Finite-State Morphological Analysis and Generation of Arabic at Xerox Research: Status and Plans in 2001

Kenneth R. Beesley
Xerox Research Centre Europe
6, chemin de Maupertuis
38240 MEYLAN, France
Ken.Beesley@xrce.xerox.com

Abstract

This paper describes a finite-state morphological analyzer of Modern Standard Arabic words that can be tested on the Internet. An overview of the system is provided, including the history, the finite-state technology, and the dictionary coverage. This research system is scheduled for testing and commercial development in 2001.

1 Introduction

In 1996, the Xerox Research Centre Europe produced a morphological analyzer for Modern Standard Arabic, henceforth Arabic (Beesley, 1996). In 1997 a Java-applet interface was added to allow testing on the Internet using standard Arabic orthography. The analyzer-generator is based on dictionaries from an earlier project at ALPNET (Beesley, 1990; Buckwalter, 1990), but the system was extensively redesigned and rebuilt using Xerox finite-state technology (Beesley and Karttunen, 2001).

The system analyzes orthographical words that may include full, partial, or no diacritics; and if diacritics are present, they automatically constrain the ambiguity of the output. A fully-voweled spelling and a terse English gloss are also returned with each analysis.

The system is intended to serve as a pedagogical aid, a comprehension-assistance tool, and as a component in larger natural-language-processing systems.

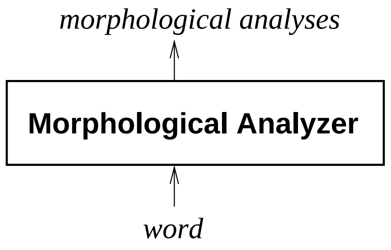

Figure 1: A Generic Morphological Analyzer as a Black Box

2 Challenges of the Arabic System

2.1 General Morphological Analysis

In its simplest diagrammatic form, a morphological analyzer can be characterized as a black-box module (see Figure 1) that accepts words and outputs morphological analyses. Such black boxes might be implemented in any number of ways.

In computer analysis of Arabic, or of any other language, the input words are in digital form, with the characters in standard encodings like ASMO8859-6 and Unicode. As for the content of the morphological analyses, they will always be somewhat theory- and application-dependent. In the broadest terms, a morphological analyzer should separate and identify the component morphemes of the input word, labeling them somehow with sufficient information to be useful for the tasks at hand.

2.2 Arabic Morphological Analysis

In the case of Arabic, one would certainly expect a morphological analyzer to separate and identify prefixed word-like morphemes such as the conjunctions **wa–** and **fa–**, prefixed prepositions such as **bi–** and **li–**, the definite article, verbal prefixes and suffixes, nominal case suffixes, and enclitic direct-object and possessive-pronoun suffixes.

Where things become more complicated is in the formal analysis of the Arabic or, more generally, the Semitic STEM; various lexicographers and linguists have proposed:

1. Stems as one-part structures: i.e. a stem is simply treated as a single monolithic morpheme.

2. Stems as two-part constructs of a ROOT morpheme and a PATTERN morpheme: e.g. root **ktb** (ك ت ب)[1] and a pattern spelled **_u_i_** or **CuCiC**, where the underscores or **C** symbols represent slots for the root consonants, sometimes termed RADICALS. In such a system, the root and pattern morphemes are said informally to "interdigitate" together to form stems like **kutib** (كُتِب).[2]

3. Stems as three-part constructs of a ROOT, a consonant-vowel (**CV**) TEMPLATE, and a VOCALIZATION: e.g. **ktb** and template **CVC̦VC** and vocalization **ui** to form **kutib** (كُتِب). The templates are drawn from a set of perhaps dozens, and the radicals and vowels are associated with the **C** and **V** slots via controversial association rules. Such a three-way division was popularized by some of the early work of McCarthy (1981).[3]

4. Stems as multi-part constructs of a root, a prosodically motivated pattern (from a severely limited set), a vocalization, and various affixes. This approach characterizes some of the more recent work of McCarthy (1993).[4]

In what follows, I shall assume that Semitic stems are built from at least two morpheme components, including a root usually consisting of three (but sometimes two or four) radicals like **ktb** (ك ت ب), **drs** (د ر س), **fʕl** (ف ع ل), **klkl** (ك ل ك ل), **sm** (س م) or **tm** (ت م).[5] I shall also assume that a crucial task for a morphological analyzer is to separate and identify these underlying roots, even if they are obscured in the surface word by morphophonological or orthographical alternations.

In addition to the arguments for the linguistic reality of Semitic roots,[6] an entirely practical motivation for identifying roots is that traditional Arabic dictionaries, such as the authoritative Hans Wehr dictionary (Wehr, 1979), are organized with root headings. To look up a word in such a dictionary, the student must know or somehow identify the root; and this is a notoriously difficult task for students as well as for many natives.

3 System Description

Users access the web-based demo via the Arabic homepage.[7] The Arabic entry page (see Figure 2) is mostly filled up with a Java applet that displays a virtual Arabic keyboard. Users can type in words either by mouse-clicking on the virtual-key objects or by typing the corresponding keys on their physical keyboard. Four alternate Arabic keyboard layouts are currently provided: PC-like, Mac-like, and the implementors' preferred Buckwalter Transliteration[8] laid out on both English and French keyboards. As words are typed, the appropriate Unicode Arabic characters are added to an internal buffer. This buffer is "observed" by an Arabic Canvas object that renders the appropriate Arabic glyphs right-to-left on the screen, updating the display interactively each time the buffer is changed in any way. Another related in-

[1]Arabic-script displays in this paper were encoded using the ArabTeX package for TeX and LaTeX by Prof. Dr. Klaus Lagally of the University of Stuttgart.

[2]For such a two-part analysis of Hebrew stems, see Harris (1941) and Lavie et al. (1988); for Akkadian, see Kataja and Koskenniemi (1988); for Arabic, see the papers cited by Beesley and Buckwalter.

[3]Kay (1987) took McCarthy's Arabic data and reformalized it using multi-level finite-state transducers.

[4]Kiraz (2000) has formalized several stages of McCarthy's theories, using multi-level finite-state transducers and applying them to Syriac.

[5]Depending on one's theory, a root like **tm** may also be formalized as an abstract triliteral **tmm** or **tmX**, where **X** indicates the copying or spreading of the previous consonant (Beesley, 1998c).

[6]For example, McCarthy (1981) cites the charming example of a Hijazi Bedouin play language wherein speakers freely scramble the order of the radicals within a root.

[7]`http://www.xrce.xerox.com/research/mltt/arabic`

[8]`http://www.xrce.xerox.com/research/mltt/arabic/info/translit-chart.html`

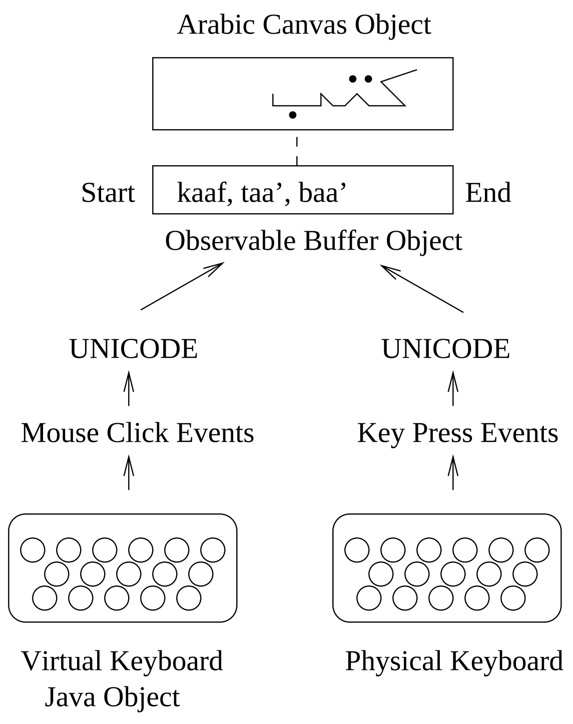

Figure 2: User Interface with Arabic Script Display in Java. Mouse clicks on the virtual keyboard or key presses on the physical keyboard are intercepted, converted to Arabic Unicode characters, and stored in a buffer, which has a start and an end but no inherent ordering. The Arabic Canvas Object observes the buffer and contains an Arabic Scribe object that renders the string of Unicode characters right-to-left as connected Arabic glyphs.

put page allows the user to cut and paste words from other Arabic-language webpages, including those encoded in MS 1256.

When the user presses the Enter key the buffer contents are sent to a Perl CGI script running on a server in Grenoble, France. The script passes each input word to the morphological analyzer (Figure 3), which is implemented as a finite-state transducer (FST). Typically there are several output strings, each representing a possible analysis of the input word.[9]

When presenting the various solutions back to the user, it is valuable to display the fully-voweled (full diacritics) spelling of each solution. So each solution is passed to a morphological generator FST, which is exactly the same as the an-

[9]The terms string and word are used interchangeably.

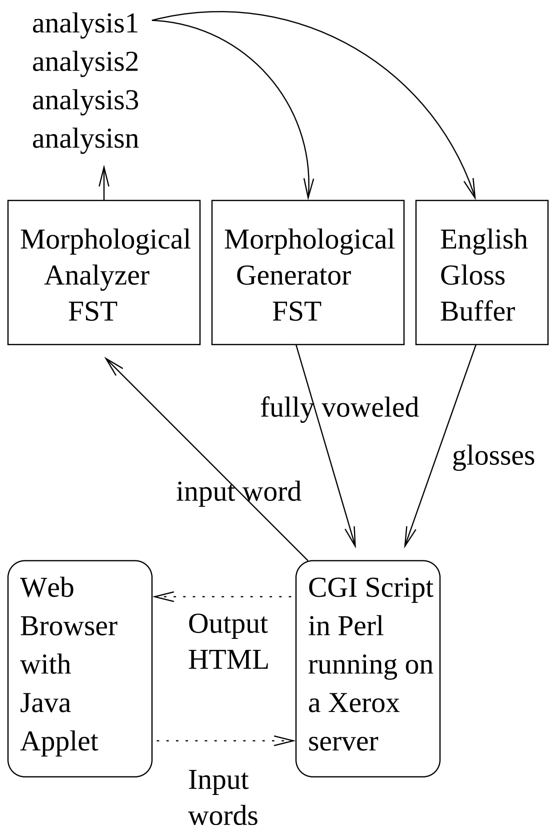
analysis1
analysis2
analysis3
analysisn
Morphological Analyzer FST
Morphological Generator FST
English Gloss Buffer
fully voweled
glosses
input word
Web Browser with Java Applet
Output HTML
CGI Script in Perl running on a Xerox server
Input words

Figure 3: Information Flow in the Xerox Arabic Demo. Input words from the user interface are transmitted across the Internet (dotted lines) and analyzed by a server, typically producing multiple analysis strings. Each analysis string is then generated in fully voweled form, combined with English glosses and then reformatted as HTML before being sent back across the Internet to the user’s browser for display. The analyzer and generator finite-state transducers (FSTs) are identical except that the lower side language of the generator is limited to contain only fully-voweled words.

alyzer except for having a lower-level language that is restricted to fully-voweled strings. The various solutions are also tokenized into morphemes, which are looked up in a dictionary of English glosses. The Perl script incorporates the analyses, the generated fully-voweled strings, and the English glosses into an HTML page that is sent back to the user's Internet browser for display.

4 Finite-State Morphological Processing

4.1 General Finite-State Theory and Techniques

The Arabic Morphological Analyzer is built using finite-state compilers and algorithms, and the results are stored and run as finite-state transducers. The finite-state approach to morphology, using a variety of software implementations, has become very popular around the world, having been used to create morphological analyzers for all the commercially important European languages, including Hungarian and Finnish, as well as Japanese, Korean, Swahili, Aymara, Malay, Klingon, etc.

While the original Two-Level implementation (Koskenniemi, 1983; Antworth, 1990) of finite-state theory was not particularly well-suited to Semitic languages, modification of the two-level paradigm and more advanced finite-state implementations have been applied successfully to a variety of Semitic languages, including Ancient Akkadian (Kataja and Koskenniemi, 1988), Hebrew (Lavie et al., 1988), Syriac (Kiraz, 2000), and Arabic.

4.2 Xerox Finite-State Tools and Techniques

The Xerox approach to finite-state morphology is well described elsewhere, and only an outline is possible and appropriate here. Lexicons and morphotactic information are encoded in the **lexc** language (Karttunen, 1993), which is a kind of right-recursive phrase-structure grammar, and are compiled into finite-state transducers.

Finite-state transducers (FSTs) are data structures that encode REGULAR RELATIONS (Kaplan and Kay, 1994), which are mappings between two regular languages.[10] For our human convenience, we can visualize a finite-state relation as having an upper-side regular language and a lower-side regular language; and each string in one language is related to one or more strings in the other language.

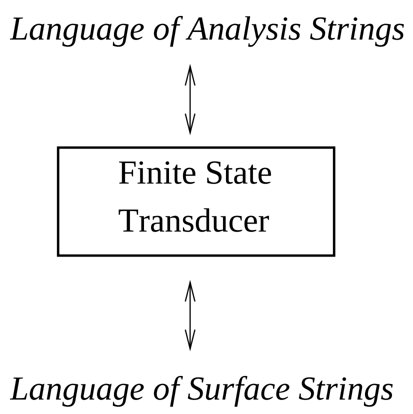

Figure 5: A morphological analyzer-generator can be implemented elegantly and efficiently as a finite-state transducer. By Xerox convention, the lower-side language consists of surface strings (words), and the upper-side language consists of strings representing analyses of the lower-side words. Such a transducer is a data structure rather than code, and the runtime code that applies such a transducer to input strings, in either direction, is completely language-independent.

By convention, the upper-side or analysis strings of an FST compiled from a **lexc** description consist of underlying morphemes (strings of phonemes and morphophonemes) and multicharacter-symbol TAGS like +Noun, +Verb, +Adj[jective], +Conj[unction], +VPref (verbal prefix), +Masc[culine], +Fem[inine], +Sing[ular], +Plur[al], etc. that identify the morphemes. These tags have multicharacter print names that are chosen and spelled according to the taste and needs of the developers, but they are manipulated internally exactly like the other typable characters. The related lower-side language consists of surface strings or at least more "surfacy" strings; they may still represent underlying strings requiring the application of alternation rules to map them into properly spelled surface strings.

Alternation rules to perform deletion, epenthesis, assimilation and metathesis are written in the **twolc** language (Karttunen and Beesley, 1992) and/or in a newer notation known as REPLACE

[10]Some authors use the term RATIONAL instead of REGULAR.

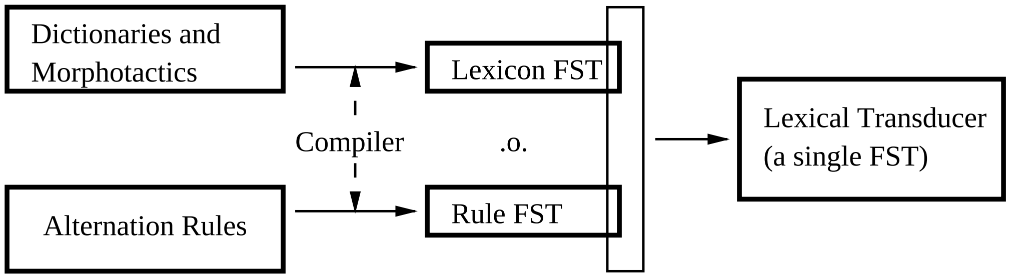
Dictionaries and Morphotactics
Alternation Rules
Compiler
Lexicon FST
.o.
Rule FST
Lexical Transducer (a single FST)

Figure 4: Creation of a Lexical Transducer. The .o. operator represents the composition operation.

RULES (Karttunen, 1995; Kempe and Karttunen, 1996). These rules also compile into finite-state transducers, and all the components of the grammar are combined into a single LEXICAL TRANSDUCER (Karttunen et al., 1992) using the finite-state composition algorithm, as shown in Figure 4. Such a lexical transducer then implements the Black Box morphological analyzer sketched above in Figure 1, mapping directly between surface strings and analysis strings as shown in Figure 5.

4.3 Arabic Finite-State Morphological Analysis

At Xerox, the treatment of Arabic starts with a **lexc** grammar where prefixes and suffixes concatenate to stems in the usual way, and where stems also are represented as a concatenation of a root and a pattern as shown in Figure 6.

The first step in the modification of such strings is to interdigitate the roots and patterns to form stems, but only on the lower side of the relation. The interdigitation is formalized in finite-state terms as intersection (Beesley, 1998b; Beesley, 1998a), but it in fact represents a special case of intersection that is performed much more efficiently by a finite-state algorithm called MERGE. The application of the merge algorithm to the lower-side of the relation is performed by the COMPILE-REPLACE algorithm (Beesley and Karttunen, 2000),[11] and the result is shown in Figure 7.

Once compile-replace has been performed on the lower side, the necessary alternation rules can be compiled and applied, via composition, in the usual way shown in Figure 4. Not surprisingly, to anyone who has studied Arabic, the rules controlling the realization of **w**, **y** and the hamza (the glottal stop) are particularly complicated. In the examples shown here, *katabat* is finished and can be displayed as كَتَبَت; the underlying final *y* radical of *banayat* disappears on the surface, leaving *banat* (بَنَت), and the underlying medial radical *w* of *qawula* disappears as well, leaving *qaala* (قالَ), with a lengthened vowel. The state of the string pairs, after composition of the alternation rules, is shown in Figure 8. Further composition of "relaxation" rules allowing the optional deletion of short vowels and other diacritics completes the picture. The final transducer will directly map from *katabat* (كَتَبَت) or *ktbt* (كتبت) or any partially-voweled variation of the spelling to the upper-side string [ktb&CaCaC]+Verb+FormI+Perf+Act +at+3P+Fem+Sg.[12] In the web demo, the various morphemes and tags in the analysis string are separated and reformatted in HTML for more perspicuous display to the user.

4.4 Advantages and Availability of Finite-State Implementations

Wherever it is possible, computing with finite-state machines is extremely attractive. By keeping within the finite-state domain, grammatical components can be defined, combined and modified using standard finite-state operations. Lexical transducers can be run forwards to generate or backwards to analyze, and they are computationally very efficient for natural-language prob-

[11]Before merge and compile-replace became available, a much less efficient algorithm produced the same result by automatically generating alternation rules, compiling them, and applying them to the lower side of the transducer.

[12]The surface string *ktbt* (كتبت) is ambiguous, and the analyzer returns all the possible solutions.

```
upper: [ktb&CaCaC]+Verb+FormI+Perf+Act+at+3P+Fem+Sg
lower: [ktb&CaCaC]                     at

upper: [bny&CaCaC]+Verb+FormI+Perf+Act+at+3P+Fem+Sg
lower: [bny&CaCaC]                     at

upper: [qwl&CaCuC]+Verb+FormI+Perf+Act+a+3P+Masc+Sg
lower: [qwl&CaCuC]                     a
```

Figure 6: Three pairs of strings in the lexicon FST compiled from the **lexc** description. These examples correspond to the words that will eventually be *katabat* (كَتَبَت), *banat* (بَنَت) and *qaala* (قَالَ). The upper-side analysis strings contain the roots *ktb*, *bny* and *qwl* respectively, the verbal form I perfect active pattern *CaCaC* or *CaCuC* and the third-person feminine singular suffix *at* or the third-person masculine singular suffix *a*. The square brackets are used for convenience to delimit the stem components from the rest of the word, and the ampersand serves here as just a delimiter between the root and the pattern, which are simply concatenated together. In the upper-side strings, the various morphemes are separate and are identified with multicharacter tags and bracketing conventions. The lower-side strings, still abstract here, will be mapped via finite-state algorithms and alternation rules into properly spelled surface strings.

```
upper: [ktb&CaCaC]+Verb+FormI+Perf+Act+at+3P+Fem+Sg
lower:  katab                          at

upper: [bny&CaCaC]+Verb+FormI+Perf+Act+at+3P+Fem+Sg
lower:  banay                          at

upper: [qwl&CaCuC]+Verb+FormI+Perf+Act+a+3P+Masc+Sg
lower:  qawul                          a
```

Figure 7: Pairs of strings from the lexicon FST after application of the compile-replace algorithm to the lower side. The lower-side strings, ignoring gaps or epsilons, are now *katabat*, which is essentially finished (كَتَبَت), and *banayat* and *qawula*, which involve weak radicals and await the application of alternation rules to map them into their final orthographical forms بَنَت and قَالَ, respectively. Note that the upper-side strings have not been modified.

```
upper: [ktb&CaCaC]+Verb+FormI+Perf+Act+at+3P+Fem+Sg
lower:  katabat

upper: [bny&CaCaC]+Verb+FormI+Perf+Act+at+3P+Fem+Sg
lower:  banat

upper: [qwl&CaCuC]+Verb+FormI+Perf+Act+a+3P+Masc+Sg
lower:  qaala
```

Figure 8: Pairs of strings from the lexicon FST after composition of the alternation rules on the lower side. The lower-side strings are here displayed contiguously.

lems. Xerox finite-state morphological analyzers, running on modern PC and workstations, typically analyze thousands of words per second.[13] The runtime code that applies lexical transducers to input strings is also completely language-independent. Thus the code that runs the Arabic morphological analyzer is exactly the same code that runs German, French, Spanish, Portuguese, Aymara, etc.

Xerox's implementation of finite-state theory has been used extensively in its own research and commercial work, and these software tools have been licensed to over 70 universities and non-commercial research centers. And Xerox is hardly alone in its enthusiasm for finite-state computing; several similar implementations of finite-state theory are now available, notably from the Summer Institute of Linguistics, AT&T and the University of Gronigen.[14]

5 Conclusion

5.1 Status

The current underlying lexicons include about 4930 roots, each one hand-encoded to indicate the subset of patterns with which it can combine. The pattern dictionary contains about 400 phonologically distinct entries, many of them ambiguous. In practice, the average root participates in about 18 morphotactically distinct stems, yielding approximately 90,000 stems based on roots and patterns. Some of these are phonologically indistinguishable on the lower side, resulting in approximately 70,000 root-pattern intersections to be performed just once at compile time.

Various combinations of prefixes and suffixes, concatenated to the intersected stems, and filtered by composition, yield over 72,000,000 abstract words. Sixty-six finite-state alternation rules map these abstract strings into fully-voweled orthographical strings, and additional orthographical relaxation rules are then applied to optionally delete short vowels and other diacritics, allowing the system to analyze unvoweled, partially voweled, and fully-voweled variant spellings of the 72,000,000 abstract words.

[13]Performance of finite-state morphological analyzers varies linearly with the length of words and with the amount of lexical ambiguity. Arabic will be slower than our European systems, in terms of input words analyzed per second, because of the unusually large number of solutions to be computed.

[14]`http://www.sil.org/computing/catalog/pc-kimmo.html`, `http://www.research.att.com/sw/tools/lextools/`, `http://odur.let.rug.nl/~vannoord/fsa/`, `http://www.pg.gda.pl/~jandac/fsa.html`

5.2 Plans

After several years of inactivity, a new project was begun in April 2001 to test, update, and redevelop this system for commercial use. This will involve modernization of the system to incorporate improvements in finite-state algorithms and techniques developed after 1996, extensive testing on real corpora, and augmentation of the underlying dictionary, especially with proper names. Testing will no doubt uncover other omissions and infelicities.

References

Evan L. Antworth. 1990. *PC-KIMMO: a two-level processor for morphological analysis*. Number 16 in Occasional publications in academic computing. Summer Institute of Linguistics, Dallas.

Kenneth R. Beesley and Lauri Karttunen. 2000. Finite-state non-concatenative morphotactics. In *COLING Finite-State Phonology Workshop Proceedings*, pages 1–12, Luxembourg, August 5-6. Association for Computational Linguistics.

Kenneth R. Beesley and Lauri Karttunen. 2001. *Finite-State Morphology: Xerox Tools and Techniques*. Cambridge University Press, Cambridge. Forthcoming.

Kenneth R. Beesley. 1990. Finite-state description of Arabic morphology. In *Proceedings of the Second Cambridge Conference on Bilingual Computing in Arabic and English*, September 5-7. No pagination.

Kenneth R. Beesley. 1996. Arabic finite-state morphological analysis and generation. In *COLING'96*, volume 1, pages 89–94, Copenhagen, August 5-9. Center for Sprogteknologi. The 16th International Conference on Computational Linguistics.

Kenneth R. Beesley. 1998a. Arabic morphology using only finite-state operations. In Michael Rosner, editor, *Computational Approaches to Semitic Languages: Proceedings of the Workshop*, pages 50–57, Montréal, Québec, August 16. Université de Montréal.

Kenneth R. Beesley. 1998b. Arabic stem morphotactics via finite-state intersection. Paper presented at the 12th Symposium on Arabic Linguistics, Arabic Linguistic Society, 6-7 March, 1998, Champaign, IL.

Kenneth R. Beesley. 1998c. Consonant spreading in Arabic stems. In *COLING-ACL'98*, volume 1, pages 117–123, Montréal, Québec, August 10-14. Université de Montréal. 36th Annual Meeting of the Association for Computational Linguistics, and 17th International Conference on Computational Linguistics.

Timothy A. Buckwalter. 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*, September 5-7. No pagination.

Zelig Harris. 1941. Linguistic structure of Hebrew. *Journal of the American Oriental Society*, 62:143–167.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Lauri Karttunen and Kenneth R. Beesley. 1992. Two-level rule compiler. Technical Report ISTL-92-2, Xerox Palo Alto Research Center, Palo Alto, CA, October.

Lauri Karttunen, Ronald M. Kaplan, and Annie Zaenen. 1992. Two-level morphology with composition. In *COLING'92*, pages 141–148, Nantes, France, August 23-28.

Lauri Karttunen. 1993. Finite-state lexicon compiler. Technical Report ISTL-NLTT-1993-04-02, Xerox Palo Alto Research Center, Palo Alto, CA, April.

Lauri Karttunen. 1995. The replace operator. In *ACL'95*, Cambridge, MA. cmp-lg/9504032.

Laura Kataja and Kimmo Koskenniemi. 1988. Finite-state description of Semitic morphology: A case study of Ancient Akkadian. In *COLING'88*, pages 313–315.

Martin Kay. 1987. Nonconcatenative finite-state morphology. In *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*, pages 2–10.

André Kempe and Lauri Karttunen. 1996. Parallel replacement in finite-state calculus. In *COLING'96*, Copenhagen, August 5–9. cmp-lg/9607007.

George Anton Kiraz. 2000. Multitiered nonlinear morphology using multitape finite automata: A case study on Syriac and Arabic. *Computational Linguistics*, 26(1):77–105.

Kimmo Koskenniemi. 1983. Two-level morphology: A general computational model for word-form recognition and production. Publication 11, University of Helsinki, Department of General Linguistics, Helsinki.

Alon Lavie, Alon Itai, Uzzi Ornan, and Mori Romon. 1988. On the applicability of two level morphology to the inflection of Hebrew verbs. In *ALLC III*, pages 246–260.

John J. McCarthy. 1981. A prosodic theory of nonconcatenative morphology. *Linguistic Inquiry*, 12(3):373–418.

John J. McCarthy. 1993. Template form in prosodic morphology. In Laurel Smith Stvan, Stephen Ryberg, Mari Broman Olsen, Talke Macfarland, Linda DiDesidero, Anne Bertram, and Larin Adams, editors, *Papers from the Third Annual Formal Linguistics Society of Midamerica Conference*, pages 187–218, Bloomington, Indiana. Indiana University Linguistics Club.

Hans Wehr. 1979. *A Dictionary of Modern Written Arabic*. Spoken Language Services, Inc., Ithaca, NY, 4 edition. Edited by J. Milton Cowan.