

# Appendix B

## Two-level Phonology Revisited

*Last modified October 20, 1995*

---

### B1 Introduction to two-level phonology

- B1.1 Computational and linguistic roots
- B1.2 Two-level rule application
- B1.3 How a two-level description works
- B1.4 With zero you can do (almost) anything

### B2 Rule interaction in two-level phonology

- B2.1 Ordered rules versus two-level rules
- B2.2 Multiple application
- B2.3 Deletion and insertion rules
- B2.4 Mutually bleeding rules
- B2.5 Input feeding rules

---

This chapter provides a general introduction to two-level phonology and a guide to handling interactions among rules. While chapter 3 of the first book on PC-KIMMO (Antworth 1990, now available as Appendix A) focused on how to translate two-level rules into finite state tables, this chapter argues that two-level phonology is an important phonological formalism not only because it is computationally tractable but also because it offers a linguistically coherent alternative to generative phonology.

### B1 Introduction to two-level phonology[fn1]

Two-level phonology is a linguistic tool developed by computational linguists. Its primary use is in systems for natural language processing such as PC-KIMMO. This section describes the linguistic and computational basis of two-level phonology.

#### B1.1 Computational and linguistic roots

As the fields of computer science and linguistics have grown up together during the past several decades, they have each benefited from cross-fertilization. Modern linguistics has especially been influenced by the formal language theory that underlies computation. The most famous application of formal language theory to linguistics was Chomsky's (1957) transformational generative grammar. Chomsky's strategy was to consider several types of formal languages to see if they were capable of modeling natural language syntax. He started by considering the simplest type of formal languages, called *finite state* languages. As a general principle, computational linguists try to use the least powerful computational devices possible. This is because the less powerful devices are better understood, their behavior is predictable, and they are computationally more efficient. Chomsky (1957:18ff) demonstrated that natural language syntax could not be effectively modeled as a finite state language; thus he rejected finite state languages as a theory of syntax and proposed that syntax requires the use of more powerful, non-finite state languages. However, there is no reason to assume that the same should be true for natural language phonology. A finite state model of phonology is especially desirable from the computational point of view, since it makes possible a computational implementation that is simple and efficient.

While various linguists proposed that generative phonological rules could be implemented by finite state devices (see Johnson 1972, Kay 1983), the most successful model of finite state phonology was developed by

Kimmo Koskenniemi, a Finnish computer scientist. He called his model two-level morphology (Koskenniemi 1983), though his use of the term morphology should be understood to encompass both what linguists would consider morphology proper (the decomposition of words into morphemes) and phonology (at least in the sense of morphophonemics). Our main interest in this article is the phonological formalism used by the two-level model, hereafter called *two-level phonology*. Two-level phonology traces its linguistic heritage to 'classical' generative phonology as codified in *The Sound Pattern of English* (Chomsky and Halle 1968). The basic insight of two-level phonology is due to the phonologist C. Douglas Johnson (1972) who showed that the *SPE* theory of phonology could be implemented using finite state devices by replacing sequential rule application with simultaneous rule application. At its core, then, two-level phonology is a rule formalism, not a complete theory of phonology. The following sections of this article describe the mechanism of two-level rule application by contrasting it with rule application in classical generative phonology. It should be noted that Chomsky and Halle's theory of rule application became the focal point of much controversy during the 1970s with the result that current theories of phonology differ significantly from classical generative phonology. The relevance of two-level phonology to current theory is an important issue, but one that will not be fully addressed here. Rather, the comparison of two-level phonology to classical generative phonology is done mainly for expository purposes, recognizing that while classical generative phonology has been superseded by subsequent theoretical work, it constitutes a historically coherent view of phonology that continues to influence current theory and practice.

One feature that two-level phonology shares with classical generative phonology is linear representation. That is, phonological forms are represented as linear strings of symbols. This is in contrast to the nonlinear representations used in much current work in phonology, namely autosegmental and metrical phonology (see Goldsmith 1990). On the computational side, two-level phonology is consistent with natural language processing systems that are designed to operate on linear orthographic input.

## B1.2 Two-level rule application

We will begin by reviewing the formal properties of generative rules. Stated succinctly, generative rules are *sequentially ordered rewriting rules*. What does this mean?

First, *rewriting* rules are rules that change or transform one symbol into another symbol. For example, a rewriting rule of the form  $a \rightarrow b$  interprets the relationship between the symbols  $a$  and  $b$  as a dynamic change whereby the symbol  $a$  is rewritten or turned into the symbol  $b$ . This means that after this operation takes place, the symbol  $a$  no longer 'exists,' in the sense that it is no longer available to other rules. In linguistic theory generative rules are known as process rules. Process rules attempt to characterize the relationship between levels of representation (such as the phonemic and phonetic levels) by specifying how to transform representations from one level into representations on the other level.

Second, generative phonological rules apply *sequentially*, that is, one after another, rather than applying simultaneously. This means that each rule creates as its output a new intermediate level of representation. This intermediate level then serves as the input to the next rule. As a consequence, the underlying form becomes inaccessible to later rules.

Third, generative phonological rules are *ordered*; that is, the description specifies the sequence in which the rules must apply. Applying rules in any other order may result in incorrect output.

As an example of a set of generative rules, consider the following rules:

- (1) Vowel Raising  
 $e \rightarrow i / \_\_C\_0 i$
- (2) Palatalization  
 $t \rightarrow c / \_\_i$

Rule 1 (Vowel Raising) states that  $e$  becomes (is rewritten as)  $i$  in the environment preceding  $Ci$  (where  $C$  stands for the set of consonants and  $C\_0$  stands for zero or more consonants). Rule 2 (Palatalization) states that  $t$  becomes  $c$  preceding  $i$ . A sample derivation of forms to which these rules apply looks like this (where UR stands for Underlying Representation, SR stands for Surface Representation):[fn2]

```

UR:   temi
(1)   timi
(2)   cimi
SR:   cimi

```

Notice that in addition to the underlying and surface levels, an intermediate level has been created as the result of sequentially applying rules 1 and 2. The application of rule 1 produces the intermediate form *timi*, which then serves as the input to rule 2.

Not only are these rules sequential, they are ordered, such that rule 1 must apply before rule 2. Rule 1 has a *feeding* relationship to rule 2; that is, rule 1 increases the number of forms that can undergo rule 2 by creating more instances of *i*. Consider what would happen if they were applied in the reverse order. Given the input form *temi*, rule 2 would do nothing, since its environment is not satisfied. Rule 1 would then apply to produce the incorrect surface form *timi*.

Two-level rules differ from generative rules in the following ways. First, whereas generative rules apply in a sequential order, two-level rules apply simultaneously, which is better described as applying in parallel. Applying rules in parallel to an input form means that for each segment in the form all of the rules must apply successfully, even if only vacuously.

Second, whereas sequentially applied generative rules create intermediate levels of derivation, simultaneously applied two-level rules require only two levels of representation: the underlying or lexical level and the surface level. There are no intermediate levels of derivation. It is in this sense that the model is called two-level.

Third, whereas generative rules relate the underlying and surface levels by rewriting underlying symbols as surface symbols, two-level rules express the relationship between the underlying and surface levels by positing direct, static correspondences between pairs of underlying and surface symbols. For instance, instead of rewriting underlying *a* as surface *b*, a two-level rule states that an underlying *a* corresponds to a surface *b*. The two-level rule does not change *a* into *b*, so *a* is available to other rules. In other words, after a two-level rule applies, both the underlying and surface symbols still 'exist.'

Fourth, whereas generative rules have access only to the current intermediate form at each stage of the derivation, two-level rules have access to both underlying and surface environments. Generative rules cannot 'look back' at underlying environments or 'look ahead' to surface environments. In contrast, the environments of two-level rules are stated as lexical-to-surface correspondences. This means that a two-level rule can easily refer to an underlying *a* that corresponds to a surface *b*, or to a surface *b* that corresponds to an underlying *a*. In generative phonology, the interaction between a pair of rules is controlled by requiring that they apply in a certain sequential order. In two-level phonology, rule interactions are controlled not by ordering the rules but by carefully specifying their environments as strings of two-level correspondences.

Fifth, whereas generative, rewriting rules are unidirectional (that is, they operate only in an underlying to surface direction), two-level rules are *bidirectional*. Two-level rules can operate either in an underlying to surface direction (generation mode) or in a surface to underlying direction (recognition mode). Thus in generation mode two-level rules accept an underlying form as input and return a surface form, while in recognition mode they accept a surface form as input and return an underlying form. The practical application of bidirectional phonological rules is obvious: a computational implementation of bidirectional rules is not limited to generation mode to produce words; it can also be used in recognition direction to parse words.

### B1.3 How a two-level description works

To understand how a two-level phonological description works, we will use the example given above involving Raising and Palatalization. The two-level model treats the relationship between the underlying form *temi* and the surface form *cimi* as a direct, symbol-to-symbol correspondence:

```

UR:   t e m i
SR:   c i m i

```

Each pair of lexical and surface symbols is a *correspondence pair*. We refer to a correspondence pair with the notation *<underlying symbol>: <surface symbol>*, for instance *e: i* and *m: m*. There must be an exact one-to-one correspondence between the symbols of the underlying form and the symbols of the surface form.

Deletion and insertion of symbols (explained in detail in the next section) is handled by positing correspondences with zero, a null segment. The two-level model uses a notation for expressing two-level rules that is similar to the notation linguists use for phonological rules. Corresponding to the generative rule for Palatalization (rule 2 above), here is the two-level rule for the  $t:c$  correspondence:

(3) Palatalization  
 $t:c \Leftrightarrow \_\_ @:i$

This rule is a statement about the distribution of the pair  $t:c$  on the left side of the arrow with respect to the context or environment on the right side of the arrow. A two-level rule has three parts: the *correspondence*, the *operator*, and the *environment*. The correspondence part of rule 3 is the pair  $t:c$ , which is the correspondence that the rule sanctions. The operator part of rule 3 is the double-headed arrow. It indicates the nature of the logical relationship between the correspondence and the environment (thus it means something very different from the rewriting arrow  $\rightarrow$  of generative phonology). The  $\Leftrightarrow$  arrow is equivalent to the biconditional operator of formal logic and means that the correspondence occurs always and only in the stated context; that is,  $t:c$  is allowed if and only if it is found in the context  $\_\_ @:i$ . In short, rule 3 is an obligatory rule. The environment part of rule 3 is everything to the right of the arrow. The long underline indicates the gap where the pair  $t:c$  occurs. Notice that even the environment part of the rule is specified as two-level correspondence pairs.

The environment part of rule 3 requires further explanation. Instead of using a correspondence such as  $i:i$ , it uses the correspondence  $@:i$ . The @ symbol is a special 'wildcard' symbol that stands for any phonological segment included in the description. In the context of rule 3, the correspondence  $@:i$  stands for all the feasible pairs in the description whose surface segment is  $i$ , in this case  $e:i$  and  $i:i$ . Thus by using the correspondence  $@:i$ , we allow Palatalization to apply in the environment of either a lexical  $e$  or lexical  $i$ . In other words, we are claiming that Palatalization is sensitive to a surface (phonetic) environment rather than an underlying (phonemic) environment. Thus rule 3 will apply to both underlying forms  $timi$  and  $temi$  to produce a surface form with an initial  $c$ .

Corresponding to the generative rule for Raising (rule 1 above) is the following two-level rule for the  $e:i$  correspondence:

(4) Vowel Raising  
 $e:i \Leftrightarrow \_\_ C:C^* @:i$

(The asterisk in  $C:C^*$  indicates zero or more instances of the correspondence  $C:C$ ) Similar to rule 3 above, rule 4 uses the correspondence  $@:i$  in its environment. Thus rule 4 states that the correspondence  $e:i$  occurs preceding a surface  $i$ , regardless of whether it is derived from a lexical  $e$  or  $i$ . Why is this necessary? Consider the case of an underlying form such as  $pememi$ . In order to derive the surface form  $pimimi$ , Raising must apply twice: once before a lexical  $i$  and again before a lexical  $e$ , both of which correspond to a surface  $i$ . Thus rule 4 will apply to both instances of lexical  $e$ , capturing the regressive spreading of Raising through the word.

By applying rules 3 and 4 in parallel, they work in consort to produce the right output. For example,

UR:	t	e	m	i
Rules	3	4		
SR:	c	i	m	i

Conceptually, a two-level phonological description of a data set such as this can be understood as follows. First, the two-level description declares an alphabet of all the phonological segments used in the data in both underlying and surface forms, in the case of our example,  $t$ ,  $m$ ,  $c$ ,  $e$ , and  $i$ . Second, the description declares a set *feasible pairs*, which is the complete set of all underlying-to-surface correspondences of segments that occur in the data. The set of feasible pairs for these data is the union of the set of *default correspondences*, whose underlying and surface segments are identical (namely  $t:t$ ,  $m:m$ ,  $e:e$ , and  $i:i$ ) and the set of *special correspondences*, whose underlying and surface segments are different (namely  $t:c$  and  $e:i$ ). Notice that since the segment  $c$  only occurs as a surface segment in the feasible pairs, the description will disallow any underlying form that contains a  $c$ .

A minimal two-level description, then, consists of nothing more than this declaration of the feasible pairs. Since it contains all possible underlying-to-surface correspondences, such a description will produce the correct output form, but because it does not constrain the environments where the special correspondences can occur, it will also allow many incorrect output forms. For example, given the underlying form *temi*, it will produce the surface forms *temi*, *timi*, *cemi*, and *cimi*, of which only the last is correct.

Third, in order to restrict the output to only correct forms, we include rules in the description that specify where the special correspondences are allowed to occur. Thus the rules function as constraints or filters, blocking incorrect forms while allowing correct forms to pass through. For instance, rule 3 (Palatalization) states that a lexical *t* must be realized as a surface *c* when it precedes *@:i*; thus, given the underlying form *temi* it will block the potential surface output forms *timi* (because the surface sequence *ti* is prohibited) and *cemi* (because surface *c* is prohibited before anything except surface *i*). Rule 4 (Raising) states that a lexical *e* must be realized as a surface *i* when it precedes the sequence *C:C @:i*; thus, given the underlying form *temi* it will block the potential surface output forms *temi* and *cemi* (because the surface sequence *emi* is prohibited). Therefore of the four potential surface forms, three are filtered out; rules 3 and 4 leave only the correct form *cimi*.

Two-level phonology facilitates a rather different way of thinking about phonological rules. We think of generative rules as processes that change one segment into another. In contrast, two-level rules do not perform operations on segments, rather they state static constraints on correspondences between underlying and surface forms. Generative phonology and two-level phonology also differ in how they characterize relationships between rules. Rules in generative phonology are described in terms of their relative order of application and their effect on the input of other rules (the so-called feeding and bleeding relations). Thus the generative rule 1 for Raising precedes and feeds rule 2 for Palatalization. In contrast, rules in the two-level model are categorized according to whether they apply in lexical versus surface environments. So we say that the two-level rules for Raising and Palatalization are sensitive to a surface rather than underlying environment.

## B1.4 With zero you can do (almost) anything

Phonological processes that delete or insert segments pose a special challenge to two-level phonology. Since an underlying form and its surface form must correspond segment for segment, how can segments be deleted from an underlying form or inserted into a surface form? The answer lies in the use of the special null symbol 0 (zero). Thus the correspondence *x:0* represents the deletion of *x*, while *0:x* represents the insertion of *x*. (It should be understood that these zeros are provided by rule application mechanism and exist only internally; that is, zeros are not included in input forms nor are they printed in output forms.) As an example of deletion, consider these forms from Tagalog (where + represents a morpheme boundary):

```
UR:    m a n + b i l i
SR:    m a m 0 0 i l i
```

Using process terminology, these forms exemplify phonological coalescence, whereby the sequence *nb* becomes *m*. Since in the two-level model a sequence of two underlying segments cannot correspond to a single surface segment, coalescence must be interpreted as simultaneous assimilation and deletion. Thus we need two rules: an assimilation rule for the correspondence *n:m* and a deletion rule for the correspondence *b:0* (note that the morpheme boundary + is treated as a special symbol that is always deleted).

```
(5)    Nasal Assimilation
        n:m <=> ____ +:0 b:@
```

```
(6)    Deletion
        b:0 <=> @:m +:0 ____
```

Notice the interaction between the rules: Nasal Assimilation occurs in a lexical environment, namely a lexical *b* (which can correspond to either a surface *b* or 0), while Deletion occurs in a surface environment, namely a surface *m* (which could be the realization of either a lexical *n* or *m*). In this way the two rules interact with each other to produce the correct output.

Insertion correspondences, where the lexical segment is 0, enable one to write rules for processes such as stress insertion, gemination, infixation, and reduplication. For example, Tagalog has a verbalizing infix *um* that attaches between the first consonant and vowel of a stem; thus the infixed form of *bili* is *bumili*. To account

for this formation with two-level rules, we represent the underlying form of the infix  $_{um}$  as the prefix  $x+$ , where  $x$  is a special symbol that has no phonological purpose other than standing for the infix. We then write a rule that inserts the sequence  $_{um}$  in the presence of  $x+$ , which is deleted. Here is the two-level correspondence:

```
UR:    x + b 0 0 i l i
SR:    0 0 b u m i l i
```

and here is the two-level rule, which simultaneously deletes  $x$  and inserts  $_{um}$ :

```
(7)    Infixation
        X:0 <=> ____ +:0 C:C 0:u 0:m V:V
```

These examples involving deletion and insertion show that the invention of zero is just as important for phonology as it was for arithmetic. Without zero, two-level phonology would be limited to the most trivial phonological processes; with zero, the two-level model has the expressive power to handle complex phonological or morphological phenomena (though not necessarily with the degree of felicity that a linguist might desire).

## B2 Rule interaction in two-level phonology

### B2.1 Ordered rules versus two-level rules

Generative phonology and two-level phonology use quite different rule formalisms. They differ from each other in two aspects: how the rules are applied and what is available in the rules' environments. In brief, generative rules are applied in an ordered sequence and their environments have access only to underlying structure, whereas two-level rules are applied simultaneously and their environments have access to both underlying and surface structure. Generative phonology claims that rule ordering is necessary to handle interactions among rules, while two-level phonology claims that if rules have access to both underlying and surface environments, rule ordering is unnecessary.

The debate over rule ordering has been around since the 1960s. An array of possible rule application models were proposed, ranging from extrinsically ordered rules as one extreme to simultaneous rules as the other extreme, with various intermediate positions such as rules that apply sequentially but are not extrinsically ordered. Of these various models, simultaneous rule application was virtually rejected out of hand because it appeared unable to handle certain types of rule interaction. But it is important to note that apparently nothing precisely like the two-level model was ever considered.

A typical textbook example of the argument against simultaneous rule application is found in Kenstowicz and Kisseberth (1979:291ff). They describe a model of simultaneous rule application which they call the *direct mapping hypothesis*. The description "direct mapping" comes from the fact that if all rules are applied simultaneously, then an underlying form is mapped directly onto its corresponding surface form (that is, there are only two levels of representation). This is in contrast to a sequential model of rule application which maps an underlying form to its surface form indirectly via several intermediate levels of derivation. First, they show an example of rule interaction that simultaneous rules can handle. Rules 8 and 9 are the same rules as first used in section B1.2 above, but they are placed in a different order.

```
(8)    Palatalization
        t --> c / ____i

(9)    Vowel Raising
        e --> i / ____C_0 i
```

A sample derivation of forms to which these rules apply looks like this :

```
UR:    temi
(8)    --
(9)    timi
SR:    timi
```

As was observed in section B1.2 above where these rules were first used, the Raising rule potentially *feeds*

the Palatalization rule, since if Raising were applied first it would increase the number of forms that could undergo Palatalization by creating more instances of *i*. However, by ordered Palatalization to apply first, it is prevented from applying before a surface *i* that is produced by Raising. In other words, the order of these rules captures the fact that Palatalization only applies before an underlying *i*. These rules demonstrate a *counterfeeding* order; that is, the feeding rule is ordered after the "fed" rule.

However, if the two generative rules 8 and 9 are allowed to apply simultaneously to the underlying form *temi*, the correct surface form *timi* will be produced. This is because the structural description of Palatalization is not met in the underlying form *temi*, thus it does not apply. But the structural description of Raising is met, so it applies and produces the correct surface form *timi*. Thus rules in a counterfeeding order do not pose a problem for simultaneous rules.

Then, Kenstowicz and Kisseberth show an example of rule interaction that simultaneous rules cannot handle. Rules 10 and 11 are the same as rules 8 and 9, but in the reverse order.

- (10) Vowel Raising  
e --> i / \_\_\_\_C\_0 i
- (11) Palatalization  
t --> c / \_\_\_\_i

A sample derivation of forms to which these rules apply looks like this:

UR:      *temi*  
(10)     *timi*  
(11)     *cimi*  
SR:      *cimi*

Rules 10 and 11 are now in a *feeding* order, since Palatalization (rule 11) applies to the output of Raising (rule 10) to produce the surface form *cimi*. The order of these rules captures the fact that Palatalization applies before a surface *i* regardless of whether its underlying source is *i* or *e* (by means of the Raising rule). We have already seen above that applying these rules simultaneously produces the surface form *timi*. Simultaneous rules cannot produce the desired surface form *cimi*, since Palatalization does not apply to the underlying form.

From this demonstration Kenstowicz and Kisseberth conclude that the direct mapping hypothesis (that is, simultaneous rules) can handle rules in a counterfeeding order but not rules in a feeding order; therefore it is an inadequate model of rule application. However, their conclusion depends on the assumption that phonological rules can refer only to the underlying environment, never the surface environment. If this restriction is removed, then the situation becomes quite different. For example, if rule 11 above could "look ahead" to the surface environment, then it could apply before a surface *i*, thus enabling rules 10 and 11 to apply simultaneously and produce the desired surface form. While there was some discussion in the phonological literature about so-called "global rules"[fn3] that could look ahead to the surface environment, the concept was never accepted into mainstream phonological theory. Two-level phonology takes this germ of an idea and develops it into a fully developed formalism for expressing rule environments that obviates the need for ordered rules.

Here are rules 8 and 9, the counterfeeding rules, expressed as two-level rules:

- (14) Palatalization  
t:c <=> \_\_\_\_ i:@
- (15) Vowel Raising  
e:i <=> \_\_\_\_ C:C\* @:i

Since even simultaneous generative rules can handle rules in a counterfeeding order, translating rules 8 and 9 into two-level rules is straightforward. We have shown above that a "counterfed" rule such as rule 8, the Palatalization rule, is stated in terms of the underlying environment. This is reflected in the environment of rule 14, the two-level rule for Palatalization, where the correspondence *i:@* refers to all feasible pairs whose underlying segment is *i*; in other words, Palatalization applies before an underlying *i* regardless of its surface realization. (Note that rule 15, the two-level rule for Vowel Raising, is discussed in section B1.3 above

as rule 4.)

Here are rules 10 and 11, the feeding rules, expressed as two-level rules:

(16) Vowel Raising  
e:i <=> \_\_\_\_ C:C\* @:i

(17) Palatalization  
t:c <=> \_\_\_\_ @:i

Now Palatalization represents the "fed" rule and therefore applies in a derived rather than underlying environment. This is reflected in the environment of rule 17, the two-level rule for Palatalization, where the correspondence @:i refers to all feasible pairs whose surface segment is i; in other words, Palatalization applies before a surface i regardless of its underlying source (either i or e).

When we look at generative rules that exemplify a bleeding relation, we get analogous results: simultaneous generative rules can handle rules in a counterbleeding relation but not a bleeding relation, but both types of rule relations can be expressed by two-level rules. To demonstrate this, consider rules 18 and 19:

(18) Vowel lowering  
i --> e / \_\_\_\_C\_0 e

(19) Palatalization  
t --> c / \_\_\_\_i

Here is a sample derivation using these rules:

```
UR:    time
(18)   teme
(19)   --
SR:    teme
```

The underlying form *timi* in the second column meets the structural descriptions of both Vowel Lowering and Palatalization. Applying Vowel Lowering first produces the intermediate form *teme* which no longer meets the structural description of Palatalization. Thus Vowel Lowering bleeds Palatalization. The order of these rules captures the fact that Palatalization applies before a surface i. Now here are rules 18 and 19 expressed as two-level rules:

(20) Vowel lowering  
i:e <=> \_\_\_\_C:C\* @:e

(21) Palatalization  
t:c <=> \_\_\_\_@:i

The fact that Palatalization only applies before a surface i is reflected in the environment of rule 17, where the correspondence @:i refers to all feasible pairs whose surface segment is i.

Rules 22 and 23 are the same as rules 18 and 19, but in the reverse order.

(22) Palatalization  
t --> c / \_\_\_\_i

(23) Vowel lowering  
i --> e / \_\_\_\_C\_0 e

Here is a sample derivation using these rules:

```
UR:    time
(22)   cime
(23)   ceme
SR:    ceme
```



For the underlying form *time*, Vowel Lowering now counterbleeds Palatalization, since it potentially bleeds Palatalization (as demonstrated above) but is ordered after it. The order of these rules captures the fact that Palatalization applies before an underlying *i* regardless of its surface realization. Now here are rules 22 and 23 expressed as two-level rules:

- (24) Palatalization  
 $t:c \Leftrightarrow \_\_\_i:@$
- (25) Vowel lowering  
 $i:e \Leftrightarrow \_\_\_C:C^* @:e$

The fact that Palatalization applies before an underlying *i* is reflected in the environment of rule 24, where the correspondence *i:@* refers to all feasible pairs whose underlying segment is *i*.

The preceding discussion shows that the notion of rule interaction is expressed quite differently in two-level phonology than it is in generative phonology. Because generative phonology is based on a formalism where the output of one rule serves as the input to another rule, it views rule interaction as one rule increasing (feeding) or decreasing (bleeding) the potential input to another rule. In contrast, two-level phonology views rule interaction in terms of environmental constraints on rules--whether they apply to underlying or surface environments.

As a practical guide for translating ordered generative rules into two-level rules, here are two rules of thumb:

- If two generative rules A and B are ordered such that rule A feeds or bleeds rule B, then construct a two-level rule for B that is constrained by the surface environment.
- If two generative rules A and B are ordered such that rule B counterfeeds or counterbleeds rule A, then construct a two-level rule for A that is constrained by the underlying environment.

## B2.2 Multiple application

One of the more vexing problems in generative phonology is the problem of multiple application. Multiple application refers to a rule that applies more than once to an input form. This occurs when the structural description of a rule is met at more than one place in a word. For example, consider a rule that lengthens vowels before a voiced consonant: it must apply twice to an underlying form such as *abapad* to produce the surface form *a:bapa:d*. Just as generative phonologists debated various models of rule application, they also debated how to handle multiple application. Chomsky and Halle (1968:344) proposed that a rule should apply simultaneously to each segment in the form that meets its structural description. Thus the vowel lengthening rule just mentioned would simultaneously apply to each vowel in the input form that met its structural description. However, this approach suffers from the same defects as described above for simultaneous generative rules: they cannot handle feeding or bleeding interactions. To see this, consider this generative rule (repeated from rule 1):

- (26) Vowel Raising  
 $e \rightarrow i / \_\_\_C_0 i$

Assume that it is intended to relate underlying and surface forms such as these:

UR: pememi  
 SR: pimimi

Only one segment of the underlying form *pememi* meets the structural description of rule 26; thus simultaneous application of rule 26 will produce the incorrect surface form *pemimi*. The intended surface form *pimimi* shows that Vowel Raising feeds itself; that is, it can apply to its own output. What we need is a derivation something like this:

UR: pememi  
 (26) pemimi  
 (26) pimimi  
 SR: pimimi

This effect can be accomplished by using the *directional iterative* mode of application (see Kenstowicz and Kisseberth 1979:326). This means that the rule passes through the word segment-by-segment to find a segment that meets the structural description of the rule. If it does, the rule applies to the segment and produces a derived form. The iterative process then looks at the next segment in the derived form, and so on, until the end of the word is reached. The iterative mode of application is clearly analogous to rule ordering.

A drawback to the directional iterative mode of application is that it is necessary to specify whether a rule should apply in a left-to-right or a right-to-left direction. For example, rule 26 above must apply right-to-left in order to produce the intended surface form. But what if the surface form is *pemimi* instead? To produce this surface form from the same underlying form, rule 26 must iteratively apply left-to-right. The interaction produced by left-to-right application can be described as "self-counterfeeding." The direction of application can also be used to handle self-bleeding and self-counterbleeding rule interactions.

While the directional iterative mode of application apparently "works," it requires each rule to carry a feature or flag indicating its directionality. This may seem a harmless notational accretion, but it actually indicates a fundamental weakness in the rule formalism: the rules are not cleanly separated from the application mechanism. In other words, rules should express purely phonological information; they should not contain procedural instructions to the application mechanism.

In contrast, the two-level model can handle so-called multiple application with no additional application principles or rule features. In fact, multiple application is simply a natural consequence of the two-level formalism itself. For example, rule 26 above translates into this two-level rule (see also rule 4 in section B1.3):

(27) Vowel Raising  
 $e:i \Leftrightarrow \_\_\_ C:C^* @:i$

Given the underlying form *pememi*, this rule will apply twice to produce the surface form *pimimi*. The correspondence  $@:i$  in the environment of rule 27 includes the feasible pairs  $i:i$  and  $e:i$ , thus enabling the rule to apply before a surface *i* regardless of its underlying source. Since rule 26 was described as self-feeding, it comes as no surprise that its two-level version, rule 27, is constrained by a surface environment (see the principles above for translating generative rules to two-level rules).

Now assume that the Vowel Raising rule must account for this derivation:

UR: pememi  
 SR: pimimi

This is the self-counterfeeding case, which the generative model handles by applying rule 26 left-to-right. The two-level solution is this rule:

(28) Vowel Raising  
 $e:i \Leftrightarrow \_\_\_ C:C^* i:@$

Whereas rule 27 used the correspondence  $@:i$  in its environment, rule 28 uses  $i:@$ . In other words, rule 28 says that Vowel Raising applies only before an underlying *i* (regardless of its surface realization). Since there is only one underlying *i* in *pememi*, rule 28 only applies once.

At this point it is instructive to compare the generative and two-level versions of the Vowel Raising rules. The generative analysis posits just one rule, namely rule 26. To account for the self-feeding case, it specifies that the rule applies right-to-left; while for the self-counterfeeding case, it specifies that the rule applies left-to-right. As noted earlier, this solution requires an extra notational device--a direction flag. The two-level analysis posits two rules, namely rules 27 and 28. To account for the self-feeding case, rule 27 applies in a surface environment; while for the self-counterfeeding case, rule 28 applies in an underlying environment. This is precisely the same way that pairs of rules in feeding and counterfeeding relationships are expressed as two-level rules; thus the two-level model needs no additional devices to handle multiple application.

## B2.3 Deletion and insertion rules

Section B1.4 above shows how two-level phonology uses the null symbol to represent phonological process that delete or insert segments. In most cases, translating generative rules for deletion or insertion into two-level

rules is straightforward; but some cases require special care, such as when a deletion rule feeds another rule.

Kenstowicz and Kisseberth (1979:57) posit these generative rules for Russian:

- (29)      1-drop  
          1 --> 0 / C\_\_\_#
- (30)      Final Devoicing  
          [+obstruent] --> [-voiced] / \_\_\_#

These rules produce the following derivations:

UR:	greb-u	greb	greb-l
(29)	--	--	greb
(30)	--	grep	grep
SR:	grebu	grep	grep

Rule 29 deletes a word-final 1 that follow a consonant and rule 30 devoices a word-final obstruent. The derivations above show that the deletion rule feeds the devoicing rule; that is, rule 29 deletes the final 1 in greb-l which enables rule 30 to apply to the now-final b.

To translate this generative analysis into a two-level analysis, we first set up the two-level correspondences:

UR:	greb-u	greb	greb-l
SR:	greb0u	grep	grep00

Ignoring for the moment the -:0 correspondence for deleting the morpheme boundary symbol, these forms show that devoicing applies even when a 1:0 pair intervenes. Thus rules 29 and 30 can be translated into these two-level rules (where OBS stands for obstruents and OBS[-vc] stands for voiceless obstruents):

- (31)      1-drop  
          1:0 <=> C\_\_\_#
- (32)      Final Devoicing  
          OBS:OBS[-vc] <=> \_\_\_1:0#

While rule 32 will work for the underlying form greb-l, it will not work for greb, which does not have an 1 in it. Therefore the 1:0 correspondence in rule 32 must be optional. Also, we must permit an optional morpheme boundary. Rule 32 is revised as 32':

- (32')      Final Devoicing  
          OBS:OBS[-vc] <=> \_\_\_(-:0)(1:0)#

## B2.4 Mutually bleeding rules

In all the examples thus far in this chapter of interaction among generative rules, the rule that feeds or bleeds another rule does so by affecting the second rule's environment. For example, rule 29 above feeds rule 30 by deleting a segment of its environment. However, a rule can also feed or bleed another rule by increasing or decreasing instances of its input, that is, the symbol on the left of the rewriting arrow. For example, consider these two schematic generative rules:

- (33)      a --> b / x\_\_\_y
- (34)      a --> c / x\_\_\_

Here is a derivation using these rules:

UR:	xay
(33)	xby
(34)	--
SR:	xby

Since rule 33 has the same input symbol as rule 34 (namely  $a$ ), rule 33 bleeds rule 34. Notice however that if rule 34 were applied first, then it would bleed rule 33 and produce the surface form  $x_cy$ . This has been called *mutual bleeding*.<sup>[fn4]</sup> Examples of mutually bleeding rules have been used as arguments for the necessity of ordered rules, since it is only by ordering the rules that the intended surface form is ensured. Thus such examples are a challenge to the two-level model.

To see how to translate these ordered rules into two-level rules, we first rewrite them using two-level notation:

(35)  $a:b \leq x\_y$

(36)  $a:c \leq x\_$

Unfortunately, these two-level rules will not work. This is because the  $\leq$  parts of the rules have a *realization conflict* (or  $\leq$  conflict; see Antworth 1990:85ff for more on rule conflicts).<sup>[fn5]</sup> A realization conflict arises due to the meaning of  $\leq$  rules. A  $\leq$  rule requires the correspondence always to occur in the specified environment. The  $\leq$  part of rule 35, then, says that when the lexical segment  $a$  occurs in the environment  $x\_y$  it must be realized as the surface segment  $b$ . Similarly, the  $\leq$  part of rule 36 says that when the lexical segment  $a$  occurs in the environment  $x\_$  it must be realized as the surface segment  $c$ . The conflict arises when the rules are given an underlying form such as  $xay$ , which meets the structural description of both rules. Rule 35 says that it must be realized as  $xb_y$ , while rule 36 says that it must be realized as  $xc_y$ . It should be remembered that in the two-level model, all rules apply simultaneously and all must apply successfully (even if only vacuously). If any individual rule fails, the derivation of the input form fails and no result is returned. Since it is impossible for both rules 35 and 36 to apply successfully to the underlying form  $xay$ , neither can succeed.

The conflict can be resolved by modifying rule 36 in such a way that it allows (but does not require) the correspondence  $a:b$  (from rule 35) in its environment. Thus in the environment  $x\_y$  rule 35 will sanction the pair  $a:b$  and rule 36 will sanction the pairs  $a:b$  and  $a:c$ . When the rules are applied together, only the intersection of these sets of pairs can succeed, namely  $a:b$ . This is accomplished by splitting rule 36 into a  $\leq$  rule and a  $\Rightarrow$  rule and writing the correspondence part of the  $\leq$  rule as  $a:[b|c]$ , which means that the underlying segment  $a$  is realized either as a surface  $b$  or as a surface  $c$ . Now see rule 36a and 36b:

(36a)  $a:[b|c] \leq x\_$

(36b)  $a:c \Rightarrow x\_$

Comparing the two-level rules 35 and 36a-b with the generative rules 33 and 34, it could be argued that the two-level rules are inferior to the generative rules, since rule 36a requires disjunctive segments, namely  $[b|c]$ , while rule 34 does not. Thus the generative rules, which requires fewer symbols, could be judged simpler or more economical than the two-level rules. Interestingly, Karttunen (1993) argues that a disjunction of symbols such as  $b | c$  is actually simpler (more general, less specific) than just the single symbol  $c$  (which in turn is simpler than a conjunction of symbols such as  $b \wedge c$ ). To see how this claim can be substantiated, we will look at a pair of rules that exemplify common phonological processes (where  $z$  stands for a voiced alveopalatal fricative):

(37) Palatalization  
 $s \rightarrow z / i\_i$

(38) Voicing  
 $s \rightarrow z / V\_V$

Here are sample derivations using the rules:

UR:	asa	isi
(37)	--	iZi
(38)	asa	--
SR:	asa	iZi

Like rules 33 and 34 above, rules 37 and 38 share the same input symbol and thus potentially bleed each other. The important thing to notice here is that the structural description of rule 37 is subsumed by (properly included in) the structural description of rule 38. That is, any input form that meets the structural description of

rule 37 will also meet the structural description of rule 38, but not the other way around. Another way of describing this situation is to say that rule 38 is a general rule that includes rule 37, a more specific rule, in its scope. Rule 38 is a general of intervocalic voicing that changes unvoiced *s* to voiced *z*. Rule 37 also voices *s* in an intervocalic environment, but also palatalizes it to become *ʃ* just in the specific intervocalic environment *i\_\_\_i*. It is no surprise that the Palatalization rule is ordered before the Voicing rule, since we expect that a specific condition will take precedence over a general condition.[fn6]

Rules 37 and 38 translate into these two-level rules:

(39) Palatalization  
 $s:Z \Leftrightarrow i\_i$

(40) Voicing  
 $s:z \Leftrightarrow V\_V$

Like rules 35 and 36 above, rules 39 and 40 have a realization conflict. This is because, given the input form *isi* which meets the structural description of both rules, rule 39 requires *s* to be realized as *ʃ*, while rule 4b requires *s* to be realized as *z*. The conflict is resolved by modifying rule 40 so that it allows (but does not require) *s:Z* to succeed in its environment:

(40a) Voicing  
 $s:[z|Z] \Leftarrow V\_V$

(40b)  $s:z \Rightarrow V\_V$

In rule 40a, the disjunctive segments  $[z|Z]$  form a natural class of segments which differ only with respect to point of articulation. In terms of distinctive features, it takes fewer features to specify  $[z|Z]$  than it takes to specify just *z*. Thus when two (or more) segments are closely related phonetically, it turns out to be true that a disjunction of them is simpler (more economical) than specifying a single segment. (Note that for PC-KIMMO, which uses subsets rather than distinctive features, the segments would be put in a subset named, say, *Z*.)

## B2.5 Input feeding rules

The previous section dealt with a type of generative rule interaction where a rule bleeds the input symbol of another rule. This section deals with an interaction where a rule feeds the input symbol of another rule. This is perhaps the most difficult type of generative rule interaction to express with two-level rules and has been cited as an argument for the necessity of ordered rules. While the following discussion shows how to handle "input feeding" with two-level rules, it may well be the case that input feeding is an overly powerful device whose use in natural language is not warranted.

Consider these two schematic rules which exemplify the input feeding relation:

(41)  $a \rightarrow b / x\_$

(42)  $b \rightarrow c / \_\_y$

The output symbol of rule 41 is identical to the input symbol of rule 42; thus rule 41 feeds the input symbol of rule 42. Applying these rules produces derivations like these:

UR:	<i>xa</i>	<i>by</i>	<i>xay</i>
(41)	<i>xb</i>	--	<i>xby</i>
(42)	--	<i>cy</i>	<i>xcy</i>
SR:	<i>xb</i>	<i>cy</i>	<i>xcy</i>

Relative to the input forms *xa* and *by* the two rules do not interact. But for the input form *xay* rule 41 feeds rule 42 by creating the intermediate form *xby*. The two-level model, of course, does not permit such intermediate forms. Thus it would seem that a two-level analysis of these forms would have to posit a third rule that realizes underlying *a* as a surface *c* just in the environment *x\\_y*.

If this were the only solution available to the two-level model, then it would be a strong point in favor of

ordered rules, since the ordered rule solution needs only two rules while the two-level solution would require a third rule that repeats much of the information already contained in the first two rules. Fortunately, there is another approach available to the two-level model. To get a clue of the possible solution, we will convert the above generative derivations into two-level correspondences:

UR:	xa	by	xay
SR:	xb	cy	xcy

From these correspondences we can make two observations: first, underlying *a* has two surface realizations, *b* and *c*; and second, surface *c* has two underlying sources, *b* and *a*. This suggests a solution using disjunctive input and output symbols. The generative rules above can be rewritten as the following two-level rules:

(43)  $a:[b|c] \Leftrightarrow x\_$

(44)  $[a|b]:c \Leftrightarrow \_\_y$

Rule 43 will sanction the pairs *a:b* and *a:c* after *x*, while rule 44 will sanction the pairs *a:c* and *b:c* before *y*. In the environment  $x\_y$ , then, both rules will apply at once, permitting only the intersection of these sets of pairs, namely *a:c*. Note that these rules can be written more felicitously if we declare a subset *P* consisting of *b* and *c* and a subset *Q* consisting of *a* and *b* and then write the correspondence in rule 43 as  $a:P$  and the correspondence in rule 44 as  $Q:c$ . This implies that *b* and *c* on the one hand and *a* and *b* on the other hand are phonetically related.

The motivation for using rules like 41 and 42 above is to provide an intermediate environment for a third rule. For example, rules 41 and 42 are repeated here as rules 45 and 47 with a third rule inserted between them:

(45)  $a \rightarrow b / x\_$

(46)  $x \rightarrow z / \_\_b$

(47)  $b \rightarrow c / \_\_y$

These rules produce the following derivations:

UR:	xa	by	xay
(45)	xb	--	xby
(46)	zb	--	zby
(47)	--	cy	zcy
SR:	zb	cy	zcy

In the derivation of the underlying form *xay*, the segment *b* only occurs in intermediate forms where it serves as the environment for rule 46. Thus, the argument runs, such an analysis demonstrates that ordered rules and intermediate forms are necessary constructs of a phonological formalism. The two-level rebuttal to this argument should sound familiar by now: rule 46 need only to be generalized to permit its application before either a surface *b* (as in the derivation for *xa*) or a surface *c* (as in the derivation for *xay*). Thus rule 46 translates into two-level rule 49, which together with two-level rules 43 and 44 account for the same derivations as the generative analysis.

(49)  $x:z \Leftrightarrow \_\_a:[b|c]$

Here is how the two-level analysis works. The set of all feasible pairs sanctioned by the rules comprises the special correspondences *a:b*, *a:c*, *b:c*, *x:z* plus default correspondences such as *a:a*, *b:b*, and *x:x*. When applied individually to the underlying form *xay*, the three two-level rules (that is, 43, 44, and 49) sanction these surface forms:

Rule 43:	xby, xcy, zby, zcy
Rule 44:	xcy, zcy
Rule 49:	zby, zcy

When the three rules are applied simultaneously, only the intersection of their outputs succeeds, namely *zcy*.

As an example of a natural language analysis that proposes a set of rules similar to rules 45 to 47, we will examine some data and rules from Lomongo as reported by Kisseberth (1976:48-49).[fn7] First, the generative rules as Kisseberth writes them:

```
(50)    Glide Formation
        [-low, +syll] --> [-syll] / ____V

(51)    Affrication
        t           c
        -->         / ____[-cons, -syll]
        {d,l}       j

(52)    y-Drop
        y --> 0 / {c,j}____
```

The rules produce the following derivations:

```
UR:      ..loa...    ..lia...
(50)     lwa         lya
(51)     jwa         jya
(52)     --         ja
SR:      jwa         ja
```

Rules 50 and 52 exemplify the "input bleeding" relation: rule 50 changes *i* to *y*, and rule 52 then deletes *y*. The *y*, which appears only in intermediate forms, serves as the environment for rule 51.

To translate these generative rules into two-level rules, we first declare the following subsets:

```
SUBSET    G           i o y w
SUBSET    G[+syll]    i o
SUBSET    G[-syll]    y w 0      ;where 0 is the NULL symbol
SUBSET    G[-bk]      i y
SUBSET    D           t d l
SUBSET    J           c j
```

The full set of high vocoids (subset G) is divided into three subsets: syllabic (*i* and *o*), nonsyllabic (*y* and *w*), and nonback (*i* and *y*). Subset D contains the apical consonants while subset J contains their affricate counterparts.

The set of all feasible pairs includes default correspondences such as *a:a*, *t:t*, and *l:l* plus these special correspondences:

```
UR:      i    o    i    y    t    d    l
SR:      y    w    0    0    c    j    j
```

Rule 50, Glide Formation, is translated into two-level rule 53, where the correspondence *G[+syll]:G[-syll]* stands for the set of feasible pairs {*i:y*, *i:0*, *o:w*}:

```
(53)    Glide Formation
        G[+syll]:G[-syll] <=> ____V
```

Rule 51, Affrication, is translated into two-level rule 54, where the correspondence *D:J* stands for the set of feasible pairs {*t:c*, *d:j*, *l:j*}:

```
(54)    Affrication
        D:J <=> ____@:G[-syll]
```

The environment of rule 54 contains the correspondence *@:G[-syll]*, which capture the fact that Affrication can apply before surface nonsyllabic glides produced by rule 53.

Rule 52, *y*-Drop, is translated into two-level rule 55, now called Front Glide Drop, where the correspondence *G[-bk]:0* stands for the set of feasible pairs {*i:0*, *y:0*}:

(55) Front Glide Deletion  
G[-bk]:0 <=> @:J\_\_\_\_

The environment of rule 55 contains the correspondence @:G[-bk], which capture the fact that Front Glide Deletion can apply after surface affricates produced by rule 54.

When applied individually to the underlying form *lia*, the three two-level rules (that is, 53, 54, and 55) sanction these surface forms:

Rule 53:     *lya, la, jya, ja*  
Rule 54:     *jya, ja*  
Rule 55:     *lia, ly, ja*

When the three rules are applied simultaneously, only the intersection of their outputs succeeds, namely *ja*.

There are two crucial ways in which the two-level rules differ from the generative rules. First, the subset G[-syll] includes not only  $\gamma$  and  $w$  but also  $\emptyset$ , the NULL symbol. This permits rule 53 (Glide Formation) to sanction two surface realizations for underlying *i*, namely  $\gamma$  and  $\emptyset$ . A likely objection to this analysis is that by definition a null segment (zero) has no phonetic content and does not belong in a class of segments described as nonsyllabic glides. The alternative is to remove  $\emptyset$  from subset G[-syll] and instead include it as a disjunctive surface segment in the correspondence part of the Glide Formation rule:

(53') Glide Formation  
G[+syll]:[G[-syll] |  $\emptyset$ ] <=> \_\_\_\_V

Thus Glide Formation says that a high vowel either is realized as a nonsyllabic glide or is deleted. However, the Affrication rule must also be revised to include  $\emptyset$  in its environment:

(54') Affrication  
D:J <=> \_\_\_\_@:[G[-syll] |  $\emptyset$ ]

Notice that these changes make rules 53' and 54' look very similar to the schemantic rules 43 and 49 above. The choice between rules 53 and 54 or rules 53' and 54' is largely a matter of taste on the part of the analyst, depending on his or her commitments to phonological theory.

The second significant difference is that the  $\gamma$ -Drop rule of the generative analysis has become the Front Glide Deletion rule in the two-level analysis. What this means is that whereas the generative  $\gamma$ -Drop rule deletes only an underlying  $\gamma$ , the two-level Front Glide Deletion rule deletes both underlying  $\gamma$  and *i*. While at first it might appear that the two-level rule is more complex than the generative rule, the two-level rule is actually simpler: because  $\gamma$  and *i* are phonetically closely related (differing only in syllabicity), it takes fewer features to refer to them as a class than it does to refer to just  $\gamma$ .

The two-level solution, then, offers a slightly different perspective on the phonological processes of Lomongo. In the generative solution, the Glide Formation and  $\gamma$ -Drop rules are formulated independent of each other. It is only an incidental fact that for some words the output of Glide Formation happens to undergo  $\gamma$ -Drop. But in the two-level solution,  $\gamma$ -Drop is generalized to become a rule of Front Glide Deletion, which in effect functions as a special case of the more general Glide Formation rule. The relation between Glide Formation and Front Glide Deletion can be stated like this: in the general case, a high vocoid preceding a vowel is realized as a nonsyllabic glide; in the specific case, a high vocoid following a palatal affricate and preceding a vowel is deleted. Thus the two-level analysis of Lomongo, rather than sacrificing descriptive insight, actually offers a felicitous alternative analysis.

---

[fn1] This section is excerpted from Antworth 1991.

[fn2] This made-up example is used for expository purposes. To make better phonological sense, the forms should have internal morpheme boundaries, for instance *te+mi* (otherwise there would be no basis for positing an underlying *e*). See the section below on the use of zero to see how morpheme boundaries are handled.

[fn3] Find a reference for this. Lakoff, maybe?



[fn4] Find reference.

[fn5] This term is due to Dalrymple and others 1987.

[fn6] This is very close to Kiparsky's Elsewhere Condition.

[fn7] The primary source of this information on which Kisseberth based his analysis is not available to me.