Languages Generated by Two-Level Morphological Rules*

Graeme Ritchie†
University of Edinburgh

The two-level model of morphology and phonology arose from work on finite-state machine descriptions of phonological phenomena. However, the two-level rule notation can be given a precise declarative semantics in terms of the segmentation of sequences of pairs of symbols, quite independently of any computational representation as sets of finite-state transducers. Thus defined, the two-level model can be shown to be less powerful, in terms of weak generative capacity, than parallel intersections of arbitrary finite-state transducers without empty transitions (the usual computational representation). However, if a special boundary symbol is permitted, the full family of regular languages can be generated. Two-level morphological grammars may, without loss of generality, be written in a simplified normal form. The set of two-level generated languages can be shown to be closed under intersection, but not under union or complementation.

1. Background

Koskenniemi (1983a, 1983b, 1984) proposed a rule-system for describing morphological regularities in a language, depending centrally on the idea of matching two sequences of symbols—a **lexical string** (made up of the lexical forms of morphemes) and a **surface string** (the sequence of characters in the normal, inflected, form of the word). (In general, symbols could be orthographic or phonological; here we shall abstract from this linguistic issue, and merely consider strings of atomic symbols, which could be phonemes, typewritten characters, or any other separate entities).

Koskenniemi (1983a) originally described the rules in two alternative forms—high-level rules and finite-state transducers. The high-level rules were given only an informal interpretation, and were used as an expository device to state the linguistic regularities more perspicuously. The formalism that was actually used to write and implement two-level morphological grammars was **parallel combinations of finite-state transducers**. Koskenniemi's own implementation was an interpreter for parallel transducers, which were directly written by the linguist as rules in their own right. Also, various linguistic analyses presented in Dalrymple et al. (1983) follow this approach, expressing rules as transition tables for transducers, and Antworth (1990) describes a recent implementation based wholly on transducers being written directly. Nevertheless, Koskenniemi conjectured that an automatic compilation procedure could be

* An earlier version of part of this work was presented at the Fourth Conference of the European Chapter of the Association for Computational Linguistics, Manchester, April 1989.
† Department of Artificial Intelligence, 80 South Bridge, Edinburgh EH1 1HN, Scotland, g.d.ritchie@ed.ac.uk

devised to transform the more readable high-level form into the more directly implementable parallel transducer form. Koskenniemi (1985) refined the notation and sketched a compilation method, although he still did not provide a detailed declarative definition of the meaning of the high-level rules. Various implementations that compile variants of the notation into arrangements of transducers have been developed (e.g. Karttunen et al. 1987; Ritchie et al. 1991).

The general view of this earlier work was that the rule notation was a mere "syntactic sugar" for parallel combinations of arbitrary transducers, and that there was no difference in meaning or power between the two formalisms.

This paper establishes the following:

(i) an alternative (declarative) statement of the meaning of the high-level rule notation is possible, without recourse to compilation into finite-state transducers;

(ii) the usual two-level morphological mechanism is more limited than arbitrary transducers in its ability to define relationships between strings;

(iii) the use of a special boundary symbol slightly increases the generative power of the model;

(iv) for any two-level morphological grammar, there is an equivalent one in a simpler **normal form**;

(v) the family of languages generated by two-level rules is not closed under union or complementation, but is closed under intersection.

2. The Two-Level Notation

The original notation proposed in Koskenniemi (1983a) included some rather complex notational conventions that have not survived into later versions. The formalization given here will deal only with the core ideas, as embodied in Koskenniemi (1985) (and other implementations such as Karttunen et al. 1987; Ritchie et al. 1991).

To understand how two-level rules operate, it is useful to consider four separate levels, as depicted in Figure 1. The topmost level contains **lexical forms**, which are strings of symbols in some suitable **lexical alphabet**. Each string can be thought of as the abstract representation of the phonology or orthography of an individual morpheme or lexeme. The second level consists of a notional automaton "tape" made up of any concatenation of strings from the first level (the lexicon), with special null symbols inserted anywhere between the symbols of the lexical alphabet (optionally). Hence a finite set of lexical strings characterizes an infinite set of possible "lexical tapes," produced by concatenation and insertion of nulls. The third level is a similar "surface tape" consisting of symbols in a **surface alphabet**, with the null symbol also appearing at arbitrary positions. The fourth level, at the bottom in Figure 1, represents the word as spelt (or phonetically represented) in the surface alphabet; the relationship between the surface tape and the surface form is that the latter can be produced from the former by removing all occurrences of the special null symbol. The ordinary

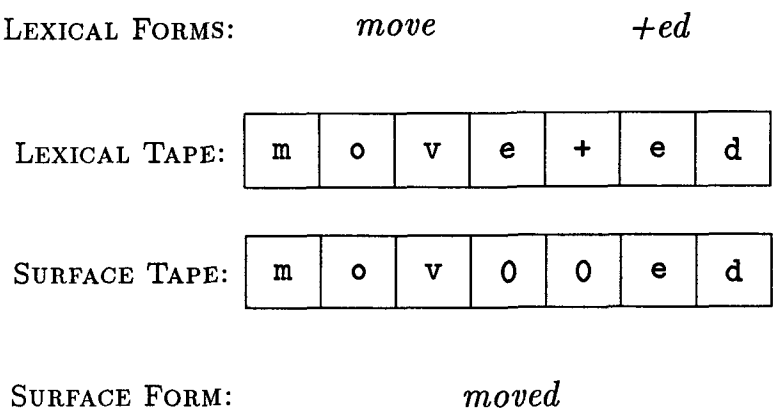

Figure 1
Lexical and surface tapes and forms.

surface form (such as `moved`) is related to a sequence of lexical forms (such as `move` and `+ed`) if a surface tape and a lexical tape exist such that:

1. the lexical tape is the concatenation of the lexical forms in question, possibly with the addition of nulls;
2. the surface tape contains the symbols of the surface form, possibly with the addition of nulls;
3. the two tapes have the same number of symbols, counting nulls;
4. the contents of the tapes are related by the two-level rules.

That is, the two-level rules define possible pairs of equal-length sequences (the two "tapes"), and the actual surface form and lexical forms are then related by general conventions acting on these sequences. The formalism as a whole can be used to describe phenomena involving a surface string of a different length from the lexical forms, via the conventions regarding null symbols, but the rules themselves are written as statements about symbol-strings of equal length. Hence, the rules can be viewed as statements about *sequences of symbol-pairs*, where each symbol-pair consists of one symbol from the lexical tape and its counterpart from the surface tape. Conventionally, symbol-pairs are written with an infix colon, rather than as a pair within parentheses. The rules specify which sequences of symbol-pairs are or are not valid, by supplying contexts (sequences of symbol-pairs) within which a particular symbol-pair may or must occur. For example, the rather trivial rule

```
e:0  =>  v:v  ___ +:0
```

would specify that the pair `e:0` could occur only if it were surrounded by `v:v` on the left and `+:0` on the right, which is a rough approximation to the phenomenon illustrated by Figure 1.

There are three types of rule, but they all have the same overall form—a single symbol-pair whose behaviour is being specified, followed by an operator (`=>`, `<=` or `<=>`), a left context, a punctuation ("___" in this paper) to indicate where the symbol pair appears relative to the context, and a right context.

The three types of rule are as follows:

Context Restriction (operator =>): states that the given symbol-pair can occur only in the given contexts, as in the simple example given above.

Surface Coercion (operator <=): states that if the contexts occur, *and* the lexical symbol is as given, then the surface symbol must be as specified. For example

```
e:l <= l:i ___ +:0
```

indicates that where a lexical `l` corresponds to a surface `i`, there is a pairing of lexical `+` with a surface null symbol one place further to the right, and the intervening symbol-pair has a lexical `e`, then the surface symbol in that pair *must* be `l`. This might be one of the rules describing the relationship between `possible+ity` and `possibility`.

Composite (operator <=>): a rule of this sort is merely an abbreviation of two separate rules (a context restriction rule and a surface coercion rule) made up of exactly the same symbol-pair, left context and right context. For example

```
i:y <=> =:= ___ e:0 +:0 i:i
```

states that lexical `i` and surface `y` are paired only in the context of anything on the left (if we assume that the "=" symbol means "any symbol") and a sequence of pairings on the right as given. This describes the association of lexical `tie+ing` with surface `tying`. The precise meaning of a composite rule is given by re-writing it as a context restriction rule and a surface coercion rule; composite rules will be ignored in the formalization below, as they add nothing to the abstract mechanism.

Fuller examples of the use of two-level rules can be found in Koskenniemi (1983a) and Ritchie et al. (1987, 1991).

The interpretation of a set of rules is as follows. For a sequence of symbol-pairs to be acceptable, if any symbol-pair in it was the subject of one or more context restriction rules, then at least one of these rules must apply to the surrounding sequence (i.e. the contexts must match). Also, if the surrounding contexts of any symbol-pair matched the context parts of any surface coercion rule, then the symbol-pair must obey that rule. That is, in a rule-set (a two-level morphological grammar), all the context restriction rules that have the same symbol-pair on the left-hand side of the $\Rightarrow$ operator were deemed to be a *disjunction* of constraints for that symbol-pair, and the surface coercion rules were a *conjunction* of constraints (regardless of which symbol-pair they constrained).

In specifying symbol-pairs in rules (whether in the central pair or within the contexts) various convenient abbreviations were available. Symbols denoting sets of characters could be used instead of individual characters, thereby making it easier to state rules that referred to "all vowels" or other classes. Variables were also available for conflating rules that were very similar.

In specifying contexts (left and right), it was possible to supply more complex expressions than just sequences of symbol-pairs. Essentially, regular expressions of symbol-pairs were allowed (**regular pair expressions** as Koskenniemi called them), since notation was available to state alternation (disjunction), optionality, and the occurrence of zero or more instances (Kleene star). Also, a single rule could contain several pairs of contexts, written as a disjunction of possibilities.

In the following illustrative example of a two-level morphological rule (taken from Ritchie et al. 1987), angle brackets indicate sequences of pairs and braces indicate alternatives; also, `C`, `V`, `C2`, and = represent subsets of the relevant symbol alphabets and + is an abstract symbol occurring in certain lexical forms.

```
e:0  <=> =:C2 ___ < +:0 V:= >
      or < C:C V:V> ___ <+:0 e:e>
      or {g:g c:c} ___ <+:0 {e:e i:i} >
      or l:0 ___ +:0
      or c:c ___ <+:0 a:0 t:t>
```

This rule describes the various contexts in which a lexical `e` is elided on the surface (as in `larger`, `continuing`, `raced`, `possibly`, or `reduction`).

The formalism here will not include symbolic mnemonics for sets of symbols, nor variables ranging over sets of symbols. The semantics of both these notations can be stated in terms of equivalent sets of rules without such abbreviatory conventions, so all that is required is a definition of the interpretation of rules containing only actual character symbols. We shall also (in some of our formalization) abstract away from the stipulation that the context-expressions must denote *regular* sets, since none of the proofs depend on that characteristic. (This is a slight generalization of the presentation in Ritchie 1989.)

One of the more peripheral aspects of two-level morphology is the role of the rules in segmenting surface input strings into lexical forms (i.e. the interface between a rule interpreter and a lexicon of morphemes). We shall ignore this issue here (but see Ritchie 1989 and Ritchie et al. 1991 for a formal statement of two possible lexical interfaces). Our formalization will deal solely with the way in which the two equal-length "tapes" are related, as this is the stage of the surface-to-lexicon mapping that is explicitly mediated by the rules.

3. Regular Relations

The two-level rule compiler of Karttunen et al. (1987) was based on sophisticated manipulation of regular expressions in order to define very precisely and rigorously the set of transducers produced by the compiler. Kaplan (1988) generalized some of this work further, by formalizing the algebraic manipulations of regular sets of sequences of pairs of symbols, which he called **regular relations**.[1] These relations bear the same relationship to finite-state transducers that regular languages bear to finite-state machines, and his formal definition is exactly analogous to that of a regular language:

> the empty set is a regular relation;
>
> the set consisting of the empty string is a regular relation;
>
> the set consisting of a single ordered pair of symbols, either of which may be the empty string, is a regular relation;
>
> if R_1 and R_2 are regular relations, so are $R_1 \cup R_2$, R_1R_2, and R_1^* (i.e. the union of the two sets, the set consisting of concatenations of elements from the two

1 The material in this section is based solely on Kaplan (1988). For the results reported, there are no formally published details that I am aware of.

sets, and the set consisting of zero or more concatenations of elements from a set).

Regular relations can then be described using regular expressions over symbol-pairs in the obvious way. Kaplan observed that every regular relation is accepted by some finite-state transducer, and that every finite-state transducer accepts some regular relation. One important subtlety is the use of the empty string as an element of a symbol-pair, with the consequence that a "regular relation" can associate strings of *unequal* lengths together, assuming the obvious equivalence between a sequence of symbol-pairs and a pair of symbol-strings (see Section 2 above). In the associated transducer form, the empty symbol corresponds to no symbol being scanned or no symbol being output (i.e. one of transducer's "heads" may advance while the other does not).

Kaplan stated various results about regular relations, regular languages, and their combinations, including the following. If R_1, R_2, are regular relations, and L_1, L_2 are regular languages, then the following are regular relations:

$R_1 \circ R_2$ (the composition of the relations);

$L_1 \times L_2$ (the direct product of the sets);

$Id(L_1)$ (the relation in which each element is paired with itself);

R_1^{-1} (the inverse relation).

Also:

$Dom(R_1)$ (the set of sequences of first elements of pairs in R_1) is a regular language.

Kaplan pointed out that intersection and complementation of regular relations do *not* in general yield regular relations if empty symbols are involved, but will do where all the symbols are nonempty. Since intersection of regular relations corresponds to a parallel combination of finite-state transducers, this means that there are combinations of transducers that will map a regular language into a nonregular language. This last point is illustrated by considering the two regular relations defined as

$(a : b)^*(\epsilon : c)^*$

$(\epsilon : b)^*(a : c)^*$

where ϵ denotes the empty string. In transducer terms, the first of these scans any number of a symbols on one tape, with the same number of b symbols on the other tape; one tape then stays stationary (empty transitions) while the other is scanning any number of c symbols. The second expression describes a situation where the first head remains stationary, while the second head scans some number of b symbols; then an equal number of a and c symbols are scanned on the two tapes. The intersection of these will have n occurrences of a on the first tape, but the second tape must contain exactly $b^n c^n$. Hence a regular language is being associated with a known nonregular (context-free) language. There is no single finite-state transducer that will define this mapping.

It is important to note that here ϵ is a genuine empty string, interpreted by a transducer as a lack of transition; this is different from two-level morphology's "null"

symbol 0, which acts as an ordinary symbol for the transducers (or for rule-matching), but is treated specially when relating the "tapes" to other linguistic levels (see Section 2 above).

Kaplan also developed more subtle results about combinations of regular relations and languages, including the result that the following operator *IF* combines two regular relations to form a regular relation:

$$IF(R_1, R_2) =_{def} \{xy \mid \text{ if } y \in R_2, \text{ then } x \in R_1\}$$

With these algebraic devices, he expressed the meaning of Koskenniemi's rule-notation as regular expressions. Two important points must be noted here—the context expressions in Koskenniemi's rules denoted regular sets (the rules formed a regular two-level morphological grammar, in the terminology of Section 5 below); also, the basic rule-mechanism operates on equal-length sequences of symbols, as outlined in Section 2 above, with no genuine *empty* or *null* symbols. A context restriction rule of the form

```
a:b  => LC ___ RC
```

defines a regular relation that can be expressed as:

$$IF(\pi^* LC, \mathtt{a}:b\pi^*) \cap IF((\pi^* \mathtt{a} : \mathtt{b})', (RC\,\pi^*)')$$

where π is the set of all possible symbol-pairs, and the prime denotes complementation with respect to π^*.

Similarly, a surface coercion rule

```
a:b <= LC ___ RC
```

defines the relation:

$$(\pi^* LC(Id(Dom(\mathtt{a} : \mathtt{b})) \circ \pi^* \sim \{\mathtt{a} : \mathtt{b}\})RC\,\pi^*)'$$

where "$\sim$" denotes set-difference. Kaplan did not explicitly define the language that would be generated by a *set* of two-level rules (a full grammar), but it would be characterized by an intersection of all the regular relations defined by the individual rules.[2]

He also showed that the uniform deletion/insertion of a particular symbol (such as the special null symbol used in two-level rules) at arbitrary points in a string could be expressed as a regular relation, so the entire lexicon-to-surface mapping sketched in Section 2 earlier could be stated as a regular relation.

The overall result of Kaplan's theoretical work is that the compilation from regular two-level rules to parallel transducers is vindicated as theoretically sound, and anything expressible in the two-level formalism (limited to regular context expressions) is expressible in the transducer (regular relation) formalism; that is, the latter is at least as powerful as the regular two-level formalism itself. The question of whether the two formalisms were *equivalent* (as had always been assumed) was left open.

2 Strictly, this would require the two-level grammar to be in the *normal form* defined in Section 10 below.

4. Basic Definitions

Having established the basis for discussion, we can now formulate a precise set-theoretic definition of two-level rules, so as to go on to investigate their formal properties.

Given any two finite symbolic alphabets, A and A′, a **symbol-pair** from A and A′ is a pair (a, a′) where a ∈ A and a′ ∈ A′. Such symbol-pairs will normally be written as "a:a′". A **symbol-pair sequence from A and A′** is simply a sequence (possibly empty) of symbol-pairs from A and A′, and a **symbol-pair language over A and A′** is a set of symbol-pair sequences (i.e. a subset of $(A \times A')^*$).

Given two alphabets A and A′, and a symbol-pair sequence Σ from A and A′, a sequence $(P_1,..P_n)$ of symbol-pair sequences from A and A′ is said to be a **partition** of Σ iff

$$\Sigma = P_1P_2....P_n$$

(i.e. Σ is made up of the concatenation of the P_i).

Given two alphabets A and A′, a **two-level morphological rule** over A and A′ consists of a pair (P, C) where P is a symbol-pair from A and A′, and C is a nonempty set of pairs (LC,RC) where LC and RC are sets of symbol-pair sequences from A and A′; each such set of symbol-pair sequences is called a **context set**. The reason for having C be a *set* of pairs of context sets, is that we must cater, in the general case, for there being a disjunction of pairs of context sets, as in the illustrative rule given in Section 2 earlier. In the case where the set is a singleton, this reduces to the simple (nondisjunctive) case. Only one of the formal proofs that follow depends upon C being a *finite* set, but if infinite sets of context-pairs were required, some suitable notation would have to be devised for expressing such infinite sets. There seems to be no linguistic motivation to go beyond finite sets of pairs of context sets. Each individual context set (i.e. LC or RC in the above notation) can be an infinite set, and often is.

The above definition is the generalization beyond regular context expressions mentioned in the previous section. To specialize it to the traditional two-level case, we need a further definition:

A two-level morphological rule is said to be **regular** if all the context sets in it are regular sets. (Notice that even here we have abstracted from the actual notation used to represent regular sets).

A set θ is said to **match at the right-end** a symbol-pair sequence Σ iff there is a partition (P_1, P_2) of Σ such that $P_2 \in \theta$.

A set θ is said to **match at the left-end** a symbol-pair sequence Σ iff there is a partition (P_1, P_2) of Σ such that $P_1 \in \theta$.

A set R of two-level morphological rules **contextually allows** a symbol-pair sequence Σ iff, for every partition $(P_1,$ a:a′, $P_2)$ of Σ, *either* there is no rule of the form (a:a′, C) ∈ R, *or* there is at least one rule (a:a′, C) ∈ R such that C contains a pair (LC, RC) such that LC matches P_1 at the right end and RC matches P_2 at the left end.

A two-level morphological rule ((a,a′), C) **coercively allows** a symbol-pair sequence Σ iff for every possible partition $(P_1,$ b:b′, $P_2)$ of Σ and every element (LC, RC) ∈ C such that LC matches P_1 at the right end, and RC matches P_2 at the left end, if b = a, then b′ = a′.

An alternative but equally useful variation on the last definition would be that a two-level morphological rule ((a,a′), C) **coercively disallows** a symbol-pair sequence Σ iff there is a possible partition $(P_1,$ b:b′, $P_2)$ of Σ and an element (LC, RC) ∈ C such that LC matches P_1 at the right end, RC matches P_2 at the left end, b = a and b′ ≠ a′.

5. Feasible Pairs

Real implementations of two-level morphology have to consider the issue of what counts as a **feasible pair** within a two-level grammar. Roughly speaking, the set of feasible pairs is the set of symbol-pairs that have to be considered as potential elements of symbol-pair sequences. The normal approach is not to allow the whole cross-product $A \times A'$ as possible symbol pairs, but to define a subset of this as being the effective alphabet under consideration. This is normally done in three ways:

- Any symbol `a` that appears in both the lexical alphabet and the surface alphabet gives rise to a feasible pair `a:a`.
- Any symbol-pair that is explicitly mentioned in a context-expression anywhere in a rule is feasible.
- Any symbol-pair that is explicitly declared to be feasible (in a list supplied along with the rules) is feasible.

The set of feasible pairs is then used in two ways—any variables or sets occurring in the statement of rules are deemed to range only over feasible pairs, not over arbitrary symbol-pairs; also, any feasible pair may occur freely in a symbol-pair sequence if not otherwise constrained by the rules of the grammar.

It can be seen that these definitions and conventions are to some extent bound up with the concrete textual representation of rules, and the way of stating symbolically the contents of the grammar. At the set-theoretic level of abstraction here, a slightly different (but compatible) set of definitions is necessary. We are assuming that no variables or symbol-set-mnemonics appear in our rules, so the question of using the feasible pair set to expand or give meaning to such abbreviations is irrelevant, but there is still the question of freedom of occurrence.

The notion of implicit definition of feasibility can be altered so that instead of referring to occurrence within a context-expression, it refers to occurrence with a set (for regular sets specified using disjunction, concatenation and indefinite iteration, the two are equivalent). Explicit listing of additional feasible pairs can be represented by including context restriction rules of the form

(a:b, { ({ ϵ }, { ϵ }) })

where ϵ denotes the empty sequence. A **trivial** rule like this allows the symbol-pair to occur freely, since any adjacent material will match the empty string. (The exact definition of matching is important here—matching the empty string or sequence means that there is some partition of the surrounding string in which the portion of the partition next to the symbol-pair of interest is the empty sequence; it does *not* mean that there are no adjacent symbol-pairs).

That is, no extra mechanism is needed for adding feasible pairs—what would appear in a practical implementation as a declaration of an enumerated list can be viewed as a convention for defining rather degenerate context restriction rules. The point is that the original textually-based definitions of feasibility are merely a notational convenience for conveying (to the human reader or a software interpreter/compiler) a set of symbol-pairs that includes at least the pairs from the contexts (and the identity pairs), and we shall abstractly regard that set as being part of the definition of the grammar, regardless of the notation used to make it manifest.

A symbol-pair a:a′ is said to be a **string-constituent** in a set θ of symbol-pair sequences if there is at least one element s $\in \theta$ such that a:a′ is an element of the sequence s.

A symbol-pair a:a′ is said to **occur** in a rule (b:b′, C) iff *either* a:a′ = b:b′ *or* for at least one element (LC, RC) of C, a:a′ is a string-constituent in at least one of LC and RC.

Given two alphabets A and A′, a **two-level morphological grammar based on A and A′** consists of a pair (CR, SC) where CR and SC are finite sets of two-level morphological rules over A and A′. (The two sets of rules are the context restriction and surface coercion rules respectively). Given such a two-level morphological grammar R = (CR, SC), the **set of feasible pairs in R** is the set of symbol-pairs:

$$\{a : a' \mid a : a' \text{ occurs in some element of CR } \cup \text{SC}\} \cup \{a : a \mid a \in A \cap A'\}$$

A two-level morphological grammar is said to be **regular** if all the rules in it are regular.

6. Languages Generated

Given a two-level morphological grammar R = (CR, SC), a symbol-pair sequence Σ is **generated** by R iff all the following hold:

1. all the symbol-pairs in Σ are feasible pairs in R;
2. each rule in SC coercively allows Σ;
3. the set CR of rules contextually allows Σ.

As mentioned earlier, the two classes of rules are treated slightly differently—surface coercion rules are conjoined, forming a set of constraints all of which must be met, and the context restriction rules for a given symbol pair are disjoined, giving a set of possible licensing contexts. If no rules apply to a particular symbol-pair, it is acceptable if and only if it is feasible.

With the above definitions, it is now possible to ask what sorts of symbol-pair languages can be characterized using a two-level morphological grammar.

Lemma 1
Let CR be a set of two-level morphological rules. Let E_1 and E_2 be symbol-pair sequences such that CR contextually allows E_1, and CR contextually allows E_2. Then CR contextually allows the concatenation E_1E_2.

Proof
Let a:a′ be a symbol-pair occurring in E_1E_2, such that (P_1, a:a′,P_2) is a partition of E_1E_2. Assume, without loss of generality, that a:a′ occurs in E_1. That is, P_1 is a proper initial subsequence of E_1 and $P_2 = S_2E_2$ for some sequence S_2. Since CR contextually allows E_1, for the partition (P_1, a:a′, S_2) of E_1 there is at least one rule C in CR that contains at least one context-pair (LC, RC) such that LC matches P_1 at the right end and RC matches S_2 at the left end. If RC matches S_2 at the left end, then RC will also match $S_2E_2 = P_2$ at the left end. Hence, for the partition (P_1, a:a′, P_2) of E_1E_2 there is at least one rule C in CR that contains at least one context-pair (LC, RC) such that LC matches P_1 at the right end and RC matches P_2 at the left end. A similar argument can be given for the occurrence of a:a′ being in E_2. Since this will be true for any such a:a′ in E_1E_2, CR contextually allows E_1E_2. ■

Corollary
If a two-level grammar is of the form (CR, ∅) (i.e. it contains no surface coercion rules), then the concatenation of any two strings in its language is also in its language.

Lemma 2
Let R = (a:a′, C) be a two-level morphological rule. Let E_1, E_2, E_3 be symbol-pair sequences such that $E_1E_2E_3$ is coercively allowed by R. Then E_2 is coercively allowed by R.

Proof
If E_2 were not coercively allowed by R, it would mean that there is a partition (S_1, a:b, S_2) of E_2 such that for some (LC, RC) in C, LC matches S_1 at the right end, RC matches S_2 at the left end, and b≠ a′. If this were the case, there would be a corresponding partition (E_1S_1, a:b, S_2E_3) of $E_1E_2E_3$, with LC matching E_1S_1 at the right end, and RC matching S_2E_3 at the left end. This would (by definition) mean that R does not coercively allow $E_1E_2E_3$, which is not the case by hypothesis. ■

Corollary (a)
Let C be a set of two-level morphological rules, all of which coercively allow a symbol-pair sequence E. Then all of the rules in C coercively allow any subsequence of E.

Corollary (b)
If all the context restriction rules in a two-level morphological grammar are trivial, in the sense of having vacuous contexts (see Section 5 above), then any substring of an element of its language is also in the language.

Lemma 3 (The Concatenation Property)
Let G be a two-level morphological grammar (CR, SC), and let L(G) be the set of symbol-pair sequences generated by G. Suppose that there are sequences E_1, E_2, E_3, E_4 such that $E_2 \in$ L(G), $E_3 \in$ L(G), and $E_1E_2E_3E_4 \in$ L(G). Then $E_2E_3 \in$ L(G).

Proof

(i) Since $E_1E_2E_3E_4 \in$ L(G), all the symbol-pairs in it are feasible with respect to G, hence all the symbol-pairs in E_2E_3 are feasible.

(ii) Since E_2 and $E_3 \in$ L(G), it follows that CR contextually allows E_2 and E_3 (by definition). By Lemma 1 above, this means that CR contextually allows E_2E_3.

(iii) Since $E_1E_2E_3E_4 \in$ L(G), it follows (by definition) that all of the rules in SC coercively allow $E_1E_2E_3E_4$. Hence, by Corollary (a) to Lemma 2 above, all of the rules in SC coercively allow E_2E_3.

This establishes the three defining conditions for $E_2E_3 \in$ L(G). ■

7. Comparison with Transducers

As mentioned in the introduction, two-level grammars have historically been written in two different ways—as rules as defined here, and as sets of finite-state transducers. In the latter case, each transducer deals with some linguistic phenomenon, and a

sequence of symbol-pairs is generated by the grammar if *every* transducer in the grammar accepts it. That is, the symbol-pair sequence must be in the *intersection* of the languages accepted by the transducers (viewed as acceptors); in procedural terms, this is often referred to as "having the transducers executed in parallel." Hence, when working with the transducer formalism the linguist has to devise independent transducers whose intersection is the required language. These transducers, like the two-level rules, define a mapping between *equal-length* symbol sequences, as described earlier; there are no "empty transitions." Under these conditions, since the intersection of a set of regular languages is also a regular language, it follows that these parallel finite-state acceptors define exactly the regular sets of symbol-pair sequences.

As observed earlier, Kaplan's work on regular relations shows that the "parallel transducer" model is at least as powerful as the two-level grammar model defined earlier. The obvious question is whether there is a difference in power; in fact, there is:

Theorem 1
There are regular sets of symbol-pair sequences (i.e. symbol-pair languages characterized by regular expressions of symbol-pairs) that cannot be generated by any two-level morphological grammar.

Proof
This follows directly from Lemma 3 above. Any language L generated by a two-level morphological grammar must have the property that if E_2, E_3, and $E_1E_2E_3E_4 \in$ L, then $E_2E_3 \in$ L. There are regular symbol-pair languages that do not have this property, such as the language defined by the regular expression

$$\texttt{b:b} \vee (\texttt{a:a b:b})^*$$

which contains `b:b` and `a:a b:b` but not `b:b a:a b:b`, even though that sequence is a subsequence of other elements of the language. ■

It was already clear that there are some regular relations that cannot be generated by a two-level grammar, since a regular relation can put into correspondence symbol-sequences of different lengths; what the above result shows is that there are some *equal-length* regular relations that cannot be generated by any two-level grammar. That is, we have the following proper inclusions:

languages generated by regular two-level morphological grammars

$\subset$ regular sets of symbol-pair sequences

$\subset$ regular relations

There is another, rather trivial, difference between the power of two-level morphological rules and arbitrary regular expressions. According to the definitions given here, the empty sequence of symbol-pairs is in every language generated by a two-level morphological grammar, since it conforms to the definition regardless of the content of the rules. The definitions could be altered to exclude the empty sequence from every language, but it is hard to see how the rule mechanism could be used to allow the empty sequence in some languages but not others.

8. Another Type of Rule

Karttunen et al. (1987) allow a further rule operator /<= in their rules. A rule of the form

```
a:b  /<=  LC --- RC
```

means, informally, that the pair a:b must *not* occur if the contexts LC and RC are present. It would be straightforward to extend the definition of a two-level morphological grammar to cover this symbol. What would be needed would be to allow a grammar to be a triple (CR, SC, CE) in which CR and SC are as before, but CE is a set of **context exclusion** rules of this new sort. The applicability of such rules could be defined thus:

> A symbol-pair sequence Σ is **contextually excluded** by a rule (a:b, C) if there is a partition (S_1, a:b, S_2) of Σ and some context-pair (LC, RC) $\in$ C such that LC matches S_1 at the right end and RC matches S_2 at the left end.

The definition of generation of a sequence Σ by a grammar (CR, SC, CE) would have to be amended to include the additional stipulation (see Section 6 above):

4. no rule in CE contextually excludes Σ.

Inclusion of this type of context exclusion rule does not affect the proof of Lemmas 1 and 2, or their corollaries, and the Concatenation Property (Lemma 3) can still be proven since the following lemma can be proven by a very simple modification to the proof for Lemma 2 :

Lemma 4
Let R = (a:a′, C) be a two-level morphological rule. Let E_1, E_2, E_3 be symbol-pair sequences such that $E_1E_2E_3$ is not contextually excluded by R. Then E_2 is not contextually excluded by R.

Hence, the use of context exclusion rules does not affect any of the results about generative power given earlier, or the lack of closure demonstrated in Section 11 below.

David Weir (personal communication) has pointed out that the use of context exclusion rules makes surface coercion rules technically redundant, since any stipulation of surface coercion can be restated as a set of context exclusion rules.

9. Boundary Markers

The two-level formalism, as defined above (and as originally defined by Koskenniemi) has no word-boundary symbol to mark the end of a sequence of symbols. Although the linguist is free to introduce any symbols that seem empirically useful, none of these symbols has any specially defined status beyond what the linguist chooses to state in the actual two-level morphological grammar, and the apparatus does not stipulate that a particular symbol occurs only at the start or end of a complete string. The

PC-KIMMO implementation of two-level morphology (Antworth 1990) has a special boundary marker "#" with the following properties:

1. # can appear in both the lexical and the surface component of a symbol-pair.
2. # can be paired only with itself, not with any other symbol.
3. The pair #:# occurs at the extreme ends of every string.
4. The pair #:# never occurs at any internal position of a string.
5. Any two-level rule may refer to the pair #:#, thereby making phenomena it describes relative to the end of the sequence.
6. The symbol-pair #:# is not regarded as part of the sequence generated; that is, the rules characterize an extended string with #:# at each end, but the generated sequence is defined to be the sequence without boundary markers.

It would be possible, with careful design of the two-level rules, to enforce points 1–5 without resorting to special treatment for #:#, but point 6 steps outside the two-level mechanism; a grammar that merely enforced 1–5 using its rules would, according to the basic definitions of generation, have in its language strings that contained #:# (at the ends). Karttunen et al. (1987) also allow a lexical boundary marker #, such that the symbol-pair #:0 meets points 3, 4, and 5 above (which can be achieved by writing suitable two-level rules without any need for special treatment). However, they do not need to rely on the additional stipulation given in 6, since the use of a surface null will give the desired effect when considering the entire mapping from lexical forms to surface form, with explicit null symbols being removed in the manner outlined in Section 2 above (see Ritchie 1989 for a formal definition of this phase).

We can show that the inclusion of a specially treated boundary symbol slightly alters the generative power of the formalism, since this allows any set that can be included as a context set to constitute the entire language, as follows.

Given two sets A and A′, a symbol $\alpha \in$ A, a symbol $\beta \in$ A′, and a two-level morphological grammar G based on alphabets A and A′, then a symbol-pair sequence $\Sigma \in (A \times A')^*$ is said to be **generated by G with boundary** $\alpha{:}\beta$ iff

1. $\alpha{:}\beta$ does not occur anywhere in Σ
2. $\alpha{:}\beta\ \Sigma\ \alpha{:}\beta$ is generated by G.

Notice that under this definition, a sequence Σ may well be "generated by G with boundary $\alpha{:}\beta$" even though Σ itself is *not* generated by G.

Theorem 2
Let A and A′ be sets of symbols, let α be some symbol not in A, β some symbol not in A′ and let L be some set of symbol-pair sequences from A × A′. Then there is a two-level morphological grammar G based on A ∪ { α } and A′ ∪ { β } such that

$$L = \{\Sigma \mid \text{ G generates } \Sigma \text{ with boundary } \alpha : \beta\}$$

Proof
Consider the two-level grammar (written in the usual textual notation):

$\alpha : \beta \Rightarrow$ --- C $\alpha : \beta$

or $\alpha : \beta$ C ---

where C is some expression denoting the set L (in the usual notation based on regular sets, C would be a regular expression). More formally, this grammar G (= (CR, SC)) could be written set-theoretically as

$$(\{(\alpha : \beta, \{(\{\epsilon\}, L_1)(L_2, \{\epsilon\})\})\}, \emptyset)$$

where L_1 is the set consisting of all possible concatenations of elements of L with α:β, and L_2 is the set consisting of all possible concatenations of α:β with elements of L.

It is straightforward to verify that this grammar generates all and only strings of the form α:$\beta\Sigma\alpha$:β for $\Sigma \in$ L. ■

This theorem may sound rather odd, since it implies that any set whatsoever can be generated in this manner (in contrast to the earlier result that there are some regular languages of symbol-pairs that cannot be generated by two-level rules). However, it is important to note that this theorem not only uses a special boundary symbol, it relies on the generalization of the usual regular-set based two-level rules to rules that allow *any* set as a context set. Essentially it says that, with the augmentation of a special boundary symbol, the generative power is limited only by the limits placed on the class of sets that can appear as contexts. A more useful and natural specialization of this theorem is as follows:

Corollary
For any regular set L of symbol-pair sequences, there is a regular two-level morphological grammar that generates L with boundary α:β, for some symbols α, β that are not in the alphabets used for L.

10. Normal Form

We define a two-level morphological grammar as being in **normal form** if there is no symbol-pair that is the subject of more than one context restriction rule or more than one surface coercion rule. More formally, a grammar (CR, SC) is said to be in normal form iff whenever (a:b, C) $\in$ CR and (a:b, C′) $\in$ CR, C = C′, and whenever (a:b, C) $\in$ SC and (a:b, C′) $\in$ SC, C = C′.

Theorem 3
For any two-level morphological grammar G, there is a corresponding two-level morphological grammar G′ in normal form that generates the same symbol-pair language.

Proof
Suppose G = (CR, SC) is not in normal form. Create the grammar G′ = (CR′, SC′) as follows. If there is exactly one rule (a:b, C) in CR for the symbol-pair a:b, include that rule in CR′. For any set of two or more rules in CR with the same symbol-pair $a : b$

$(a : b, C_1), ..., (a : b, C_n)$

include in CR′ a rule:

$$(a : b, C_1 \cup C_2 \ldots C_n).$$

Create SC′ from SC in exactly the same way. It should be clear from this that (a) any symbol-pair a:b occurs in the rules in G iff it occurs in the rules in G′; (b) any pair of context sets (LC, RC) appears in a rule in CR iff it appears in a rule in CR′ with the same symbol-pair; (c) any pair of context sets (LC, RC) appears in a rule in SC iff it appears in a rule in SC′ with the same symbol-pair.

It is straightforward to prove the equivalence for G and G′ of the three defining criteria for language membership. ■

11. Closure Properties

Most classes of formal languages (e.g. regular languages, context-free languages) are closed under at least some of the simple set-theoretic operations such as union and intersection. Certain results can be proved for two-level morphological languages concerning their closure (or lack of it).

Theorem 4
The set of languages generated by two-level morphological grammars is not closed under union.

Proof
The proof follows from the "concatenation property" of Lemma 3 above, which allows a simple counterexample to be constructed. Consider the grammar G_1 which would conventionally be written as

```
b:b  =>  ___
```

(if we follow the normal practice of omitting textually any context set containing only the empty sequence). This generates the language $(\mathtt{b} : \mathtt{b})^*$; that is,

$$\{\ \epsilon,\ \mathtt{b{:}b},\ \mathtt{b{:}b\ b{:}b},\ \ldots\}$$

The grammar G_2 given by the rules

```
b:b  =>  a:a ___
a:a  =>   ___ b:b
```

(again, omitting mention of contexts that are trivially satisfiable) generates the language $(\mathtt{a{:}a\ b{:}b})^*$:

$$\epsilon,\ \mathtt{a{:}a\ b{:}b},\ \mathtt{a{:}a\ b{:}b\ a{:}a\ b{:}b},\ \ldots$$

The union of these languages contains the sequences

```
b:b      a:a b:b      a:a b:b a:a b:b
```

but does not contain the sequence

```
b:b a:a b:b
```

and hence, by Lemma 3, cannot be a language generated by a two-level language. ■

The following result and construction are due to David Weir (personal communication).

Theorem 5
The set of languages generated by two-level morphological grammars is closed under intersection.

Proof
Let L_1 and L_2 be two-level morphological languages, generated by the grammars G_1=(CR_1, SC_1) and G_2=(CR_2, SC_2) respectively. It is possible to construct a two-level grammar for the intersection $L = L_1 \cap L_2$. Without loss of generality, assume (merely to simplify the construction):

1. For every rule in CR_1 or CR_2 of the form (P,C) let the set C contain just one pair of the form (LC,RC). This assumes that the set of context-pairs in a rule is never an infinite set—see Section 4 above. There can be several such rules for any given symbol-pair P (i.e. we have to allow the grammar to be not in the "normal form" defined in Section 10 above). Note that we do not need to make this assumption for the rules in SC_1 and SC_2.
2. For every pair $a : b$ that is a feasible pair in both G_1 and G_2 there is at least one rule in CR_1 and CR_2 for the symbol pair $a : b$. In the absence of any other rule include $(a : b, (\{\epsilon\}, \{\epsilon\}))$.

Suppose that Φ is the set of symbol-pairs that are feasible in both G_1 and G_2, so that Φ^* is the set of sequences of such symbol-pairs.

To create the context restriction rules for the intersection grammar G, take

$$CR = CR' \cup CR''$$

where CR' is defined to be:

$$\{(p, \{(\Phi^* LC_1 \cap \Phi^* LC_2, RC_1\Phi^* \cap RC_2\Phi^*)\}) \mid$$

$$(p, \{(LC_1, RC_1)\}) \in CR_1, \text{ and } (p, \{(LC_2, RC_2)\}) \in CR_2\}$$

and CR'' is defined as:

$$\{(a : b, \{(\{a' : a'\}, \epsilon)\} \mid a : b \text{ is feasible in exactly one of } G_1 \text{ and } G_2\}$$

where a' is a new symbol that is not in a feasible pair in either G_1 or G_2.

The set SC' defined below has the effect of making the context $a' : a'$ unobtainable. Hence, the above rule makes each $a : b$ that is not feasible in both grammars behave as if it were not feasible in the new grammar. There will be no symbol-pair that is the subject of a rule in CR' and in CR'', since the first of these sets covers the jointly feasible pairs and the latter handles others.

To construct the surface coercion rules for the intersection grammar G use

$$SC = SC_1 \cup SC_2 \cup SC'$$

where SC' is defined as follows:

$$SC' = \{(a' : a', \{(\{\epsilon\}, \{\epsilon\})\}), (a' : a, \{(\{\epsilon\}, \{\epsilon\})\})\}$$

where a' is as above and a is an arbitrary symbol other than a'.

There is then a fairly straightforward proof that this grammar characterizes a symbol-pair sequence if and only if that sequence is in $L(G_1) \cap L(G_2)$. The only slightly complicated step involves the proof that if an occurrence of a symbol-pair matches a rule in CR_1, and also matches a rules in CR_2, it must match a rule in CR (in CR', in fact). The argument goes as follows. If a partition $(S, a : b, S')$ is contextually allowed by original rules $(a : b, \{(LC_1, RC_1)\})$ and $(a : b, \{(LC_2, RC_2)\})$, then it follows from the definitions in Section 4 above that there is a P_1 at the right end of S that is in LC_1, and a P_2 at the right end of S that is in LC_2. Hence $S \in \Phi^* LC_1$, and $S \in \Phi^* LC_2$. Hence there is a string (namely S) that is at the right end of S and is in $\Phi^* LC_1 \cap \Phi^* LC_2$ (and similarly for the right contexts). ■

Corollary (a)
The set of regular two-level morphological languages is closed under intersection.

Proof
The above construction uses combinations that yield regular sets from regular sets. Hence if G_1 and G_2 are regular two-level morphological grammars, so is G. ■

Corollary (b)
The set of two-level morphological languages is not closed under complementation.

Proof
Follows directly from Theorem 4 and Theorem 5, since

$$A \cup B = (A' \cap B')'$$

(a direct counter-example can also be constructed using Lemma 3). ■

12. The Next Stage—Compilation and Complexity

The above results define more clearly the family of two-level languages, but they say nothing about mechanisms for recognizing strings of symbol-pairs, nor how such recognition can be integrated with a lexicon of symbol-strings. There are at least two methods of compilation and subsequent interpretation for regular two-level grammars (i.e. those whose context sets can be written as regular expressions, which is the normal practice), both of which rely heavily on finite-state transducers in their compiled form. Karttunen et al. (1987) produce an intersecting set of traditional transducers, and the recognition process is then a straightforward interpretation of the equivalent combined automaton. Ritchie et al. (1991) compile rules into a slightly peculiar form of transducer, in which states are marked to indicate different processing requirements; a nonstandard interpreter then scans the resulting annotated automaton. There are no

published proofs of correctness of either mechanism, but Karttunen et al.'s exposition relies heavily on Kaplan's work on regular relations (see Section 3 above), which appears to justify its procedures.

The usual assumed interface to a lexicon (e.g. Koskenniemi 1983a, 1983b; Karttunen 1983) is via sublexicons for different lexeme classes, with cross-pointers indicating allowable sequences of lexemes (a formal definition of this can be found in Ritchie 1989). Another possibility is to impose a wholly separate rule-driven layer of morphotactic description upon the lexical forms; this approach is coupled with a two-level system in Ritchie et al. (1991).

The complexity of the problem of recognizing a string of symbols using the parallel transducer formulation of two-level morphology with a single (or cross-linked) lexicon has been shown to be NP-complete by Barton et al. (1987), and Ritchie et al. (1991) suggest that the proof of NP-hardness could be modified to cover the recognition problem for the definition of two-level rules given here (and a single lexicon), but exact complexity or decidability results for the compilation/interpretation procedure of Ritchie et al. (1991) have not been produced, nor is there any characterization of the complexity of determining whether a sequence of symbol-pairs belongs to the language generated by a two-level language (without regard for the lexical segmentation issue).

All these issues lie outside the scope of this paper, but it would be desirable for them to be settled.

Acknowledgments

I would like to thank Alan Black, Ron Kaplan, and David Weir for useful discussions of this material.

References

Antworth, Evan L. (1990). *PC-KIMMO : A Two-Level Processor for Morphological Analysis.* Occasional Publications in Academic Computing No. 16, Summer Institute of Linguistics, Dallas, TX.

Barton, G. Edward; Berwick, Robert C.; and Ristad, Eric Sven (1987). *Computational Complexity and Natural Language.* Cambridge: The MIT Press.

Dalrymple, Mary; Doron, Edit; Goggin, John; Goodman, Beverley; and McCarthy, John (eds.) (1983). Texas Linguistic Forum 22, Department of Linguistics, University of Texas at Austin.

Kaplan, Ronald M. (1988). "Regular models of phonological rule systems." Talk on finite-state transducers given at the Alvey Workshop on Parsing and Pattern Recognition, Oxford.

Karttunen, Lauri (1983). "KIMMO: A general morphological analyser." In *Texas Linguistic Forum* 22, edited by M. Dalrymple, E. Doran, J. Goggin, B. Goodman, and J. McCarthy: 165–186.

Karttunen, Lauri; Koskenniemi, Kimmo; and Kaplan, Ronald M. (1987). "A compiler for two-level phonological rules." Unpublished manuscript.

Koskenniemi, Kimmo (1983a). *Two-level Morphology: a General Computational model for Word-Form Recognition and Production.* Publication No. 11, University of Helsinki, Finland.

Koskenniemi, Kimmo (1983b). "Two-level model for morphological analysis." In *Proceedings, Eighth International Joint Conference on Artificial Intelligence,* Karlsruhe: 683–685.

Koskenniemi, Kimmo (1984). "A general computational model for word-form recognition and production." In *Proceedings, 10th International Conference on Computational Linguistics/22nd Annual Meeting of the ACL,* Stanford, CA: 178–181.

Koskenniemi, Kimmo (1985). "Compilation of automata from morphological two-level rules." *Papers from the Fifth Scandinavian Conference of Computational Linguistics,* Helsinki, Finland: 143–149.

Ritchie, Graeme (1989). "On the generative power of two-level morphological rules." In *Proceedings, Fourth Conference of the European Chapter of the Association for Computational Linguistics,* Manchester: 51–57.

Ritchie, Graeme D.; Pulman, Stephen G.; Black, Alan W.; and Russell, Graham J. (1987). "A computational framework for lexical description." *Computational Linguistics 13* (3–4):290–307.

Ritchie, Graeme D.; Russell, Graham J.; Black, Alan W.; and Pulman, Stephen G. (1991). *Computational Morphology.* Cambridge: The MIT Press.