Multitiered Nonlinear Morphology Using Multitape Finite Automata: A Case Study on Syriac and Arabic

George Anton Kiraz*
Bell Laboratories, Lucent Technologies

> He who corrects the alphabetical ordering [of this *Lexicon*] does not take into account the letters which are added, or those that change, here and there, due to inflexion at the beginning of the lexis [i.e., word], but goes back to the base form of the word.
>
> – Abu Ḥasan Bar Bahlūl (fl. ca. 963)
> *Syriac Lexicon*, Duval's edition (Paris, 1888)

This paper presents a computational model for nonlinear morphology with illustrations from Syriac and Arabic. The model is a multitiered one in that it allows for multiple lexical representations corresponding to the multiple tiers of autosegmental phonology. The model consists of three main components: (i) a lexicon, which is made of sublexica, with each sublexicon representing lexical material from a specific tier, (ii) a rewrite rules component that maps multiple lexical representations into one surface form and vice versa, and (iii) a morphotactic component that employs regular grammars. The system is finite-state in that lexica and rules can be represented by multitape finite-state machines.

1. Introduction

This paper is concerned with presenting a general finite-state framework for computing complex nonlinear (i.e., nonconcatenative) morphophonological descriptions. The framework subsumes previous models in that it is not only capable of handling the complex nonlinear morphological phenomena we are about to describe, but also the usual linear ones found in many languages. The framework is a multitiered model that encompasses linear and nonlinear morphology.

The elegance of the proposed framework lies in the fact that it is capable of handling sophisticated linguistic descriptions, such as those of autosegmental phonology, in a computationally tractable way. The model consists of three components. The first is a multitier lexicon that consists of several sublexica, each describing a lexical tier as in autosegmental phonology. The second component is a bidirectional rewrite rules system that maps, inter alia, the multitier lexical representations to a linearized surface form and vice versa. The third component provides morphotactic constraints. The proposed framework has the computational property that its lexical and rule formalisms are compiled algorithmically into finite-state machinery using operators under which finite-state machines are closed.

* 700 Mountain Ave., Murray Hill, NJ 07974. E-mail: gkiraz@research.bell-labs.com

The rest of this section discusses the importance of nonlinear morphology and outlines the research objectives of this work.

1.1 Nonlinear Morphology

Early generative morphophonological theory was mainly based on the linear segmental approach of *The Sound Pattern of English* (Chomsky and Halle 1968). In the mid seventies, however, linguists had departed from this linear framework to a nonlinear one. Goldsmith (1976), working on the phonology of African tone languages, proposed autosegmental phonology with multitiered representations. Goldsmith made use of two tiers to describe tone languages: one to represent sequences of vowels and consonants, and another to describe tone segments. McCarthy (1979) applied autosegmental phonology to Semitic root-and-pattern morphology resulting in what is now known as the theory of nonconcatenative (or nonlinear) morphology, as opposed to concatenative morphology. McCarthy's findings have become ubiquitous. In fact, "every aspect of the theory of morphology and morphophonology," remarks Spencer (1991, 134), "has had to be reappraised in one way or another in the wake of [McCarthy's] analysis of Semitic and other languages."

Spencer's statement could well apply to the theory of computational morphology. Two-level morphology (Koskenniemi 1983), as well as its predecessor in the work of Kay and Kaplan (1983), is also deeply rooted in the linear concatenative tradition. Indeed, "if Koskenniemi had been interested in Arabic or Warlpiri rather than Finnish," notes Sproat (1992, 206), "his system might have taken on a rather different character from the start." It would prove difficult, if not impossible, to implement Semitic languages using linguistically motivated theoretical models such as those of McCarthy and others in the field with traditional two-level morphology.

Kay (1987) was the first computational linguist to make use of McCarthy's findings. He proposed that a four-tape finite-state machine, as opposed to the traditional two-tape machines of two-level morphology, be used to describe the autonomous morphemes of Arabic. Kay devised a system for manipulating the multitape machine, albeit using an ad hoc procedure to control the movements of the machine's head(s). We shall build upon Kay's work by providing higher-level lexical and rule formalisms, and algorithms for compiling the formalisms into multitape machines, eliminating the need for the ad hoc procedure that controls head movements. We shall revisit Kay's approach in Section 6.1.

Previous work to implement Semitic languages, namely, Akkadian (Kataja and Koskenniemi 1988), Arabic (Beesley, Buckwalter, and Newton 1989; Beesley 1990, 1991, 1996, 1998a, 1998b, 1998c, forthcoming) and Hebrew (Lavie, Itai, and Ornan 1990), employed traditional two-level morphology with some augmentation to handle the nonlinearity of stems, but did not make any use of the then-available theory of nonconcatenative morphology. The challenge here lies in the fact that two-level morphology assumes the lexical representation of a surface form to be the concatenation of the corresponding lexical morphemes in question. To resolve the problem, these authors (with the exception of Beesley's work from 1996 on) provided for a simultaneous search of various root and affix lexica, the result of which served as the lexical tape of the two-level system. We shall revisit these approaches in Sections 6.2 and 6.3.

Other publications dealing with Semitic computational morphology are confined to proposals for compiling autosegmental descriptions into automata (Kornai 1991; Wiebe 1992; Bird and Ellison 1992). They revolve around encoding autosegmental representations (by various encoding mechanisms) and providing ways for compiling such encodings into finite machines. None provide for lexicon and rule formalisms that can be compiled into their respective encodings or directly into automata. No

Semitic language, to the best of the author's knowledge, has been implemented with these proposals. We shall revisit these approaches in Section 6.4.

1.2 Research Objectives

The purpose of this work is to provide a theoretical computational framework under which nonlinear morphology can be handled in a linguistically and computationally motivated manner with the following objectives in mind:

1. The framework is to present a general multitiered computational morphology model that allows for both linear and nonlinear morphophonological descriptions.
2. The formalism of the framework is to handle various linguistic theories and models including McCarthy's initial findings as well as later models for Hebrew (Bat-El 1989), moraic theory (McCarthy and Prince 1990a, 1995), the affixational approach to handling templates in Arabic (McCarthy 1993), and others. That is, a flexible formalism that leaves the grammar writer with ample room to choose the appropriate linguistic theory for an application.
3. Multitiered lexica and grammars written in the formalism are to be compiled into finite-state machines (multitape automata in this case). The multitape machines are created by a compiler that employs a finite-state engine with an algebraic interface to *n*-way regular expressions.
4. The multitape machines are to be as close as possible in spirit to two-level morphology in that surface forms map to lexical morphemes. Our lexical level is a multitiered representation.

This paper provides an overall description of the theoretical framework, compilation algorithms, and illustrations. Additionally, it discusses other related topics crucial to developing Semitic grammars.

Results that emerged earlier from this work appear elsewhere (Kiraz 1994b, 1996, 1997a, 1997b, 1997c, in press), but have been thoroughly enhanced and reworked since. New contributions include enhancing the theoretical framework (Section 3), compiling lexica and rules into multitape finite-state machines (Section 4), and evaluating the current model with respect to previous ones (Section 6).

2. Problems in Semitic Morphophonology

The challenges in implementing Semitic morphophonological grammars are numerous. Here, we shall concentrate only on nonlinearity that poses computational difficulties to traditional finite-state models.

2.1 The Nonlinear Stem

The main characteristic of Semitic derivational morphology is that of the "root and pattern." The "root" represents a morphemic abstraction, usually consisting of consonants, e.g., {ktb} 'notion of writing'. Stems are derived from the root by the superimposition of patterns. A "pattern" (or "template") is a sequence of segments containing Cs that represent the root consonants, e.g., $C_1aC_2C_2aC_3$ and $maC_1C_2aC_3$ (the indices refer to root consonants). Root consonants are slotted in, in place of the Cs, to derive stems, e.g., Arabic /kattab/ 'wrote' and /maktab/ 'office' from the root {ktb} and the above two patterns, respectively.

Table 1
Syriac verbal stems with the root {ktb}. The data provides stems in underlying morphological forms; surface forms are parenthesized. The passive is marked by the reflexive prefix {ʔet}.

Measure	Active	Passive ({ʔet}+)
Pʕal (1)	katab (ktab)	kateb (ʔetkteb)
PaʕʕeI (2)	katteb	kattab
ʔafʕeI (3)	ʔakateb (ʔakteb)	ʔakatab (ʔettaktab)

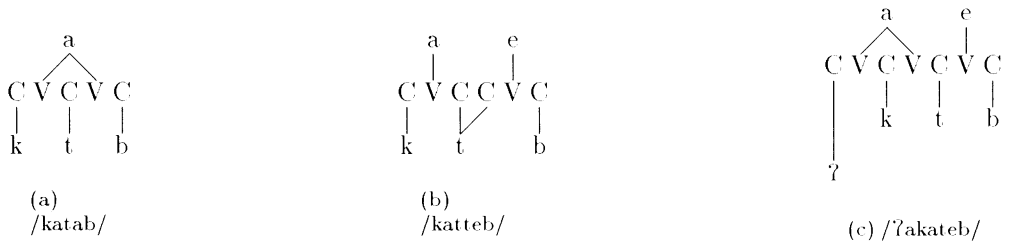

Figure 1
Autosegmental representations of Syriac Measures 1-3. Each morpheme sits on its own autonomous tier.

2.2 Various Linguistic Models

There are various linguistic models for representing patterns. In traditional accounts, the "vocalism" (i.e., vocalic elements) is collapsed with the pattern (Harris, 1941), e.g., maC_1C_2aC_3 above. Since the late 1970s, however, more sophisticated descriptions have emerged, most notably by McCarthy (1981, 1986, 1993) and McCarthy and Prince (1990a, 1990b, 1995).

Consider the Syriac verbal data in Table 1 containing verbs classified according to various measures.[1] Glancing over the table horizontally, one notes the same pattern of consonants and vowels per measure; the only difference is that the vowels are invariably {ae} in active stems, but {aa} in passive stems (apart from Measure 1 whose vocalism is idiosyncratic, a general Semitic phenomenon). Surface forms that result after phonological rules are applied appear in parentheses. The vowel after [k] in the Measure 1 and 3 forms, for example, is deleted because it is a short vowel in an open syllable, a general phenomenon of Aramaic (of which Syriac is a dialect). Further, [ʔ] in the passive of Measure 3 is assimilated into the preceding [t] of the reflexive prefix. Both rules are employed in */ʔetʔakatab/ → /ʔettaktab/.

Under McCarthy's autosegmental analysis, a stem is represented by three autonomous morphemes: a consonantal root, a vocalism, and a pattern that consists of Cs and Vs. For example, the analysis of /katteb/ (Measure 2) produces three morphemes: the root {ktb} 'notion of writing', the vocalism {ae} 'PERF ACT' and the pattern {CVCCVC} 'Measure 2'. Some stems include affix morphemes, e.g. the prefix {ʔa} of Measure 3; these sit on their own tier. The segments are linked together by association lines as in Figure 1. Note the spreading of [a] in /katab/ and the gemination of [t] in /katteb/.

1 The term "measure" is employed by native grammarians to denote a specific pattern.

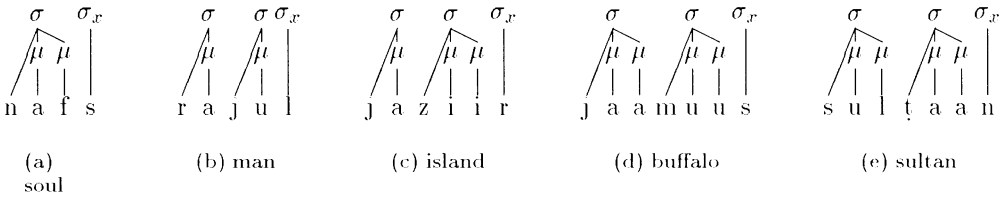

Figure 2
Moraic analysis of the Arabic nominals. Glosses appear under each autosegmental structure. Consonants and vowels, although shown on the same line, belong to different tiers.

While McCarthy's initial model is the most visible in the computational linguistics literature, it is by no means the only one. McCarthy and Prince (1990b) argue for representing the pattern morpheme in terms of the authentic units of prosody. Consider the Arabic nominal stems and their moraic analysis in Figure 2. Here, a pattern is represented by a sequence of syllables, σs. Association takes the following form: a syllable node σ takes a consonant, and its mora μ takes a vowel. In bimoraic syllables, the second μ may be associated with either a consonant or a vowel. At the right edge, all stems end in an extrametrical consonant, denoted by σ_x.

Other linguistic models have been described by Hammond (1988), Bat-El (1989), and McCarthy (1993).

2.3 Orthographic Issues

In addition to the linguistic peculiarities of Semitic, the writing system adds further complications. The vast majority of Semitic texts (apart from Ethiopic) are orthographically underspecified. Short vowels are absent from the orthography. (Imagine the previous sentence had been written "shrt vwls ʔr ʔbsnt frm th ʔrthgrph!") We shall deal with vocalization issues in Section 5.4.

3. The Theoretical Framework: A Multitiered Model

The model presented here assumes two levels of linguistic description in recognition and synthesis. The lexical level employs *multiple* representations (e.g., for patterns, roots, and vocalisms), while the surface level employs only *one* representation. The upper bound on the number of lexical representations is not only language-specific, but also grammar-specific.

The proposed model is divided into three components: (i) a lexicon component, which consists of multiple sublexica, each representing entries for a particular lexical representation or tier, (ii) a rewrite rules component, which maps the multiple lexical representations to a surface representation, and (iii) a morphotactic component, which enforces morphotactic constraints.

Throughout the following discussion, a tuple of strings represents a surface-to-lexical mapping, where the first element of the tuple represents the surface form and the remaining elements represent lexical forms. For instance, the tuple of strings that represents the mapping of /katab/ to its lexical forms is ⟨katab,cvcvc,ktb,a⟩.[2] The elements are: surface, pattern, root, vocalism.

2 Capital-initial strings will be used shortly to denote variables. For this reason, we represent the pattern using small letters.

3.1 The Lexicon Component

Here, the lexicon consists of multiple sublexica, each sublexicon containing entries for one particular lexical representation (or tier in the autosegmental analysis). Since an n-tuple contains $n \Leftrightarrow 1$ lexical elements (the first element is the surface representation), the lexicon component consists of $n \Leftrightarrow 1$ sublexica. A Syriac lexicon for the data in Table 1 requires a pattern sublexicon, a root sublexicon, and a vocalism sublexicon. Other affixes that do not conform to the root-and-pattern nature of Semitic morphology (e.g., the reflexive prefix {ʔet}) can either be given their own sublexicon or placed in one of the three sublexica. Since pattern segments are the closest—in terms of number—to surface segments, such morphemes are represented in the pattern sublexicon by convention.

As a way of illustration, the first sublexicon for the Syriac data in Table 1 contains the following entries: {ʔet} (representing the reflexive prefix) and {cvcvc} (for the verbal pattern). Here, we have chosen to derive all verbs from this pattern in a way reminiscent of McCarthy (1993) rather than entering separate patterns for each morpheme. The second sublexicon maintains roots, e.g., {ktb} 'notion of writing', {pnq} 'notion of delight', and {qrb} 'notion of approaching'. The third sublexicon maintains vocalisms: {ae} for active stems and {a} (with spreading) for passive ones.

3.2 The Rewrite Rules Component

The rewrite rules component maps the multiple lexical representations to a surface representation and vice versa. It also provides for phonological, orthographic, and other rules. The current model adopts the formalism presented by Ruessink (1989) and Pulman and Hepple (1993) with additional extensions to handle multiple lexical forms. Below, the top line represents the lexical tiers and the bottom line represents the corresponding surface form:

$$\begin{array}{ccccccc} \text{LLC} & \Leftrightarrow & \textsc{Lex} & \Leftrightarrow & \text{RLC} & \{\Rightarrow, \Leftrightarrow\} \\ \text{LSC} & \Leftrightarrow & \textsc{Surf} & \Leftrightarrow & \text{RSC} & \end{array}$$

LLC denotes the left lexical context, LEX denotes the lexical form, and RLC denotes the right lexical context. LSC, SURF and RSC are the surface counterparts. The context denoted by "*" represents Kleene star as applied to the grammar alphabet (i.e., matching anything). When all four contexts are "*", they are omitted from rules; i.e., the formalism becomes:

$$\begin{array}{cc} \textsc{Lex} & \{\Rightarrow, \Leftrightarrow\} \\ \textsc{Surf} & \end{array}$$

Further, capital-initial expressions are variables over predefined finite sets of symbols.

The operator $\Rightarrow$ is the optional operator. It states that LEX *may* surface as SURF in the given context, but may surface otherwise if sanctioned by another rule. The operator $\Leftrightarrow$ adds obligatory constraints: when LEX appears in the given context, then the surface description *must* satisfy SURF. A lexical string maps to a surface string if and only if they can be partitioned into pairs of lexical-surface subsequences, where (i) each pair is licensed by a $\Rightarrow$ rule, and (ii) no sequence of zero or more adjacent pairs violates a $\Leftrightarrow$ rule. The interpretation of the latter condition is based on Grimley-Evans, Kiraz, and Pulman (1996). (See Kiraz [in press] for the historical development of the formalism.)

Several extensions are introduced into the formalism to handle multitiered representations. Expressions on the upper lexical side (LLC, LEX, and RLC) are tuples of strings of the form $\langle x_1, x_2, \ldots, x_{n-1} \rangle$. The ith element in the tuple refers to symbols in the ith sublexicon of the lexical component. When a lexical expression makes use of

$\langle$c,X,$\varepsilon\rangle$ $\Rightarrow$
R1: X
where X is a consonant.
$\langle v, \varepsilon, X\rangle$ $\Rightarrow$
R2: X
where X is a vowel.
* $\Leftrightarrow$ $\langle$v, ε, X$\rangle$ $\Leftrightarrow$ $\langle$cv, *, $^*\rangle$ $\Leftrightarrow$
R3: * $\Leftrightarrow$ ε $\Leftrightarrow$ *
where *X* is a vowel.

Figure 3
Rules for the derivation of Syriac /ktab/. R1 and R2 sanction root consonants and vowels, respectively, while R3 handles vowel deletion.

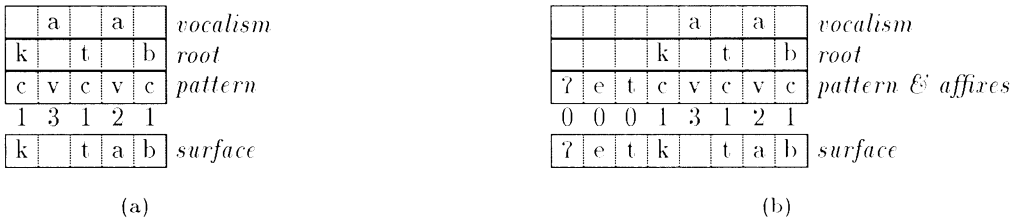

Figure 4
Lexical-surface analysis of Syriac /ktab/ and /ʔetktab/. Vocalic spreading is ignored in this example (see Section 5.1).

only the first sublexicon, the angle brackets can be ignored. Hence, the LEX expression $\langle x, \epsilon, \ldots, \epsilon\rangle$ and x are equivalent; in lexical contexts, $\langle x, *, \ldots, *\rangle$ and x are equivalent. Additionally, the symbol "*" now denotes Kleene star as applied to the alphabet of the respective tier.

The formalism is illustrated in Figure 3. The rules derive Syriac /ktab/ (underlying */katab/) from the pattern morpheme {cvcvc} 'verbal Measure 1', the root morpheme {ktb} 'notion of writing', and the vocalism morpheme {aa} 'PERF ACT' (ignoring spreading for the moment). Rule R1 sanctions root consonants by mapping a [c] from the first (pattern) sublexicon, a consonant [X] from the second (root) sublexicon, and no symbol from the third (vocalism) sublexicon to surface [X]. Rule R2 sanctions vowels in a similar manner. The obligatory rule R3 deletes the first vowel of */katab/ in the given context. The mapping is illustrated in Figure 4(a). The numbers between the surface and lexical expressions indicate the rules in Figure 3 that sanction the shown subsequences. Empty slots represent the empty string ϵ.

As stated above, morphemes that do not conform to the root-and-pattern nature of Semitic (e.g., prefixes, suffixes, particles) are given in the first sublexicon. The identity rule:

X $\Rightarrow$
R0 X
where X $\notin$ {c, v}

maps such morphemes to the surface. The rule basically states that any symbol not

Table 2
Syriac circumfixes of the imperfect verb.

Number	Gender	Circumfix
Sing.	masc.	ne-
Sing.	fem.	te-
Pl.	masc.	ne-ūn
Pl.	fem.	te-ån

in { c,v } from the first sublexicon may optionally surface. Figure 4(b) illustrates the analysis of /ʔetkatab/ from the morphemes given earlier.

3.3 The Morphotactics Component

Semitic morphotactics is divided into two categories: **Templatic morphotactics** occurs when the pattern, root, vocalism, and possibly other morphemes, join together in a nonlinear manner to form a stem. **Non-templatic morphotactics** takes place when the stem is combined with other morphemes to form larger morphological or syntactic units. The latter is divided in turn into two types: **linear** nontemplatic morphotactics, which makes use of simple prefixation and suffixation, and **nonlinear** nontemplatic morphotactics, which makes use of circumfixation.

Templatic morphotactics is handled implicitly by the rewrite rules component. For example, the rules in Figure 3 implicitly dictate the manner in which pattern, root, and vocalism morphemes combine. Hence, the morphotactic component need not worry about templatic morphotactics.

Linear nontemplatic morphotactics is handled via regular operations, usually n-way concatenation (Kaplan and Kay 1994) in the multitiered case. Consider for example Syriac /ʔetktab/ and its lexical analysis in Figure 4(b). The lexical analysis of the prefix is $\langle$ʔet, ε, $\varepsilon\rangle$ and that of the stem is $\langle$cvcvc, ktb, aa$\rangle$. Their n-way concatenation gives the tuple $\langle$ʔet cvcvc, ktb, aa$\rangle$. One may also use the "continuation classes" paradigm familiar from traditional two-level systems (Koskenniemi 1983, inter alia), in which lexical elements on each sublexicon are marked with the set of morpheme classes that can follow on the same sublexicon.

The last case is that of nonlinear nontemplatic morphotactics. Normally this arises in circumfixation operations. The following morphotactic rule formalism is used to describe such operations:

$$\begin{aligned} A &\rightarrow \overbrace{P\,B\,S} \\ (P,S) &\rightarrow (p_1,s_1) \\ (P,S) &\rightarrow (p_2,s_2) \\ &\vdots \\ (P,S) &\rightarrow (p_n,s_n) \end{aligned}$$

A circumfix here is a pair (P,S) where P represents the prefix portion of the circumfix and S represents the suffix portion. The circumfixation operation $\overbrace{PBS}$ applies the circumfix (P,S) to B. By way of illustration, consider the Syriac circumfixes of the imperfect verb in Table 2. The circumfixation of the circumfixes to the stem Syriac

/ktob/ 'to write – IMPF' is:

$$\begin{array}{rcl} \text{Verb} & \rightarrow & \overbrace{P \text{ ktob } S} \\ (P, S) & \rightarrow & (\text{ne}, \epsilon) \\ (P, S) & \rightarrow & (\text{te}, \epsilon) \\ (P, S) & \rightarrow & (\text{te}, \bar{\text{u}}\text{n}) \\ (P, S) & \rightarrow & (\text{te}, \text{å}\text{n}) \end{array}$$

Unlike traditional finite-state methods in morphology that employ two-tape transducers, the proposed multitiered model requires multitape transducers. The algorithms for compiling the three components into such machines are given next.

4. Algorithms for Compilation into Multitape Automata

Multitape finite-state machines were first introduced by Rabin and Scott (1959), and Elgot and Mezei (1965). An n-tape finite-state automaton (FSA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, δ: $Q \times (\Sigma^\epsilon)^n \rightarrow 2^Q$ is a transition function (where $\Sigma^\epsilon = \Sigma \cup \{\epsilon\}$ and ϵ is the empty string), $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a set of final states. An n-tape FSA accepts an n-tuple of strings if and only if starting from the initial state q_0, it can simultaneously scan all the symbols on every tape $i, 1 \leq i \leq n$, and end up in a final state $q \in F$. An n-tape finite-state transducer (FST) is simply an n-tape finite-state automaton but with each tape marked as to whether it belongs to the domain or range of the transduction.

In addition to common operators under which finite machines are closed, the algorithms discussed below make use of the following three operators:

Definition
Let L be a regular language. $\texttt{Id}_n(L) = \{ X \mid X$ is an n-tuple of the form $\langle x, \ldots, x \rangle, x \in L \}$ is the n-way identity of L.

Definition
Let R be a regular relation over the alphabet Σ and let m be a set of symbols not necessarily in Σ. $\texttt{Insert}_m(R)$ inserts the relation $\texttt{Id}_n(a)$ for all $a \in m$ freely throughout R.

The identity and insert operators are the n-tape version of their counterparts in Kaplan and Kay (1994).[3]

Definition
Let S and S' be same-length n-tuples of strings over some alphabet Σ, $I = \texttt{Id}_n(a)$ for some $a \in \Sigma$, and $S = S_1 I S_2 I \ldots S_k, k \geq 1$, such that S_i does not contain I; i.e., $S_i \in (\Sigma^n \Leftrightarrow \{I\})^*$. We say that $\texttt{Substitute}_{(S', I)}(S) = S_1 S' S_2 S' \ldots S_k$ substitutes every occurrence of I in S with S'.

3 Insert is also called the ignore operator, e.g., in the Xerox finite-state compiler (Karttunen and Beesley 1992; Karttunen 1993).

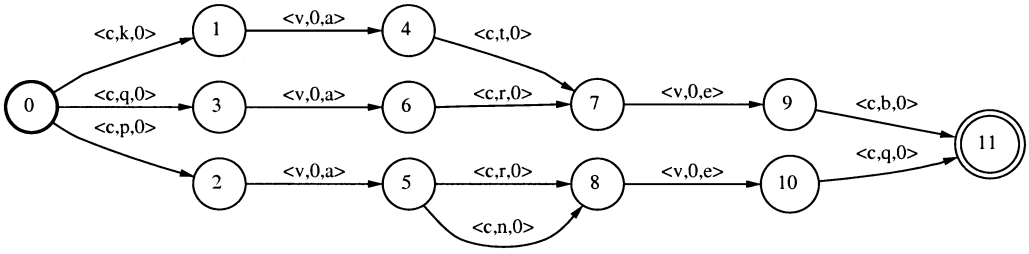

Figure 5
Multitape representation of the lexicon.

4.1 Building a Multitape Lexicon

The compilation process builds a one-tape automaton for each sublexicon. The sublexica are then put together using the cross product operator with the effect that the resulting machine accepts entries from the *i*th sublexicon on its *i*th tape.

Representing a lexical entry W in an automaton is achieved by concatenating the symbols of W one after the other. Now let $\mathcal{L}_i = \{ W_1, W_2, \dots \}$ be the set of lexical entries in the *i*th sublexicon. The expression for the *i*th sublexicon becomes:

$$L_i = \bigcup_{W \in \mathcal{L}_i} W \tag{1}$$

(Daciuk et al. [2000] give a more sophisticated incremental algorithm for compiling acyclic lexica.)

The overall lexicon can then be expressed by taking the cross product of all the sublexica. To make the final lexicon accept same-length tuples, we insert 0s throughout,

$$\mathit{Lexicon} = \left(\prod_{i=1}^{n-1} \texttt{Insert}_{\{0\}}(L_i) \right) \cap \pi^* \tag{2}$$

All invalid tuples resulting from the cross product operation (e.g., $\langle 0,0,\dots,0 \rangle$) are removed by the intersection with π^*, where π is the set of all feasible tuples computed from rules (see Section 4.2). By way of illustration, Figure 5 gives the lexicon for the pattern {cvcvc}, the roots {ktb}, {pnq}, {qrb}, and {prq}, and the vocalism {ae}.

4.2 Compiling the Rewrite Rules Component

The algorithm for compiling rewrite rules is based on collaborative work by the author with E. Grimley-Evans and S. Pulman (Grimley-Evans, Kiraz, and Pulman 1996). The compilation process is preceded by a preprocessing stage during which all mappings of unequal lengths are made same-length mappings by inserting a special symbol, 0, when necessary. (The grammar writer need not worry about this special symbol, but cannot use it in the grammar.) This is necessary because ϵ-containing transducers are not closed under intersection and subtraction. Additionally, during preprocessing the following sets are computed: the set of all feasible tuples sanctioned by the grammar, π (used in expression (2) above); and the set of feasible surface symbols, π_s (to be used in expression (23) in Appendix A).

The actual compiler takes as its input rules that have been preprocessed. The algorithm is subtractive in nature: it starts off by creating an automaton that accepts sequences of feasible tuples that are sanctioned by rules regardless of context, then starts subtracting strings which violate the rules. This subtractive approach was first suggested by E. Grimley-Evans.

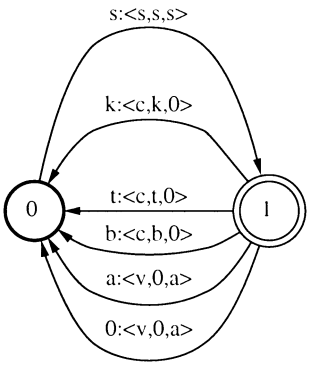

Figure 6
A four-tape machine for accepting centers. The symbol s denotes the partition symbol σ. The surface symbol appears to the left of ":" and the lexical tuple to its right.

4.2.1 Accepting Centers. Recall that in the formalism of Section 3.2, a lexical string maps to a surface string if and only if they can be partitioned into pairs of lexical-surface subsequences, where (i) each pair is licensed by a $\Rightarrow$ rule, and (ii) no sequence of zero or more adjacent pairs violates a $\Leftrightarrow$ rule.

Let $c = s{:}\langle l_1, l_2, \ldots \rangle$ be the center of a rule where s is the surface form and l_i are the lexical forms, and let C be the set of all such centers in the grammar. Further, let σ be a special symbol (not in the grammar's alphabet) to denote a subsequence boundary within a partition, and let $\sigma' = \mathtt{Id}_n(\sigma)$. The automaton that accepts the centers of the grammar is described by the relation

$$\mathit{Centers} = \sigma'(C\,\sigma')^* \tag{3}$$

Centers accepts any sequence of the centers described by the grammar (each center surrounded by σ') irrespective of their contexts. Assuming that /ktab/ is under consideration, Figure 6 gives the four-tape machine for the centers of the rules from Figure 3.

4.2.2 Valid Contexts. Now we eliminate all the sequences whose left and right contexts violate the grammar. For each center $c \in C$ in the entire grammar, let $LR_c = \{(\lambda_1, \rho_1), (\lambda_2, \rho_2), \ldots\}$ be the set of valid left and right context pairs for that center. The *invalid* contexts for c, are expressed by:

$$\mathit{Restrict} = \pi^* c\,\pi^* \Leftrightarrow \bigcup_{(\lambda,\rho)\in LR_c} \lambda c \rho \tag{4}$$

The first component of expression (4) gives all the possible contexts for c. The second component gives all the valid contexts for c. The subtraction results in all the invalid contexts for c. However, since σ appears freely in expression (3), it needs to be introduced in expression (4) as well, resulting in:

$$\mathit{Restrict} = \mathtt{Insert}_{\{\sigma\}} \left(\pi^* c \pi^* \Leftrightarrow \bigcup_{(\lambda,\rho)\in LR_c} \lambda c \rho \right) \tag{5}$$

The relation in expression (5) works only if the center consists of just one tuple. In order to allow it to be a sequence of tuples, c must be surrounded by σ' on both sides

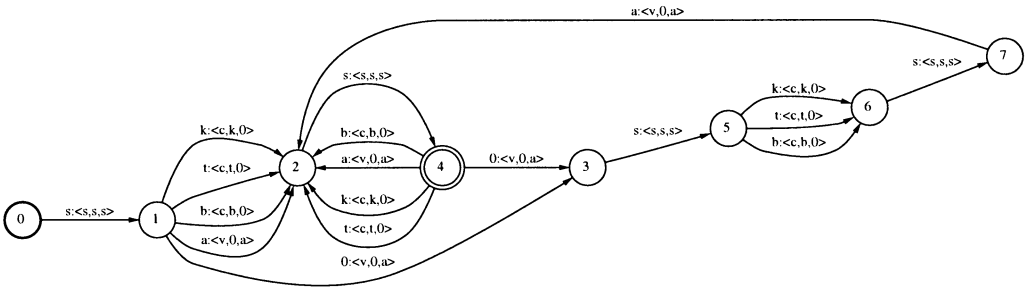

Figure 7
The machine from Figure 6 is repeated here after processing the rules in Figure 3.

to mark it as one subsequence. It also must be devoid of any σ'. The first condition is accomplished by simply placing σ' to the left and right of c. As for the second condition, an auxiliary symbol ω is used as a placeholder representing c in order to avoid inserting σ' within the tuples of c by `Insert`. Hence, first we introduce σ freely using `Insert`, then substitute c back in place of ω,

$$Restrict = \tag{6}$$
$$\texttt{Substitute}_{(c,\omega')}\left(\texttt{Insert}_{\{\sigma\}}\left(\pi^*\sigma'\omega'\sigma'\pi^* \Leftrightarrow \bigcup_{(\lambda,\rho)\in LR_c} \lambda\sigma'\omega'\sigma'\rho\right)\right)$$

where $\omega' = \texttt{Id}_n(\omega)$. Finally, for each c we subtract all such invalid relations from *Centers*, yielding the relation,

$$ValidContexts = Centers \Leftrightarrow \bigcup_{c} Restrict \tag{7}$$

ValidContexts now accepts all the sequences of tuples described by the grammar based on their contexts; however, it does not enforce obligatory rules. Figure 7 gives the machine after the center of R3 from Figure 3 has been processed.

4.2.3 Obligatory Rules. For each obligatory rule, let $\mathcal{C}$ represent the center c with the *correct* lexical expressions and the *incorrect* surface expression. The following relation describes all sequences of tuples that contain an unlicensed segment:

$$Coerce = \bigcup_{(\lambda,\rho)\in LR} \texttt{Insert}_{\{\sigma\}}(\lambda\sigma'\mathcal{C}\sigma'\rho) \tag{8}$$

The two σ's surrounding $\mathcal{C}$ ensure that obligatoriness applies to at least one lexical-surface subsequence. The `Insert` operator inserts additional σ's through the contexts *and* the center. The insertion of σ' through the center allows *Coerce* to apply to a series of lexical-surface subsequences. To handle the case of epenthetic rules, one needs to allow *Coerce* to apply on zero subsequences as well. In such a case, one takes the union of expression (8) with $\texttt{Insert}_{\{\sigma\}}(\lambda\sigma'\rho)$, i.e., the empty subsequence.

Finally, we subtract *Coerce* from the *ValidContexts* relation, yielding the relation:

$$Rules = ValidContexts \Leftrightarrow \bigcup_{\mathcal{C}} Coerce \tag{9}$$

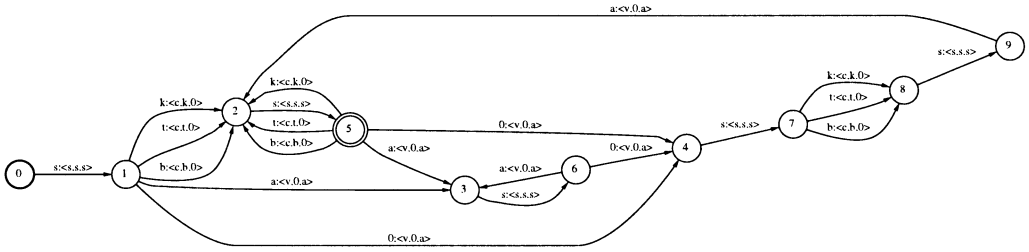

Figure 8
The machine from Figure 7 is repeated here after processing the obligatory rule R3 from Figure 3.

The relation accepts *all* and *only* the sequences of tuples described by the grammar. Figure 8 gives the machine after processing the obligatoriness of rule R3. The last step in compiling rules is to remove all instances of the symbol σ and the symbol 0.

4.3 Compiling the Morphotactic Component

The only class of morphotactics that is of interest here is nonlinear nontemplatic circumfixation per the formalism in Section 3.3. Let $(P, S) = \{ (p_1, s_1), (p_2, s_2), \ldots, (p_n, s_n) \}$ be a set of circumfixes and let B be the domain of the circumfixation operation. A rule $A \rightarrow \overbrace{PBS}$ is compiled into an automaton by the expression in (10):

$$A = \bigcup_{(p,s) \in (P,S)} pBs \tag{10}$$

An equivalent approach to handling circumfixation is to follow the subtractive approach used in compiling the rewrite rules component above by first overgenerating and then subtracting all invalid forms. The two approaches, union or subtraction, are formally equivalent in that they result in the same machine. Compilation using the union approach, however, is more efficient in terms of time complexity than the subtractive approach. The latter requires invoking negation and intersection algorithms, both of which are computationally expensive. It must be noted that the union approach requires a great deal of care in creating a machine that accepts only grammatical forms with no overgeneration.

Beesley (1998c) employed a similar approach for eliminating invalid forms in Arabic long-distance dependencies by means of composition.

5. Developing Semitic Grammars

When developing Semitic grammars, various issues and problems arise that normally do not occur with linear grammars. This section aims at pointing out some of these issues.

5.1 Handling the Nonlinear Stem

The example lexica and rules in Sections 3.1 and 3.2 demonstrate how the CV-based templates are implemented in the current framework. Further, spreading and gemination, which tend to cause difficulties in other computational frameworks (see Section 6), are represented easily in the multitiered model without having to resort to ad hoc notation. For example, the following rule, which uses prosodic templates, demonstrates

the intrasyllabic spreading of vowels:

$$\text{Intrasyllabic spreading:} \quad \underset{\text{CVV}}{\langle \sigma_{\mu\mu}, \text{C}, \text{V} \rangle} \quad \Leftrightarrow$$

$$\text{Extrametricality:} \quad \underset{\text{C}}{\langle \sigma_x, \text{C}, \varepsilon \rangle} \quad \Leftrightarrow$$

The symbol $\sigma_{\mu\mu}$ above is a templatic segment denoting a bimoraic syllable. The rule states that when $\sigma_{\mu\mu}$ appears on the pattern tape, a consonant C and a vowel V are read from the root and vocalism tapes, respectively; the corresponding surface segments are CVV. In conjunction with the extrametricality rule above, one obtains surface-lexical tuples like ʃaamuus:$\langle \sigma_{\mu\mu}\sigma_{\mu\mu}\sigma_x$, ʃms, au$\rangle$, where the element to the left of ":" is the surface form and the tuple to its right represents the lexical forms (see Figure 2(d)).

Gemination is handled in one of two ways: The first marks pattern consonantal segments (C or σs) with a subscript (e.g., $c_1vc_2vc_3$) and provides rules to geminate the appropriate consonant, e.g.,

$$\text{Gemination in /kattab/:} \quad \underset{XX}{\langle c_2, \text{X}, \varepsilon \rangle} \quad \Leftrightarrow$$

The second approach leaves pattern segments unmarked, but provides the proper left and right contexts in rules. The former approach requires more annotations in lexica and rules, but provides for smaller machines since no context expressions are used. The opposite holds for the later approach. Both, however, require rule features (see Section 5.2) to ensure that the rule applies only to the desired measures.

As the multitiered framework does not put any limitations on the number of lexical tapes, the grammar writer may also choose to place affixes on their own autonomous tape. Hence, one produces surface-lexical tuples like ʔakateb:$\langle$ʔa, cvcvc, ktb, ae$\rangle$ (see Figure 1(c)).

5.2 Using Features to Handle Idiosyncracies

Various lexical and morphotactic constraints exist in Semitic. For instance, roots do not appear in all verbal measures; rather, each root occurs in the literature in a subset of the measures, e.g., {pnq} does not exist in Measure 1 (one gets such information from dictionaries and corpora). Another example is that the vocalisms of Measure 1 in almost all Semitic languages are lexically marked for each root. In the current framework, categories of the form:

$$\begin{bmatrix} \text{cat-name} \\ \text{ATTRIBUTE}_1 = \text{VALUE}_1 \\ \text{ATTRIBUTE}_2 = \text{VALUE}_2 \\ \vdots \end{bmatrix}$$

are used in the lexicon as well as rules to resolve such issues (Bear 1988; Ritchie et al. 1992). Here, a value can be either an atom or a set of atoms. For example, the set of measures in which a root occurs are given in the attribute MEASURE below:

$$\text{ktb} \begin{bmatrix} \text{pattern} \\ \text{MEASURE} = \{1,2,3\} \\ \text{VOWEL} = \text{a} \end{bmatrix}, \qquad \text{pnq} \begin{bmatrix} \text{pattern} \\ \text{MEASURE} = \{2,3\} \\ \text{VOWEL} = \text{NA} \end{bmatrix},$$

$$\text{qrb} \begin{bmatrix} \text{pattern} \\ \text{MEASURE} = \{1,2,3\} \\ \text{VOWEL} = \text{e} \end{bmatrix}$$

The vocalisms of Measure 1 for various roots are marked with the attribute VOWEL. Hence, one gets [a] in /ktab/, but [e] in /qreb/.

Categories are also used in rules. For example, the gemination rule above is associated with a category that indicates the measures in which gemination is valid. The definition of rule obligatoriness is extended to include categories (Pulman and Hepple 1993). The categories are incorporated in the automata compilation process following the algorithms in Kiraz (1997a).

5.3 Linear versus Nonlinear Grammars

Considering that nonlinearity in Semitic occurs mainly in the stem, maintaining a nonlinear lexical representation in rewrite rules causes rules that describe one phonological/orthographic phenomenon to be duplicated. This becomes a challenge to the grammar writer since Semitic employs very rich phonological rules: assimilation, dissimilation, prosthesis, anaptyxis, syncope, haplology, etc. (Moscati et al. 1969, Section 9.1 ff.).

Consider the derivation of Syriac /ktab/ using the rules in Figure 3. Since vowel deletion in Syriac applies right-to-left, when adding the object pronominal suffix {eh} 'MASC 3RD SING', the second vowel should be deleted, */katabeh/ → /katbeh/. By virtue of its right lexical context, however, R3 in Figure 3 can only apply to the *first* stem vowel. Another rule (R4 below) is required for deriving /katbeh/ from */katab/ and the suffix {eh}, where the *second* stem vowel is deleted.

$$\begin{array}{lccccccc} & * & \Leftrightarrow & \langle \mathrm{v}, \varepsilon, * \rangle & \Leftrightarrow & \langle \mathrm{cV}, *, * \rangle & \Leftrightarrow \\ \mathrm{R4} & * & \Leftrightarrow & \epsilon & \Leftrightarrow & * & \end{array}$$

where V is a vowel.

The c in the right lexical context is a concrete symbol from a pattern morpheme, while V represents the class of all vowels.

This does not resolve the problem. Both R3 and R4 fail when the deleted vowel itself appears in the prefix, e.g. /wakatbeh/ → /wkatbeh/ (with the prefix {wa}), requiring another rule. An additional rule is also needed to delete prefix vowels when the right context belongs to a (possibly another) linear prefix, e.g., prefixing the sequence {wa} 'and', {la} 'to', and {da} 'of' to the stem /katab/ giving /waldaktab/ (the [a] of {la} and the first stem vowel are deleted).

The above examples clearly illustrate the proliferation that would result. Considering that such phonological rules do not depend on the nonlinear lexical structure of the stem, a better approach divides the lexical-surface mappings into two separate problems. The first handles the templatic nature of morphology, mapping the multiple lexical representation into a linearized lexical form, somewhat corresponding to McCarthy's notion of tier conflation (McCarthy 1986). Linearization of autosegmental representations in general has been suggested earlier by Kornai (1991, 1995).

The second takes care of phonological/orthographic/graphemic mappings between the linearized lexical form and the actual surface. The entire grammar is taken as the composition of two sets of rules (Karttunen, Kaplan, and Zaenen 1992). Composition, however, needs to be defined for multitape machines. First, we redefine an n-tape finite-state machine as $(Q, \Sigma, \delta, q_0, F, d)$, where the first five elements are as before and $d, 1 \leq d < n$, is the number of domain tapes (the number of range tapes is simply $n \Leftrightarrow d$).

Definition

Let $A = (Q_1, \Sigma_1, \delta_1, q_1, F_1, d_1)$ and $B = (Q_2, \Sigma_2, \delta_2, q_2, F_2, d_2)$ be two multitape machines over n_1 and n_2 tapes, respectively. Further, let s_i denote the symbol on the ith tape.

There is a composition of A and B, denoted by C, if and only if $d_2 = n_1 \Leftrightarrow d_1$ with $C = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, [q_1, q_2], F_1 \times F_2, d_1)$ where for all $p_1 \in Q_1$ and $p_2 \in Q_2$,

$$\begin{aligned}\delta([p_1, p_2], s_1 : \cdots : s_{d_1} : s'_{d_2+1} : \cdots : s'_{n_2}) = \\ [\delta_1(p_1, s_1 : \ldots : s_{d_1} : s_{d_1+1} : \ldots : s_{n_1}), \\ \delta_2(p_2, s'_1 : \ldots : s'_{d_2} : s'_{d_2+1} : \ldots : s'_{n_2})]\end{aligned}$$

if and only if $s_{d_1+1} = s'_1, \ldots, s_{n_1} = s'_{d_2}$.

The resulting machine is an k-tape machine, where $k = d_1 \Leftrightarrow d_2 + n_2$. In our implementation (see Section 7, and Appendix A), the domain and range tapes are given as arguments to the composition function, rather than coding them in machines, in order to allow for flexibility in using machines.

5.4 Vocalization

Semitic texts appear in three forms: consonantal texts, which do not incorporate any vowels but *matres lectionis*; partially vocalized texts, which incorporate some vowels to clarify ambiguity; and vocalized texts, which incorporate full vocalization.

Handling all such forms is resolved in line with the previous discussion on linearization. The grammar writer should assume full vocalization when writing grammars. This will not only get rid of the duplicated rules for the same phonological/orthographic phenomenon, but will also make understanding and debugging rules an easier task. Once a lexical-surface rewrite rules system has been achieved, a set of rules that optionally delete vowel segments are specified and composed with the entire system.

6. Other Approaches to Finite-State Semitic Morphology

6.1 Kay's Multitape Approach

Kay (1987) proposed handling the autosegmental analysis of Arabic by means of multitape automata. Kay adds some extensions to traditional FSTs. Transitions are marked with quadruples of elements (for vocalism, root, pattern, and surface form, respectively), where each element is a pair: a symbol and an instruction concerning the movement of the tape's head. Kay uses the following notation: An unadorned symbol is read and the tape's head moves to the next position. A symbol in brackets, [ ], is read and the tape's head remains stationary. A symbol in braces, { }, is read and the tape's head moves only if the symbol is the last one on the tape.

The transitions for the analysis of Syriac /kattab/ 'to write—CAUSATIVE, PASSIVE', excluding the reflexive prefix {ʔet}, are shown in Figure 9. After the first transition on the quadruple ⟨[ ], k, C, k⟩ in Figure 9(a): no symbol is read from the vocalism tape, [k] is read from the root tape and the tape's head is moved, [C] is read from the pattern tape and the tape's head is moved, and [k] is written on the surface tape and the tape's head is moved. At the final configuration, all the tapes have been exhausted. Kay makes use of a special symbol, G, to handle gemination; when read, a symbol from the root tape is scanned without advancing the read head of that tape.

The model suffers from a number of shortcomings, some of which have already been pointed out (Bird and Ellison 1992, Section 5.1). Firstly, the use of various bracketing notations to control the moves of the machine head(s) causes the read heads of the three upper input tapes to move independently of each other; this put the expressiveness of the device under question: Bird and Ellison (1992, but not 1994) questioned the formal power of the device. Wiebe (1992), citing formal results from Fischer (1965),

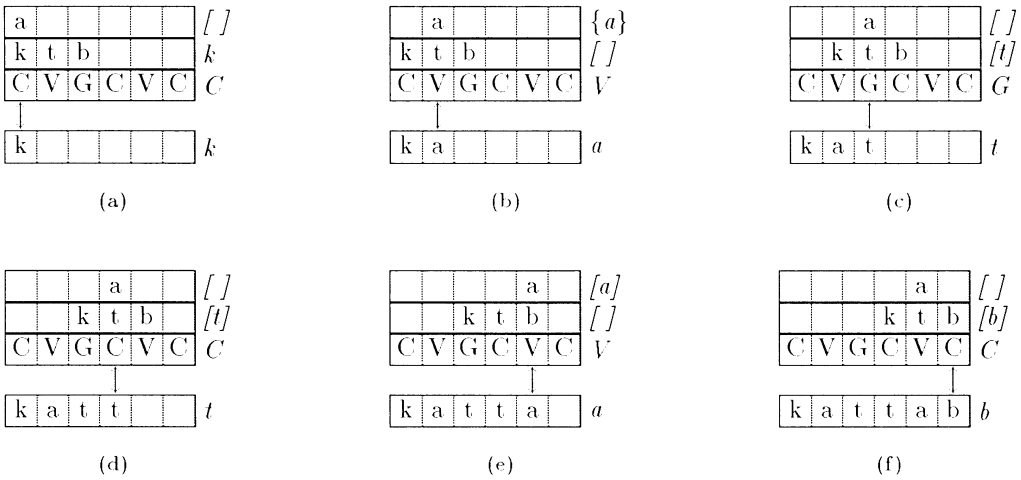

Figure 9
Kay's Analysis of Syriac /kattab/. The four tapes are (from top to bottom): vocalism tape, root tape, pattern tape, and surface tape. Transition quadruples are shown at the right side of the tapes. The symbol "↕" between the lower surface tape and the lexical tapes indicates the current symbols under the read/write heads.

stated that Kay's machine goes beyond finite-state power. No consensus has been reached on the matter to the best of the author's knowledge. In contrast, our proposed *n*-tape machines move all the read heads simultaneously ensuring finite-state expressive power. Secondly, the introduction of ad hoc symbols to templates (e.g., G for gemination) moves away from the spirit of association in autosegmental phonology that Kay wanted to model; other special symbols must also be added to completely implement the rest of the paradigm in question.

We have demonstrated, however, the usefulness of Kay's proposal. Indeed, if one eliminates the ad hoc controls of the read head(s) and provides a rule formalism from which machines can be compiled algorithmically, the multitape model is quite adequate for describing autosegmental representations.

6.2 The Intersection of Lexica Approach

Kataja and Koskenniemi (1988), working on Akkadian, developed a system under traditional two-level morphology. It was mentioned earlier (see Section 1.1) that the challenge of handling Semitic morphology within traditional two-level morphology is that the lexical level is not merely the concatenation of the morphemes in question.

Kataja and Koskenniemi resolved this by devising a "lexicon component" that makes use of two lexica: one for roots and the other for stem patterns and affixes. Entries in the former leave affix elements unspecified, while entries in the latter leave root elements unspecified. For example, the lexical entry for the Arabic root morpheme {ktb} takes the form

$$\Sigma_p^* \, \mathrm{k} \, \Sigma_p^* \, \mathrm{t} \, \Sigma_p^* \, \mathrm{b} \, \Sigma_p^* \tag{11}$$

where Σ_p is the alphabet of nonroot segments; likewise, the entry for the perfect passive vocalism {ui} takes the form:

$$\Sigma_r^* \, \mathrm{u} \, \Sigma_r^* \, \mathrm{i} \, \Sigma_r^* \tag{12}$$

where Σ_r is the alphabet of root segments. The intersection of both expressions allows for the well-formed string /kutib/, as well as numerous ill-formed sequences such as */ktbui/, inter alia. Under this framework, the result of the intersection becomes the

lexical level of a traditional two-level morphology system. The two-level system then takes care of other morphological, phonological, and orthographic rules, all of which are linear in nature. Kataja and Koskenniemi suggested simulating the intersection by having the lexical lookup explore both lexica simultaneously.

The following computational shortcomings of the intersection approach come to mind. The intersection of the two lexica works only if Σ_p and Σ_r are disjoint. As this is not the case in Semitic, one has to introduce ad hoc symbols in the alphabet to make the two alphabets disjoint. Alternatively, Beesley (forthcoming) introduces an ingenious, but cumbersome, bracketing mechanism. Expression (11) above becomes:

$$A^* \langle \mathrm{k} \rangle A^* \langle \mathrm{t} \rangle A^* \langle \mathrm{b} \rangle A^* \tag{13}$$

where $A = \Sigma \Leftrightarrow \{ \langle, \rangle \}$ (I have changed Beesley's curly brackets into angle brackets in order to avoid confusion with set notation). Expression (12) then becomes:

$$B^* \,\mathrm{u}\, B^* \,\mathrm{i}\, B^* \tag{14}$$

where $B = \Sigma \Leftrightarrow V$, and V is the disjunction of all vowels. Finally, each measure is given by an expression; for instance, Arabic Form V (e.g., /takattab/ where the first [t] is an affix not related to the [t] of the root) is:

$$\mathrm{t}VCVCXVC \tag{15}$$

where C is

$$\langle \{ \mathrm{k,t,b} \} \rangle \tag{16}$$

(i.e., the disjunction of the root symbols surrounded by angle brackets). The symbol X in expression (15) indicates gemination in a way reminiscent of Kay's G symbol. The intersection of expressions (13), (14), and (15) results in /takatXab/ (X is dealt with by later rules). The disjunction of all such intersections results in what one may call a "quasi lexicon," i.e, the lexical side of subsequent two-level transducers that deal with linear phenomena (setting aside long-distance dependencies). Given r roots (approximately 4,000 in Modern Standard Arabic), v vocalisms, and p patterns (a few hundred for $v \times p$ depending on the linguistic framework used), Beesley's bracketing algorithm needs to perform m intersections, where $r \ll m < r \times v \times p$ (since each root only intersects with lexically defined subsets of the patterns). In contrast, such a bracketing mechanism is not necessary in our multitape approach, since the alphabet of one tape does not interfere with the alphabets of other tapes. Further, our lexicon compiler needs to perform only $n \Leftrightarrow 1$ cross product operations (where n is the number of lexical tapes, usually 3). There is a substantial time complexity difference with practical effects. A faithful implementation of Beesley's bracketing approach and ours was performed using the Bell Labs Lextools compiler (Sproat 1995; Kiraz 1997b). (See Sproat [1997, Section 3.2] for a brief description of Lextools.) The test was also performed by M. Jansche using a neutral finite-state library (van Noord 1997) to ensure partiality. Jansche was able to substantially enhance the performance of Beesley's method. The results of compiling various numbers of roots with the 24 Arabic verbal patterns appear in Table 3. The table indicates that for a full-scale system, the proposed multitier compilation method is far more efficient. Details of the tests appear in Appendix B.

More serious is the fact that bidirectionality of two-level morphology (i.e., morphemes mapping to surface forms and vice versa) is lost. Once intersection is performed, the result is an accepting automaton that represents stems rather than independent morphemes. In contrast, using our multitape model, the original morphemes

Table 3
Evaluation of lexical compilation of Beesley's bracketing mechanism vs. Kiraz's multitiered method using Lextools and van Noord's FSA utilities. The times of the compilation process for the latter are based on an enhanced implementation of Beesley's method by M. Jansche (see Appendix B). The ratio columns show the order of complexity (e.g., for 100 roots, the multitiered lexical compilation runs 23.7 times faster than Beesley's bracketing mechanism using Lextools and 4.9 times faster than Jansche's enhancements to Beesley's algorithm).

	Lextools			Van Noord's		
Roots	Beesley (h min sec)	Kiraz (sec)	Ratio	Beesley-Jansche (sec)	Kiraz (sec)	Ratio
20	4.97s	0.64	7.8	7.4	3.9	1.9
40	9.85s	0.79	12.5	15.6	6.1	2.6
60	14.64s	0.81	18.1	25.7	7.4	3.5
80	19.51s	0.95	20.5	38.1	9.3	4.1
100	24.03s	1.13	23.7	52.9	10.8	4.9
200	48.52s	1.39	34.9	177.0	22.7	7.8
300	1m 10.95s	1.67	42.5	387.7	37.6	10.4
400	1m 37.42s	1.87	52.1	677.1	56.3	12.0
500	2m 6.94s	2.62	48.4	1124.4	79.2	14.2
1,000	5m 0.79s	5.64	53.4			
2,000	22m 19.53s	10.20	131.3			
3,000	2h 5m 35.38s	12.60	598.0			

(root, pattern, and vocalism) can be reconstructed from the multitape lexicon by a projection operation. Hence, projection, under which automata are closed, acts as the "reciprocal" operator for cross product in expression (2) ensuring means for bidirectionality. There is no such reciprocal operator for intersection: it is a destructive operator in the sense that its arguments cannot be recovered from the result.

6.3 Beesley's Other "Intersection" Approach

Beesley and his colleagues (Beesley, Buckwalter, and Newton 1989; Beesley 1990, 1991) developed a large-scale Arabic system under two-level morphology, which even has the ability to handle regional Egyptian spelling (Beesley, p. c.). The lexical lookup of the two-level model was augmented by a technique called "detouring" to access roots and affixes from different lexica (see Sproat [1992, 163–165] for details on "detouring"). In his 1996 paper, and subsequent work, Beesley reimplemented the system using the Xerox lexical and rule compilers (Karttunen 1993; Karttunen and Beesley 1992). An on-line demo of the reimplementation was also developed (Beesley 1998a).[4]

Bidirectionality is maintained in Beesley's system by a direct mapping of each root and pattern pair to their respective surface realizations. The lexical description gives the root and pattern superficially concatenated in the form (Beesley 1996, p. c.):

$$[\text{ktb\&CaCaC}] \tag{17}$$

The square brackets are special symbols that delimit the stem, and "&" is another special symbol that separates the root from the pattern; it is not the intersection operator.

4 The new URL is http://www.xrce.xerox.com/research/mltt/arabic.

For each root and pattern pair, a rule of the following form is generated automatically:

$$[\text{ktb\&CaCaC}] \rightarrow \text{katab} \tag{18}$$

Each rule of the form in (18) is compiled into a transducer, which is then applied by composition to the identity transducer of the corresponding lexical description in (17). The result is a transducer that maps the string "[ktb&CaCaC]" into "katab". It is worth noting that rules of the form in (18) are reminiscent of Chomsky's early transformational rules for Hebrew stems (Chomsky 1951).

(As "&" in (17) and (18) is a concrete symbol, no real intersection, in the set-theoretic sense, takes place, though Beesley refers to this method as well as to the bracketing mechanism described in the previous section as "intersection".)

This method requires m (where $r \ll m < r \times v \times p$ as before) rules of the form in (18) to be compiled into their respective transducers using algorithms of the Xerox replace operator (Karttunen 1997), literally thousands of rules. Additionally, the entire set of m transducers (or subsets, one subset at a time) needs to be put together into one (or more) transducer(s) by means of intersection (if the transducers are ϵ-free) or composition. Although this takes place during developing the grammar, rather than at run-time, the inefficiency that results in the compilation process is apparent from the fact that a linguistic phenomenon (here, the linearization of stems) is conveyed by applying a rule to every single stem of the language. As one does not provide a rule to delete [e] in English /move+ing/ and another to delete the same in /charge+ing/, etc., but a single [e] deletion rule that applies throughout the entire language, stems in Semitic ought to be realized by rules that represent the phenomenon itself, not every single instance of the phenomenon.

In contrast, the proposed multitier model requires only three rules throughout the entire language to model Beesley's roots and patterns (i.e., with X to denote gemination and hard-coding vocalic spreading): R1 (for stem consonants) and R2 (for stem vowels) from Figure 3, in addition to the following gemination rule (see Section 5.1 for our handling of gemination and spreading):

$$\text{Gemination:} \quad \begin{array}{ccccccc} \langle \text{x}, \text{C}, \varepsilon \rangle & \Leftrightarrow & \langle \text{x}, \varepsilon, \varepsilon \rangle & \Leftrightarrow & ^{*} & \Leftrightarrow \\ ^{*} & \Leftrightarrow & \text{C} & \Leftrightarrow & ^{*} & \end{array}$$

where C is a consonant.

The result of the three rules is a mere $(|R| + 1)$-state machine, where R is the set of all root segments (= 28 for Arabic, 22 for Syriac), which is then applied to the multitiered lexicon. Figure 10 gives such a machine for R={ k,t,b } and the vowels { a,u,i }.

Another disadvantage, although a minor one, of rules of the form in (18) is the loss of alignment between surface segments and their lexical counterparts. While this does not affect the behavior of the resulting machines, having segments aligned helps in debugging at the grammar-design stage. A cursory look at the transitions in Figure 10 indicates to the grammar writer the lexical segments that correspond to a surface segment.

Having said that, Beesley's system remains the largest reported Semitic grammar written within finite-state morphology to date. The system, however, relies on old linguistic models, as old as Harris (1941). No move has been reported to employ modern linguistic models such as the autosegmental framework and other developments mentioned in Section 1, although this seems to be the direction of modern research in computational Semitic morphology as well as linguistics (see the bibliographical entries cited in Section 1).

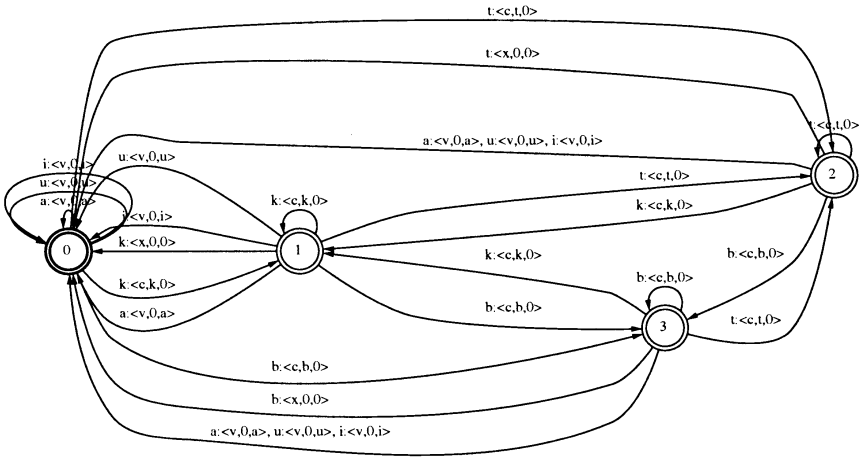

Figure 10
Rules for mapping multitier lexica into stems. The symbol 0 represents ϵ.

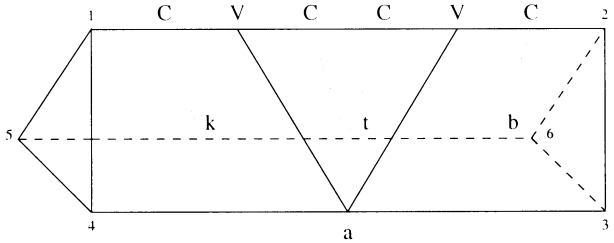

Figure 11
Triangular prism demonstrating the autosegmental representation of Arabic /kattab/.

6.4 Encoding Autosegmental Representations Approach

There have been a number of proposals to encode autosegmental representations. Kornai (1991, 1995) gives a linear coding; Wiebe (1992) and Bird and Ellison (1992) give a multitiered encoding. We shall illustrate this approach from Bird and Ellison's work.

Every pair of autosegmental tiers constitutes a chart (or plane). The representation of Arabic /kattab/, for example, takes the form of a triangular prism as in Figure 11 (Pulleyblank 1986). Each morpheme sits on one of the prism's three longitudinal edges: the pattern on edge 1-2, the vocalism on edge 3-4, and the root on edge 5-6. The prism has three longitudinal charts: pattern-vocalism (1-2-3-4), pattern-root (1-2-6-5), and root-vocalism (3-4-5-6). The corresponding encoding of the diagram is:

Tier 1			a:2:0:0			
Tier 2	C:0:1:0	V:1:0:0	C:0:1:0	C:0:1:0	V:1:0:0	C:0:1:0
Tier 3			k:0:1:0	t:0:2:0	b:0:1:0	

Each expression is an $(n+1)$-tuple, where n is the number of charts. The first element in the tuple represents the autosegment. The positions of the remaining elements in the tuple indicate the chart in which an association line occurs, and the numerals indicate the number of association lines on that chart. For example, the expression a:2:0:0 states that the autosegment "a" has two association lines on the first pattern-vocalism chart, zero lines on the second pattern-root chart, and zero lines on the third root-vocalism chart.

No implementation of a Semitic language, to the best of the author's knowledge, has been carried out using these methods. Bird and Ellison, however, give an example of how they envisage implementing Semitic using their framework. For each measure, they provide an expression that generalizes over that particular measure, e.g. for {CVCCVC}:

$$\text{C:1 V:0 C:1 C:1 V:0 C:1} \tag{19}$$

The numbers after the colon refer to the autosegmental tier with which the segment is linked (0 for vowels and 1 for root segments). A second expression generalizes over all stems constructed from a particular root, e.g., for {ktb}:

$$\Sigma{:}0^* \text{ k:1 } (\Sigma{:}0^* \text{ t:1})^+ \text{ b:1} \tag{20}$$

where "*" and "+" denote Kleene star and plus, respectively. A third expression describes the vocalism, e.g., for {a} with spreading:

$$(\text{a} \cup \text{C})^* \tag{21}$$

The intersection of the three expressions, per their intersection algorithm for such encodings, yields:

$$\text{k:1 a:0 t:1 t:1 a:0 b:1} \tag{22}$$

Superficially, this approach may seem equivalent to the other intersection approaches mentioned in Section 6.2. The methodology here, however, is formally more appealing and linguistically more sound since it provides for a mechanism to describe autosegmental association. Bird and Ellison (1992, 87) question if their approach will "cover all possible generalizations about Arabic verbal structure." It would definitely be worth investigating how a higher-level autosegmental description of Semitic can be compiled algorithmically into their machines directly.

We mentioned above (Section 6.2) that the intersection approach lacks bidirectionality. It is possible, though this has not been tested, that the indices in Bird and Ellison's method can play a role in claiming the various morphemes of a particular surface form.

7. Implementation

There are two aspects of the implementation: implementing the theoretical model based on the algorithms presented in Section 4 and implementing Semitic grammars for the case study. As for the former, the algorithms in Section 4 were implemented by the author in SICStus Prolog. Details of the implementation are given in Appendix A.

As for the grammars themselves, a small-scale Syriac grammar was implemented based on the 100 most frequent roots, including their numerous inflexions, of the Syriac New Testament (Kiraz 1994a). Care was taken, however, to ensure that most of the verbal and nominal classes of the language were exhaustively covered. Additionally, sample Arabic grammars—but with full coverage of the phenomena under question—have been implemented to test various linguistic models of Semitic, including: CV templates, moraic templates, affixational templatic morphology, prosodic circumscription, and broken plurals. A detailed description of handling these linguistic models appears elsewhere (Kiraz, in press).

8. Conclusion

This paper presented a multitier morphology model that can cope with the nonlinear operations of Semitic root-and-pattern morphology in a linguistically motivated way.

The system consists of three main components: a lexicon made of multiple sublexica, where each sublexicon represents entries from a particular tier; a rewrite rules system that maps multiple lexical representations to a surface representation; and a morphotactic component that makes use of regular rewrite rules. Rules and lexica are compiled algorithmically into multitape finite-state machines. There is no reason to believe that our multitiered rewrite rules component cannot be applied to other rewrite rules systems (Koskenniemi 1983; Kaplan and Kay 1994; Mohri and Sproat 1996; Karttunen 1997).

It was demonstrated that the proposed framework is capable of modeling sophisticated linguistic descriptions, especially those of autosegmental phonology. Having said that, considering that the morphology of Semitic languages is notoriously difficult to analyze—partly because of its root-and-pattern nature and the existence of many morphologically distinct homographic morphemes, but mostly because the orthographic system is underspecified (see Section 5.4)—the current work, as well as all reported work on Semitic morphology, is far from providing a *usable* morphological system. Much work in the area of morphological disambiguation, which must venture into the realms of syntax and semantics, awaits research.

The proposed multitape approach may not be confined to Semitic and may prove useful for autosegmental representation in general. Since a segment in modern phonological theory is a mere shorthand for multitiered features, the model may be applied to concatenative languages when descriptions are required at the multitiered feature level. While this may be cumbersome for morphological applications, it may prove useful for automatic speech recognition and other applications that require subsegmental analyses.

Acknowledgments

This research was supported by a St. John's Benefactors' scholarship and was carried out under the supervision of Dr. Stephen Pulman (University of Cambridge). Thanks are due to the Master and Fellows of St. John's College and the Computer Laboratory, Cambridge, for various grants. Much revision and enhancement took place at Bell Laboratories. Comments by the anonymous reviewers helped in reshaping the presentation of this paper. Ken Beesley kindly made a forthcoming paper available to me and answered many questions. Martin Jansche performed the test detailed in Appendix B.2 and provided comments. Christine Nakatani provided many useful editorial comments.

Appendix A: Implementation

The algorithms in Section 4 were implemented by the author in SICStus Prolog using a finite-state library provided by E. Grimley-Evans. The library allows the creation, manipulation, and destruction of multitape finite-state machines with an easy algebraic interface to *n*-way regular expressions. The Prolog term `regexp_to_fsa(+RegExp,?Automaton)` constructs the machine `Automaton` for the extended regular expression `RegExp`, e.g., expression 3 (Section 4.2.1) is turned into a four-tape machine with the expression

```
regexp_to_fsa(t(σ,σ,σ,σ)^([t(k,c,k,0),...)]^t(σ,σ,σ,σ))*, Centers)
```

The predicate `t(+terminal)` denotes a terminal tuple, infix `^` denotes concatenation, postfix `*` denotes Kleene star, and a list denotes union over its elements. Primi-

tive Prolog procedures (e.g., `union/3`, `kleene/2`, etc.) are also provided for (Kiraz and Grimley-Evans 1998).

The algorithms given in Section 4 are closely followed. First, the lexical compiler is invoked creating a multitape machine for each sublexicon, and then putting them together with the cross product operator. The rule compiler then compiles all the rules into another machine. The entire language description is then created with the operation:

$$Language = (\pi_s \times Lexicon) \cap Rules \tag{23}$$

where π_s denotes the set of all surface symbols. The first component maps the lexicon to all surface symbols. The intersection leaves *Language* with the valid expressions only. (One can also use composition instead of intersection depending on the nature of the rules.)

There is some room to enhance the implementation, especially in time and memory overhead. While the finite-state library is capable of handling automata for small-scale grammars, larger grammars would stretch its capabilities. The library was successfully used, however, to implement the algorithms in the current work as well as those presented by Grimley-Evans (1997) and Kiraz (1997a). It must be stressed that the finite-state calculus library, as well as the rule and lexicon compilers, are prototype implementations written for research purposes. An interpreter version of the work presented here was described earlier (Kiraz 1996). The interpreter works on rules directly and can handle larger grammars.

Appendix B: Beesley's Bracketing Algorithm vs. Kiraz's Multitape Algorithm

B.1 Using Bell Labs' Lextools

This test was performed using the Lextools lexical compiler. Each line in the input file is an extended regular expression. The compiler turns each line into an FSA, then takes the union of all FSAs. The file may contain a header, surrounded between two %%s, for alias definitions. In Beesley's case, the following source file was used (only two roots and two vocalisms are shown):

```
%%
$(NotRoot) = {Sigma} - ({<}|{>}) ;              // This is A in eq. 13
$(Con) = {<} {Letters} {>} ;                    // See eq. 16
$(ktb) = ({<}k{>} {<}t{>} {<}b{>})~$(NotRoot) ; // Root ktb; see eq. 13
$(qbr) = ({<}q{>} {<}b{>} {<}r{>})~$(NotRoot) ; // Root qbr
$(aa) = $(Con)a$(Con)a$(Con) ;                  // Pattern CaCaC; eq. 15 with
                                                //    vowels instantiated
$(ui) = $(Con)u$(Con)i$(Con) ;                  // Pattern CuCiC
%%
$(ktb) & $(aa)                                  // Intersection for /katab/
$(ktb) & $(ui)                                  // Intersection for /kutib/
$(qbr) & $(aa)                                  // Intersection for /qabar/
$(qbr) & $(ui)                                  // Intersection for /qubir/
```

Each line in the header consists of `$(name)`, followed by `=`, followed by an extended regular expression. The compiler builds an FSA for the expression and stores it under `name`. In regular expressions, curly brackets are used to denote multicharacter symbols (e.g., `{Sigma}`) or special symbols (e.g., `<` and `>` which otherwise are used for weights in weighted FSAs). The operators are: `-` for subtraction, `|` for union, `~` for insert or ignore, `&` for intersection, and `$(name)` for reference to a machine already defined in the header.

The implementation of the Lextools insert operator was modified to provide for a fair representation of Beesley's method. The initial runs took quite some time (many

hours for 100 roots!) since the insert operator was initially implemented by the expression $Range(A \circ (\Sigma \cup \epsilon{:}B)^*)$ for inserting B into A (Kaplan and Kay 1994). The new algorithm inserts B directly into A by iterating over states and adding new states and arcs. Additionally, since Beesley's algorithm makes heavy use of alias definitions, access to them was enhanced by applying a binary search mechanism rather than Lextools's original linear search.

For Kiraz's method, two source files were used. The root file is simply a list of all roots, e.g.,

```
ktb             // root ktb
qbr             // root qbr
```

and the pattern file is a listing of all vocalisms, e.g.,

```
aa              // vocalism aa
ui              // vocalism ui
```

The two FSAs generated from the two files were fed into a regular expression compiler to compute expression (2) (Section 4.1).

Table 3 (Section 6.2) shows the times spent on compiling up to 3,000 roots with the 24 patterns described in (Beesley, forthcoming). The test was performed on a MIPS R5000 180 MHz based Unix system with a memory size of 64 MB.

Caveat is required here: To the disadvantage of Kiraz, the two FSAs resulting from the root and vocalism files are saved on disk, then loaded again by the regular expression compiler to compute their cross product adding unnecessary expensive I/O operations. Otherwise, the results would be even more in favor of the multitiered model.

B.2 Using van Noord's FSA Utilities (by Martin Jansche)

An independent comparison between the approach described here and the one in (Beesley, forthcoming) was carried out using van Noord's FSA Utilities (van Noord 1997) (we use its notation for regular expressions throughout this section). We compared the time spent on compiling lexica of various sizes for both frameworks. Given a set of roots and a set of patterns, each compiled into a finite automaton as outlined in Section 4.1, the key difference between the work of Kiraz and that of Beesley is that the former uses the cross product operation to compile the full lexicon, while the latter uses intersection. Since these two operations are very similar, one would not expect much of a difference if the elements in each sublexicon were the same. However, the different formal renderings of roots and patterns in the two approaches results in a significant difference.

	Beesley	Kiraz
lexicon by	intersection	cross product
root e.g.	`ignore([<,k,>,<,t,>,<,b,>], ? -{<,>})`	`[k,t,b]`
pattern e.g.	`[<,?,>,u,<,?,>,i,<,?,>]`	`['C',u,'C',i,'C']`

Using Beesley's approach directly made the task of compiling the root lexicon intractable for more than 10 roots. After modifications to Beesley's version—such as intersecting the disjunction of roots with the disjunction of patterns once,[5] delimiting

5 We realize that since roots do not apply to all patterns, but to lexically defined subsets, applying intersection once does not work in practice. It was applied here to enhance the performance of Beesley's algorithm.

root consonants with one special symbol only, and inserting other symbols only between root consonants rather than at arbitrary positions so that a typical root now has the shape `[sym*,<,k,sym*,<,t,sym*,<,b,sym*]`, where `syms` expands to `(? - <)`—it became tractable, but was still much slower than the alternative discussed here.

Both approaches were implemented as Prolog code and macros on top of the FSA Utilities. The two implementations share common code that contains, among other things, a database of roots and patterns in the form

```
pattern(['C',a,'C',a,'C']).  % Form I, Perfect Active
pattern(['C',u,'C',i,'C']).  % Form I, Perfect Passive
%%  etc.
root(k,t,b).
root(p,n,q).
%%  etc.
```

The data represented there are transformed in different ways by the two implementations:

```
%%  Kiraz
root([X,Y,Z]) :- root(X,Y,Z).
%%  Beesley
root([sym*,<,X,sym*,<,Y,sym*,<,Z,sym*]) :- root(X,Y,Z).
macro(sym, ? - <).  %% Beesley's "nonRoot"
```

There is a common macro `sublexicon(Pred,N)` that calls a metapredicate like `findall/3` to find the first `N` solutions to a call to `Pred(W)` and collects all the `W`s thus obtained into a big disjunction. The compilation of the lexicon is then very simple:

```
%%  Kiraz
macro(lexicon(R,P),
   ignore(sublexicon(pattern,P),0) x ignore(sublexicon(root,R),0)).
%%  Beesley
macro(lexicon(R,P), sublexicon(pattern,P) & sublexicon(root,R)).
macro('C', [<,sym]).
```

While the pattern database is shared between the two implementations, the meaning of `'C'` within the patterns is different: for Kiraz, it is just an ordinary symbol, but for Beesley it expands into a marker for a root consonant followed by any other symbol (the root consonant itself).

Table 3 shows the times (in seconds) spent on compiling full lexica with different numbers of roots and the 24 patterns described in (Beesley, forthcoming). Each number is the minimum obtained after several trials with the "walltime" statistics of SICStus Prolog 3.7.1 running on an Intel Pentium 233 MHz based Linux system.

References

Bat-El, O. 1989. *Phonology and Word Structure in Modern Hebrew*. Ph.D. thesis, University of California at Los Angeles.

Bear, J. 1988 Morphology with two-level rules and negative rule features. In *COLING-88: Papers Presented to the 12th International Conference on Computational Linguistics*, volume 1, pages 28–31.

Beesley, K. 1990. Finite-state description of Arabic morphology. In *Proceedings of the Second Cambridge Conference: Bilingual Computing in Arabic and English*.

Beesley, K. 1991. Computer analysis of Arabic morphology: A two-level approach with detours. In B. Comrie and M Eid, editors, *Perspectives on Arabic Linguistics III: Papers from the Third Annual Symposium on Arabic Linguistics*. John Benjamins, Amsterdam, pages 155–72.

Beesley, K. 1996. Arabic finite-state morphological analysis and generation. In *COLING-96: Papers Presented to the 16th International Conference on Computational Linguistics*, volume 1, pages 89–94.

Beesley, K. 1998a. Arabic morphological analysis on the Internet. In *Proceedings of the 6th International Conference and Exhibition on Multi-Lingual Computing*, pages 3.1.1–10, Cambridge.

Beesley, K. 1998b. Arabic morphology using only finite-state operations. In M. Rosner, editor, *Proceedings of the Workshop on Computational Approaches to Semitic Languages*, pages 50–57, Montreal.

Beesley, K. 1998c. Constraining separated morphotactic dependencies in finite-state grammars. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pages 118–127, Ankara, Turkey.

Beesley, K. Forthcoming. Arabic stem morphotactics via finite-state intersection. In E. Benmamoun, editor, *Perspectives on Arabic Linguistics XII: Papers from the Twelfth Annual Symposium on Arabic Linguistics*, pages 85–100. John Benjamins, Amsterdam.

Beesley, K., T. Buckwalter, and S. Newton. 1989. Two-level finite-state analysis of Arabic morphology. In *Proceedings of the Seminar on Bilingual Computing in Arabic and English*. The Literary and Linguistic Computing Centre, Cambridge.

Bird, S. and T. Ellison. 1992. One-level phonology: Autosegmental representations and rules as finite-state automata. Technical Report Research Paper EUCCS/RT-51, University of Edinburgh.

Bird, S. and T. Ellison. 1994. One-level phonology. *Computational Linguistics*, 20(1):55–90.

Chomsky, N. 1951. Morphophonemics of modern Hebrew. Master's thesis, University of Pennsylvania. Published by Garland Press, New York, 1979.

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

Daciuk, J., S. Mihov, B. Watson, and R. Watson. 2000. Incremental construction of minimal acyclic finite-state automata. *Computational Linguistics*, 26(1):3–16.

Elgot, C. and J. Mezei, 1965. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9:47–68.

Fischer, P. 1965. Multi-tape and infinite-state automata—A survey. *Communications of the ACM*, 8:799–805.

Goldsmith, J. 1976. *Autosegmental Phonology*. Ph.D. thesis, MIT. Published as *Autosegmental and Metrical Phonology*, Oxford, 1990.

Grimley-Evans, E. 1997. Approximating context-free grammars with a finite-state calculus. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 452–9, Madrid, Spain.

Grimley-Evans, E., G. Kiraz, and S. Pulman. 1996. Compiling a partition-based two-level formalism. In *COLING-96: Papers Presented to the 16th International Conference on Computational Linguistics*, volume 1, pages 454–59.

Hammond, M. 1988. Templatic transfer in Arabic broken plurals. *Natural Language and Linguistic Theory*, 6:247–70.

Harris, Z. 1941. Linguistic structure of Hebrew. *Journal of the American Oriental Society*, 62:143–67.

Kaplan, R. and M. Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–78.

Karttunen, L. 1993. Finite-state lexicon compiler. Technical Report, XEROX Palo Alto Research Center, April.

Karttunen, L. 1997. The replace operator. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*. MIT Press, chapter 4, pages 117–47.

Karttunen, L. and K. Beesley. 1992. Two-level rule compiler. Technical Report, XEROX Palo Alto Research Center.

Karttunen, L., R. Kaplan, and A Zaenen. 1992. Two-level morphology with composition. In *COLING-92: Papers Presented to the 15th* [sic] *International Conference on Computational Linguistics*, volume 1, pages 141–48, Nantes, France. International Committee on Computational Linguistics.

Kataja, L. and K. Koskenniemi. 1988. Finite state description of Semitic morphology. In *COLING-88: Papers Presented to the 12th International Conference on Computational Linguistics*, volume 1, pages 313–15.

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

Kay, M. and R. Kaplan. 1983. Word recognition. Unpublished paper. The core ideas are published in Kaplan and Kay (1994).

Kiraz, G. 1994a. *Lexical Tools to the Syriac New Testament*. JSOT Manuals 7. Sheffield Academic Press.

Kiraz, G. 1994b. Multi-tape two-level morphology: A case study in Semitic non-linear morphology. In *COLING-94: Papers Presented to the 15th International Conference on Computational Linguistics*, volume 1, pages 180–86.

Kiraz, G. 1996. ṢEMḤE: A generalised two-level system. In *Proceedings of the 34th Annual Meeting*, pages 159–66, Association for Computational Linguistics.

Kiraz, G. 1997a. Compiling regular formalisms with rule features into finite-state automata. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 329–36.

Kiraz, G. 1997b. Lextools 2.0: Tools for finite-state linguistic analysis. Technical Report 011334-970625-04TM, Bell Laboratories.

Kiraz, G. 1997c. Linearization of nonlinear lexical representations. In J. Coleman, editor, *Proceedings of the Third Meeting of the ACL Special Interest Group in Computational Phonology*, pages 57–62.

Kiraz, G. In press. *Computational Nonlinear Morphology: With Emphasis on Semitic Languages*. Cambridge University Press.

Kiraz, G. and E. Grimley-Evans. 1998. Multi-tape automata for speech and language systems: A Prolog implementation. In Derick Wood and Sheng Yu, editors, *Automata Implementation*. Lecture Notes in Computer Science, Number 1436. Springer Verlag, pages 87–103.

Kornai, A. *Formal Phonology*. Ph.D. thesis, Stanford University. Published in 1995.

Kornai, A. 1991. *Formal Phonology*. Garland Publishing.

Koskenniemi, K. 1983. *Two-Level Morphology*. Ph.D. thesis, University of Helsinki.

Lavie, A., A. Itai, and U. Ornan. 1990. On the applicability of two level morphology to the inflection of Hebrew verbs. In Y. Choueka, editor, *Literary and Linguistic Computing 1988: Proceedings of the 15th International Conference*, pages 246–60.

McCarthy, J. 1979. *Formal Problems in Semitic Phonology and Morphology*. Ph.D. thesis, MIT, Cambridge, MA.

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

McCarthy, J. 1986. OCP effects: Gemination and antigemination. *Linguistic Inquiry*, 17:207–63.

McCarthy, J. 1993. Template form in prosodic morphology. In Stvan, L. et al., editors, *Papers from the Third Annual Formal Linguistics Society of Midamerica Conference*, pages 187–218. Indiana University Linguistics Club, Bloomington.

McCarthy, J. and A. Prince. 1990a. Foot and word in prosodic morphology: The Arabic broken plural. *Natural Language and Linguistic Theory*, 8:209–83.

McCarthy, J. and A. Prince. 1990b. Prosodic morphology and templatic morphology. In M. Eid and J. McCarthy, editors, *Perspectives on Arabic Linguistics II: Papers from the Second Annual Symposium on Arabic Linguistics*. John Benjamins, Amsterdam, pages 1–54.

McCarthy, J. and A. Prince. 1995. Prosodic morphology. In J. Goldsmith, editor, *The Handbook of Phonological Theory*. Blackwell, chapter 9, pages 318–66.

Mohri, M. and R. Sproat. 1996. An efficient compiler for weighted rewrite rules. In *Proceedings of the 34th Annual Meeting*, pages 231–8. Association for Computational Linguistics.

Moscati, S., A. Spitaler, E. Ullendorff, and W. von Soden. 1969. *An Introduction to the Comparative Grammar of the Semitic Languages: Phonology and Morphology*. Porta Linguarum Orientalium. Otto Harrassowitz, Wiesbaden, Second edition.

Pulleyblank, D. 1986. *Tone in Lexical Phonology*. Studies in Natural Language and Linguistic Theory. Reidel.

Pulman, S. and M. Hepple. 1993. A feature-based formalism for two-level phonology: A description and implementation. *Computer Speech and*

Language, 7:333–58.
Rabin, M. and D. Scott. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–25. Reprinted in E. Moore, editor, *Sequential Machines*. Addison-Wesley, Reading, MA. 1964, pages 63–91.
Ritchie, G., A. Black, G. Russell, and S. Pulman. 1992. *Computational Morphology: Practical Mechanisms for the English Lexicon*. MIT Press, Cambridge, MA.
Ruessink, H. 1989. Two level formalisms. Technical Report 5, Utrecht Working Papers in NLP.
Spencer, A. 1991. *Morphological Theory*. Basil Blackwell.
Sprout, R. 1992. *Morphology and Computation*. MIT Press, Cambridge, MA.
Sprout, R. 1995. Lextools: Tools for finite-state linguistic analysis. Technical Report 11522-951108-10TM, Bell Laboratories.
Sprout, R., editor. 1997. *Multilingual Text-to-Speech Synthesis: The Bell Labs Approach*. Kluwer, Boston, MA.
van Noord, Gertjan. 1997. FSA Utilities: A toolbox to manipulate finite-state automata. In Darrell Raymond, Derick Wood, and Sheng Yu, editors, *Automata Implementation*, Lecture Notes in Computer Science, Number 1260. Springer Verlag, pages 87–108.
Wiebe, B. 1992. Modelling autosegmental phonology with multi-tape finite state transducers. Master's thesis, Simon Fraser University.