Arabic Morphology Using Only Finite-State Operations

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

Abstract

Finite-state morphology has been successful in the description and computational implementation of a wide variety of natural languages. However, the particular challenges of Arabic, and the limitations of some implementations of finite-state morphology, have led many researchers to believe that finite-state power was not sufficient to handle Arabic and other Semitic morphology. This paper illustrates how the morphotactics and the variation rules of Arabic have been described using only finite-state operations and how this approach has been implemented in a significant morphological analyzer/generator.

1 Introduction

In Arabic, as in other natural languages, the two challenges of morphological analysis are the description of 1) the morphotactics and 2) the variation rules. Morphotactics is the study of how morphemes combine together to make well-formed words. Variations are the discrepancies between the underlying or morphophonemic strings and their surface realization, which are either phonological or orthographical strings depending on the purpose of the grammar.

The key insight and claim of the finite-state approach to morphology (Karttunen, 1991; Karttunen et al., 1992; Karttunen, 1994) is that both morphotactics and variation grammars can be written as regular expressions, which are compiled and implemented on computers as finite-state automata. Such grammars are interesting theoretically because they are highly constrained; and in practical computational linguistics for natural languages, finite-state automata are fast, usually compact in size, bidirectional, combinable using all valid finite-state operations, and consultable using language-independent lookup code.

Finite-state approaches to morphology, including the readily available implementation known as Two-Level Morphology (Koskenniemi, 1983; Antworth, 1990), have been shown to work in significant projects for French, English, Spanish, Portuguese, Italian, Finnish, Turkish and a wide variety of other natural languages. But despite the high attractiveness of finite-state computing, many investigators have concluded that finite-state techniques are not adequate for describing Semitic root-and-pattern morphology. This paper will present the case that fully implemented finite-state morphology can be and has been used successfully for Arabic.

2 Regular Expressions

When writing a finite-state morphological grammar, linguists state morphotactic and variation rules in the metalanguage of regular expressions or in higher-level languages that are convenient shorthand notations for complex regular expressions.

2.1 Regular Expressions, Regular Relations, and Finite-State Transducers

A regular expression that contains an alphabet of one-level symbols defines a regular language and compiles into a finite-state machine (FSM) that accepts this regular language. A regular expression that contains an alphabet of paired symbols defines a regular relation (a relation between two regular languages) and compiles into a finite-state transducer (FST) that maps from every string of one language into strings of the other. If the necessary finite-state algorithms and compilers are available, components of the grammar, including various sublexicons and rules, can be compiled into separate transducers and then combined together using any operations that are mathematically valid.

The Xerox implementation of finite-state morphology includes a complete range of fundamental algorithms (concatenation, union, intersection, complementation, etc.) plus higher-level shorthand languages such as **lexc** (Karttunen, 1993), **twolc** (Karttunen and Beesley, 1992) and Replace Rules (Karttunen, 1995; Karttunen and Kempe, 1995; Karttunen, 1996).

2.2 Finite-State Operations

When defining morphotactics or variations via regular expressions, the linguist has access to all the operations that are mathematically valid on regular languages and relations. The following is a brief outline of regular expressions in the Xerox notation:

For each symbol s, the regular expression **s** denotes a regular language consisting of the single string "s". If **A** and **B** are regular languages, then the regular expressions in Figure 1 also denote regular languages. The cross-product of **A** and **B**, denoted **A .x. B**, relates each string in **A**, the upper language, to every string of **B**, the lower language, and vice versa. **A .x. B** thus denotes a regular relation. Where **u** and **l** are symbols, **u:l** is a notation equivalent to **u .x. l**. For formal reasons, relations are not quite as manipulable as simple languages; in particular, relations are closed under concatenation, union, and iteration, but not under intersection, subtraction or complementation.

Relations are closed under composition, a somewhat more difficult operation to conceptualize. Let **A**, **B** and **C** denote regular languages; let **X** denote a regular relation between an upper-side language **A** and a lower-side language **B**; and let **Y** denote a regular relation between the upper-side language **B** and a lower-side language **C**. Then the composition of **Y** under **X**, denoted **X .o. Y**, denotes a regular relation **Z** that maps directly between languages **A** and **C**; the intermediate language **B** disappears in the process of composition.

In defining natural-language morphotactics, union and concatenation are the basic operations required. Variation rules and long-distance-dependency filters are applied using composition. And we shall illustrate below how Arabic root-and-pattern interdigitation can be performed via intersection and composition.

3 Regular-Expression Grammars

3.1 Concatenative Morphotactics

Individual morphemes of natural language typically consist of one or more symbols, simply concatenated together. Thus the English morphemes *s*, *ed* and *ing* represent the concatenations **[s]**, **[e d]** and **[i n g]** respectively. Where **0** represents ϵ (the zero-length string), the set of regular verb suffixes of English can be represented as the union **[[s] | [e d] | [i n g] | 0]**. The set of verb stems taking these endings includes *wreck*, *walk*, and *talk*, which can also be formalized using concatenation and union: **[[w r e c k] | [w a l k] | [t a l k]]**. The union of endings can then be concatenated on the end of the union of verb stems to form a larger expression that denotes a language that looks like a subset of English verbs: **[[w r e c k] | [w a l k] | [t a l k]] [[s] | [e d] | [i n g] | 0]**.

If the linguist defines the symbols **+Verb**, **+3PS** (for "third personal singular"), **+Past**, **+PrPart** (for "present participle") and **+Bare**, the following expression denotes the relation that maps lower-side (surface) string like "talks" to the upper-side string "talk+Verb+3PS", and vice-versa. The preceding plus signs of these "tag" symbols are included simply to improve the human readability of the resulting strings; because the plus sign is normally a special Kleene Plus symbol in regular expressions, it is literalized in the examples below with a preceding percent sign.

[[w:w r:r e:e c:c k:k] | [w:w a:a l:l k:k] | [t:t a:a l:l k:k]] %+Verb:0 [[%+3PS:s] | [%+Past:e 0:d] | [%+PrPart:i 0:n 0:g] | %+Bare:0]

By convention in Xerox regular expressions denoting relations, the relation s:s can be written simply as s, as in the following:

[[w r e c k] | [w a l k] | [t a l k]] %+Verb:0 [[%+3PS:s] | [%+Past:e 0:d] | [%+PrPart:i 0:n 0:g] | %+Bare:0]

The English-verb fragment shown here was carefully chosen to be simple. However, there are three classes of phenomena for which union and concatenation, by themselves, are generally inadequate or at least very inconvenient for describing all and only the strings that appear in a natural language:

`0`	the zero-length string (often called ϵ)
`[A]`	bracketing; denotes the same language as A
`A B`	the concatenation of B after A
`A | B`	the union of A and B
`A & B`	the intersection of A and B
`(A)`	optionality, equivalent to [A | 0]
`A*`	Kleene star iteration, zero or more concatenations of A
`A+`	equivalent to [A A*], i.e. one or more concatenations of A
`A/B`	the regular language A, ignoring any instances of B
`?`	any symbol, i.e. the union of all single-symbol strings
`A - B`	language A, minus all strings in language B
`\B`	equivalent to [? - B], the union of all single-symbol strings minus the strings in B
`~B`	equivalent to [?* - B], the complement of B

Figure 1: Some Finite-State Notations

1. Discontiguous dependencies between morphemes in a word,
2. Non-concatenative morphotactic processes such as reduplication and Semitic interdigitation, and
3. Variations, typically assimilations, deletions and epentheses, that map between the abstract morphophonemic strings and their correct surface realizations.

We continue with illustrations of how such phenomena can be handled in a finite-state grammar.

3.2 Discontiguous Dependencies

To illustrate discontiguous dependencies, let us ignore for a second the internal structure of Arabic stems and postulate a set of noun stems including *kaatib* ("scribe"), *kitaab* ("book"), and *daaris* ("student"), formalized as [[k a a t i b] | [k i t a a b] | [d a a r i s]]. The set of possible case endings includes the definite set *u* (nominative), *a* (accusative) and *i* (genitive) as well as the indefinite set *un* (nominative), *an* (accusative) and *in* (genitive).[1] The most straightforward way to proceed to describe the morphotactics of a fragment of Arabic nouns is to concatenate the possible case endings onto the noun stems. Informative multicharacter symbols, **+Noun**, **+Def** (for "definite"), **+Indef** (for "indefinite") and **+Nom**, **+Acc** and **+Gen** are defined for the upper-side language.

(Prep%+:b 0:i) (Art%+:l) [[k a a t i b] | [k i t a a b] | [d a a r i s]] %+Noun:0 [[%+Def:0 [%+Nom:u | %+Acc:a | %+Gen:i]] | [%+Indef:0 [%+Nom:u 0:n | %+Acc:a 0:n | %+Gen:i 0:n]]]

Figure 2: An Overgenerating Lexicon Fragment for Arabic Nouns

[[k a a t i b] | [k i t a a b] | [d a a r i s]] %+Noun:0 [[%+Def:0 [%+Nom:u | %+Acc:a | %+Gen:i]] | [%+Indef:0 [%+Nom:u 0:n | %+Acc:a 0:n | %+Gen:i 0:n]]]

The resulting relation includes pairs of strings like

```
Upper:   kaatib+Noun+Indef+Acc
Lower:   kaatiban
```

Arabic nouns can also have a prefixed definite article, which we will represent as *l*, and prefixed prepositions like *bi*. Both are optional, and if *bi* and *l* cooccur, then *bi* must come first. The most straightforward way to allow these prefixes is to concatenate them on the front of the regular expression as in Figure 2. **Prep+** and **Art+** are interpreted as multicharacter symbols, and the parentheses indicate optionality, as shown in Figure 1.

[1]The spellings *un*, *an* and *in* roughly represent the pronunciation. Orthographically, the indefinite case endings consist of single symbols that are distinct from the single symbols used for definite endings.

However, Arabic words with a prefixed definite article *l* are in fact precluded from taking indefinite case suffixes. And words with a prefixed *bi* are compatible only with genitive case suffixes. The expression, as written in Figure 2, overgenerates, producing ill-formed string pairs like the following:

```
Upper:  Art+kaatib+Noun+Indef+Acc
Lower:  lkaatiban

Upper:  Prep+Art+kaatib+Noun+Def+Nom
Lower:  bilkaatibu
```

It is possible to rewrite the regular expression in various ways to eliminate the overgeneration, but this is tedious and dangerous, requiring the making and subsequent parallel maintenance of multiple copies of the noun stems. In practice, it is much more convenient to let the core lexicon overgenerate and subsequently filter out the bad strings, either at compile time or at runtime. The most straightforward method is to remove the ill-formed strings via composition of finite-state filters. Starting with the overgenerating grammar of Figure 2, one set of illegal strings to be eliminated contains both the **Art+** and the **+Indef** symbols on the upper side. We can characterize these illegal strings in a regular expression:

?* Art%+ ?* %+Indef ?*

Other illegal strings contain **Prep+** and then either **+Nom** or **+Acc** on the upper side.

?* Prep%+ ?* [%+Nom | %+Acc] ?*

The union of these two expressions characterizes the ill-formed upper-side strings to be eliminated. and the complement (notated ˜) of that union denotes the good strings.

˜[[?* Art%+ ?* %+Indef ?*] | [?* Prep%+ ?* [%+Nom | %+Acc] ?*]]

When this "filter" expression is composed on top of the overgenerating lexicon transducer, only the legal strings are matched, and the illegal strings are in fact eliminated from the result, which is again a finite-state transducer. There are several variations of this method that produce the same effect (Beesley, 1998d), with different penalties in the size of the resulting transducer or in the performance; but in the end the constraint of discontiguous dependencies is easily accomplished using finite-state techniques.

3.3 Non-Concatenative Morphotactics

While the morphotactic structure of many natural languages can be satisfactorily described using just concatenation, perhaps with subsequent filtering to constrain discontiguous dependencies, there are other languages with morphotactic phenomena that are notoriously non-concatenative, in particular reduplication, infixation and Semitic stem interdigitation (also known as intercalation). We will concentrate on Arabic here, arguing that roots, patterns and vocalizations can be formalized as regular expressions denoting regular languages, and that stems are formed by the intersection of these regular languages.

For illustration, let us assume, following the influential McCarthy (1981) analysis fairly closely, that Arabic stems consist of a root like **ktb**, a consonant-vowel template such as **CVCVC**, and a vocalization like **ui**. Where McCarthy proposed an extension of autosegmental theory, placing each of these morphemes on a separate tier, and proposing "association rules" to combine and linearize them into the stem *kutib*, we propose to formalize the same data in purely finite-state terms.

Let each root like **ktb** be formalized as [k t b]/?, i.e. as the language consisting of all strings containing **k**, **t** and **b**, in that order, ignoring the presence of any other symbols. (The notation [k t b]/? is equivalent to [?* k ?* t ?* b ?*].) Let **C** denote the union of all radical consonants, and let **V** denote [a | i | u], the union of all vowels. CV templates are defined as concatenations of **C**s and **V**s. Using the Xerox **xfst** interface, these definitions can be computed as

```
define ktb [k t b]/? ;
define drs [d r s]/? ;
define C [ k | t | b | d | r | s ] ;
define V [ a | i | u ] ;
define FormI    [ C V C V C ] ;
define FormII   [ C V C X V C ] ;
define formIII  [ C V V C V C ] ;
```

Vocalizations are also defined as regular expressions denoting regular languages, e.g. Perfect Active as [a*]/\V, the set of all strings containing zero or more *a*s, ignoring all other symbols except vowels. Other vocalizations are defined similarly:

```
define PerfActive [a*]/\V ;
define PerfPassive [u* i]/\V ;
define ImperfPassive [u a*]/\V ;
```

Given the definitions above, **xfst** will evaluate the expressions on the left below, indicating the intersection of a root, a pattern and a vocalization, and return a language consisting of the single string on the right, an interdigitated but still morphophonemic stem (Beesley, 1998a).

```
[ktb & FormI   & PerfActive ]  ⇒ katab
[ktb & FormIII & PerfPassive ] ⇒ kuutib
[drs & FormII  & PerfActive ]  ⇒ darXas
```

The X in the Form II pattern indicates the gemination (or lengthening) of the previous consonant, and its realization is controlled by variation rules. Consonant spreading, as in Form IX and Form XII, and biliteral roots also use the morphophonemic X symbol (Beesley, 1998c). Form I vocalizations are in fact idiosyncratic for each root, and those for the Imperfect Active are more troublesome, but the same kind of formalism applies.[2] If patterns are allowed to contain non-radical consonants, as in the analyses of Harris (1941) and Hudson (1986), then the definitions must be complicated slightly to prevent radicals from intersecting with the non-radical consonants (Beesley, 1998b). For a different formalization of this and other models proposed by McCarthy, but using techniques that go beyond finite-state power, see Kiraz (1996).

3.4 Defining Variation Rules

When underlying morphemes are concatenated and intersected together, the resulting strings are often still very abstract or morphophonemic; there may be many phonological or orthographical variations between these morphophonemic strings and their ultimate surface pronunciation or spelling. For example, English nouns usually pluralize by taking an *s* suffix, as in *book/books*, but words like *fly* pluralize as *flies* rather than **flys*. The variation between underlying *y* and the surface *ie* can be defined in terms of two-level rules or Replace Rules, which partially mimic traditional rewrite rules in their superficial syntax (Chomsky and Halle, 1968). Johnson (1972) demonstrated that rewrite-rules, as used by linguists, had only finite-state power and could be implemented as finite-state transducers; this important result, unfortunately overlooked at the time and later rediscovered by Kaplan and Kay (1981) (see also Kaplan and Kay (1994)) is a key mathematical foundation for finite-state morphology and phonology.

The variation rules required for Arabic were relatively difficult to write, but they are not different in kind or power from the rules required for other languages. The most difficult challenges involve the so-called weak roots, those containing a w (و), y (ي) or hamza (glottal stop) as one of the radicals.

Via concatenation and intersection, the lexicon produces morphophonemic strings like **katab+a**, the Form I perfect active of **ktb**, with a masculine singular **+a** suffix; similarly for **daras+a**, based on **drs**. These particular strings are very surfacy already, being realized in their fully-voweled form as *kataba*, rendered as (كَتَبَ), and *darasa*, rendered as (دَرَسَ). When trivial "relaxation" rules are composed on the bottom of the lexicon, allowing optional deletion of the short vowels, the system is also able to analyze the surface forms *ktb* (كتب) and *drs* (درس) and all the other partially voweled variations.

With weak roots, however, such as the finally weak **bny**, the dictionary generates parallel morphophonemic forms like **banay+a**, but the surface form is properly spelled with a y-like 'alif maqṣuura, بنى, rather than with a normal **y** with two dots (بني is not a possible spelling for underlying **banay+a**). This or-

[2] The Form I perfect active stem vowel for **ktb** happens to be /a/, so the general PerfectActive vocalization [a*]/\V works in this case; other roots will require [a i]/\V or [a u]/\V. For the Imperfect Passive, the vocalization is [u a*]/\V for all forms. For the Imperfect Active, the least attractive case for vowel abstraction, the Form I voweling is [a*]/\V, [a i]/\V or [a u]/\V, depending on the root; the Form II through IV voweling is [u a* i]/\V; the Form V and VI voweling is [a*]/\V; and the remaining forms VII to XV use [a* i]/\V. If such generalization of vocalization appears tenuous, the alternative is simply to keep the vowels in the patterns, resulting in a two-way intersection of roots and patterns (Harris, 1941; Kataja and Koskenniemi, 1988).

thographical change reflects the fact that the word is pronounced /banaː/ rather than /banaja/. The perfect passive **buniy+a**, however, is still spelled as *bny* (بني), reflecting a pronunciation of /bunija/, although in Egyptian orthographical practice the dots are usually dropped here as well, yielding بنى again. With the feminine ending, **banay+at**, the underlying y disappears completely, both phonologically and orthographically, yielding surface *bnt* (بنت).

With a medially-weak root like **qwl**, the morphophonemic Form I perfect active **qawul+a** gets realized as *qAl* (قال), reflecting the pronunciation /qaːla/. When the suffix begins with a consonant, as in **qawul+ta**, the surface spelling is *qlt*, reflecting the pronunciation /qulta/. An initially weak example like **ta+wlid+u**, based on root **wld**, yields تلد, with the deletion of the initial radical **w**, while **tu+wlad+u**, with an initial **tu+** prefix, yields تولد with the **w** intact. Similarly for root **w'd**, but with hamza complications: **ya+w'id+u** yields يئد while **yu+w'ad+u** yields يوأد.

The rule writer must also handle a number of assimilations, as in the Form VIII of root **ðkr**, underlying ðtakar+a, which is pronounced /ʔiddakara/ and written accordingly, including diacritics for clarity, as اِدَّكَرَ. Similary, for roots with an initial pharyngealized ṣaad (ص) or ḍaad (ض) radical, such as **ḍrb**, the underlying Form VIII is **ḍtarab+a**, emerging with the infixed Form VIII **t** assimilating to its pharyngealized version ṭ in اِضْطَرَبَ. None of these phenomena is phonologically surprising; local assimilations and contextual instabilities in semiconsonants like /w/ and /y/ are garden-variety variations, elegantly handled with finite-state variation rules.

4 Practical Applications

4.1 History of Computing Semitic Stems via Intersection

Classic Two-Level (Koskenniemi, 1983; Karttunen, 1983; Antworth, 1990) and finite-state lexicons (Karttunen, 1993) build underlying strings via concatenation only, but this limitation is not characteristic of the overall theory but only of the computational implementations. Kataja and Koskenniemi (1988) were apparently the first to understand that concatenating languages were just a special case; they showed that by generalizing lexicography to allow regular expressions, Semitic (specifically Akkadian) roots and patterns could denote regular languages, and that stems could be computed as the intersection of these regular languages.[3]

This principle was borrowed in the ALPNET prototype analyzer for Arabic morphology (Beesley, 1989; Beesley, 1991); but it used an implementation of Two-Level Morphology enhanced with a "detouring" mechanism that simulated the intersection of roots and patterns at runtime. This prototype grew into a large commercial system in 1989 and 1990 (Beesley et al., 1989; Beesley, 1990). In 1989, Lauri Karttunen (personal communication) also proposed and demonstrated in an Interlisp script the intersection of roots, patterns and vocalizations as an alternative to the finite-state solution of (Kay, 1987), which used a four-tape finite-state transducer transducer.

4.2 Current Xerox System

The current Xerox morphological analyzer for Arabic is based on dictionaries licensed from ALPNET, but the rules and organization of the system have been extensively rewritten.

4.2.1 System Components

The Arabic morphological analyzer starts out as a dictionary database containing entries for prefixes, suffixes, roots and patterns of Arabic. The database also includes morphotactic codings. Perl scripts extract the pertinent information from this database, reformatting it as **lexc** files, which are then compiled into a finite-state transducer that we label the "core" lexicon transducer. On top of the core FST, filters are composed to remove the strings that are ill-formed because of discontiguous dependencies. Finite-state rules that intersect roots and patterns are compiled into transducers and composed on the bottom of the core, leaving

[3]Kataja (personal communication) wrote comparative two-level grammars of the Neo-Babylonian and Neo-Assyrian dialects of Akkadian. The source dictionaries contained separate sublexicons for roots and patterns; these were intersected via awk scripts into Koskenniemi's TwoL format, which was then compiled.

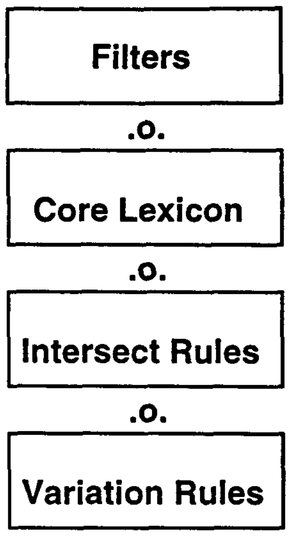

Figure 3: Constructing the Common FST

linearized lexical strings for the variation rules (also compiled into FSTs) to apply to, as shown in Figure 3. The result of the composition is a single "common" FST, with slightly enhanced fully-voweled strings in the lower language.

For generation purposes, where the user probably wants to see only formally correct fully-voweled strings, the bottom level is trivially cleaned up by yet another layer of composed rules. For recognition purposes, the rules applied to the bottom side include

```
[ a | i | u | o | ~ ] (->) 0 ;
```

which optionally maps the fatȟa (a), kasra (i), ḍamma (u), sukuun (o) and shadda (~) to the empty string. The resulting "analysis" transducer recognizes fully-voweled, partially voweled, and the usual unvoweled spellings. Where diacritics are present in the input, the output is correspondingly less ambiguous.

4.2.2 System Status

The current dictionaries contain 4930 roots, each one hand-coded to indicate the subset of patterns with which it legally combines (Buckwalter, 1990). Various combinations of prefixes and suffixes, concatenated to the intersected stems, and filtered by composition, yield over 72,000,000 abstract, fully-voweled words. Sixty-six finite-state variation rules map these abstract strings into fully-voweled orthographical strings, and additional rules are then applied to optionally delete short vowels and other diacritics, allowing the system to analyze unvoweled, partially voweled, and fully-voweled orthographical variants of the 72,000,000 abstract words. New entries are added easily to the original lexical database.

A full-scale version of the current system is available for testing on the Internet at http://www.xrce.xerox.com/research/mltt/arabic. A Java interface renders Arabic words in traditional Arabic script, both for input and output.

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, Tim Buckwalter, and Stuart N. Newton. 1989. Two-level finite-state analysis of Arabic morphology. In *Proceedings of the Seminar on Bilingual Computing in Arabic and English*, Cambridge, England, September 6-7. No pagination.

Kenneth R. Beesley. 1989. Computer analysis of Arabic morphology: A two-level approach with detours. In *Third Annual Symposium on Arabic Linguistics*, Salt Lake City, March 3-4. University of Utah. Published as Beesley, 1991.

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. 1991. Computer analysis of Arabic morphology: A two-level approach with detours. In Bernard Comrie and Mushira Eid, editors, *Perspectives on Arabic Linguistics III: Papers from the Third Annual Symposium on Arabic Linguistics*, pages 155–172. John Benjamins, Amsterdam. Read originally at the Third Annual Symposium on Arabic Linguistics, University of Utah, Salt Lake City, Utah, 3-4 March 1989.

Kenneth R. Beesley. 1998a. Arabic morphological analysis on the Internet. In *ICEMCO-98*, Cambridge, April 17-18. Centre for Middle Eastern Studies. Proceedings of the 6th International Conference and Exhibition on Multilingual Computing. Paper number 3.1.1; no pagination.

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'98*.

Kenneth R. Beesley. 1998d. Constraining separated morphotactic dependencies in finite-state grammars. In *FSMNLP-98*, Bilkent. Bilkent University.

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.

Noam Chomsky and Morris Halle. 1968. *The Sound Pattern of English*. Harper and Row, New York.

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

Grover Hudson. 1986. Arabic root and pattern morphology without tiers. *Journal of Linguistics*, 22:85–122. Reply to McCarthy:1981.

C. Douglas Johnson. 1972. *Formal Aspects of Phonological Description*. Mouton, The Hague.

Ronald M. Kaplan and Martin Kay. 1981. Phonological rules and finite-state transducers. In *Linguistic Society of America Meeting Handbook. Fifty-Sixth Annual Meeting*, New York. December 27-30. Abstract.

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 and André Kempe. 1995. The parallel replacement operation in finite-state calculus. Technical Report MLTT-021, Rank Xerox Research Centre, Grenoble, France, December. Available at http://www.xrce.xerox.com/publis/mltt/mltttech.html.

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. 1983. KIMMO: a general morphological processor. In Mary Dalrymple, Edit Doron, John Goggin, Beverley Goodman, and John McCarthy, editors, *Texas Linguistic Forum*, number 22, pages 165–186. Department of Linguistics, The University of Texas at Austin, Austin, TX.

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

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. 1994. Constructing lexical transducers. In *COLING'94*, Kyoto, Japan.

Lauri Karttunen. 1995. The replace operator. In *Proceedings of the 33rd Annual Meeting of the ACL*, Cambridge, MA. Available at http://www.xrce.xerox.com/publis/mltt/mltttech.html.

Lauri Karttunen. 1996. Directed replacement. In *Proceedings of the 34rd Annual Meeting of the ACL*, Santa Cruz, CA.

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.

George Anton Kiraz. 1996. Computing prosodic morphology. In *COLING'96*, pages 664–669.

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.

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