Compiling Regular Formalisms with Rule Features into Finite-State Automata

George Anton Kiraz
Bell Laboratories

Abstract

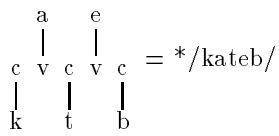
This paper presents an algorithm for the compilation of regular formalisms with rule features into finite-state automata. Rule features are incorporated into the right context of rules. This general notion can also be applied to other algorithms which compile regular rewrite rules into automata.

1 Introduction

The past few years have witnessed an increased interest in applying finite-state methods to language and speech problems. This in turn generated interest in devising algorithms for compiling rules which describe regular languages/relations into finite-state automata.

It has long been proposed that regular formalisms (e.g., rewrite rules, two-level formalisms) accommodate rule features which provide for finer and more elegant descriptions (Bear, 1988). Without such a mechanism, writing complex grammars (say two-level grammars for Syriac or Arabic morphology) would be difficult, if not impossible. Algorithms which compile regular grammars into automata (Kaplan and Kay, 1994; Mohri and Sproat, 1996; Grimley-Evans, Kiraz, and Pulman, 1996) do not make use of this important mechanism. This paper presents a method for incorporating rule features in the resulting automata.

The following Syriac example is used here, with the infamous Semitic root {ktb} 'notion of writing'. The verbal *pa"el* measure[1], /katteb/[2] 'wrote CAUSATIVE ACTIVE', is derived from the following morphemes: the pattern {cvcvc} 'verbal pattern', the above mentioned root, and the vocalism {ae} 'ACTIVE'. The morphemes produce the following underlying form:[3]



= */kateb/

/katteb/ is derived then by the gemination, implying CAUSATIVE, of the middle consonant, [t].[4]

The current work assumes knowledge of regular relations (Kaplan and Kay, 1994). The following convention has been adopted. Lexical forms (e.g., morphemes in morphology) appear in braces, { }, phonological segments in square brackets, [], and elements of tuples in angle brackets, ⟨ ⟩.

Section 2 describes a regular formalism with rule features. Section 3 introduce a number of mathematical operators used in the compilation process. Sections 4 and 5 present our algorithm. Finally, section 6 provides an evaluation and some concluding remarks.

2 Regular Formalism with Rule Features

This work adopts the following notation for regular formalisms, cf. (Kaplan and Kay, 1994):

$$\tau \{ \Rightarrow, \Leftarrow, \Leftrightarrow \} \lambda \_\_ \rho \qquad (1)$$

where τ, λ and ρ are n-way regular expressions which describe same-length relations.[5] (An n-way regular expression is a regular expression whose terms

[1] Syriac verbs are classified under various measures (i.e., forms), the basic ones being *p'al, pa"el* and *'af'el*.

[2] Spirantization is ignored here; for a discussion on Syriac spirantization, see (Kiraz, 1995).

[3] This analysis is along the lines of (McCarthy, 1981) – based on autosegmental phonology (Goldsmith, 1976).

[4] This derivation is based on the linguistic model proposed by (Kiraz, 1996).

[5] More 'user-friendly' notations which allow mapping expressions of unequal length (e.g., (Grimley-Evans, Kiraz, and Pulman, 1996)) are mathematically equivalent to the above notation after rules are converted into same-

R1	k:c_1:k:0	⇒	__
R2	b:c_3:b:0	⇒	__
R3	a:v:0:a	⇒	__
R4	e:v:0:e	⇒	__
R5	t:c_2:t:0 t:0:0:0	⇔	__
	⟨[cat=verb], [measure=pa"el], []⟩		
R6	t:c_2:t:0	⇔	__
	⟨[cat=verb], [measure=p'al], []⟩		
R7	0:v:0:a	⇔	__ t:c2:t:0 a:v:0:a

Figure 1: Simple Syriac Grammar

are n-tuples of alphabetic symbols or the empty string ϵ. A same-length relation is devoid of ϵ. For clarity, the elements of the n-tuple are separated by colons: e.g., *a:b:c* q:r:s* describes the 3-relation $\{ \langle a^m q, b^m r, c^m s \rangle \mid m \geq 0 \}$. Following current terminology, we call the first j elements 'surface'[6] and the remaining elements 'lexical'.) The arrows correspond to context restriction (CR), surface coercion (SC) and composite rules, respectively. A compound rule takes the form

$$\tau \{ \Rightarrow, \Leftarrow, \Leftrightarrow \} \lambda^1 \_\_ \rho^1; \lambda^2 \_\_ \rho^2; \ldots \qquad (2)$$

To accommodate for rule features, each rule may be associated with an $(n-j)$-tuple of feature structures, each of the form

$$[attribute_1{=}val_1, attribute_2{=}val_2, \ldots] \qquad (3)$$

i.e., an unordered set of *attribute=val* pairs. An *attribute* is an atomic label. A *val* can be an atom or a variable drawn from a predefined finite set of possible values.[7] The ith element in the tuple corresponds to the $(j+i)$th element in rule expressions. As a way of illustration, consider the simplified grammar in Figure 1 with $j = 1$.

The four elements of the tuples are: surface, pattern, root, and vocalism. R1 and R2 sanction the first and third consonants, respectively. R3 and R4 sanction vowels. R5 is the gemination rule; it is only triggered if the given rule features are satisfied: [cat=verb] for the first lexical element (i.e., the pattern) and [measure=pa"el] for the second element (i.e., the root). The rule also illustrates that τ can be a sequence of tuples. The derivation of /katteb/ is illustrated below:

length descriptions at some preprocessing stage.

[6] In natural language, usually $j = 1$.

[7] It is also possible to extend the above formalism in order to allow *val* to be a category-feature structure, though that takes us beyond finite-state power.

Sublexicon	Entry	Feature Structure
Pattern	$c_1 v c_2 v c_3$	[cat=verb]
Root	ktb	[measure=(p'al,pa"el)†]
Vocalism	ae	[voice=active, measure=pa"el]
	aa	[voice=active, measure=p'al]

†Parenthesis denote disjunction over the given values.

Figure 2: Simple Syriac Lexicon

0	a	0 0	e	0	*vocalism*
k	0	t 0	0	b	*root*
c_1	v	c_2 0	v	c_3	*pattern*
1	3	5	4	2	
k	a	tt	e	b	*surface*

The numbers between the lexical expressions and the surface expression denote the rules in Figure 1 which sanction the given lexical-surface mappings.

Rule features play a role in the semantics of rules: a ⇒ states that if the contexts *and* rule features are satisfied, the rule is triggered; a ⇐ states that if the contexts, lexical expressions *and* rule features are satisfied, then the rule is applied. For example, although R5 is devoid of context expressions, the rule is composite indicating that if the root measure is *pa"el*, then gemination must occur and vice versa. Note that in a compound rule, each set of contexts is associated with a feature structure of its own.

What is meant by 'rule features are satisfied'? Regular grammars which make use of rule features normally interact with a lexicon. In our model, the lexicon consists of $(n-j)$ sublexica corresponding to the lexical elements in the formalism. Each sublexical entry is associate with a feature structure. Rule features are satisfied if they match the feature structures of the lexical entries containing the lexical expressions in τ, respectively. Consider the lexicon in Figure 2 and rule R5 with τ = t:c_2:t:0 t:0:0:0 and the rule features ⟨[cat=verb], [measure=pa"el], []⟩. The lexical entries containing τ are {$c_1 v c_2 v c_3$} and {ktb}, respectively. For the rule to be triggered, [cat=verb] of the rule must match with [cat=verb] of the lexical entry {$c_1 v c_2 v c_3$}, and [measure=pa"el] of the rule must match with [measure=(p'al,pa"el)] of the lexical entry {ktb}.

As a second illustration, R6 derives the simple *p'al* measure, /ktab/. Note that in R5 and R6,

1. the lexical expressions in both rules (ignoring 0s) are equivalent,

2. both rules are composite, and

3. they have *different* surface expression in τ.

In traditional rewrite formalism, such rules will be contradicting each other. However, this is not the case here since R5 and R6 have different rule features. The derivation of this measure is shown below (R7 completes the derivation deleting the first vowel on the surface[8]):

0	a	0	a	0	*vocalism*
k	0	t	0	b	*root*
c_1	v	c_2	v	c_3	*pattern*
1	7	6	3	2	
k	0	t	a	b	*surface*

Note that in order to remain within finite-state power, both the attributes and the values in feature structures must be atomic. The formalism allows a value to be a variable drawn from a predefined finite set of possible atomic values. In the compilation process, such variables are taken as the disjunction of all possible predefined values.

Additionally, this version of rule feature matching does not cater for rules whose τ span over two lexical forms. It is possible, of course, to avoid this limitation by having rule features match the feature structures of both lexical entries in such cases.

3 Mathematical Preliminaries

We define here a number of operations which will be used in our compilation process.

If an operator `Op` takes a number of arguments $(a_1, \cdots, a_k)$, the arguments are shown as a subscript, e.g. $\texttt{Op}_{(a_1,\cdots,a_k)}$ – the parentheses are ignored if there is only one argument. When the operator is mentioned without reference to arguments, it appears on its own, e.g. `Op`.

Operations which are defined on tuples of strings can be extended to sets of tuples and relations. For example, if S is a tuple of strings and $\texttt{Op}(S)$ is an operator defined on S, the operator can be extended to a relation R in the following manner

$$\texttt{Op}(R) = \{\, \texttt{Op}(S) \mid S \in R \,\}$$

Definition 3.1 (Identity) Let L be a regular language. $\texttt{Id}_n(L) = \{\, X \mid X$ is an n-tuple of the form $\langle x, \cdots, x \rangle, x \in L \,\}$ is the n-way identity of L.[9]

Remark 3.1 If `Id` is applied to a string s, we simply write $\texttt{Id}_n(s)$ to denote the n-tuple $\langle s, \ldots, s \rangle$.

Definition 3.2 (Insertion) 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. $\texttt{Insert}_m^{-1} \circ \texttt{Insert}_m(R) = R$ removes all such instances if m is disjoint from Σ.[10]

Remark 3.2 We can define another form of `Insert` where the elements in m are tuples of symbols as follows: Let R be a regular relation over the alphabet Σ and let m be a set of tuples of symbols not necessarily in Σ. $\texttt{Insert}_m(R)$ inserts a, for all $a \in m$, freely throughout R.

Definition 3.3 (Substitution) Let S and S' be same-length n-tuples of strings over the alphabet $(\Sigma \times \cdots \times \Sigma)$, $I = \texttt{Id}_n(a)$ for some $a \in \Sigma$, and $S = S_1 I S_2 I \cdots S_k, k \geq 1$, such that S_i does not contain I – i.e. $S_i \in ((\Sigma \times \cdots \times \Sigma) - \{I\})^*$. $\texttt{Substitute}_{(S',I)}(S) = S_1 S' S_2 S' \cdots S_k$ substitutes every occurrence of I in S with S'.

Definition 3.4 (Projection) Let $S = \langle s_1, \ldots, s_n \rangle$ be a tuple of strings. $\texttt{Project}_i(S)$, for some $i \in \{\, 1, \ldots, n \,\}$, denotes the tuple element s_i. $\texttt{Project}_i^{-1}(S)$, for some $i \in \{\, 1, \ldots, n \,\}$, denotes the $(n-1)$-tuple $\langle s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n \rangle$.

The symbol π denotes 'feasible tuples', similar to 'feasible pairs' in traditional two-level morphology. The number of surface expressions, j, is always 1.

The operator $\circ$ represents mathematical composition, not necessarily the composition of transducers.

4 Compilation without Rule Features

The current algorithm is motivated by the work of (Grimley-Evans, Kiraz, and Pulman, 1996).[11]

Intuitively, the automata is built by three approximations as follows:

1. Accepting τs irrespective of any context.

2. Adding context restriction ($\Rightarrow$) constraints making the automata accept only the sequences which appear in contexts described by the grammar.

3. Forcing surface coercion constraints ($\Leftarrow$) making the automata accept all and only the sequences described by the grammar.

[8] Short vowels in open unstressed syllables are deleted in Syriac.

[9] This is a generalization of the operator *Id* in (Kaplan and Kay, 1994).

[10] This is similar to the operator *Intro* in (Kaplan and Kay, 1994).

[11] The subtractive approach for compiling rules into FSAs was first suggested by Edmund Grimley-Evans.

4.1 Accepting τs

Let $\mathcal{T}$ be the set of all τs in a regular grammar, $\wp$ be an auxiliary boundary symbol (not in the grammar's alphabets) and $\wp' = \mathtt{Id}_n(\wp)$. The first approximation is described by

$$Centers = \wp'[(\bigcup_{\tau \in \mathcal{T}} \tau)\wp']^* \tag{4}$$

Centers accepts the symbols, $\wp'$, followed by zero or more τs, each (if any) followed by $\wp'$. In other words, the machine accepts all centers described by the grammar (each center surrounded by $\wp'$) irrespective of their contexts.

It is implementation dependent as to whether $\mathcal{T}$ includes other correspondences which are not explicitly given in rules (e.g., a set of additional feasible centers).

4.2 Context Restriction Rules

For a given compound rule, the set of relations in which τ is *invalid* is

$$Restrict(\tau) = \pi^*\tau\pi^* - \bigcup_k \pi^*\lambda^k\tau\rho^k\pi^* \tag{5}$$

i.e., τ in any context minus τ in all valid contexts. However, since in §4.1 above, the symbol $\wp$ appears freely, we need to introduce it in the above expression. The result becomes

$$\begin{aligned} Restrict(\tau) \quad = \quad & \mathtt{Insert}_{\{\wp\}} \circ \\ & \pi^*\tau\pi^* - \bigcup_k \pi^*\lambda^k\tau\rho^k\pi^* \end{aligned} \tag{6}$$

The above expression is only valid if τ consists of only one tuple. However, to allow it to be a sequence of such tuples as in R5 in Figure 1, it must be

1. surrounded by $\wp'$ on both sides, and
2. devoid of $\wp'$.

The first condition is accomplished by simply placing $\wp'$ to the left and right of τ. As for the second condition, we use an auxiliary symbol, ω, as a place-holder representing τ, introduce $\wp$ freely, then substitute τ in place of ω. Formally, let ω be an auxiliary symbol (not in the grammar's alphabet), and let $\omega' = \mathtt{Id}_n(\omega)$ be a place-holder representing τ. The above expression becomes

$$\begin{aligned} Restrict(\tau) \quad = \quad & \mathtt{Substitute}(\tau, \omega') \circ \\ & \mathtt{Insert}_{\{\wp\}} \circ \\ & \pi^*\wp'\omega'\wp'\pi^* - \bigcup_k \pi^*\lambda^k\wp'\omega'\wp'\rho^k\pi^* \end{aligned} \tag{7}$$

For all τs, we subtract this expression from the automaton under construction, yielding

$$CR = Centers - \bigcup_\tau Restrict(\tau) \tag{8}$$

CR now accepts only the sequences of tuples which appear in contexts in the grammar (but including the partitioning symbols $\wp'$); however, it does not force surface coercion constraints.

4.3 Surface Coercion Rules

Let τ' represent the center of the rule with the *correct* lexical expressions and the *incorrect* surface expressions with respect to π^*,

$$\tau' = \overline{\mathtt{Project}_1(\tau)} \times \mathtt{Project}_1^{-1}(\tau) \tag{9}$$

The coerce relation for a compound rule can be simply expressed by[12]

$$\begin{aligned} Coerce(\tau') \quad = \quad & \mathtt{Insert}_{\{\wp\}} \circ \\ & \bigcup_k \pi^*\lambda^k\wp'\tau'\wp'\rho^k\pi^* \end{aligned} \tag{10}$$

The two $\wp'$s surrounding τ' ensure that coercion applies on at least one center of the rule.

For all such expressions, we subtract *Coerce* from the automaton under construction, yielding

$$SC = CR - \bigcup_\tau Coerce(\tau) \tag{11}$$

SC now accepts *all* and *only* the sequences of tuples described by the grammar (but including the partitioning symbols $\wp'$).

It remains only to remove all instances of $\wp$ from the final machine, determinize and minimize it. There are two methods for interpreting transducers. When interpreted as acceptors with n-tuples of symbols on each transition, they can be determinized using standard algorithms (Hopcroft and Ullman, 1979). When interpreted as a transduction that maps an input to an output, they cannot always be turned into a deterministic form (see (Mohri, 1994; Roche and Schabes, 1995)).

5 Compilation with Rule Features

This section shows how feature structures which are associated with rules and lexical entries can be incorporated into FSAs.

[12] A special case can be added for epenthetic rules.

Entry	Feature Structure
abcd	f_1
ef	f_2
ghi	f_3

Figure 3: Lexicon Example

5.1 Intuitive Description

We shall describe our handling of rule features with a two-level example. Consider the following analysis.

a	b	c	d	♭	e	f	♭	g	h	i	♭	*Lexical*
1	2	3	4	5	6	7	5	8	9	10	5	
a	b	c	d	0	e	f	0	g	h	i	0	*Surface*

The lexical expression contains the lexical forms {abcd}, {ef} and {ghi}, separated by a boundary symbol, ♭, which designates the end of a lexical entry. The numbers between the tapes represent the rules (in some grammar) which allow the given lexical-surface mappings.

Assume that the above lexical forms are associated in the lexicon with the feature structures as in Figure 3. Further, assume that each two-level rule $m, 1 \leq m \leq 10$, above is associated with the feature structure F_m. Hence, in order for the above two-level analysis to be valid, the following feature structures must match

All the structures ...	must match ...
F_1, F_2, F_3, F_4	f_1
F_6, F_7	f_2
F_8, F_9, F_{10}	f_3

Usually, boundary rules, e.g. rule 5 above, are not associated with feature structures, though there is nothing stopping the grammar writer from doing so.

To match the feature structures associated with rules and those in the lexicon we proceed as follows. Firstly, we suffix each lexical entry in the lexicon with the boundary symbol, ♭, and it's feature structure. (For simplicity, we consider a feature structure with instantiated values to be an atomic object of length one which can be a label of a transition in a FSA.)[13] Hence the above lexical forms become: 'abcd♭f_1', 'ef♭f_2', and 'ghi♭f_3'. Secondly, we incorporate a feature structure of a rule into the rule's right context, ρ. For example, if ρ of rule 1 above is b:b c:c, the context becomes

$$\text{b:b c:c } \pi^* \text{ 0:}F_1 \tag{12}$$

(this simplified version of the expression suffices for the moment). In other words, in order for a:a to be sanctioned, it must be followed by the sequence:

1. b:b c:c, i.e., the original right context;
2. any feasible tuple, π^*; and
3. the rule's feature structure which is deleted on the surface, 0:F_1.

This will succeed if only if F_1 (of rule 1) and f_1 (of the lexical entry) were identical. The above analysis is repeated below with the feature structures incorporated into ρ.

a	b	c	d	♭	f_1	e	f	♭	f_2	g	h	i	♭	f_3	*Lexical*
1	2	3	4	5		6	7	5		8	9	10	5		
a	b	c	d	0	0	e	f	0	0	g	h	i	0	0	*Surface*

As indicated earlier, in order to remain within finite-state power, all values in a feature structure must be instantiated. Since the formalism allows values to be variables drawn from a predefined finite set of possible values, variables entered by the user are replaced by a disjunction over all the possible values.

5.2 Compiling the Lexicon

Our aim is to construct a FSA which accepts any lexical entry from the ith sublexicon on its $j+i$th tape.

A lexical entry μ (e.g., morpheme) which is associated with a feature structure ϕ is simply expressed by $\mu ♭ \phi$, where ♭ is a (morpheme) boundary symbol which is not in the alphabet of the lexicon. The expression of sublexicon i with r entries becomes,

$$L_i = \bigcup_r \mu^r ♭ \phi^r \tag{13}$$

We also compute the feasible feature structures of sublexicon i to be

$$F_i = \bigcup_r \phi^r \tag{14}$$

and the overall feasible feature structures on all sublexica to be

$$\Phi = 0^* \times F_1 \times F_2 \times \cdots \tag{15}$$

The first element deletes all such features on the surface. For convenience in later expressions, we incorporate features with π as follows

$$\pi_\phi = \pi \cup \Phi \tag{16}$$

The overall lexicon can be expressed by,[14]

$$Lexicon = L_1 \times L_2 \times \cdots \tag{17}$$

[13] As to how this is done is a matter of implementation.

[14] To make the lexicon describe equal-length relations, a special symbol, say 0, is inserted throughout.

The operator $\times$ creates one large lexicon out of all the sublexica. This lexicon can be substantially reduced by intersecting it with $\texttt{Project}_1^{-1}(\pi_\phi)^*$.

If a two-level grammar is compiled into an automaton, denoted by $Gram$, and a lexicon is compiled into an automaton, denoted by Lex, the automaton which enforces lexical constraints on the language is expressed by

$$L = (\texttt{Project}_1(\pi)^* \times Lex) \cap Gram \tag{18}$$

The first component above is a relation which accepts any surface symbol on its first tape and the lexicon on the remaining tapes.

5.3 Compiling Rules

A compound regular rule with m context-pairs and m rule features takes the form

$$\tau \quad \{\Rightarrow, \Leftarrow, \Leftrightarrow\} \quad \lambda^1 \_\_ \rho^1; \lambda^2 \_\_ \rho^2; \ldots; \lambda^m \_\_ \rho^m \quad [\phi^1, \phi^2, \ldots, \phi^m] \tag{19}$$

where τ, λ^k, and $\rho^k, 1 \leq k \leq m$ are like before and ϕ^k is the tuple of feature structures associated with rule k.

The following modifications to the procedure given in section 4 are required.

Forgetting contexts for the moment, our basic machine scans sequences of tuples (from $\mathcal{T}$), but requires that any sequence representing a lexical entry be followed by the entry's feature structure (from Φ). This is achieved by modifying eq. 4 as follows:

$$Centers = \wp'([(\bigcup_{\tau \in \mathcal{T}} \tau)\wp']^+ \Phi \wp')^* \tag{20}$$

The expression accepts the symbols, $\wp'$, followed by zero or more occurrences of the following:

1. one or more τ, each followed by $\wp'$, and
2. a feature tuple in Φ followed by $\wp'$.

In the second and third phases of the compilation process, we need to incorporate members of Φ freely throughout the contexts. For each λ^k, we compute the new left context

$$\mathcal{L}^k = \texttt{Insert}_\Phi(\lambda^k) \tag{21}$$

The right context is more complicated. It requires that the first feature structure to appear to the right of τ is ϕ^k. This is achieved by the expression,

$$\mathcal{R}^k = \texttt{Insert}_\Phi(\rho^k) \cap \pi^* \phi^k \pi_\phi^* \tag{22}$$

The intersection with $\pi^* \phi^k \pi_\phi^*$ ensures that the first feature structure to appear to the right of τ is ϕ^k: zero or more feasible tuples, followed by ϕ^k, followed by zero or more feasible tuples or feature structures.

Now we are ready to modify the $Restrict$ relation. The first component in eq. 5 becomes

$$A = (\pi \cup \pi\Phi)^* \tau \pi_\phi^* \tag{23}$$

The expression allows Φ to appear in the left and right contexts of τ; however, at the left of τ, the expression $(\pi \cup \pi\Phi)$ puts the restriction that the first tuple at the left end must be in π, not in Φ.

The second component in eq. 5 simply becomes

$$B = \bigcup_k \pi_\phi^* \mathcal{L}^k \tau \mathcal{R}^k \pi_\phi^* \tag{24}$$

Hence, $Restrict$ becomes (after replacing τ with ω' in eq. 23 and eq. 24)

$$Restrict(\tau) = \texttt{Substitute}(\tau, \omega') \circ \texttt{Insert}_{\{\wp\}} \circ A - B \tag{25}$$

In a similar manner, the $Coerce_R$ relation becomes

$$Coerce(\tau') = \texttt{Insert}_{\{\wp\}} \circ \bigcup_k \pi_\phi^* \mathcal{L}^k \wp' \tau' \wp' \mathcal{R}^k \pi_\phi^* \tag{26}$$

6 Conclusion and Future Work

The above algorithm was implemented in Prolog and was tested successfully with a number of sample-type grammars. In every case, the automata produced by the compiler were manually checked for correctness, and the machines were executed in generation mode to ensure that they did not over generate.

It was mentioned that the algorithm presented here is based on the work of (Grimley-Evans, Kiraz, and Pulman, 1996) rather than (Kaplan and Kay, 1994). It must be stated, however, that the intuitive ideas behind our compilation of rule features, viz. the incorporation of rule features in contexts, are independent of the algorithm itself and can be also applied to (Kaplan and Kay, 1994) and (Mohri and Sproat, 1996).

One issue which remains to be resolved, however, is to determine which approach for compiling rules into automata is more efficient: the standard method of (Kaplan and Kay, 1994) (also (Mohri and Sproat, 1996) which follows the same philosophy) or

Algorithm	Intersection (N^2)	Determinization (2^N)
KK	$(n-1)+3\sum_{i=1}^{n} k_i$	$8\sum_{i=1}^{n} k_i$
EKP	$1+\sum_{i=1}^{n} k_i$	$1+\sum_{i=1}^{n} k_i$

where n = number of rules in a grammar,
and k_i = number of contexts for rule $i, 1 \leq i \leq n$.

Figure 4: Statistics of Complex Operations

the subtractive approach of (Grimley-Evans, Kiraz, and Pulman, 1996).

The statistics of the usage of computationally expensive operations – viz., intersection (quadratic complexity) and determinization (exponential complexity) – in both algorithms are summarized in Figure 4 (KK = Kaplan and Kay, EKP = Grimley-Evans, Kiraz and Pulman). Note that complementation requires determinization, and subtraction requires one intersection and one complementation since

$$A - B = A \cap \overline{B} \tag{27}$$

Although statistically speaking the number of operations used in (Grimley-Evans, Kiraz, and Pulman, 1996) is less than the ones used in (Kaplan and Kay, 1994), only an empirical study can resolve the issue as the following example illustrates. Consider the expression

$$A = \overline{a_1 \cup a_2 \cup \cdots \cup a_n} \tag{28}$$

and the De Morgan's law equivalent

$$B = \overline{a_1} \cap \overline{a_2} \cap \cdots \cap \overline{a_n} \tag{29}$$

The former requires only one complement which results in one determinization (since the automata must be determinized before a complement is computed). The latter not only requires n complements, but also $n-1$ intersections. The worst-case analysis clearly indicates that computing A is much less expensive than computing B. Empirically, however, this is not the case when n is large and a_i is small, which is usually the case in rewrite rules. The reason lies in the fact that the determinization algorithm in the former expression applies on a machine which is by far larger than the small individual machines present in the latter expression.[15]

Another aspect of rule features concerns the morphotactic unification of lexical entries. This is best dealt with at the morphotactic level using a unification based formalism.

[15]This important difference was pointed out by one of the anonymous reviewers whom I thank.

Acknowledgments

I would like to thank Richard Sproat for commenting on an earlier draft. Many of the anonymous reviewers' comments proofed very useful. Mistakes, as always, remain mine.

References

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.

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

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.*

Hopcroft, J. and J. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley.

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

Kiraz, G. 1995. *Introduction to Syriac Spirantization.* Bar Hebraeus Verlag, The Netherlands.

Kiraz, G. [1996]. Syriac morphology: From a linguistic description to a computational implementation. In R. Lavenant, editor, *VIItum Symposium Syriacum 1996*, Forthcoming in Orientalia Christiana Analecta. Pontificio Institutum Studiorum Orientalium.

Kiraz, G. [Forthcoming]. *Computational Approach to Nonlinear Morphology: with emphasis on Semitic languages.* Cambridge University Press.

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

Mohri, M. 1994. On some applications of finite-state automata theory to natural language processing. Technical report, Institut Gaspard Monge.

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

Roche, E. and Y. Schabes. 1995. Deterministic part-of-speech tagging with finite-state transducers. *CL*, 21(2):227–53.