FORMAL ASPECTS OF PHONOLOGICAL DESCRIPTION

by

C. DOUGLAS JOHNSON

University of California, Santa Barbara

1972

MOUTON

THE HAGUE · PARIS

CONTENTS

1

INTRODUCTION

The phonology of a language must provide a phonetic representation for each of the infinitely many sentences generated by the syntax. Hence the phonology as well as the syntax is a finite device that accounts for an infinite set of cases. To be sure, the phonology differs in several important respects from the syntax. Where the phrase-structure component takes a single symbol S as initial input and responds with one of an infinite set of alternative outputs, the phonology behaves as a mapping device accepting any of an infinite set inputs and responding in each case with one or at most a small set of alternative outputs. Under current views of syntax, the transformations too constitute a mapping device, though of a radically different sort from the phonology.

A phonological theory must characterize precisely the form of phonologies and the way they process strings. Among the results of such a theory will be a prediction as to what sorts of mappings can be effected by the phonologies of natural languages. Such a prediction would be analogous to a hypothesis in syntax that all natural languages are, say, context-sensitive. Although it has proven difficult to formulate and sustain strong hypotheses of this sort in syntax, we will try to show that in phonology we are somewhat more fortunate.

When confronted with several phonological formalisms with the same mapping capacity, there are several lines of action we could take. We could simply regard the formalisms as empirically equivalent and choose one of them on the basis of practical convenience. The standard view, however, is that formal theories

of phonology carry an empirical burden far greater than the mere prediction of phonologically possible mappings and that there are therefore additional and perhaps even more important criteria for choosing among them. In particular, it has been held that the naturalness or plausibility of a phonological process ought to be reflected formally by a corresponding simplicity of formulation. Though this kind of consideration is much less well-defined than mapping capacity, it has played a central role in most discussions of notational devices in phonology and will serve as an important guide to our present investigation.

Our first step will be to examine, in Chapter 2, certain mechanisms for characterizing sets of phonological strings. Although the major focus of our attention will be on how such sets should be represented in the contextual portions of rules, some incidental consideration will be given to the problem of representing morpheme structure. The major result of Chapter 2 will be that the familiar schematic notation formalized in Chomsky and Halle (1968) is quite restricted in its capacity to represent sets of strings and consequently reflects a strong empirical claim concerning the nature of such sets as they occur in phonology. Specifically, we shall see that schemata can represent just the regular sets in the technical sense of automata theory. In Chapter 3 we consider some notational devices which do not add to the representational capacity of schemata but which seem to be necessary for linguistically satisfactory formulations. Most of these devices are already familiar from the literature, but we introduce them systematically to show how they fit into a formal system and to establish the notation to be employed in subsequent chapters. One new departure is suggested: the use of bracket notation to represent set intersection.

It is not until Chapter 4 that rules are formally introduced. There we consider some properties of two diametrically opposed types of rule, the iterative and the simultaneous. Our conclusion is that the iterative type is excessively powerful, being able to effect virtually any computable mapping, while the simultaneous type is highly restricted indeed, being able to effect only the sort of

mapping known in automata theory as a finite-state transduction. Thus to confine phonological rules to the simultaneous type is to make another strong claim about phonology, a claim that is essentially correct as far as I can tell.

In Chapter 5 we consider some rule types that are equivalent to the simultaneous in mapping capacity but yield superior formulations in many cases. These new types of rules are called right-linear and left-linear. A body of empirical evidence is considered which leads us to the conclusion that right-linear and left-linear rules should both be allowed in phonological descriptions, although the simultaneous type can apparently be dispensed with.

Linear rules are formalizations of processes which proceed from left-to-right or from right-to-left through a string. Two other ways of formalizing these processes, the restricted iterative and the cyclic, are considered in Chapter 6 and rejected after a review of some empirical evidence.

Distinctive features, originally introduced in Chapter 3, are assumed to be binary up through Chapter 6. In Chapter 7 we consider the effects of allowing integers as feature coefficients. It seems clear that integer coefficients are necessary with at least certain prosodic features. The formal consequence is that certain rules are not strictly finite state and therefore stand as exceptions, albeit of a highly restricted nature, to one of our assertions in Chapter 4. A right-linear tone rule is discussed which manipulates an integrally-valued pitch feature, and it is shown that this rule cannot possibly be formulated with the standard notational devices if simultaneous application is presupposed; a right-linear formulation, on the other hand, seems quite satisfactory. We continue with a discussion of the stress feature, also integrally valued. Our general conclusion is that when stress is a culminative feature, being placed on at most one vowel in any given rule application, then it is either the rightmost or leftmost vowel fitting the structural description of the rule that is affected. Thus in particular we consider unnecessary the complex ordering relations among the subcases of the English Main Stress Rule as given by Chomsky and Halle. It is shown that an alternative formulation,

due to Ross, fits quite neatly into the more restricted formalism that we propose.

Certain general conventions to be used throughout should be taken note of. We use $\emptyset$ to designate the null string; thus $X\emptyset Y = XY$ for all strings X and Y. Also, if X is any string then $X^0 = \emptyset$ and $X^i = X^{i-1}X$ for each positive integer i. Thus $X_1 = X$, $X^2 = XX$, $X^3 = XXX$, and so on. It is important to keep in mind that the notational devices just discussed will not be thought of as actually occurring in expressions that appear in phonological descriptions. Rather, they are part of the metalanguage we use to talk about such expressions. This practice is different from that of Chomsky and Halle (1968, Appendix to Chapter 8), who regard $\emptyset$ and superscript and subscript integers as part of the notation of phonological rules.

Certain portions of the text can be skipped without loss of continuity. The beginning and end of such a portion is signaled by (* and *), respectively.

2

SCHEMATA

We will assume that the phonology of any natural language can be described in terms of a fixed universal alphabet of phonological units. For simplicity of exposition we will usually assume that all phonological units are segments, boundary symbols being usually excluded from consideration. A string of phonological units will be called a phonological string.

It is generally accepted that the phonological component of a generative grammar consists wholly or largely of rules which rewrite phonological strings. Each of these rules operates by appropriately altering short substrings (usually single segments) that satisfy certain conditions. Some of these conditions are contextual: a segment will be rewritten in the specified way only if the substring to the left belongs to a certain set (the left environment) and the substring to the right belongs to a certain set (the right environment). Consider, for example, the Sanskrit rule which changes a dental *n* into a retroflex *ṇ* when the *n* is

(a) preceded somewhere in the same word by a retroflex continuant without an intervening palato-alveolar, retroflex, or dental consonant, and
(b) followed immediately by a sonorant.[1]

[1] For other descriptions of the Sanskrit nasal retroflexion rule see Allen (1951), Emeneau and van Nooten (1968: 7), Langendoen (1968: 84), and Whitney (1889: 64-66). Our way of representing vowels and semivowels in underlying forms is similar to that of Zwicky (1965). I wish to thank Professor Murray Emeneau for personally clarifying certain points of Sanskrit grammar. Any errors that remain are entirely my own.

(a) characterizes the left environment of the rule, (b) the right environment. Now suppose that the segments which may appear in phonological representations when this rule applies fall into the classes indicated in the following table.

CONSONANTAL SEGMENTS

	Velar	*Palato-alveolar*	*Retro-flex*	*Dental*	*Labial*	
	k	c	ṭ	t	p	Noncontinuant
	kh	ch	ṭh	th	ph	
(1)	g	j	ḍ	d	b	
	gh	jh	ḍh	dh	bh	
	ŋ	ñ	ṇ	n	m	
		ś	ṣ	s		Continuant
			r	l		

NONCONSONANTAL SEGMENTS

a i u h

Then some instances of the left environment of the rule will be

uṣ...	(e.g. *uṣnam* becomes *uṣṇam*)
kṣubhaa...	(e.g. *kṣubhaanaam* becomes *kṣubhaaṇaam*)
draui...	(e.g. *drauinam* becomes *drauiṇam*)

The following strings, however, will not be instances of the left environment:

upaakhiaa...	(e.g. *upaakhiaanaam* remains unchanged)
rathii...	(e.g. *rathiinaam* remains unchanged)

Characterizing a rule environment, then, is a matter of describing a set of phonological strings. How such sets are to be represented formally is a crucial question of phonological theory.

In the past it has been proposed that a phonological description also contain a set of statements or rules which characterize the set of phonologically admissible morphemes in a language. The

status and organization of this morpheme-structure component, as we shall call it, is a matter of some controversy, but if we grant it some sort of existence we are again faced with the problem of representing a set of strings, namely, those strings that are phonologically admissible as morpheme shapes. (On the other hand, there seems to be no need at all for a special component to describe the set of admissible phonetic strings, since this set is determined indirectly by the morpheme-structure component and the phonological rules).

Several possible mechanisms for representing sets of phonological strings immediately suggest themselves. We might, for example, generate such sets by means of phrase-structure or even transformational grammars. A number of authors (Romeo 1964, Waratomasikkhadit 1964) have proposed that the admissible phonetic or phonemic strings of a language be described in this manner. However, no one to my knowledge has suggested the use of such powerful devices to characterize the environment of a phonological rule. Morpheme structure too has usually been described otherwise by generative phonologists.

Another approach would be to allow the use of string variables and truth-functional conditions. The left environment of the Sanskrit nasal retroflexion rule could then be given as

PAQ
Conditions: A is a retroflex continuant;
Q $\neq$ RBS if B is a palato-alveolar, retroflex or dental consonant.

This way of describing environments is often found in the literature, though usually conjunction with other formal devices and not in the pure form exemplified here.

Most formalisms actually used or proposed for writing phonological rules and representing morpheme structure appear to be versions of a schematic notation involving two fundamental devices:

(2) (a) Explicit finite lists. The usual notation is $\{X_1, \ldots, X_n\}$, where $X_1, \ldots, X_n$ are the listed items.

(b) Some means of representing an infinite list of the form $X^0, X^1, X^2, X^3, \ldots$. Both $(X)_0$ and $(X)^*$ have been used in this function. We will use $(X)^*$.

Using this notation we can represent the left environment of the Sanskrit rule by the expression

(3) $(\{A_1, \ldots, A_q\})^*$ {r, ʂ} ({k, kh, g, gh, ŋ, p, ph, b, bh, m, a, i, u, h})*

where $A_1, \ldots, A_q$ are all the segments of table (1).

Any meaningful expression involving brace and star notation will be called a schema. From the informal explanation just given it is probably fairly easy to construct and interpret the schemata needed in phonology. However, we will not leave the matter to the reader's intuition but proceed to a formal development. To begin with, we characterize well-formed schemata recursively in (4). The term "elementary symbol", used in (4), is for present purposes equivalent to "phonological unit".

(4) (a) $\emptyset$ is a schema and each elementary symbol is a schema.
(b) If $X_1, \ldots, X_n$ are schemata then so is $X_1 \ldots X_n$.
(c) If $X_1, \ldots, X_n$ are schemata then so is $\{X_1, \ldots, X_n\}$.
(d) If X is a schema then so is $(X)^*$.

To verify that (3), for example, is a schema, we first note that since each segment is a schema by virtue of (4a), the following are also schemata by virtue of (4c):

$\{A_1, \ldots, A_q\}$
{r, ʂ}
{k, kh, g, gh, ŋ, p, ph, b, bh, m, a, i, u, h}

Then because of (4d) each of the following is a schema too

$(\{A_1, \ldots, A_q\})^*$
({k, kh, g, gh, ŋ, p, ph, b, bh, m, a, i, u, h})*

That (3) is a schema now follows directly by virtue of (4b).

At this point it would be well to establish explicitly certain notational conventions which we have in fact been tacitly observing.

Late capital letters U, ..., Z range over arbitrary schemata, the earlier letters P, ..., T over elementary strings (strings of elementary symbols), and the very early letters A, ..., E over elementary symbols.

The interpretation of schemata in phonology is usually given in the form of conventions which expand schemata into explicit lists, which may be finite or infinite. The standard conventions for brace and star are given in (5).

(5) (a) $X\{Y_1, \ldots, Y_n\}Z$ expands to the finite list $XY_1Z, \ldots, XY_nZ$
(b) $X(Y)^*Z$ expands to the infinite list $XY^0Z, XY^1Z, XY^2Z, XY^3Z, \ldots$

(cf. Chomsky and Halle, 1968: 394, 398). By recursively applying these conventions one presumably ends up with a list of elementary strings. These strings constitute the set represented by the original schema.

In general, of course, we cannot literally write out lists resulting from expansions since these may very well be infinite. (5) is apparently just a metaphorical version of (6).

(6) (a) Each elementary string represents the set consisting of that string alone.
(b) $X\{Y_1, \ldots, Y_n\}Z$ represents the union of the sets represented by $XY_1Z, \ldots, XY_nZ$.
(c) $X(Y)^*Z$ represents the union of the sets represented by $XY^0Z, XY^1Z, XY^2Z, XY^3Z, \ldots$

It is convenient to say that a string is subsumed under a schema if it belongs to the set represented by that schema.

Alternatively, then, we state (6) as in (7).

(7) (a) Each elementary string subsumes itself and only itself.
(b) P is subsumed under $X\{Y_1, \ldots, Y_n\}Z$ if and only if it is subsumed under XY_iZ for some i, $1 \leq i \leq n$.
(c) P is subsumed under $X(Y)^*Z$ if and only if it is subsumed under XY^iZ for some $i \geq 0$.

Another way of interpreting (5) is this. Suppose that instead of replacing a schema by an expanded list we replace the schema by an arbitrarily chosen item in that list. Continuing in this way we eventually arrive at a particular elementary string. Suppose we say that the sequence of items written down in this process is a chain, and that adjacent items constitute links. We can formalize this alternative interpretation of (5) as in (8).

(8) (a) A brace-removing link is an ordered pair of the form $(X\{Y_1, \ldots, Y_n\}Z, XY_iZ)$ where $1 \leq i \leq n$.
(b) A star-removing link is an ordered pair of the form $(X(Y)^*Z, XY^iZ)$ where $i \geq 0$.
(c) A link is an ordered pair that is either a brace-removing link or a star-removing link.
(d) A chain is a nonempty sequence $(X_1, \ldots, X_n)$ in which (X_i, X_{i+1}) is a link for each i, $1 \leq i \leq n-1$.

Under this conception a schema X represents the set of all P which occur at the end of chains beginning with X. An example of a chain beginning with schema (3) and ending in the string *brahma* is given in (9). For brevity we have set

$U = \{A_1, \ldots, A_n\}$
$V = \{k, kh, g, gh, ŋ, p, ph, b, bh, m, a, i, u, h\}$.

For each line we indicate whether it forms a brace-removing link or a star-removing link with its predecessor.

(9)

(U)*{r, s} (V)*	
U {r, s} (V)*	star-removing
b {r, s} (V)*	brace-removing
br(V)*	brace-removing
brVVVV	star-removing
braVVV	brace-removing
brahVV	brace-removing
brahmV	brace-removing
brahma	brace-removing

Some fairly obvious consequences of the definitions given so far are stated in (10).

(10) (a) (X, X′) is a link if and only if there are schemata W, Y, Y′, Z such that $X = WYZ$, $X' = WY'Z$, and (Y, Y′) is a link.
(b) If $(X_1, \ldots, X_n)$ and $(X_n, \ldots, X_{n+m})$ are chains, then so is $(X_1, \ldots, X_n, \ldots, X_{n+m})$.
(c) If $P \neq X$, then P is subsumed under X if and only if there is a link (X, X′) such that P is subsumed under X′.

The question of primary interest, of course, is whether or not (7) and (8) yield equivalent interpretations of schemata. The answer is yes; that is,

(11) P is subsumed under X if and only if some chain begins with X and ends in P.

(*First we prove (12)).

(12) If there is a chain beginning with X and ending in P, then X subsumes P.

Let the chain be $(X_1, \ldots, X_n)$. Suppose $n = 1$. Then $X_1 = X_n = P$. But by (7a) P subsumes itself. Suppose next that $n > 1$ and that (12) is true for all chains of length less than n. By the inductive hypothesis P is subsumed under X_2 and hence, because (X_1, X_2) is a link, under X_1 (cf. (10c)).

To prove the converse of (12) we make use of a depth measure. The depth of any schema X, denoted d(X), is defined recursively as follows:

(13) (a) If X is an elementary string then $d(X) = 1$.
(b) If $X = YZ$ then d(X) is the maximum of d(Y), d(Z).
(c) If $X = \{Y_1, \ldots, Y_n\}$ then d(X) is one greater than the maximum of $d(Y_1), \ldots, d(Y_n)$.
(d) If $X = (Y)^*$ then d(X) is one greater than d(Y).

This definition assigns a unique depth to every schema and is so framed that every schema is at least as deep as any schema it contains.

Schema (3) provides convenient examples. Letting U and V have the same values as in (9) we have:

Expression	*Depth*
any segment	1
brahma	1
{r, s}	2
U	2
V	2
U {r, s}	2
{r, s} V	2
(U)*	3
(V)*	3
(U)* {r, s}	3
{r, s} (V)*	3
(U)* (V)*	3
(U)* {r, s} (V)*	3

For each schema X of depth greater than 1 there will be schemata W, Y, and X such that

(i) X = WYZ;
(ii) Y has the form {...} or (...)*; and
(iii) Y has the same depth as X.

The number of distinct triples (W, Y Z) satisfying (i)-(iii) will be called the linear complexity of X, denoted lc (X). Thus if X is schema (3), X will have a linear complexity of 2 because both of the following triples, but no others, satisfy (i)-(iii):

(∅, (U)*, {r, s} (V)*)
((U)* {r, s}, (V)*, ∅)

where U and V are as in (9). Now suppose that P is subsumed under X and that (W, Y, Z) is a triple satisfying (i)-(iii). Because of (10a) and (10c) there is a schema Y′ such that (Y, Y′) is a link and P is subsumed under WY′Z. The schema Y′ must be shallower than Y and hence shallower than X (cf. (13c) and (13d)), while Y itself is as deep as X because by hypothesis it satisfies (iii) above.

Consequently, if the linear complexity of X is greater than 1, the linear complexity of WY′Z will be one less than that of X, although the depths of X and WY′Z will be the same. On the other hand, if the linear complexity of X is 1, the depth of WY′Z will be one less than that of X.

We can now show (14)

(14) If the depth of X is greater than 1 and P is subsumed under X, then there is a chain beginning with X and ending in some schema which is shallower than X and which subsumes P.

Suppose first that lc (X) = 1. It follows from the remarks in the preceding paragraph that there is a link (X, X′) where X′ is shallower than X and subsumes P. But this link is also a chain. Now suppose that (14) is true for lc(X) $\leq$ k. Consider the case where lc (X) = k + 1. It follows from the remarks of the preceding paragraph that there is a link (X, X′) where X′ has the same depth as X, has a linear complexity less than that of X, and subsumes P. By the inductive hypothesis there is a chain (X′, X_1, ..., X_n) such that X_n is shallower than X′ and subsumes P. But (X, X′, X_1, ..., X_n) is also a chain, in fact, a chain that begins with X and ends in a schema that is shallower than X and subsumes P.

We are now ready to consider the converse of (12) directly. This is given in (15).

(15) If P is subsumed under X, then there is some chain beginning with X and ending in P.

Suppose d(X) = 1. Then P = X, and the sequence consisting of P alone is a chain beginning with X and ending in P. Suppose next that (15) is true for d(X) $\leq$ k. Consider the case where d(X) = k + 1. By (14) there is a chain (X, X_1, ..., X_n) in which X_n is shallower than X and X_n subsumes P. By the inductive hypothesis there is a chain (X_n, ..., X_{n+m}) where X_{n+m} = P. But (X, X_1, ..., X_{n+m}) is a chain beginning with X and ending in P. This concludes the proof of (11).*)

A third way of interpreting schemata, the last we shall consider, is given in (16).

(16) (a) $\emptyset$ subsumes itself and only itself, and each elementary symbol subsumes itself and only itself.
(b) P is subsumed under XY if and only if there are strings Q and R such that $P = QR$, Q is subsumed under X, and R is subsumed under Y.
(c) P is subsumed under $\{X_1, \ldots, X_n\}$ if and only if it is subsumed under some X_i $(1 \leq i \leq n)$.
(d) P is subsumed under $(X)^*$ if and only if it is subsumed under some schema of the form X^i $(i \geq 0)$.

(16) interprets schemata in precisely the same way as (7) and (8). (*To demonstrate this we need only show that (16) implies (7) and conversely. That (16) implies (7) can be seen from the fact that (7a), (7b) and (7c) can be derived from (16) by taking (16a), (16c), and (16d), respectively, in conjunction with (16b). That (7) implies (16) can be seen from the following considerations. (16a), (16c) and (16d) are just special cases of (7a), (7b), and (7c), respectively. As to (16b), we can reason as follows. First, suppose we know that $P = QR$, where Q is subsumed under X and R is subsumed under Y. Then there will be chains $(X_1, \ldots, X_n)$ and $(Y_1, \ldots, Y_m)$ such that $X = X_1$, $X_n = Q$, $Y = Y_1$, and $Y_n = R$. However, $(X_1Y_1, \ldots, X_nY_1, \ldots, X_nY_m)$ is also a chain (cf. (10a)); in fact, it is a chain beginning with XY and ending in P. On the other hand, suppose we know that P is subsumed under XY. Then there will be a chain beginning with XY and ending in P, and this chain must have the form $(X_1Y_1, \ldots, X_nY_n)$ where $X = X_1$, $Y = Y_1$, and for each i, $1 \leq i \leq n-1$, either

(i) (X_i, X_{i+1}) is a link and $Y_i = Y_{i+1}$, or else
(ii) $X_i = X_{i+1}$ and (Y_i, Y_{i+1}) is a link.

(Again, cf. (10a).) Since $(X_1, \ldots, X_n)$ and $(Y_1, \ldots, Y_m)$ can be turned into chains merely by deleting repetitions, $X_1 = X$ subsumes X_n and $Y_1 = Y$ subsumes Y_n. But $P = X_nY_n$, so that we can take X_n and Y_n, respectively, as Q and R.*)

The result just obtained is of some significance because it means that schemata are just notational variants of regular expressions,

familiar from theory of finite automata (cf. the definitions in McNaughton and Yamada 1960, where the term 'denote' corresponds to our 'subsume' and where the notation $X_1 \cup \ldots \cup X_n$ corresponds to our $\{X_1, \ldots, X_n\}$). The only discrepancy is the trivial one that we have no method of representing the null set, a gap we can immediately remedy by introducing the symbol O (to be distinguished from $\emptyset$ denoting the empty string in our metalanguage). We can, then, represent with schemata all and only those sets that we can represent with regular expressions. These sets, which are said to be regular, constitute a highly restricted family, ranking lowest in the following familiar hierarchy:

Recursively enumerable sets
Context-sensitive sets
Context-free sets
Regular sets.

As is well known, each family of sets in this list is a proper subfamily of any family listed above it.

To propose, then, that schematic notation is adequate for writing phonological descriptions is to assert that all the sets that need to be referred to are regular. This is a strong claim in view of the highly restricted nature of regular sets. If the claim is correct, as I believe it is by and large, phonology stands in sharp contrast to syntax, where comparably strong hypotheses seem not to be tenable. Furthermore, we now have a profound reason for rejecting the proposal discussed earlier that phrase-structure or transformational grammars be used to generate sets of phonological strings, for many grammars of these types generate nonregular sets (indeed, some recent work of Kimball (1967), of Peters and Ritchie (1969) and of Ginsburg and Partee (1969) suggests that we can generate any recursively enumerable set with a transformational grammar). The free use of string variables is also excluded by the regularity claim, for with such variables we can generate such sets as (i) and (ii), which are known not to be regular.

(i) The set of all strings of the form PAP.
(ii) The set of all strings of the form PAQ, where $P \neq Q$.

To conclude this chapter, let us consider which of the three equivalent ways of interpreting schemata should be taken as axiomatic. This question is of no great theoretical import, but it must be decided before we proceed to further development of the formalism in the next chapter. In fact, we will find it most convenient to take (16) as axiomatic, and we will henceforth refer to clauses (16a)-(16d) as interpretive axioms. (8) then becomes a set of auxiliary definitions and (7) becomes a set of theorems along with the other assertions made in this section about subsumption. The clauses (4a) through (4d), which define well-formed schemata, will be known as constructive axioms.

3

REFINEMENTS OF THE FORMALISM

Neither braces nor star can be removed from the formalism of schemata without drastically reducing the family of representable sets. Hence both braces and star must be regarded as primitive notational devices. However, we can exclude from the primitive notation every expression of the form $X\{Y_1, \ldots, X_n\}Z$ in which X or Z is nonnull, since such an expression represents exactly the same set as $\{XY_1Z, \ldots, XY_nZ\}$. For an example we can turn again to schema (3) of the preceding section. Letting U and V be as in (9), we can write this schema as $(U)^*\{r, s\}(V)^*$. This schema, which is not primitive, represents the same set as the primitive schema

$$\{(U)^*r(V)^*, (U)^*s(V)^*\}$$

Although we can represent all phonologically relevant sets with primitive schemata, assuming these sets to be regular, we cannot always do so in a way that is satisfactory for linguistic purposes, because we frequently cannot express linguistically significant generalizations concerning the subsumed strings. For this reason it is essential to introduce nonprimitive devices of an abbreviatory nature into the formalism. The classic example of such a device is nonprimitive brace notation, as exemplified in schema (3). Since this use of braces is so well established in phonology, we will accept it here without further comment.

Another aspect of our formalism that needs clarification is the status of phonological units. We will make the usual assumption that it is not the alphabet of units that is to be taken as primitive

but rather some fixed finite universal set of distinctive features F_1, ..., F_d, each of which is associated with a set of possible coefficients. Phonological units are then defined by the constructive axiom (17).

(17) Each expression of the form $[K_1F_1, \ldots, K_dF_d]$, where K_i is a possible coefficient of F_i, is a phonological unit.

The expressions K_iF_i will be called feature specifications. Of course, since certain combinations of features specifications are inherently impossible, (17) can be regarded only as a first approximation to the definition of unit. However, we will attempt no further refinement here.

The distinctive features to be assumed here unless otherwise noted will be those proposed by Chomsky and Halle (1968: Chapter 7) as amended on page 354 to include a feature of syllabicity. We also assume that the possible coefficients of each feature constitute a finite set; in fact, we generally take these coefficients to be just plus and minus. Under these assumptions the alphabet of units is still finite, and the hypothesis of regularity put forward in the previous chapter remains unaffected. The consequences of allowing an infinite set of coefficients for some feature (all the positive integers, say), which may involve a departure from regularity, will be considered in a later chapter.

The distinctive features form the basis of a highly important system of abbreviatory notation. A simple version of this notation is given by (18), which includes one constructive axiom (18a) and one interpretive axiom (18b).

(18) (a) Each feature specification is a schema.
(b) P is subsumed under a feature specification if and only if P is a unit containing that specification.

Thus *t* is subsumed under +cor(onal) but not under −cor.

Suppose now we extend the formalism to include the new axioms (19a) and (19b), which are constructive and interpretive, respectively.

(19) (a) If $X_1, \ldots, X_n$ are schemata, then $[X_1, \ldots, X_n]$ is a schema.

(b) P is subsumed under $[X_1, \ldots, X_n]$ if and only if it is subsumed under each X_i.

In other words, $[X_1, \ldots, X_n]$ represents the intersection of the sets represented by $X_1, \ldots, X_n$. Since the intersection of two or more regular sets is regular, the bracket notation hereby introduced does not increase the family of representable sets and can therefore be regarded as nonprimitive.

The main justification for (19) is that it yields the full apparatus of distinctive feature notation when conjoined with (18). Thus *t* is subsumed under [+cor, −voice] because it is subsumed under +cor and −voice, but is not subsumed under [+cor, +voice] because it is not subsumed under +voice. To consider a slightly more complex example possible under the extended formalism, *a* is subsumed under [+syl, {+tns, −high}] because it is subsumed under both +syl and {+tns, −high}, and it is subsumed under {+tns, −high} because it is subsumed under −high. Two minor differences between our notation and the customary one should be noted. First, we allow a feature specification to stand unbracketed to represent the set of all units containing that specification. However, since [X] is equivalent to X, we can if we wish bracket otherwise unadorned feature specifications, a practice we will follow. The other difference is that our definitions imply [ ] = [0]; hence, [ ] stands for the set containing just the null string rather than the set of all phonological units. For the latter set we will use the special symbol $.

The need for distinctive feature notation is glaringly apparent in schema (3), and we are now able to recast it more adequately. Suppose that the segments of table (1) are specified in part as follows (this analysis is based on Chomsky and Halle 1968: 314):

	Coronal	*Distributed*	*Anterior*	*Continuant*
k, kh, g, gh, ŋ	−		−	−
c, ch, j, jh, ñ	+	+	−	−
ṭ, ṭh, ḍ, ḍh, ṇ	+	−	−	−
t, th, d, dh, n	+	−	+	−
p, ph, b, bh, m	−		+	−

ŝ	+	+	−	+
ʂ	+	−	−	+
s	+	−	+	+
r	+	−	−	+
l	+	−	+	+
a, i, u, h	−		−	+

Schema (3) can then be reformulated as:

($)*[+cont, +cor, −ant, −distr] ([−cor])*

Although bracket notation has been motivated chiefly by its utility in representing classes of phonological units, it has, under our formalism, a potentially much vaster range of application. Consider, for example, the readjustment rule of English (Chomsky and Halle 1968: 175) which marks a vowel as exempt from part of a certain laxing rule if that vowel is followed by a string subsumed under

[+cons, +ant, +cor] [+cons, +cor]

To be subsumed under this schema a string must have the following characteristics:

(i) it must consist of two segments,
(ii) it must consist of consonantal coronal segments, and
(iii) it must begin with an anterior segment.

Notice, however, that the schema does not directly express these three generalizations; rather, it describes each of the two segments in the cluster separately. It might be held that this is undesirable. If so, we can use bracket notation to express (i)-(iii) directly, as follows:

[$$, ([+cons, +cor])*, [+ant]$]

Whether this is to be regarded as an improvement is disputable, of course. Consider another case. It has become customary to use an expression of the form X_1 to stand for $X^1(X)^*$. Thus $[-syl]_2$ would subsume clusters of two or more nonsyllabics. We can reproduce the essence of this subscript notation by means of

brackets and $ if we want; for example, we can write [([−syl])*, $$($)*]. This expression says directly that (i) the subsumed strings consist entirely of nonsyllabics and (ii) the subsumed strings consist of two or more units. The bracket and $ notation may in some respects be superior to the subscript notation. Consider the following phenomenon, discussed by Kiparsky (1968: 180). Old English had a rule that laxed a vowel before a cluster of at least three consonants, and Early Middle English generalized the environment to include clusters of two consonants as well. Without subscripts or extended use of brackets we could describe this change as simplification of the schema [−syl] [−syl] [−syl] to [−syl] [−syl]. However, a rule using this schema does not directly express the generalization that members of the clusters are all consonants; the segments are specified independently. We might then want to write the schemata of the two historic versions of the rule as $[-syl]_3$ and $[-syl]_2$. Then, however, if the subscripts are an official part of the notation there is no difference in simplicity, and one way of explaining the historic change in the rule is lost. It would seem that the schemata [([−syl])*, $$$] and [([−syl])*, $$] would resolve this dilemma if $ were assigned some small fractional cost less than that of a distinctive feature.

The examples just discussed are intended only as illustrative. I have not pursued a thorough investigation of bracket notation, and it is entirely unclear to me how far its use should be extended. Henceforth, therefore, we will usually confine our use of brackets to the representation of classes of phonological units in conformity with the usual custom.

Still another abbreviatory device that has come into wide use is the variable. In the preceding section we saw that variables ranging freely over strings make it possible to represent nonregular sets, and since we are assuming that nonregular sets do not come up in phonology, we will want to constrain variable notation in some appropriate way. What we will do is allow variables to range only over feature coefficients and phonological units.

The basic formalism for variable notation is given in (20). We have adopted the convention of using capital letters from G through

O for strings that are not necessarily either schemata or phonological strings.

(20) (a) If K is a coefficient (resp.: unit) and L is a coefficient (resp.: unit) variable, then K is a substitution instance of L.
(b) If GKH is a schema and K is a substitution instance of L, then GLH is a schema.
(c) Let X be a schema of the form $G_0L_1G_1...L_nG_n$, where each L_i is a variable and each G_i is free of variables. A substitution instance of X is a schema of the form $G_0K_1G_1...K_nG_n$, where K_i is a substitution instance of L_i and $K_i = K_j$ if $L_i = L_j$.
(d) If X contains variables, P is subsumed under X if and only if it is subsumed under a substitution instance of X.

In deference to custom, but not from any consideration of principle, we will use early Greek letters α, β, γ, ... as coefficient variables. Late Greek letters σ, τ, υ, ... will be used as unit variables. Actually, unit variables have appeared in a different guise under another name, but we will not be ready to discuss this matter until we treat rule formalism.

As the formalism now stands it is inconsistent because paradoxes such as the following arise at every turn. According to (20) both *t* and *a* are substitution instances of σ and hence both are subsumed under σ. Also, according to (20), the strings *tt* and *aa*, but not *ta* or *at*, are subsumed under $\sigma\sigma$. But by (16b) both *ta* and *at* are subsumed under $\sigma\sigma$. To avoid this contradiction we must replace (16b) by (21).

(21) If neither X nor Y contain variables, then P is subsumed under XY if and only if there are strings Q and R such that P = QR, Q is subsumed under X, and R is subsumed under Y.

Now for a simple example involving unit variables. The set of all strings of the form ABBA, where A is a vowel and B a consonant, can be represented by the schema

[+syl, σ] [−syl, τ] [−syl, τ] [+syl, σ]

Replacing the variables by *a* and/or *t* in all the allowable ways, we obtain the following as substitution instances of the above schema:

[+syl, a] [−syl, t] [−syl, t] [+syl, a]
[+syl, a] [−syl, a] [−syl, a] [+syl, a]
[+syl, t] [−syl, t] [−syl, t] [+syl, t]
[+syl, t] [−syl, a] [−syl, a] [+syl, t]

The first of these substitution instances subsumes only the string *atta*. The remaining three subsume nothing whatever, because nothing can be subsumed under [−syl, a] or [+syl, t]. This is precisely the desired result.

There are certain auxiliary devices which are of little use by themselves but can be employed to good effect in conjunction with variable notation. Among these are conditions, counterparts, and coefficient strings.

Conditions are frequently used to further restrict the ways in which variables can be replaced. Let us say that an expression of the form (K = L), where K and L are both units or both coefficients is an atomic condition. Nonatomic conditions can be constructed by using negation, denoted by a prefixed hyphen, or any of the sentential connectives 'and', 'or', 'if...then'. To bring conditions into schemata we add (22) to the existing axioms. (22a) is constructive, (22b) interpretive.

(22) (a) If X is a schema containing no condition and if K is a condition, then X:K is a schema.
(b) P is subsumed under X:K if and only if P is subsumed under X and K is true.

Thus X:K represents either the same set as X or the null set, according as K is true or false. Consider, for example, the schema $\sigma\tau$:−($\sigma = \tau$), which represents the set of all clusters consisting of two nonidentical units. Among the substitution instances of this schema are *nt*:−(*n* = *t*) and *tt*:−(*t* = *t*). *nt*:−(n = t) has a true

condition, n and t being in fact nonidentical; consequently it represents the same set as nt (which subsumes only itself). tt: $-(t = t)$ has a false condition and subsumes nothing.

Schemata that might be referred to as counterpart expressions are introduced in (23).

(23) (a) If $X_1, \ldots, X_n$ are feature specifications and A is a phonological unit, then $[X_1, \ldots, X_n, (A)]$ is a schema.
(b) A is subsumed under
$[X_1, \ldots, X_p, KF_i, ([Y_1, \ldots, Y_q, LF_l, Z_1, \ldots, Z_r])]$
if and only if it is subsumed under
$[X_1, \ldots, X_p, ([Y_1, \ldots, Y_q, KF_i, Z_1, \ldots, Z_r])]$.
(c) $[(A)]$ subsumes A and only A.

As an example, consider the counterpart expression [−voice, (d)]. (23b) tells us that to interpret this expression we should substitute −voice for the feature specification +voice in d; the result will be the voiceless counterpart of d, which is t. The primary use of counterpart expressions will be in the structural change portion of rules, treated in the next chapter.

Coefficient variables become more flexible instruments if strings of coefficients are allowed to occur in feature specifications and if the familiar equivalences $++ = +$, $-- = +$, $+- = -$, and $-+ = -$ are adopted as interpretive principles. We introduce this commonplace notation in (24).

(24) (a) If K is a nonempty string of coefficients of F_i, then KF_i is a schema.
(b) Let K be a possibly empty string of coefficients of F_i. Then P is subsumed under $++KF_i$ (resp.: $--KF_i$, $+-KF_i$, $-+KF_i$) if and only if it is subsumed under $+KF_i$ (resp.: $+KF_i$, $-KF_i$, $-KF_i$).

The variable notation and associated auxiliary devices just introduced do not increase the family of sets representable by schemata. It is obvious that this is true of the auxiliary devices considered by themselves; for the very axioms that interpret them are essentially statements of equivalence between schemata that contain

them and schemata that do not; thus X:K is equivalent to X or $\varnothing$, each counterpart expression is equivalent to a specific phonological unit, and each schema of the form KF_i is equivalent to either $+F_i$ or $-F_i$. As to unit and coefficient variables, each of them has only a finite set of substitution instances. Hence any schema X containing such variables, but no other variables, will have a finite set of substitution instances $Y_1, \ldots, Y_n$ and will be equivalent, therefore, to the variable-free schema $\{Y_1, \ldots, Y_n\}$.

The schematic notation as it now stands will form the basis of our discussion of rules, to which we turn in the next chapter. Some customary devices that have not so far been mentioned, such as angle brackets and parentheses, will be discussed in Chapter 7.

Some conventions adopted for solely typographical reasons are the following. A list of items separated by commas, whether enclosed in brackets or braces, will frequently be displayed vertically. In addition, we will write X^* instead of $(X)^*$ where X is a braced or bracketed schema or \$. Thus our final reformulation of schema (3) is

$$\$^* \begin{bmatrix} +\text{cont} \\ +\text{cor} \\ -\text{ant} \\ -\text{distr} \end{bmatrix} [-\text{cor}]^*$$

4

ITERATIVE AND SIMULTANEOUS RULES

A generative phonology is a system of rules for mapping phonological strings into phonetic realizations. We will suppose that the most rudimentary form such a rule can have is $P \rightarrow P'/Q - R$, where P, Q, R, and P′ are phonological strings. This type of rule will be referred to as elementary and will be said to have QPR as its input and QP′R as its output. Usually such a rule is thought of as applying to any string of the form SQPRT, but for convenience in formalization we will take the more atomistic view that it applies only to the specific string QPR. If $P = P'$, the rule is said to be vacuous.

In general it is neither possible nor desirable to represent a phonology as an explicit list of elementary rules. Usually, in order to preserve the finiteness of the grammar and to express significant generalizations, we must resort to nonelementary rules which, by appropriately economical means, can achieve the effect of a large or even infinite series of elementary rules. It is a widely accepted view that these nonelementary rules should be constructed by means of a schematic notation similar to that developed in the preceding chapters. Accordingly we will allow arrow, slash, and dash to be elementary symbols along with phonological units, thereby implicitly introducing schemata that subsume strings containing these symbols. A nonelementary rule will then be thought of as having the general structure G:X, where G is a symbol designating a particular mode of application and X is a schema subsuming elementary rules, these being referred to as the subrules of G:X. For the time being we will not associate a particular kind

of application with a particular kind of schematic expression; for example, we will make no special association between conjunctive ordering and braces. Rather, we will treat all types of schematic expression in a uniform manner, assuming that the rule G:X treats an input string in a way that is determinable entirely from

(i) the input string,
(ii) the mode of application G, and
(iii) the set of elementary rules subsumed under X.

Just what modes of application should be made available to phonological rules is the question to which we now turn.

There are some phonological rules which make changes at no more than one place in any input form. A rule which places a stress on the first vowel of a word or which devoices word-final obstruents is a rule of this type. For any such rule a very simple method of application suffices. Given the string P, we try to find out whether the rule which we wish to apply to P has a subrule whose input is P. If we find such a subrule, we take its output as the output of the application; if we find no such rule we take P itself as the output.

If all phonological rules were of the sort just described, our discussion would be at an end. There are, of course, many rules that can change several different places in an input string; in fact, there may be no principled upper bound to the number of places affected or to the distance separating these places. The Sanskrit rule discussed toward the beginning of Chapter 2, for example, must retroflex every *n* occurring in the appropriate environment, regardless of how many such *n*'s there may be in an input word. An elementary rule, however, changes just one place in an input string. In general, then, we must allow several subrules to be involved in each rule application. One way we might do this is as follows. Instead of stopping after a single one-place application we continue performing such applications until we obtain a string which cannot be further changed. This sort of application, which we shall call iterative, was once proposed by Harms (1966a: 608) and has been discussed by McCawley (1968: 20-22) in connection

with sets of rules that are not necessarily elementary in our sense. We can formalize the notion of iterative rule as in (25).

(25) (a) An iterative rule is an expression of the form *I*:X, where X is a schema subsuming elementary rules (the subrules of *I*:X) and *I* is a constant denoting the mode of application next to be defined.

(b) Let P and Q be phonological strings and let *I*:X be an iterative rule. Then *I*:X maps P into Q if and only if there is a sequence $(P_1, \ldots, P_n)$ of phonological strings such that

(i) $P_1 = P$;

(ii) for each i, $1 \leq i \leq n-1$, some nonvacuous subrule of *I*:X has P_i as input and P_{i+1} as output;

(iii) *I*:X has no nonvacuous subrule with P_n as input; and

(iv) $P_n = Q$.

The sequence $(P_1, \ldots, P_n)$, if it exists, can be referred to as an application of *I*:X to P.

An iterative version of the Sanskrit nasal retroflexion rule is given in (26).

(26) $$I\text{:}\begin{bmatrix}+\text{cor}\\+\text{nas}\\ \sigma\end{bmatrix} \rightarrow \begin{bmatrix}-\text{ant}\\(\sigma)\end{bmatrix} \Big/ \$^* \begin{bmatrix}+\text{cont}\\+\text{cor}\\-\text{ant}\\-\text{distr}\end{bmatrix} [-\text{cor}]^*-[+\text{son}]\ \$^*$$

An application of this rule is displayed in (27). Each line other than the first is derived from the preceding line by virtue of the indicated subrule of (26).

(27) uṣnataraanaam
uṣṇataraanaam (n → ṇ/uṣ—ataraanaam)
uṣṇataraaṇaam (n → ṇ/uṣṇataraa—aam)

Note that application (27) is complete despite the fact that the schema in (26) subsumes both the following:

ṇ → ṇ/uṣ—ataraaṇaam
ṇ → ṇ/uṣṇataraa—aam

Because these subrules are vacuous they cannot be applied to the output of (27). If they were allowed to apply, they would do so forever, and (26) would not have provided *uṣṇataraaṇaam* with any realization at all. The desired output could have been obtained only by complicating the expression to the left of the arrow to read

$$\begin{bmatrix}+\text{cor}\\+\text{nas}\\+\text{ant}\\ \sigma\end{bmatrix}$$

There is another application of (26) that has the same input and output as (27), namely (28).

(28) uṣnataraanaam
uṣnataraaṇaam (n → ṇ/uṣnataraa—aam)
uṣṇataraaṇaam (n → ṇ/uṣ—ataraaṇaam)

A method of application diametrically opposed to the iterative would be this. Instead of applying subrules in series, changing the string under consideration step by step, we extract just those subrules which have this string as their common input and then make simultaneously all the changes that these subrules call for. This kind of application has been discussed by McCawley (1968: 20-22) and has been proposed by Chomsky and Halle (1968: 392, 398) as the appropriate way of interpreting rules of certain restricted forms.

Formalization of simultaneous rules would be straightforward if every subrule were of the form A → P/R—S, where A is a phonological unit. Then, given the input string $A_1 \ldots A_n$ to which we wish to apply the simultaneous rule N, we would say that the string T was a possible output if T had the form $Q_1 \ldots Q_n$, where for each i one of the following conditions held:

(i) $A_i \rightarrow Q_i / A_1 \ldots A_{i-1} - A_{i+1} \ldots A_n$ is a subrule of N; or

(ii) Q_i is identical to A_i and N has no subrule of the form $A_i \rightarrow Q'_i / A_1 \ldots A_{i-1} - A_{i+1} \ldots A_n$

However, we will also want to account for subrules of the form $P \to Q/R - S$ where P consists of several phonological units, since certain processes cannot be naturally formulated if such subrules are excluded. For example, metathesis of two adjacent vowels followed by a vowel is naturally formulated as

$$\begin{bmatrix} +\text{syl} \\ \sigma \end{bmatrix} \begin{bmatrix} +\text{syl} \\ \tau \end{bmatrix} \to \tau\sigma / \$^* - [+\text{syl}]\ \$^*$$

but not as

$$\left\{ \begin{array}{l} [+\text{syl}] \to \tau / \$^* - [+\text{syl}] \\ [+\text{syl}] \to \sigma / \$^* \begin{matrix} [+\text{syl}] \\ \tau \end{matrix} - \\ \qquad\qquad \sigma \end{array} \right\} [+\text{syl}]\ \$^*$$

On the other hand, we will make no attempt to accommodate subrules of the form $\varnothing \to Q/R-S$. It seems that in general we can avoid such subrules with little if any loss of naturalness. Thus a rule that inserts a schwa between the second and third members of a triconsonantal cluster could be written

$$\begin{bmatrix} -\text{syl} \\ \sigma \end{bmatrix} \begin{bmatrix} -\text{syl} \\ \tau \end{bmatrix} \to \sigma\text{ə}\tau / \$^* - [-\text{syl}]\ \$^*$$

just as well as

$$\varnothing \to \text{ə} / \$^* [-\text{syl}] [-\text{syl}] - [-\text{syl}]\ \$^*$$

In the next chapter we will consider the matter of insertion rules again and find a natural place for them.

In formalizing simultaneous rules we will make use of the notion of overlay, defined as follows. Let $M = P \to Q/R-S$ and $N = P' \to Q'/R'-S'$ be elementary rules. Then M overlays N if and only if

(i) $RPS = R'P'S'$,
(ii) R is not longer than R', and
(iii) S is not longer than S'.

Thus, for example, M will overlay N if

M = uşnataraa → uşnataraa/—naam,
N = n → ṇ/uş—ataraanaam

On the other hand, neither of the following two elementary rules overlays the other:

n → ṇ/uş—astaraanaam
n → ṇ/uşnataraa—aam

We can now define simultaneous rules and their mode of application as in (29).

(29) (a) A simultaneous rule is an expression of the form S:X, where X is a schema subsuming elementary rules (the subrules of S:X) and S is constant designating the mode of rule application defined in (b) below. Each subrule of S:X is assumed to have the form $P \to Q/R-S$ where $P \neq \varnothing$.

(b) Let P and Q be phonological strings and let S:X be a simultaneous rule. Then S:X maps P into Q if and only if there is a finite sequence $((P_1, Q_1), \ldots, (P_n, Q_n))$ of ordered pairs of strings such that
  (i) n is odd;
  (ii) $P = P_1 \ldots P_n$;
  (iii) for each even i, $1 < i < n$, the elementary rule
    $P_i \to Q_i / P_1 \ldots P_{i-1} - P_{i+1} \ldots P_n$
    is a subrule of S:X;
  (iv) for each odd i, $1 \leqslant i \leqslant n$, $P_i = Q_i$ and the elementary rule
    $P_i \to Q_i / P_1 \ldots P_{i-1} - P_{i+1} \ldots P_n$
    overlays no subrule of S:X;
  (v) $Q = Q_1 \ldots Q_n$.

The sequence $((P_1, Q_1), \ldots, (P_n, P_n))$ will be called an application of S:X with input P and output Q. The elementary rule

$$P_i \to Q_i / P_1 \ldots P_{i-1} - P_{i+1} \ldots P_n$$

will be referred to as the ith step of the application (thus the even steps are subrules of S:X and the odd steps are vacuous and

overlay no subrules of S:X). Frequently we will display a simultaneous application in the form

$$P_1 \begin{matrix} P_2 \\ Q_2 \end{matrix} P_3 \ldots \begin{matrix} P_{n-1} \\ Q_{n-1} \end{matrix} P_n$$

We can create a simultaneous version of the Sanskrit nasal retroflexion rule by simply substituting S for I in (26). We then have (30).

(30) $$S: \begin{bmatrix} +\text{cor} \\ +\text{nas} \\ \sigma \end{bmatrix} \rightarrow \begin{bmatrix} -\text{ant} \\ (\sigma) \end{bmatrix} \Big/ \$^* \begin{bmatrix} +\text{cont} \\ +\text{cor} \\ -\text{ant} \\ -\text{distr} \end{bmatrix} [-\text{cor}]^*-[+\text{son}]\$^*$$

This rule gives the same result as (26) but is applied in a radically different way. For example, (30) converts *uṣnataraanaam* into *uṣṇataraaṇaam* by virtue of the following application:

$$\text{uṣ} \begin{matrix} \text{n} \\ \text{ṇ} \end{matrix} \text{ataraa} \begin{matrix} \text{n} \\ \text{ṇ} \end{matrix} \text{aam}$$

This application consists of the following five steps:

1. uṣ → uṣ/—nataraanaam
2. n → ṇ/uṣ—ataraanaam
3. ataraa → ataraa/uṣn—naam
4. n → ṇ/uṣnataraa—aam
5. aam → aam/uṣnataraan—

The even-numbered steps are subrules of (30), and the odd-numbered steps are vacuous elementary rules that overlay no subrules of (30).

Consider the array

$$\text{uṣnataraa} \begin{matrix} \text{n} \\ \text{ṇ} \end{matrix} \text{aam}$$

This sequence consists of three steps:

1. uṣnataraa → uṣnataraa/—naam
2. n → ṇ/uṣnataraa—aam
3. aam → aam/uṣnataraan—

However, we have no application of rule (30) here because the putative first step overlays

n → ṇ/uṣ—ataraanaam.

which is a subrule of (30).

The vowel metathesis rule discussed earlier is a less restricted version of a rule proposed by Chomsky and Halle (1968: 361) for the Kasem language. We give the rule again in (31) in a simultaneous formulation.

(31) $$S: \begin{bmatrix} +\text{syl} \\ \sigma \end{bmatrix} \begin{bmatrix} +\text{syl} \\ \tau \end{bmatrix} \rightarrow \tau\sigma/\$^* - [+\text{syl}]$$

This rule changes *piai* into *paii* by virtue of the application

$$\text{p} \begin{matrix} \text{ia} \\ \text{ai} \end{matrix} \text{i}$$

There are three steps:

1. p → p/—iai
2. ia → ai/p—i
3. i → i/pia—

Note that under our formalism (31) is ambiguous. The hypothetical input string *piaia* would be changed into either *paiia* or *piiaa* depending on which of the following applications was taken:

$$\text{p} \begin{matrix} \text{ia} \\ \text{ai} \end{matrix} \text{ia}, \qquad \text{pi} \begin{matrix} \text{ai} \\ \text{ia} \end{matrix} \text{a}$$

In order to get an idea of the relative power of iterative and simultaneous rules we will compare them with certain mapping devices whose formal properties have been well studied. We will assume that a finite sequence of mapping devices is also a mapping device, operating according to principle (32).

(32) Let M = (M_1, ..., M_n) be a sequence of mapping devices (possibly rules), and let I and J be arbitrary strings (not necessarily phonological). Then M maps I into J if and only if there is a sequence I_1, ..., I_{n+1} of strings such that

(i) $I_1 = I$,
(ii) for each i, $1 \leq i \leq n$, M_i maps into I_i into I_{i+1}, and
(iii) $I_{n+1} = J$.

If M maps P into Q, we will sometimes say, using phonological terminology, that M provides Q as a realization of P. Note that a realization must be a phonological string; if M maps P into I alone and I is not a phonological string, then M provides P with no realization. It should be borne in mind that in general a mapping device may provide a given string with zero, one, several, or infinitely many realizations. A device will be called monogenic if it provides each string with exactly one realization. Phonological rules are typically monogenic. Alternative realizations provided to a string by a nonmonogenic rule are said to be in free variation.

One highly generalized and unstructured form of mapping device is the (unrestricted) rewriting system. Chomsky (1963: 357) has discussed devices of this sort from the point of view of string generation, but here we are interested in the way they convert strings into strings. We can assume that each rewriting system is characterized by (i) a finite alphabet of symbols, including the boundary symbol h and the start symbol s, and (ii) a finite set of instructions of the form $I \rightarrow J$, where I and J are strings of symbols in the system's alphabet. We define an immediate derivation of a rewriting system M as an expression of the form $GIH \rightarrow GJH$, where $I \rightarrow J$ is an instruction of M, and G and H are strings of symbols in the alphabet of M. A string I is automatically put in the form of hsIh when it is presented as input to a rewriting system, and the system then applies in a series of immediate derivations as long as possible. In other words, rewriting systems operate according to (33).

(33) Let M be a rewriting system and let I and J be strings consisting of symbols of M but containing no occurrences of h or s. Then M maps I into J if and only if there is a sequence $(I_1, \ldots, I_n)$ strings such that
(i) $I_1 = hsIh$;

(ii) for each i, $1 \leq i \leq n-1$, $I_i \rightarrow I_{i+1}$ is an immediate derivation of M;
(iii) M has no immediate derivation of the form $I_n \rightarrow I'$ for any I'; and
(iv) $I_n = hJh$.

It is a well-established principle that any mapping whatever that can be computed by a finitely statable, well-defined procedure can be effected by a rewriting system (in particular, by a Turing machine, which is a special kind of rewriting system). Hence any theory which allows phonological rules to simulate arbitrary rewriting systems is seriously defective, for it asserts next to nothing about the sorts of mappings these rules can perform. It is rather alarming, then, that we can prove (34).

(34) If M is a monogenic rewriting system there is a finite sequence of phonological rules, each simultaneous or iterative, which provides every phonological string with the same realization that M does.

(*In fact, the behavior of a monogenic rewriting system M can be simulated by a single iterative rule if we suitably code the symbols $D_1, \ldots, D_a$ of that system in terms of phonological units. To do this we can pick two arbitrary distinct phonological units A and B as coding elements. Assuming the symbols $D_1, \ldots, D_a$ to be pairwise distinct and to include all phonological units, we define the coding of D_i, denoted $\langle D_i \rangle$, to be the string $A^iB^{a-i}B$. If $q_1, \ldots, q_n$ are symbols of M (not necessarily distinct), we define $\langle q_1 \ldots q_n \rangle$ (the coding of the string $q_1 \ldots q_n$) to be the string $\langle q_1 \rangle \ldots \langle q_n \rangle$. Thus $\langle GH \rangle = \langle G \rangle \langle H \rangle$ for any strings G and H over the alphabet of M. Furthermore, since the correspondence between the individual symbols and their encodings is one for one, every string is unambiguously recoverable from its encoding.

Now suppose M has the instructions $I_1 \rightarrow J_1, \ldots, I_m \rightarrow J_m$. Since M is monogenic we can assume that each phonological string is the input of exactly one application of M. This follows from the way rewriting systems can be constructed to imitate Turing ma-

chines (cf. Chomsky 1963: 358-9). We can therefore assume that $I_i \neq J_i$ for each i. For if any application of M contained an immediate derivation of the form $GI_iH \rightarrow GI_iH$, this application would never terminate, and at least one input string would have no realization, contrary to the assumption that M is monogenic. Consequently if $I_i = J_i$, the instruction $I_i \rightarrow J_i$ would never be invoked in any application and would therefore be eliminable.

Now let Y be the schema $\{\langle D_1\rangle, \ldots, \langle D_a\rangle\}^*$, and let X be the schema $\{\langle I_1\rangle \rightarrow \langle J_1\rangle, \ldots, \langle I_m\rangle \rightarrow \langle J_m\rangle\}/Y - Y$. We can show that $K \rightarrow L$ is an immediate derivation of M if and only if X subsumes some nonvacuous elementary rule whose input is $\langle K\rangle$ and whose output is $\langle L\rangle$. Suppose first that $K \rightarrow L$ is an immediate derivation of M. Then there will be G, H, and i such that $K = GI_iH$, $L = GJ_iH$, and $I_i \rightarrow J_i$ is an instruction of M. But then X will subsume $\langle I_1\rangle \rightarrow \langle J_1\rangle / \langle G\rangle - \langle H\rangle$, whose input and output are $\langle K\rangle$ and $\langle L\rangle$, respectively. Furthermore, because $I_i \neq J_i$, $\langle I_i\rangle \neq \langle J_i\rangle$. Suppose next that $\langle K\rangle$ and $\langle L\rangle$ are the input and output, respectively, of some elementary rule subsumed under X. Since this elementary rule has the form $\langle I_i\rangle \rightarrow \langle J_i\rangle / \langle G\rangle - \langle H\rangle$ for some G, H, and i ($1 \leqslant i \leqslant m$), we have $\langle K\rangle = \langle GI_iH\rangle$ and $\langle L\rangle = \langle GJ_iH\rangle$. Consequently, because the coding is biunique, $K = GI_iH$ and $L = GJ_iH$. But then, since $I_i \rightarrow J_i$ is an instruction of M, $K \rightarrow L$ is an immediate derivation of M. By considering these remarks and comparing (33) with (25b) one can easily verify that M maps P into Q if and only if I:X maps $\langle$hsPh$\rangle$ into $\langle$hQh$\rangle$, where P and Q are phonological strings.

Consider again the symbols $D_1, \ldots, D_a$ of M. We can assume that for some k, $3 \leqslant k \leqslant a$, the symbols $D_k, \ldots, D_a$ are phonological units and that the remaining symbols (which must include at least h and s) are special symbols of M used for internal computation. Thus the encoding of each phonological unit will have the form $A^iB^{a-i}B$ where $i \geqslant k$.

We can now design a rule sequence which will literally map P into Q if and only if M does so. The sequence has the form (N_1, ..., N_5) where

$N_1 = S$: $\{D_k \rightarrow \langle D_k\rangle, \ldots, D_a \rightarrow \langle D_a\rangle\}/\$^* - \$^*$,
$N_2 = I$: $\{\varnothing \rightarrow \langle$hs$\rangle$ /$-A^k\$^*$, $\varnothing \rightarrow \langle$h$\rangle/\$^*A^kA^*B^* -\}$,
$N_3 = I$: $\varnothing \rightarrow \langle$hsh$\rangle$/$-$,
$N_4 = I$: X,
$N_5 = S$: $\{\langle$h$\rangle \rightarrow \varnothing, \langle D_k\rangle \rightarrow D_k, \ldots, \langle D_a\rangle \rightarrow D_a\}/\$^* - \$^*$

(N_1, N_2) will convert P into $\langle$hsPh$\rangle$ when P is nonempty, and N_3 will do the same job when P is empty. N_4 then maps $\langle$hsPh$\rangle$ into $\langle$hQh$\rangle$ if and only if M maps P into Q, as explained previously. N_5 then converts $\langle$hQh$\rangle$ into Q.*)

It would seem, then, that a rule formalism which permits both simultaneous and iterative rules is excessively powerful. Notice that we reduce this power very little if we exclude simultaneous rules and permit only iterative ones. (*For let M, $N_1, \ldots, N_5$ be as above. Obtain N'_1 from N_1 by replacing S with I and deleting $D_k \rightarrow \langle D_k\rangle$ and $D_{k+1} \rightarrow \langle D_{k+1}\rangle$ from the braced expression. Obtain N'_5 from N_5 by simply replacing S with I. Then (N'_1, N_2, N_3, N_4, N'_5) will provide each phonological string that is free of occurrences of D_{k+1} and D_k with the same realization that M does. Strings containing occurrences of D_k and D_{k+1} will not in general be handled correctly, of course; D_k and D_{k+1} have been sacrified to the cause of coding.*)

Suppose we try the polar alternative of excluding iterative rules altogether and allowing simultaneous rules only. We will try to show that simultaneous rules are equivalent to finite-state machines, which are highly restricted in structure and are incapable, in fact, of performing a vast array of computable mappings, though the mappings they do perform seem to include most of those that arise in phonology.

In its most general form a finite-state machine is characterized by

(i) a finite alphabet of input symbols,
(ii) a finite alphabet of output symbols,
(iii) a finite set of symbols referred to as states,
(iv) a subset of states designated as left terminal,
(v) a subset of states designated as right terminal, and
(vi) a finite set of instructions of the form q K → L q′,

where q and q′ are states, K is an input symbol, and L is a string of output symbols.

By a computation of a finite-state machine M we will mean an expression of the form $q_1K_1 \rightarrow L_1 \ldots q_nK_n \rightarrow L_nq_{n+1}$ where for each i, $1 \leq i \leq n$, $q_iK_i \rightarrow L_iq_{i+1}$ is an instruction of M. $K_1 \ldots K_n$ will be referred to as the input of the computation, and $L_1 \ldots L_n$ will be referred to as the output. If q_1 is a left terminal state of M and q_{n+1} is a right terminal state of M, the computation is said to be terminated. By definition M maps I into J if and only if some terminated computation of M has I as input and J as output.

A simple example of a finite-state machine is the one characterized as follows:

Input alphabet: A, B
Output alphabet: A, B, C
States: 0, 1, 2
Left terminal states: 0
Right terminal states: 0, 1
Instructions: 0A → A1
0A → C2
0B → B0
1A → C0
1B → B1
2A → A1

One of the computations of this machine is

$$0B \rightarrow B0A \rightarrow A1B \rightarrow B1A \rightarrow C0B \rightarrow B0A \rightarrow C2A \rightarrow A1$$

The input of the computation is BABABAA, and the output is BABCBCA. Since the first symbol of the computation is a left-terminal state of the machine and the last symbol is a right terminal state, the computation is terminated. Hence we may conclude that the machine maps BABABAA into BABCBCA.

We will say that a finite-state machine is right-deterministic if it has exactly one left terminal state and, for each state q and each input symbol K, exactly one instruction of the form $qK \rightarrow Lq'$.

If a finite-state machine has exactly one right terminal state and, for each state q and input symbol K, exactly one instruction of the form $q'K \rightarrow Lq$, we will say that the machine is left-deterministic. We denote by lt(M) the unique left terminal state of a right-deterministic machine M, and use rt(M) for the unique right terminal state of a left-deterministic machine M.

We will be especially concerned with two varieties of finite state machines called right transducers and left transducers. A right transducer is a right-deterministic finite-state machine all of whose states are right terminal. Clearly a device M of this sort will convert I into J if and only if J is the output of that unique computation which has I as input and had lt(M) as its leftmost state. A left transducer is defined in completely symmetric manner as a left-deterministic finite state machine all of whose states are left terminal. A right transducer is also known as a generalized sequential machine (Ginsburg 1962: 5, 20), the unique left terminal state being also known as the start state or the initial state.

Although we have been regarding mapping devices only as mechanisms for converting strings into strings, this being the function of phonological rules, we can also look upon them as devices for defining or representing sets of strings. Suppose we say that a phonological string P is accepted by a mapping device M if and only if M provides P with at least one realization. Those phonological strings that are accepted by the device constitute the defined set. It is well known that a set is regular if and only if it is defined by some finite-state machine. Hence phonological schemata and finite-state machines are entirely equivalent in their capacity to represent sets.

If we wish to construct a finite-state machine solely for the purpose of accepting strings there is no point in providing it with output symbols. We might just as well let each instruction be of the form $qK \rightarrow q'$. Then the machine accepts P if and only if it maps P into $\varnothing$. A finite-state machine of this sort is called a finite-state automaton. If such a machine is right- (left-) deterministic it is said to be a right- (left-) automaton.

What we wish to show now is (35).

(35) If N is a monogenic simultaneous rule then there is a left transducer M_1 and a right transducer M_2 such that (M_1, M_2) provides each phonological string with the same realization that N does.

(*We let $N = S{:}X$ be an arbitrary monogenic simultaneous rule, fixed throughout the proof. If N has no subrules it leaves input strings unchanged, and there is a trivial transducer that can accomplish this identity mapping. Henceforth we assume N to have at least one subrule.

Without loss of generality we can assume that $X = \{X_1, \ldots, X_m\}$ is a primitive schema. Each of the X_i will then have the form $Y_i \rightarrow Y'_i/U_i - V_i$, where Y_i, Y'_i, U_i, and V_i subsume only phonological strings. To see this consider each of the X_i in turn, letting $(Z_1, E_1, \ldots, Z_p, E_p, Z_{p+1})$ be the longest sequence of schemata such that $X_i = Z_1E_1 \ldots Z_pE_pZ_{p+1}$ and such that each E_h $(h = 1, \ldots, p)$ is an arrow, a slash, or a dash. Since this sequence is maximally long none of the Z_j can have the form $Z'E'Z''$ where E' is arrow, slash, or dash and Z' and Z'' are schemata. Hence if arrow, slash, or dash appears anywhere within a Z_j it must be inside a braced or starred expression (these being the only nonelementary schematic expressions allowed in primitive schemata). Now, any braced expression that is a proper part of a primitive schema must be enclosed in starred parentheses. Consequently, an arrow, slash, or dash appearing within any of the Z_j must be inside a starred expression. However, if an arrow, slash or dash appeared inside a starred expression it could be repeated an unbounded number of times in subsumed strings, and X would subsume some expressions that were not elementary rules.

Because simultaneous rules have no subrules of the form $\varnothing \rightarrow Q/R - T$, we can assume that the Y_i subsume only nonempty strings. We can also assume that each Y'_i is a phonological string, since if $S{:}X$ is indeed monogenic none of the Y'_i need subsume more than one phonological string. We will set $S_i = Y'_i$ and $W_i = Y_iV_i$.

The first step in creating a pair of transducers to simulate N is

to construct, for each $k = 1, \ldots, m$, two left automata $\overline{W}_k$ and $\overline{\nabla}_k$ that represent the same regular sets as W_k and V_k, respectively, and two right automata $\overline{U}_k$ and $\overline{Y}_k$ that represent the same regular sets as U_k and Y_k, respectively. In addition we define $\overline{U}_0 = \overline{Y}_0 = \overline{W}_0 = \overline{\nabla}_0$ to be the automaton having just the one state 0 and having the instruction $0A \rightarrow 0$ for each phonological unit A. 0 is taken to be both left terminal and right terminal. This device is at once a right automaton and a left automaton, and it accepts every phonological string. Suppose now we define $((z_0, \ldots, z_m))_k = z_k$ for any $(m+1)$ -tuple $(z_0, \ldots, z_m)$ and any $k = 0, \ldots, m$. An $(m+1)$ -tuple x will be said to be a W-state (resp.: V-state, U-state) if and only if $(x)_k$ is a state of $\overline{W}_k$ (resp.: $\overline{\nabla}_k$, $\overline{U}_k$) for each $k = 0, \ldots, m$. Henceforth we use the letters w, v, and u to stand for W-states, V-states, and U-states respectively. The letter y will range over states of all the $\overline{Y}_i$. Also we define

$$\overline{\mathfrak{W}} = (\mathrm{rt}(\overline{W}_0), \ldots, \mathrm{rt}(\overline{W}_m))$$
$$\mathfrak{v} = (\mathrm{rt}(\overline{\nabla}_0), \ldots, \mathrm{rt}(\overline{\nabla}_m))$$
$$\mathfrak{U} = (\mathrm{lt}(\overline{U}_0), \ldots, \mathrm{lt}(\overline{U}_m))$$

We now construct the first transducer M_1. The input alphabet of M_1 is the phonological alphabet and the output alphabet consists of triples of the form (w, v, A). The states of M_1 are couples of the form (w, v). Each state of M_1 is left terminal, and M_1 has the unique right terminal state $(\overline{\mathfrak{w}}, \mathfrak{v})$. For each input symbol A and each state (w, v), M_1 has the instruction $(w', v')A \rightarrow (w', v', A)$ (w, v) where for each $k = 0, \ldots, m$, $(w')_kA \rightarrow (w)_k$ and $(v')_kA \rightarrow (v)_k$ are instructions of $\overline{W}_k$ and $\overline{\nabla}_k$, respectively. M_1 is a left transducer which, when presented with an input string $A_1 \ldots A_n$, responds with the output string $(w_1, v_1, A_1) \ldots (w_n, v_n, A_n)$ where the conditions of (35) hold.

(35) (a) for each k, $0 \leq k \leq m$, $(w_n)_kA_n \rightarrow \mathrm{rt}(\overline{W}_k)$ and $(v_n)_kA_n \rightarrow \mathrm{rt}(\overline{\nabla}_k)$ are instructions of $\overline{W}_k$ and $\overline{\nabla}_k$, respectively.

(b) for each k, $0 \leq k \leq m$, and each i, $1 \leq i \leq n-1$, $(w_i)_kA_i \rightarrow (w_{i+1})_k$ and $(v_i)_kA_i \rightarrow (v_{i+1})_k$ are instructions of $\overline{W}_k$ and $\overline{\nabla}_k$, respectively.

The second transducer M_2 is constructed as follows. The input alphabet of M_2 is identical to the output alphabet of M_1, and the output alphabet of M_2 is again the phonological alphabet. The states of M_2 are triples of the form (u, y, k) where $0 \leq k \leq m$, y is a state of $\bar{Y}_k$, and u is a U-state. M_2 has the unique left terminal state $(\bar{u}, 0, 0)$, and all of the states of M_2 are right terminal. The instructions of M_2 are given by (36).

(36) For each input symbol (w, v, A) and each state (u, y, k), M_2 has the instruction (u, y, k) (w, v, A) $\rightarrow$ Q(u′, y′, k′) where

(a) for each k, $0 \leq k \leq m$, $(u)_k A \rightarrow (u')_k$ is an instruction of $\bar{U}_k$;

(b_1) if y is a right terminal state of $\bar{Y}_k$ and $(v)_k$ is a left terminal state of $\bar{V}_k$, then

(i) k′ is the highest integer such that $(u)_{k'}$ is a right terminal state of $\bar{U}_{k'}$ and $(w)_{k'}$ is a left terminal state of $\bar{W}_{k'}$ (such an integer will always exist because 0, at least, is such an integer),

(ii) Q = A or $S_{k'}$ according as k′ is equal to or greater than zero, and

(iii) $1t(\bar{Y}_{k'})A \rightarrow y'$ is an instruction of $\bar{Y}_{k'}$;

(b_2) if y is not a right terminal state of $\bar{Y}_k$ or $(v)_k$ is not a left terminal state of $\bar{V}_k$, then

(i) $k' = k$,

(ii) $Q = \varnothing$, and

(iii) $yA \rightarrow y'$ is an instruction of $\bar{Y}_k = \bar{Y}_{k'}$.

Each string $(w_1, v_1, A_1) \ldots (w_n, v_n, A_n)$ produced as output by M_1 will be the input of exactly one terminated computation of the right transducer M_2, and this computation will have the general form $K_0 \ldots K_n$ where

$K_0 = (u_0, y_0, k_0)$ and

$K_i = (w_i, v_i, A_i) \rightarrow T_i(u_i, y_i, k_i)$ for $i = 1, \ldots, n$.

(M_1, M_2), then, maps $A_1 \ldots A_n$ into $T_1 \ldots T_n$. We need to show that the simultaneous rule N also maps $A_1 \ldots A_n$ into $T_1 \ldots T_n$.

Since (u_0, y_0, k_0) is the left-terminal state of M_2 we have (37).

(37) $u_0 = \bar{u}$, $y_0 = 0$, and $k_0 = 0$.

For each $i = 1, \ldots, n$ and each $r = 0, \ldots, m$, the expression $(u_{i-1})_r A_i \rightarrow (u_i)_r$ is an instruction of $\bar{U}_r$. This follows by virtue of (36a). Because of (37) $(u_0)_r = 1t(\bar{U}_r)$. Also $\bar{U}_r$ is right-deterministic. Hence

$$(u_0)_r A_1 \rightarrow (u_1)_r \ldots A_i \rightarrow (u_i)_r$$

is that unique computation of $\bar{U}_r$ that begins with $1t(\bar{U}_r)$ and has $A_1 \ldots A_i$ as input. Therefore $A_1 \ldots A_i$ is accepted by $\bar{U}_r$ if and only if $(u_i)_r$ is a right-terminal state of $\bar{U}_r$. Reasoning in a similar way from (35) we can show that $A_1 \ldots A_n$ is accepted by $\bar{W}_r$ (resp.: $\bar{V}_r$) if and only if $(w_i)_r$ (resp.: $(v_i)_r$) is a left-terminal state of $\bar{W}_r$ (resp.: $\bar{V}_r$). Hence we have (38).

(38) If $1 \leq i \leq n$ and $1 \leq r \leq m$, then

(a) $A_1 \ldots A_i$ is subsumed under U_r if and only if $(u_i)_r$ is a right-terminal state of $\bar{U}_r$; and

(b) $A_1 \ldots A_n$ is subsumed under W_r (resp.: V_r) if and only if $(w_i)_r$ (resp.: $(v_i)_r$) is a left-terminal state of $\bar{W}_r$ (resp.: $\bar{V}_r$).

π For each $i = 1, \ldots, n$ we will say that i is a type 1 or type 2 index according as the instruction

$$(u_{i-1}, y_{i-1}, k_{i-1})\ (w_i, v_i, A_i) \rightarrow T_i(u_i, y_i, k_i)$$

satisfies condition ($36b_1$) or ($36b_2$). Note that these conditions are mutually exclusive and exhaust the possibilities, so that each i from 1 through n must be either a type 1 index or a type 2 index, but cannot be both.

Now suppose that i is a type 1 index such that $k_i = 0$. By ($36b_1$.ii) $T_i = A_i$. Furthermore, if $i < n$, then $i + 1$ is a type 1 index; this follows from the fact that $y_i = 0 = (v_{i+1})_0 = (v_{i+1})_{k_i}$ is both a right-terminal state and a left-terminal state of $\bar{Y}_{k_i} = \bar{Y}_0 = \bar{V}_0 = \bar{V}_{k_i}$ (cf. $36b_1$). By ($36b_1$.i) there is no $r > k_i$ such that $(u_{i-1})_{k_i}$ is a right-terminal state of $\bar{U}_r$ and $(w_i)_{k_i}$ is a

left-terminal state of $\bar{W}_r$. Hence by (38) there is no $r > k_1$ such that $A_1 \ldots A_{l-1}$ is subsumed under U_r and $A_l \ldots A_n$ is subsumed under W_r. Consequently, since $W_r = Y_r V_r$ for each $r = 1, \ldots, m$, there is no j such that $A_l \ldots A_j$ is subsumed under Y_r and $A_{j+1} \ldots A_n$ is subsumed under V_r (taking $A_{j+1} \ldots A_n = \varnothing$ if $j = n$). Therefore there is no j such that

$$A_l \ldots A_j \rightarrow T_1 \ldots T_j / A_1 \ldots A_{l-1} - A_{j+1} \ldots A_n$$

is a subrule of N.

We can immediately extend the results of the preceding paragraph to (39).

(39) Let i and j be indices such that $i \leq j$ and such that each i', $i \leq i' \leq j$, is a type 1 index for which $k_{i'} = 0$. Then
(a) $T_1 \ldots T_j = A_1 \ldots A_j$;
(b) if $j < n$, $j + 1$ is a type 1 index;
(c) there are no i', i'' such that $i \leq i' \leq i'' \leq j$ and such that the elementary rule
$A_{i'} \ldots A_{i''} \rightarrow T_{i'} \ldots T_{i''} / A_1 \ldots A_{i'-1} - A_{i''+1} \ldots A_n$
is a subrule of N. Consequently the elementary rule
$A_l \ldots A_j \rightarrow T_1 \ldots T_j / A_1 \ldots A_{l-1} - A_{j+1} \ldots A_n$
does not overlay any subrule of N.

Now let i be a type 1 index for which $k_i > 0$. Because of (38) $A_1 \ldots A_{i-1}$ is subsumed under U_{k_i} and $A_i \ldots A_n$ is subsumed under W_{k_i}. Also, $T_i = S_{k_i}$. Now let j be the highest integer not greater than n such that each h, $i < h \leq j$, is a type 2 index. (If $j = i$, then of course there is no such h; but if $j > i$, then j at least will be such an h.) For each such h we have $T_h = \varnothing$ by ($36b_2$.ii), and therefore $T_i = S_{k_i} = T_i \ldots T_j$. Now because of ($36b_2$), we have (i) and (ii):

(i) $k_h = k_{h'}$ for $i \leq h \leq h' \leq j$, and
(ii) $y_{h-1} A_h \rightarrow y_h$ is an instruction of $\bar{Y}_{k_i}$ for each h, $i < h \leq j$.

Also, because of ($36b_1$.iii), $1t(\bar{Y}_{k_i}) A_i \rightarrow y_i$ is an instruction of

$\bar{Y}_{k_i}$. Recall, too, that $\bar{Y}_{k_i}$ is right-deterministic. Consequently, for $i \leq h \leq j$,

$$1t(\bar{Y}_{k_i}) A_i \rightarrow y_i \ldots A_h \rightarrow y_h$$

is that unique computation of $\bar{Y}_{k_i}$ which has $A_i \ldots A_h$ as input and $1t(\bar{Y}_{k_i})$ as its leftmost state. Therefore $A_i \ldots A_h$ is accepted by $\bar{Y}_{k_i}$ if and only if y_h is a right-terminal state of $\bar{Y}_{k_i}$. Therefore $A_i \ldots A_h$ is subsumed under Y_{k_i} if and only if y_h is a right-terminal state of $\bar{Y}_{k_i}$. Now consider the two cases $j < n$ and $j = n$.

Case 1: $j < n$. $j+1$ must then be type 1 index. But then because of ($36b_1$) y_j is a right-terminal state of $\bar{Y}_{k_i}$ ($= \bar{Y}_{kj}$) and $(v_{j+1})_{k_j}$ is a left-terminal state of $\bar{V}_{k_j}$ ($= \bar{V}_{k_i}$). From this it follows that $A_i \ldots A_j$ is subsumed under Y_{k_i} and $A_{j+1} \ldots A_n$ is subsumed under V_{k_i}.

Case 2: $j = n$. Because $A_i \ldots A_n$ is subsumed under W_{k_i} and $W_{k_i} = Y_{k_i} V_{k_i}$ one of the following conditions must hold:
(i) $\varnothing$ is subsumed under Y_{k_i} and $A_i \ldots A_n$ is subsumed under V_{k_i};
(ii) for some h, $i \leq h < j$, $A_i \ldots A_h$ is subsumed under Y_{k_i} and $A_{h+1} \ldots A_n$ is subsumed under V_{k_i}; or
(iii) $A_i \ldots A_n$ is subsumed under Y_{k_i} and $\varnothing$ is subsumed under V_{k_i}.
(i) is not satisfied because Y_{k_i} subsumes only nonempty strings (recall that rule N has no insertion subrules). (ii) would imply that y_h is a right-terminal state of $\bar{Y}_{k_i}$ and $(v_{h+1})_{k_i}$ is a left-terminal state of $\bar{V}_{k_i}$; therefore, since $k_h = k_j$, h+1 would be a type 1 index (cf. $36b_1$). This would contradict the assumption that each integer greater than i and equal to or less than j is a type 2 index. The only remaining possibility is (iii).

What we have managed to show is (40).

(40) Let i be a type 1 index for which $k_i > 0$ and let j be the highest integer such that each h, $i < h \leq j$, is a type 2 index. Then

$A_i...A_j \rightarrow T_i...T_j/A_1...A_{i-1} - A_{j+1}...A_n$
is subsumed under $Y_{k_i} \rightarrow S_{k_i}/U_{k_i} - V_{k_i}$ and is therefore a subrule of N.

Now let $s_1, ..., s_p$ be all and only the type 1 indices for which $k_{s_i} > 0$. $(1 \leq i \leq p)$. We can assume $s_i < s_{i'}$ if $i < i'$. For each i, $1 \leq i \leq p$, let t_i be the highest index such that each h, $s_i < h \leq t_i$, is a type 2 index. Also, define $t_0 = 0$ and $s_{p+1} = n+1$. Clearly, $t_i < s_{i+1}$. Now for $i = 0, ..., p$ we define

$P_{2i+1} = A_{t_i}...A_{s_{i+1}-1}$;
$Q_{2i+1} = T_{t_i}...A_{s_{i+1}-1}$;
$P_{2i+2} = A_{s_i}...A_{t_i}$; and
$Q_{2i+2} = T_{s_i}...T_{t_i}$.

From these definitions it follows that $A_1...A_n = P_1...P_{2p+1}$ and $T_1...T_n = Q_1...Q_{2p+1}$. Now for each j, $1 \leq j \leq 2p+1$, let M_j be the elementary rule

$$P_j \rightarrow Q_j/P_1...P_{j-1} - P_{j+1}...P_{2p+1}$$

If j is even, then $j = 2i+2$ for some i, $0 \leq i \leq p$. Then M_j is the elementary rule

$$A_{s_i}...A_{t_i} \rightarrow T_{s_i}...T_{t_i}/A_1...A_{s_i-1} - A_{t_i+1}...A_n$$

which, by virtue of (40), is a subrule of N. If j is odd then $j = 2i+1$ for some i, $0 \leq i \leq p$, and then M_j is the elementary rule

$$A_{t_i+1}...A_{s_{i+1}-1} \rightarrow T_{t_i+1}...T_{i+1}/A_1...A_{t_i} - A_{s_{i+1}}...A_n$$

There are now two cases to consider, $t_i = s_{i+1}$ and $t_i < s_{i+1}-1$

Case 1: $t_i = s_{i+1}-1$. Then $P_j = Q_j = \varnothing$ and M_j overlays no subrule of N. The reason for this is that M_j is an insertion rule, and an insertion can overlay only another insertion rule. However, N has no insertion subrules.

Case 2: $t_i < s_{i+1}-1$. We consider first the type of index that $t_i + 1$ is.

Subcase 1. $i = 0$. Then $t_i + 1 = 1$, which is a type 1 index because $k_0 = 0$, $y_0 = 0$ is a right-terminal state of $\bar{Y}_0$, and $(v_1)_0 = 0$ is a left-terminal state of $\bar{V}_0$.

Subcase 2. $i > 0$. Then $t_i + 1$ is less than $n = s_{p+1}-1$ and is a type 1 index. For if $t_i + 1$ were a type 2 index t_i would not be the highest index such that each h, $s_i < h \leq t_i$, is a type 2 index, contrary to the definition of t_i.

Now although t_i+1 is a type 1 index it is not among the $s_1, ..., s_p$, because it is larger than s_i and less than s_{i+1}. Therefore $k_{t_i+1} = 0$. Hence, because of (39c), M_j overlays no subrule of N. Furthermore, (39a) implies that $P_j = Q_j$.

We have now shown that when j is even M_j is a subrule of N and that when j is odd $P_j = Q_j$ and M_j overlays no subrule of N. Consequently the sequence $((P_1, Q_1), ..., (P_{2p+1}, Q_{2p+1}))$, whose jth step is M_j, is an application of N. But this application has $P_1...P_{2p+1} = A_1...A_n$ as input and $Q_1...Q_{2p+1} = T_1...T_n$ as output. Thus N maps $A_1...A_n$ into $T_1...T_n$. Q.E.D.*)

A left (right) transducer takes into account only what is to the right (left) of an input symbol in determining how to replace that symbol in the output. Schützenberger (1961) has investigated a finite-state device which takes into account both the left and right contexts of each input symbol. This machine, referred to simply as a finite transducer, is characterized by the following structures:

(i) Two finite alphabets $\bar{I}$ and $\bar{O}$ (the input symbols and the output symbols).
(ii) A right automaton $\bar{U}$ whose input alphabet is $\bar{I}$ and whose states are all right terminal.
(iii) A left automaton $\bar{V}$ whose input alphabet is $\bar{I}$ and whose states are all left terminal.
(iv) A finite set of output instructions of the form $uKv \rightarrow L$, where u is a state of $\bar{U}$, K is an input symbol, v is a state of $\bar{V}$, and L is a string of output symbols. There is exactly one instruction $uKv \rightarrow L$ for each uKv.

The way a finite transducer works is this. Suppose $K_1, \ldots, K_n$ are input symbols. For each i ($i = 1, \ldots, n$) let $\left\{\begin{matrix} u_i \\ v_i \end{matrix}\right\}$ be the $\left\{\begin{matrix} \text{rightmost} \\ \text{leftmost} \end{matrix}\right\}$ state in that terminated computation of $\left\{\begin{matrix} \bar{0} \\ \bar{\nabla} \end{matrix}\right\}$ which has $\left\{\begin{matrix} K_1 \ldots K_{i-1} \\ K_{i+1} \ldots K_n \end{matrix}\right\}$ as its input, and suppose $u_i K_i v_i \rightarrow L_i$ is the output instruction for $u_i K_i v_i$. Then the transducer converts $K_1 \ldots K_n$ into $L_1 \ldots L_n$.

Schützenberger has pointed out that an ordered pair of machines, one of which is a left transducer and the other of which is a right transducer, is equivalent to some finite transducer in the mapping its effects. Consequently every monogenic simultaneous rule is similarly equivalent to some finite transducer. Thus a theory which permits only simultaneous rules in a phonological description embodies a very strong hypothesis concerning the sorts of string mappings that are possible in phonology, for finite transducers rank next to the bottom in the following hierarchy.

1. Rewriting machines (in particular, Turing machines).
2. Linear bounded transducers.
3. Pushdown-store transducers.
4. Finite transducers.
5. Right transducers; left transducers.

It is well known that every mapping effected by a device of type n in this list can be effected by a device of type n−1 ($n = 2, \ldots, 5$), but not conversely. (Right transducers and left transducers are not comparable with each other in this way because some right transducers can do things that left transducers can't and vice-versa.) There seem, in fact, to be few phonological processes that exceed the capacity of finite transducers; the ones known to me belong to the very restricted types to be discussed in Chapter 7. In at least one respect, then, simultaneous rules are far more appropriate to phonology than iterative ones.

If the finite-state claim is correct, there is a clear choice between two recent proposals concerning the way such processes as contraction, metathesis, gemination, and degemination should be handled. Postal (1969: 298) has suggested the use of variables ranging over lists of feature specifications. He had in mind a vowel-doubling rule, but his considerations extend to the other types of processes just mentioned. Thus according to Postal's proposal the rule mentioned earlier that metathesized two vowels before another vowel could be written:

$$\begin{bmatrix} +\text{syl} \\ A \end{bmatrix} \begin{bmatrix} +\text{syl} \\ B \end{bmatrix} \rightarrow \begin{bmatrix} +\text{syl} \\ B \end{bmatrix} \begin{bmatrix} +\text{syl} \\ A \end{bmatrix} / - [+\text{syl}]$$

Under our view of brackets as representing set intersection, Postal's variable can in most cases be regarded as a unit variable. It is possible, therefore, to simplify slightly the above rule to

$$\begin{bmatrix} +\text{syl} \\ A \end{bmatrix} \begin{bmatrix} +\text{syl} \\ B \end{bmatrix} \rightarrow B\ A\ / - [+\text{syl}]$$

This, of course, is just a notational variant of our rule (31).

Another way of writing the kind of rule under discussion is with transformational format:

$$\underset{1}{X},\ \underset{2}{[+\text{syl}]},\ \underset{3}{[+\text{syl}]},\ \underset{4}{[+\text{syl}]},\ \underset{5}{Y} \Rightarrow 1, 3, 2, 4, 5$$

Langacker (1969: 858-9) has correctly observed that all phonological rules can be written in this format, and has proposed that this be done. Notice, however, that in this format integers are used as general string variables. Consequently many nonfinite-state operations can be performed, such as the reduplication of whole words. Such reduplications occur, to be sure (for example, in Indonesian). However, reduplication seems to be a morphological process spelling out grammatical elements or features (e.g. plurality or distribution) and seems to have no phonological origin or motivation. It belongs rather with what Chomsky and Halle (1968: 9-11) have referred to as readjustment rules. I would claim that phonological rules proper never require general string variables for their expression and would therefore reject Langacker's proposal as insufficiently restrictive.