed are then implemented as program code that handles co-occurrence restrictions and analysis of possible orthographic variants, finally producing a parse of the input word. Some systems designed along these lines are meant for general use, such as the *hunspell* tool (Hala´csy et al., 2004) which allows users to specify lexicons and constraints, while oth- ers are language-dependent, such as the Buckwalter Arabic Morphological Analyzer (*BAMA*) (Buckwal- ter, 2004).

In this paper we examine the possibility of con- verting such morphological analysis tools to FSTs

that perform the same task. As a case study, we have chosen to implement a one-to-one faithful conver- sion of the Buckwalter Arabic analyzer into a finite- state representation using the *foma* finite state com- piler (Hulden, 2009b), while also adding some ex- tensions to the original analyzer. These are useful extensions which are difficult to add to the original Perl-based analyzer because of its procedural nature, but very straightforward to perform in a finite-state environment using standard design techniques.

There are several advantages to representing mor- phological analyzers as FSTs, as is well noted in the literature. Here, in addition to documenting the con- version, we shall also discuss and give examples of the flexibility, extensibility, and speed of application which results from using a finite-state representation of a morphology.1

**2 The Buckwalter Analyzer**

Without going into an extensive linguistic discus- sion, we shall briefly describe the widely used Buck- walter morphological analyzer for Arabic. The *BAMA* accepts as input Arabic words, with or with- out vocalization, and produces as output a break- down of the affixes participating in the word, the stem, together with information about conjugation

classes. For example, for the input word **ktb**/ ..\_. �s',

*BAMA* returns, among others:

LOOK-UP WORD: ktb

SOLUTION 1: (kataba) [katab-u\_1]

katab/VERB\_PERFECT

+a/PVSUFF\_SUBJ:3MS (GLOSS): + write + he/it <verb>

1 The complete code and analyzer are available at

<http://buckwalter-fst.googlecode.com/>

70

*Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, pages 70–74, Donostia–San Sebastia´n, July 23–25, 2012. *Q*c 2012 Association for Computational Linguistics

Figure 1: The Buckwalter Arabic Morphological Analyzer’s lookup process exemplified for the word **lilkitAbi**.

**2.1 *BAMA* lookup**



In the *BAMA* system, every Arabic word is assumed to consist of a sometimes optional prefix, an oblig- atory stem, and a sometimes optional suffix.2 The system for analysis is performed by a Perl-script that carries out the following tasks:

1. Strips all diacritics (vowels) from the input word (since Arabic words may contain vocal- ization marks which are not included in the lex-

icon lookup). Example: kataba *→* ktb

2. Factors the input word into all possible combinations of prefix-stem-suffix. Stems may not be empty, while affixes are optional.

Example: ktb *→ { <*k,t,b*>*,*<* kt,b,*∅>*,

*<*k,tb,*∅>*, *<∅*,k,tb*>*, *<∅*,kt,b*>*,

*<∅*,ktb,*∅> }*.

3. Consults three lexicons (**dictPrefixes**, **dict- Stems**, **dictSuffixes**) for ruling out impossi-

ble divisions. For example, *<*kt,b,*∅>*, is

rejected since **kt** does not appear as a prefix in **dictPrefixes**, while *<*k,tb,*∅>* is accepted

since **k** appears in **dictPrefixes**, **tb** in **dict- Stems**, and *∅* in **dictSuffixes**.

4. Consults three co-occurrence constraint lists for further ruling out incompatible prefix- stem combinations, stem-suffix combinations, and prefix-suffix combinations. For example,

2 In reality, these are often conjoined prefixes treated as a single entry within the system.

*<*k,tb,*∅>*, while accepted in the previous

step, is now rejected because the file **dict-**

**Prefixes** lists **k** as a prefix belonging to class **NPref-Bi**, and the stem **tb** belonging to one of **PV V**, **IV V**, **NF**, **PV C**, or **IV C**. However, the compatibility file **tableAB** does not permit a combination of prefix class **NPref-Bi** and any of the above-mentioned stem classes.

5. In the event that the lookup fails, the analyzer considers various alternative spellings of the in- put word, and runs through the same steps us- ing the alternate spellings.

The *BAMA* lookup process is illustrated using a different example in figure 1.

**3 Conversion**

Our goal in the conversion of the *Perl*-code and the lookup tables is to produce a single transducer that maps input words directly to their morphological analysis, including class and gloss information. In order to do this, we break the process down into three major steps:

(1) We construct a transducer Lexicon that ac- cepts on its output side strings consisting of any combinations of fully vocalized prefixes, stems, and suffixes listed in **dictPrefixes**, **dict- Stems**, and **dictSuffixes**. On the input side, we find a string that represents the class each morpheme on the output side corresponds to, as well as the line number in the correspond-

LEXICON Prefixes

Prefixes ;

**tableBC**, which lists co-occurrence constraints between stems and suffixes contains only the

[Pref-%0]{P%:34}:0 Stems;

[Pref-Wa]{P%:37}:wa Stems;

...

LEXICON Stems

[Nprop]{S%:23}:|b Suffixes; [Nprop]{S%:27}:%<ib˜ Suffixes;

...

LEXICON Suffixes

[Suff-%0]{X%:34}:0 #; [CVSuff-o]{X%:37}:o #;

...

Figure 2: Skeleton of basic lexicon transducer in LEXC

generated from *BAMA* lexicons.

ing file where the morpheme appears. For ex- ample, the Lexicon transducer would contain the mapping:

[Pref-0]{P:34}[PV]{S:102658}[NSuff-a]{X:72}

kataba

indicating that for the surface form

s'

**kataba**/ ..\_ . � , the prefix class is **Pref-0**

appearing on line 34 in the file **efixes**,

**dictPr**

the stem class is **PV**, appearing on line

102,658 in **dictStems**, and that the suffix class is **NSuff-a**, appearing on line 72 in **dictSuffixes**.

To construct the Lexicon, we produced a *Perl*-script that reads the contents of the *BAMA* files and automatically constructs a LEXC- format file (Beesley and Karttunen, 2003), which is compiled with *foma* into a finite trans- ducer (see figure 2).

(2) We construct rule transducers that filter out im- possible combinations of prefix classes based on the data in the constraint tables **tableAB**, **tableBC**, and **tableAC**. We then iteratively compose the Lexicon transducer with each rule transducer. This is achieved by converting each suffix class mentioned in each of the class files to a constraint rule, which is compiled

following lines beginning with Nhy:

Nhy NSuff-h

Nhy NSuff-iy

indicating that the Nhy-class only combines with Nsuff-h or Nsuff-iy. These lines are converted by our script into the constraint re- striction regular expression:

def Rule193 "[Nhy]" => \_ ?\*

"[NSuff-h]"|"[NSuff-iy]"];

This in effect defines the language where each instance [Nhy] is always followed some- time later in the string by either [NSuff-h], or [NSuff-iy]. By composing this, and the other constraints, with the Lexicon- transducer, we can filter out all illegitimate combinations of morphemes as dictated by the original Buckwalter files, by calculating:

def Grammar Lexicon.i .o.

Rule1 .o.

... RuleNNN ;

In this step, it is crucial to note that one cannot in practice build a separate, single transducer (or automaton) that models the intersection of *all* the lexicon constraints, i.e. Rule1 .o. Rule2 .o. ... RuleNNN, and then compose that transducer with the Lexicon transducer. The reason for this is that the size of the intersection of all co-occurrence rules grows exponentially with each rule. To avoid this intermediate exponential size, the Lexicon transducer must be composed with the first rule, whose composition is then com- posed with the second rule, etc., as above.

(3) As the previous two steps leave us with a trans- ducer that accepts only legitimate combina- tions of fully vocalized prefixes, stems, and suffixes, we proceed to optionally remove short vowel diacritics as well as perform optional normalization of the letter Alif ( ) from the

for instance, that an intermediate **kataba**/ ..\_. �s', would be mapped to the surface forms **kataba**,

**katab**, **katba**, **katb**, **ktaba**, **ktab**, **ktba**, and **ktb**. This last step assures that we can parse partially vocalized forms, fully vocal- ized forms, completely unvocalized forms, and common variants of Alif.

def RemoveShortVowels

[a|u|i|o|%˜|%‘] (->) 0;

def NormalizeAlif ["|"|"<"|">"] (->) A .o. "{" (->) [A|"<"] ;

def RemovefatHatAn [F|K|N] -> 0;

def BAMA 0 <- %{|%} .o.

Grammar .o. RemoveShortVowels .o. NormalizeAlif .o. RemovefatHatAn;

**4 Results**

Converting the entire *BAMA* grammar as described above produces a final FST of 855,267 states and

1,907,978 arcs, which accepts 14,563,985,397 Ara- bic surface forms. The transducer occupies 8.5Mb. An optional auxiliary transducer for mapping line numbers to complete long glosses and class names occupies an additional 10.5 Mb. This is slightly more than the original *BAMA* files which occupy

4.0Mb. However, having a FST representation of the grammar provides us with a number of advan- tages not available in the original *BAMA*, some of which we will briefly discuss.

**4.1 Orthographical variants**

The original *BAMA* deals with spelling variants and substandard spelling by performing *Perl*-regex re- placements to the input string if lookup fails. In the *BAMA* documentation, we find replacements such as:

- word final Y’ should be y’

- word final Y’ should be }

- word final y’ should be }

In a finite-state system, once the grammar is con- verted, we can easily build such search heuristics

rules and various composition strategies such as pri- ority union (Kaplan, 1987). We can thus mimic the behavior of the *BAMA*, albeit without incurring any extra lookup time.

**4.2 Vocalization**

As noted above, by constructing the analyzer from the fully vocalized forms and then optionally remov- ing vowels in surface variants allows us to more ac- curately parse partially vocalized Arabic forms. We thus rectify one of the drawbacks of the original *BAMA*, which makes no use of vocalization informa- tion even when it is provided. For example, given an input word **qabol**, *BAMA* would as a first step strip off all the vocalization marks, producing **qbl**. Dur- ing the parsing process, *BAMA* could then match **qbl** with, for instance, **qibal**, an entirely different word, even though vowels were indicated. The FST de- sign addresses this problem elegantly: if the input word is **qabol**, it will never match **qibal** because the vocalized morphemes are used throughout the con- struction of the FST and only optionally removed from the surface forms, whereas *BAMA* used the un- vocalized forms to match input. This behavior is in line with other finite-state implementations of Ara- bic, such as Beesley (1996), where diacritics, if they happen to be present, are taken advantage of in order to disambiguate and rule out illegitimate parses.

This is of practical importance when parsing Ara- bic as writers often partially disambiguate words depending on context. For example, the word

**Hsbt**/ ..\_ . -�is ambiguous (**Hasabat** = compute,

charge; **Hasibat** = regard, consider). One would

partially vocalize **Hsbt** as **Hsibt** to denote “she regards”, or as **Hsabt** to imply “she computes.” The FST-based system correctly narrows down the parses accordingly, while *BAMA* would produce all ambiguities regardless of the vocalization in the in- put.

**4.3 Surface lexicon extraction**

Having the *BAMA* represented as a FST also al- lows us to extract the output projection of the gram- mar, producing an automaton that only accepts le- gitimate words in Arabic. This can be then be used in spell checking applications, for example, by integrating the lexicon with weighted transduc-

ers reflecting frequency information and error mod- els (Hulden, 2009a; Pirinen et al., 2010).

**4.4 Constraint analysis**

Interestingly, the *BAMA* itself contains a vast amount of redundant information in the co- occurrence constraints. That is, some suffix-stem- lexicon constraints are entirely subsumed by other constraints and could be removed without affecting the overall system. This can be observed during the chain of composition of the various transducers rep- resenting lexicon constraints. If a constraint *X* fails to remove any words from the lexicon—something that can be ascertained by noting that the number of paths through the new transducer is the same as in the transducer before composition—it is an indi- cation that a previous constraint *Y* has already sub- sumed *X* . In short, the constraint *X* is redundant.

The original grammar cannot be consistently ana- lyzed for redundancies as it stands. However, redun- dant constraints can be detected when compiling the Lexicon FST together with the set of rules, offer- ing a way to streamline the original grammar.

**5 Conclusion**

We have shown a method for converting the table- based and producedural constraint-driven Buckwal- ter Arabic Morphological Analyzer into an equiva- lent finite-state transducer. By doing so, we can take advantage of established finite-state methods to pro- vide faster and more flexible parsing and also use the finite-state calculus to produce derivative applica- tions that were not possible using the original table- driven Perl parser, such as spell checkers, normaliz- ers, etc. The finite-state transducer implementation also allows us to parse words with any vocalization without sacrificing accuracy.

While the conversion method in this case is spe- cific to the BAMA, the general principle illustrated in this paper can be applied to many other procedu- ral morphologies that rule out morphological parses by first consulting a base lexicon and subsequently applying a batch of serial or parallel constraints over affix occurrence.

**References**

Attia, M., Pecina, P., Toral, A., Tounsi, L., and van Genabith, J. (2011). An open-source finite state morphological transducer for modern stan- dard Arabic. In *Proceedings of the 9th Interna- tional Workshop on Finite State Methods and Nat- ural Language Processing*, pages 125–133. Asso- ciation for Computational Linguistics.

Beesley, K. R. (1996). Arabic finite-state morpho- logical analysis and generation. In *Proceedings of COLING’96—Volume 1*, pages 89–94. Associ- ation for Computational Linguistics.

Beesley, K. R. and Karttunen, L. (2003). *Finite State*

*Morphology*. CSLI Publications, Stanford, CA. Buckwalter, T. (2004). Arabic Morphological Ana-

lyzer 2.0. *Linguistics Data Consortium (LDC)*.

Habash, N. (2010). Introduction to Arabic natural language processing. *Synthesis Lectures on Hu- man Language Technologies*.

Hala´csy, P., Kornai, A., Ne´meth, L., Rung, A., Sza- kada´t, I., and Tro´ n, V. (2004). Creating open lan- guage resources for Hungarian. In *Proceedings of Language Resources and Evaluation Conference (LREC04). European Language Resources Asso- ciation.*

Hulden, M. (2009a). Fast approximate string match- ing with finite automata. *Procesamiento del lenguaje natural*, 43:57–64.

Hulden, M. (2009b). Foma: a finite-state compiler and library. In *Proceedings of the 12th confer- ence of the European Chapter of the Association for Computational Linguistics,*, pages 29–32. As- sociation for Computational Linguistics.

Kaplan, R. M. (1987). Three seductions of computa- tional psycholinguistics. In Whitelock, P., Wood, M. M., Somers, H. L., Johnson, R., and Bennett, P., editors, *Linguistic Theory and Computer Ap- plications*, London. Academic Press.

Pirinen, T., Linde´n, K., et al. (2010). Finite-state spell-checking with weighted language and error models. In *Proceedings of LREC 2010 Workshop on creation and use of basic lexical resources for less-resourced languages*.