Modelling Autosegmental Phonology with Multi-Tape Finite State Transducers

by

Bruce Wiebe

B.Sc. Simon Fraser University 1988

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

Approval

Name:	Bruce Wiebe
Degree:	Master of Science
Title of Thesis:	Modelling Autosegmental Phonology with Multi-Tape Finite State Transducers

Examining Committee:

Chair: Dr. Binay Bhattacharya

Dr. Fred Popowich
Senior Supervisor

Dr. Paul McFetridge

Dr. Veronica Dahl

Dr. Tom Perry
External Examiner

December 17, 1992
Date Approved

Title of Thesis/Project/Extended Essay

Modelling Autosegmental Phonology with Multi-Tape Finite State Transducers

Author: ______________________

(signature)

Bruce M. Wiebe

(name)

December 17, 1992

(date)

Abstract

Phonology may be briefly defined as the study of sound patterns in spoken language. One of the most well-known computational models of phonology, Koskenniemi's two-level phonology, is based on an underlying linguistic theory that has been superseded by autosegmental phonology, which began with the work of Goldsmith. There is a need for computational models that are faithful to this more recent theory. Such a model can form the basis of a computational tool that can quickly and accurately check the validity of a phonological analysis on a large amount of phonetic data, freeing the linguist from the tedious and error-prone task of doing this by hand.

This thesis presents a new computational model of phonology that is faithful to standard autosegmental theory, that has clearly adequate expressive power, and that is suitable as the basis for a tool for phonological analysis. It follows on very recent efforts by Kornai and Bird & Ellison to model autosegmental phonology. The model is based on a view of phonology that sees phonological *representations* as data and phonological *rules* as procedures that manipulate them. It models rules using multi-tape state-labelled finite transducers (MSFTs), a natural extension of finite state transducers obtained by adding multiple input and output tapes. MSFTs are shown to be powerful enough to express a wide range of autosegmental rules. We also investigate the class of formal languages accepted by multi-tape state-labelled finite automata (MSFAs) when their input tapes are considered to encode a single word in parallel. This class is quite large, including some languages that are not context free. Given that our model is faithful to autosegmental theory, this gives an upper bound on the computational power required to model autosegmental phonological rules.

Acknowledgements

It is my privilege and joy to express my appreciation to many people whose help has made this thesis possible.

First, to the Lord, who gave me the ability and opportunity to do a masters degree, convinced me that I should take it, gave me a vision for using this research to benefit the work of Bible translation, and sent many people and circumstances my way to encourage me and assure me that I was following his will.

To Keith Snider, for introducing me to autosegmental phonology during CSIL '91, and for his contagious excitement which sparked in me the desire to focus my own research in on this area of linguistics.

To Fred Popowich, for his encouragement, enthusiasm, availability, and guidance; for helping me to establish contacts in the academic world; and for giving me freedom to pursue my research interests.

To Steven Bird, for being a friend and long distance supervisor; for his constant encouragement along the way; for taking the time to read through several versions of my work and give me guidance.

To Andras Kornai, whose own research inspired mine, for stimulating discussions that helped keep my thinking clear.

To Jo Calder, for being an unoffical committee member and local computational phonology resource person.

To Mark Ellison for his input and encouragement, especially at the DIMACS workshop.

To Veronica Dahl, Paul McFetridge, and Tom Shermer, who asked questions and offered ideas that had a significant impact on this research.

To Evan Antworth, John Goldsmith, Doug Pulleyblank, and Gary Simons, for their interest and encouragement.

To Binay Bhattacharya, for encouraging me to apply for an NSERC postgraduate scholarship, and to NSERC for their support of this research through two postgraduate scholarships and through Fred Popowich's NSERC Operating Grant 0GP0041910.

To the Natural Language Lab and the School of Computing Science at Simon Fraser University for the use of their facilities.

And to my parents, for their support and encouragement throughout my education.

Table of Contents

List of Figures

Chapter 1
Introduction

1.1. Outline

Phonology may be briefly defined as the study of sound patterns in language. The phonologist attempts to determine the way language utterances are represented in the mind of the speaker. He also analyzes the speech sounds that the speaker actually produces, which are usually different from what is stored mentally. Both the mental representation and the spoken word are described using *phonological representations*. Then the relationship between the two is explained by means of *phonological rules*.

There are competing theories about what phonological representations should look like, and how phonological rules should be described. However, there is widespread agreement among phonologists today that the non-linear representations and rules of the *autosegmental* theory of phonology are superior to the linear representations and rules of the classical theory of generative phonology from which autosegmental theory developed.

The goal of this thesis is to construct a new computational model of autosegmental phonology that will form an adequate basis for a computational tool for phonologists. Such a tool would be invaluable both to theoretical linguists who desire to test their theories, and to field linguists who want to analyze data. It would provide a way to efficiently and accurately test the validity of a phonological analysis on a large amount of data. This task is typically neglected by theoretical linguists; theories are often based on a small amount of data because of the difficulty of doing large analyses. By contrast, analyzing large amounts of data is a fact of life for field linguists, but it is tedious and is subject to human error. Thus, a computational tool for phonological analysis would benefit both theoretical and working phonologists.

Although there have been other recent efforts along this line, there are some areas in which a new model can make a unique contribution. In particular, the question of how much formal computational power is needed to model non-linear phonology is a thread that runs throughout this work. This model is a response to the need for a procedural implementation of autosegmental phonology with clearly adequate expressive power.

The method is to take as the basis of the model a computational device we call multi-tape state-labelled finite automata (MSFAs), and in particular, multi-tape state-labelled finite transducers (MSFTs), which are a special case of MSFAs. Using suitably defined operations, simple MSFTs that describe various aspects of autosegmental rules can be combined to form more complex MSFTs that simulate the action of any autosegmental rule. The architecture of MSFTs also gives rise to a simple non-linear encoding of autosegmental representations.

The scope of this thesis includes the development and theoretical evaluation of the model, but does not include an implementation of the model. The evaluation includes a comparison with other recent work in this area. The model is evaluated on the basis of three general criteria: expressive adequacy, faithfulness to the standard theory of autosegmental phonology, and suitability as the basis of a computational tool for verifying phonological analyses.

In Chapter 2 we give a brief introduction to autosegmental phonology for computer scientists, and to automata theory for linguists. Chapter 3 defines the automata that will form the basis of our model: MSFAs and MSFTs. Chapter 4 defines the non-linear encoding of autosegmental representations that allows MSFTs to process phonological forms, and considers its advantages from a linguistic point of view. Chapter 5 defines various operations that can be used to combine MSFAs and MSFTs into new ones. The operations of *product* and *disjoint product* play a central role in our model.

Having defined all the formal computational tools we need, we then show how to model an autosegmental rule with an MSFT in Chapter 6. Chapter 7 gives a brief sketch of the formal computational power of MSFAs and MSFTs. Finally, Chapter 8 concludes with an evalutation of our model, especially in comparison with the work of Kay [29], Koskenniemi [33], Kornai [32], and Bird & Ellison [8], suggesting some directions for further research along the way.

1.2. Background

The use of finite state transducers in modelling phonological rules has its roots early in the history of phonology. Early generative phonology took a procedural view of rules as processes that manipulated representations as data. Chomsky and Halle, in their landmark work *The Sound Pattern of English* [10], used transformational grammars to describe these processes. Given the correspondence between formal grammars and automata (see Section 2.3.), it was natural to investigate the use of automata in implementing phonological rules.

In 1972, Johnson [25] showed that the phonological relations described by generative rules were regular, in the sense that they can be described by regular grammars. Since then there have been many efforts to apply finite state transducers (FSTs) to phonology [3, 8, 18, 29, 32, 33], since FSTs compute regular relations. This is true in spite of the fact that the representations and rules of generative phonology have changed, and hence, presumably, the relations have changed as well.

In this work, we have chosen to model autosegmental phonology, a contemporary theory of generative phonology that has been applied to a wide range of languages. For reasons that we will discuss in the last chapter, we have taken a procedural approach to this problem. Kornai [32] has recently proposed a procedural model of autosegmental phonology based on rules as FSTs. However, for reasons that we will discuss in detail in Chapter 8, we believe that FSTs are not powerful enough to model autosegmental rules. There are certain phonological operations that cannot be captured with just finite state power. Therefore, we have chosen to use a more powerful class of automata.

In the procedural view, autosegmental rules are processes that transform autosegmental representations. Thus, we use transducers working on input and output tape(s) to simulate this process. In view of the multi-tiered nature of autosegmental representations, it seems natural to use transducers with multiple input and output tapes to model autosegmental rules. Each tier of a representation could be encoded and written on one input tape of a transducer. (Some provision for representing the association lines *between* tiers would also be needed.) The transducer would then process this input and produce an output representation on its multiple output tapes.

We will in fact encode autosegmental representations in this fashion, and model autosegmental rules as MSFTs, a natural extension of FSTs. This idea is in one sense not new. Kay suggested using multi-tape FSTs to deal with Arabic phonology and morphology in 1987 [29]. However, Kay assumed that these devices had no more than finite state power, an assumption that turns out to be false. As a result, however, MSFTs prove to have clearly adequate computational power to describe not just Arabic phonology and morphology, but also autosegmental rules that can be used to describe any language.

Chapter 2
Preliminaries

This thesis presents an automata-based computational model of autosegmental phonology. In order to make this work accessible to computer scientists not familiar with phonology, and to linguists not familiar with automata theory, this chapter contains a brief, basic introduction to some fundamental concepts of phonology (and in particular, autosegmental phonology); to the related field of morphology; and to automata theory.

2.1. Introduction to Phonology and Morphology

2.1.1. Morphology

The meaning of a language depends on the meaning of the parts. A morpheme is the smallest unit of meaning in a language. In English, this unit is (usually) smaller than a word and larger than a letter. For instance, the word "liveliness" can be divided up into three morphemes as live-li-ness, and each morpheme contributes a unit of meaning to the word as a whole.

The morphology of a language is an account of how the language combines morphemes into words. In English, this process is very simple compared to other languages of the world; English morphology consists mainly of concatenating a small number of prefixes and suffixes to the beginning and end of word roots. By contrast, Finnish has a more complicated morphology. Finnish words can have very long strings of prefixes and suffixes agglutinated to them, so that a noun can have some 2,000 inflected forms, and a verb can have over 12,000. In other languages we find more complicated phenomena like *reduplication* (repeated morphemes) and *discontinuous morphemes* (morphemes whose parts are distributed throughout a word). We give examples of these below:

(1) Reduplication (Maori)

dik	*thick*
dikdik	*very thick*

mero	*boy*
memero	*boys*
meromero	*little boy*
memeromemero	*little boys*

(2) Discontinuous morphemes (Arabic)

k-t-b	*to write*
katab	*he wrote*
jiktib	*he will write*
maktuub	*written*

In reduplication, whole morphemes or syllables are repeated in order to add a new shade of meaning to the original word. Discontinuous morphemes are even more complex. In Arabic, the verb root meaning "to write" is the three consonants k-t-b, but these never appear as a word by themselves in the language. Vowels and other consonants are added around and in between these three root consonants in order to add verb tense and subject information.

2.1.2. Phonology

The phonology of a language can be described in simple terms as the sound patterns of that language. An utterance in a language is not a random sequence of sounds. There is structure and order in the sounds that one hears and produces. Each language has a different set of constraints on what constitutes a well-formed sequence of speech sounds. For example, an English speaker knows that "strunk" might possibly be an English word, but that "mzwlunk" could not be. This is because one conforms to the phonological constraints of English, and the other does not. Somehow, the English speaker knows that "mzwl" is not allowed at the beginning of a word, but "str" is.

If a situation arises where a phonological constraint is violated, a *phonological rule* springs into action to correct the violation.[1] An easy way to illustrate this general principle of phonology is to look at morphological processes. When a morphological process puts two morphemes together, a constraint can be temporarily violated. Then it becomes necessary for a rule to make appropriate changes.

For example, in English, adding the plural suffix -s to the end of a word causes the phonological changes exemplified by the words below:

(3) caps
cats
tacks

(4) cabs (cabz)
rods (rodz)
dogs (dogz)

1. Phonological constraints tend to be implicit in a generative theory of phonology, where rules often "endeavor to achieve [a specific] end without ever directly acknowledging it" ([22], p. 87). By contrast, in a declarative theory of phonology, constraints are taken to be central, and they are explicitly stated. The means by which the constraints are satisfied is of secondary importance, and the details can be left unspecified, leaving open the possibilities for implementation.

In (3), the plural suffix -s is pronounced as an 's'. But in (4), it is pronounced as a 'z'. The only difference between 's' and 'z' is whether or not the vocal chords are vibrating. (You can feel this difference by placing your hand on your Adam's apple while pronouncing the two sounds.) The technical term for this is voicing — 'z' is voiced and 's' is not. Notice also the consonants next to the -s in (3) and (4) -- 'b', 'd', and 'g' are voiced, while 'p', 't' and 'k' are not. So in effect, the plural suffix -s has taken on the voicing characteristic of the neighboring consonant. This is an example of a very common type of phonological rule that is found in many languages, where voicing spreads to neighboring speech sounds.

We could summarize this situation by saying that English has a phonological constraint to the effect that two consonants pronounced next to each other in a word must agree in voicing. (If you were to pronounce the plural of *cab* as *cabss* instead of *cabz*, it would sound strange.) A phonological rule applies to situations where this is not true, making a phonological change to ensure that the constraint is satisfied.

From this example, one can see that there is a close connection between morphology and phonology. When the morphological component of a language dictates that two morphemes are to be concatenated, often there will be phonological changes taking place at the boundary between the two morphemes.

An important point to notice is that speakers of English do not think of there being two different plural suffixes (-s and -z), but only one. This is evidence that we have stored in our minds a single *underlying form* for the plural suffix, which takes on different *surface forms* when we actually pronounce it. The relationship between underlying and surface forms is determined by phonological rules and constraints.

Given this general framework, we are faced with some questions. How should we represent phonological forms (underlying and surface)? What is the nature of phonological constraints and rules, and by what mechanisms should rules change phonological representations? The answers to these and other questions lead to different theories of phonology.

2.2. Introduction to Autosegmental Phonology

2.2.1. History

A phonological representation is a representation of knowledge about the structure of speech sounds in an utterance of a language. Goldsmith ([22], p. 331) defines phonology this way:

> A theory of phonology is built of three parts: it is a theory of the nature of phonological representations; it is an inventory of levels of representation, and a characterization of each level; and it is a theory of phonological rules, the statements that relate representations on each level."

A theory of phonology has at least two levels of representation: the deepest level contains underlying forms, which are meant to represent the knowledge stored in the speaker's mind about the sounds of an utterance; and the highest level contains surface forms, which are obtained from the underlying forms by applying various phonological rules, yielding a representation which describes the properties of the actual utterance. To return to a previous example, an English speaker might have the underlying form "cab+s" stored in his mind, but by means of a phonological rule about voicing he produces the surface form "cabz".

For classical generative phonology, of which the definitive work is *The Sound Pattern of English* (SPE) by Chomsky and Halle [10], the phonological representations were simply linear strings of symbols, and the phonological rules were string rewriting rules (such as "b+s $\rightarrow$ bz"). Each symbol in a string represented a segment, which was thought of as a speech sound that was articulated during a particular slice of time with a definite beginning and ending point. Each segment had various features that might be related to the vocal apparatus, such as tone, position of the tongue, position of the lips, etc. For example, the vowel 'ú' (where the acute accent indicates high tone) might have the features [+H],

[+back], [+high], [+round], meaning that when the segment is pronounced, the tone is high, the tongue is back and high in the mouth, and the lips are rounded, or pursed. Similarly, the vowel 'à' (where the grave accent indicates low tone) would have the features [-H], [+back], [-high], [-round], meaning that the tone is not high, the tongue is back but not high in the mouth, and the lips are not rounded. Notice that the features are assumed to be binary, having either a '+' or a '-' value. Segments are usually written as a matrix of feature values:

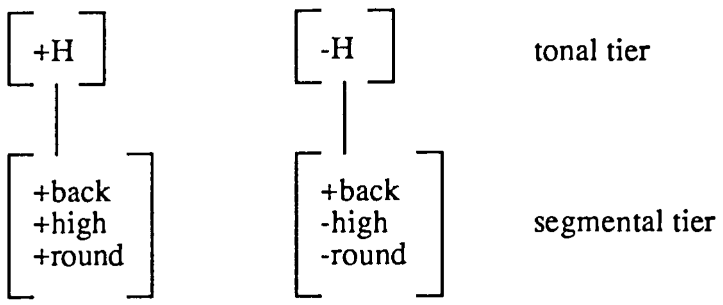
(5)

$$\begin{array}{cc} \begin{bmatrix} \text{+H} \\ \text{+back} \\ \text{+high} \\ \text{+round} \end{bmatrix} & \begin{bmatrix} \text{-H} \\ \text{+back} \\ \text{-high} \\ \text{-round} \end{bmatrix} \\ \text{'ú'} & \text{'à'} \end{array}$$

Beginning in 1976 with the work of Goldsmith [19,20], a new approach known as autosegmental (or non-linear) phonology began to take over from linear (SPE) phonology. The key difference in non-linear phonology is that phonological representations are no longer linear strings, but multiple tiers of segments with associations between the tiers. The idea for this sprang mainly from work on tonal languages. Tone was formerly considered to be simply a feature of a vowel, as above. However, it became clear that tone should be separated from the vowel of which it was formerly considered a part, and become an independent segment in its own right. (This is where the term autosegmental comes from--autonomous segment.)

So the feature(s) associated with tone were bundled into a segment, placed on a different tier in the representation (called the tonal tier), and associated with the vowel's remaining features by means of association lines. For example, the representation of the two vowels discussed above would now become:

(6)



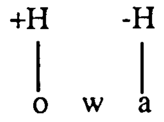
2.2.2. An Example

In the Etsako language, a whole word is reduplicated to add the meaning of “each” ([24], pp. 11-12). In the case of the word for “house”, we have:

ówà	*house*
ówõwà	*each house*

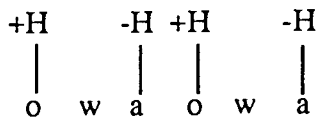
where the ‘˜’ accent means a rising (low to high) tone. Note that the final ‘a’ has been deleted in the first copy of the word. The autosegmental explanation of the rising tone on the ‘o’ is as follows. The word “house” is represented like this (here we write letters as shorthand for a feature matrix):

(7)



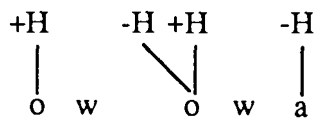
Then the word is reduplicated:

(8)



and finally, the first ‘a’ is deleted and its tone is reassociated to the next vowel:

(9)



and that is where the rising tone comes from: a sequence of a low tone followed by a high tone on the same vowel. The crucial point here is that the low tone is clearly behaving in an independent fashion by disassociating from its original vowel and reassociating to another, resulting in the other vowel having two tones (or what is known as a contour tone). This is something that was impossible to account for in a linear theory, where tone was simply a single feature of a vowel.

Eventually, this idea of features behaving as independent segments was extended to many other kinds of features, leading to a multi-tiered representation where each independent group of features had its own tier. If this idea is carried to its logical conclusion, each individual feature could have its own tier. One can think of an autosegmental representation as an orchestral score. Each staff, or line, in a score specifies what an individual instrument is doing at each point in time. In the same way, each tier in an autosegmental representation specifies the value of a feature or collection of features at successive points in time, which might correspond to the actions of the various articulators in the mouth (vocal chords, tongue, lips, etc). The association lines tell us which “notes” should overlap in time [6, 39].

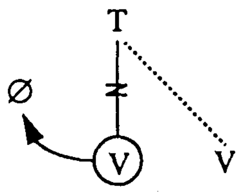
This is a completely different notion of a segment than was entertained in SPE phonology. No longer does a segment represent a whole speech sound, being pronounced over a time interval with definite beginning and ending points. A segment now is “no more than the minimal unit of phonological representation” ([22], p. 10), and the overlapping of many segments represents a speech sound. Because the segments that overlap to form a speech sound may not all start or end at the same point in time, it does not make sense to try to slice up the time line into intervals.

A phonological representation may have many tiers, and association lines between segments on any pair of tiers. A pair of tiers together with all the association lines between segments on those tiers is referred to collectively as a *chart.*

Thus, phonological rules in autosegmental theory operate on these multi-tiered representations. The elementary operations of which these rules are composed are addition and deletion of a segment on a tier, or addition and deletion of an association line. These operations are represented in a somewhat informal, ad hoc, and at times unclear notation in the literature [6].

Returning to our example, we show in (10) a rule that would accomplish the result in (9), where a vowel is deleted and its tone is reassociated to the following vowel (T stands for any tone and V for any vowel):

(10)



This rule demonstrates three of the elementary operations mentioned above. The circle around the first V with an arrow pointing to the null symbol indicates that the vowel is to be deleted. The small 'z' through the association line indicates that that association line is to be deleted. And the dotted association line indicates that an association line is to be added there. If we strip the rule of all these notations indicating the elementary operations to be performed, we are left with the structural description (SD) of the rule:

(11)

$$\begin{tikzcd} T \arrow[d, dash] & \\ V & V \end{tikzcd}$$

If a phonological representation contains the SD of a rule, then the rule applies to that representation.

2.2.3. Rule Ordering

The practicing phonologist proposes a large set of phonological rules to account for various phonological phenomena observed in the data. This raises an important question: does it matter what order we apply these rules in? The answer is "yes": the same rules applied in a different order can possibly derive a different surface form.

One of the major disadvantages of linear (SPE) phonology is the prohibitive amount of ordering of phonological rules. In order to generate the right surface forms, it became necessary in many cases to specify an ordering of the phonological rules that could not be deviated from without getting incorrect results. So rule A would have to apply before rule B, and rule B before rule C, etc. In fact, sometimes in order to explain two different phenomena, the same rules would have to be ordered one way to explain one phenomenon, and a different way to explain the other. So it was impossible to explain both at once! This was unsatisfactory, to say the least.

In autosegmental theory, much (although not all) of this need for rule ordering has been eliminated. This is partly due to the fact that features can be separated from each other onto different tiers, and thus do not interfere with each other in ways that linear phonology had to prevent by carefully ordering its rules. If rule ordering could be completely eliminated, the rules could be applied simultaneously instead of sequentially, which has significant implications for a computational model of phonology.

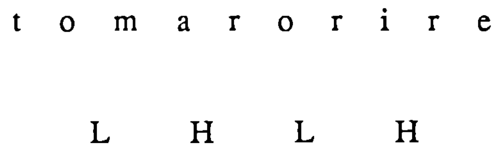
2.2.4. The Association Convention

It may happen that the underlying form of a word does not have any associations between tones and vowels. In this case, the task of associating the tones and the vowels is divided between language-specific rules and general conventions that apply to all languages. We have seen an example of the former in (10).

The Association Convention is an example of the latter. It is a kind of default association rule. It may affect a representation if it has at least one association line. It adds associations lines outward in a one-to-one fashion from the already present association line, associating from either tier only elements that are currently unassociated ([22], pp. 13-14).

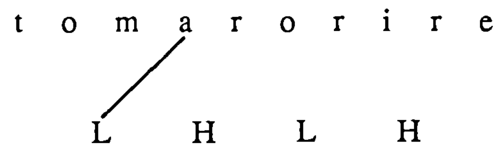
Goldsmith gives an example of a word in the Kikuyu language of Kenya whose underlying form is:

(12)



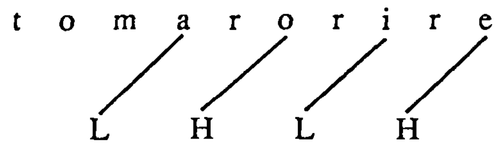
A language-specific rule (whose details need not concern us) associates the first tone to the second vowel:

(13)



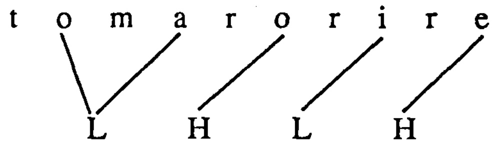
Then the Association Convention associates the rest of the tones:

(14)



Finally, another language specific rule gives the final, surface form:

(15)



This shows how the Association Convention interacts with language-specific rules to produce surface forms.

2.3. Introduction to Automata Theory

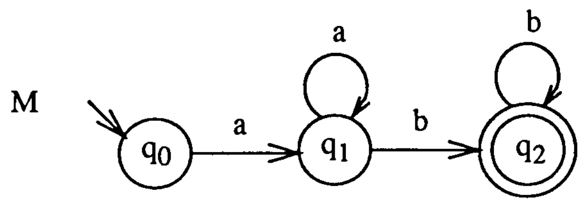
We will now give the reader a whirlwind tour of the subject of automata theory, touching on those topics that will arise later on in the thesis. For a more complete introduction to automata theory, we refer the reader to the standard texts by Hopcroft and Ullman [23] and Denning et. al. [14].

2.3.1. Finite State Automata

A *finite state automaton* (FSA) is an abstract mathematical device that reads input symbols on a tape and responds by either accepting or rejecting the input. It does this by changing its internal state each time it reads a symbol, and when it has read all input symbols on the tape, its internal state determines whether it accepts or rejects.

FSAs are typically represented using diagrams such as the one below:

(16)

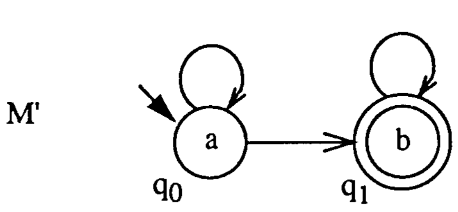


The circles represent internal states; the state in which the FSA always begins is marked by an incoming arrow. The accepting states (also called *final states*) are identified with double circles. The arrows, each labelled with an input symbol, indicate how the machine will change states when it reads that symbol. For example, if the above machine M is in state q_0 and reads an 'a' on the input tape, it will change (make a *transition*) to state q_1.

M accepts the input word "aabbb", because it will make transitions ending in state q_2, a final state. It does not accept the word "aaa", because it would end in state q_1, a non-final state. Let S be the set of all strings of a's and/or b's. M defines a subset of S consisting of all those strings that it accepts. This is called L(M), the *language* accepted by M. L(M) is referred to as a *finite state language* because it is defined by a finite state automaton.

M is a *transition-labelled* FSA. It is also possible to define *state-labelled* FSAs, which are represented in diagrams like this:

(17)

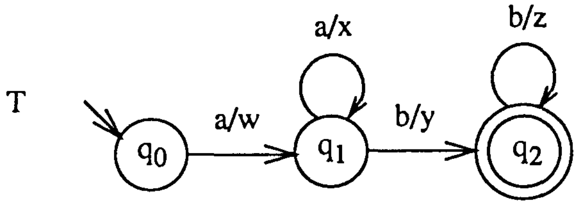


Each state is labelled with an input symbol. (The name of the state is written outside the circle.) A state-labelled FSA makes a transition along an arrow and into a state labelled with a symbol s iff it can read the symbol s from the input tape. For example, if M' above is in state q_0 and it reads a 'b' from the input tape, it makes a transition to state q_1. Notice that L(M') does not contain any strings beginning with 'b', because M' must always begin by reading an 'a' from the input tape.

Transition labelled FSAs and state-labelled FSAs are *equivalent.* That is, for each machine of one type, a machine of the other type can always be defined which accepts the same language. Notice that M and M' are equivalent: L(M) = L(M') = $\{a^m b^n \mid m,n > 0\}$[2].

A *finite state transducer* (FST) is an FSA which reads pairs of symbols from its input tape, as if it were reading two tapes at once:

(18)



T accepts pairs of words, including <ab, wy> and <aaabb, wxxyz>. An FST accepts a set of pairs of words, or a binary relation on words. For example, The binary relation R(T) accepted by T is $\{\langle aa^m bb^n, wx^m yz^n\rangle \mid m,n \geq 0\}$. Notice that FSTs can only accept pairs of words of the same length.

We can consider an FST to be scanning two tapes simultaneously. On each transition, it scans two symbols. But we can change things somewhat, and think of each of those symbols as being either read from an input tape or written onto an output tape. Thus, an FST can be viewed as either accepting pairs of words, or generating pairs of words, or

2. By convention, the solid arrow-head in (17) indicates that M' does not accept the empty input. A hollow arrow-head would indicate that an empty input tape is accepted.

reading an input word on one tape and writing an output word on the other. The latter interpretation of FSTs is useful, because one can view them as computing a function of the input word. In this case the FST is said to *transduce* the input word to the output word.

A *generalized sequential transducer* (GST) is like an FST, but it is able to accept pairs of words of different length, as if it were scanning two tapes independently. One way of doing this is to allow an FST to write strings (including the empty string) on its output tape instead of only symbols. An equivalent way is to divide up the states of an FSA into two partitions, and call one partition the "scan" states and the other the "print" states. Then the machine can freely alternate back and forth between reading input symbols (when it is in the "scan" states) and writing output symbols (when it is in the "print" states).

A relation accepted by an FST or a GST is called, for reasons that we shall soon see, a *regular relation.*

So far, we have considered only *deterministic* FSAs, FSTs, and GSTs. A deterministic machine is one that, at each step of its execution, has only one choice, so its behavior is uniquely determined by the input. A non-deterministic machine is one that may have more than one choice at each step. For an automaton, this means there may be several possible sequences of transitions by which a given input can be accepted. For a transducer, this in turn means that an input word might be transduced to more than one output word.

2.3.2. Regular Languages

A *grammar* is a string rewriting system which contains a special *start symbol* S, some *non-terminal* symbols (by convention, capital letters), some *terminal* symbols (any other symbols), and a finite set of *productions*, rules which transform strings to other strings. These rules are of the form $x \rightarrow y$, where x and y are strings of terminal and/or non-terminal symbols; a rule is interpreted as meaning "replace any one occurence of x by y". The idea is to begin with S and apply a finite sequence of productions until a string of terminal symbols is obtained. The string thus obtained is said to be in the *language* of the grammar;

the language *generated* by the grammar is the set of all strings that can be produced in this manner.

A *regular grammar* is a grammar in which the productions are all of the form $A \rightarrow wB$ or $A \rightarrow w$, where A and B are any non-terminal symbols, and w is any string of terminal symbols (including the empty string). For example, the following regular grammar generates the language $L(M) = L(M') = \{a^m b^n \mid m,n > 0\}$:

(19) $S \rightarrow aS$
$S \rightarrow aT$
$T \rightarrow bT$
$T \rightarrow b$

A language generated by a regular grammar is called a *regular language*.

A *regular expression* is a sequence of symbols from some alphabet A, connected by the binary operators + and · (usually omitted when writing expressions) and the unary operator *. A regular expression denotes a language. Regular expressions and the languages they denote, are defined as follows:

- $\varnothing$ is a regular expression and denotes the language $\{\}$
- ε is a regular expression and denotes the language $\{\varepsilon\}$
- for all $a \in A$, a is a regular expression and denotes the language $\{a\}$
- if r and s are regular expressions denoting the languages R and S respectively, then $(r+s)$, (rs) and (r^*) are regular expressions denoting $R \cup S$, $RS = \{rs \mid r \in R, s \in S\}$, and $R^* = \{r^i \mid i \geq 0\}$ respectively

For example, a regular expression denoting the language $L(M) = L(M') = \{a^m b^n \mid m,n > 0\}$ would be (aa^*bb^*).

The languages described by regular expressions are exactly the regular languages. Also, the finite state languages are exactly the regular languages. In this sense, regular grammars, regular expressions, and FSAs are equivalent: they all describe the same class of languages.

Regular languages are *closed* under certain operations, including union, intersection, complement, concatenation, reversal, and Kleene closure (the * operation on languages). That is, these operations produce languages which are also regular (see [14]). So, for example, the intersection of two regular languages is also a regular language.

2.3.3. Other Classes of Languages

Regular (R) languages can be described by both automata and grammars. There are other, larger classes of languages which also have this dual characterization. These include the most well-known classes of context free (CF), context sensitive (CS), and recursively enumerable (RE) languages (RE languages correspond to well-known automata called Turing machines). These four classes of languages are strict subsets of one another, in the given order, and form what is known as the Chomsky hierarchy, which is pictured in Figure 2-1. Also shown in that diagram, as a shaded circle, is the class of indexed languages, which fall between the CF and CS language classes.

Given a language over some alphabet A, a homomorphism is a mapping from symbols in A to strings of symbols in A. For example, if A = {a, b}, we could define the mapping $h = \{a \rightarrow aa, b \rightarrow aba\}$. If every word in a language is transformed by such a homomorphism, the result is another language. So the language $L(M) = aa^*bb^*$ is transformed by the homomorphism h into the language $aa(aa)^*aba(aba)^*$. All the classes of languages mentioned above, and many others, are closed under homomorphism.

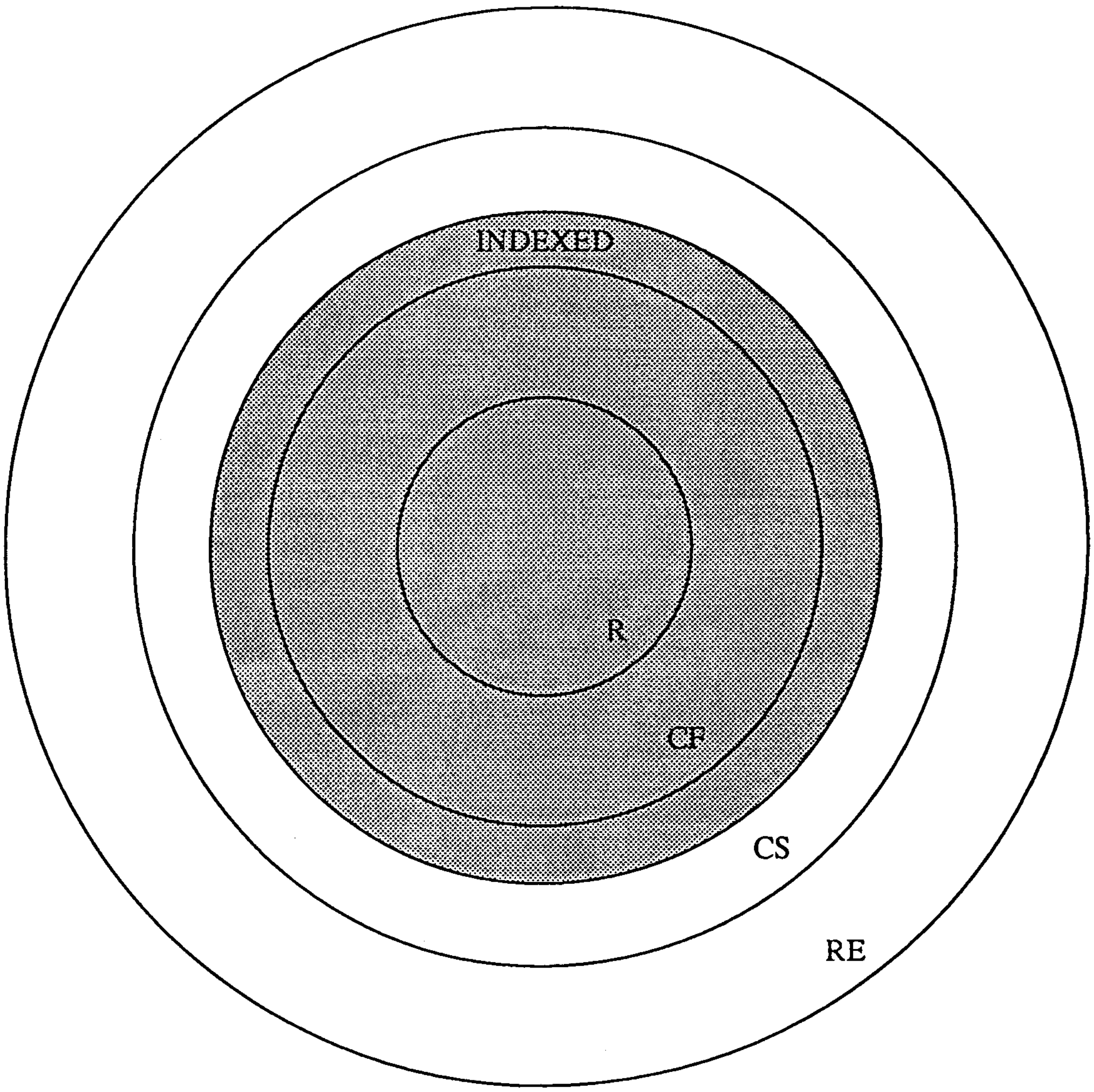


Figure 2-1: The Chomsky Hierarchy and Indexed Languages

Chapter 3
MSFAs and MSFTs

In this chapter we describe and define the automata that form the basis of our model of autosegmental phonology, and discuss their relation to other automata in the literature.

3.1. Intuitive Description

3.1.1. General Remarks

Multi-tape FSAs were first introduced by Rabin and Scott in 1959 [34]. Intuitively, a multi-tape FSA is an FSA with multiple input tapes instead of just one. These tapes are independent — that is, each tape has its own independent read head,[1] unlike the finite state transducer, which has two tapes but only a single read head. The finite state control is similar except that it must specify which tape is being scanned (ie. which read head is being advanced) on each transition. Also, the automaton halts only when all read heads have reached the ends of their respective tapes. Notice that a single execution of a multi-tape FSA with n tapes accepts an n-tuple of words (one word on each tape), rather than a single word like an FSA. In other words, multi-tape FSAs accept n-ary relations instead of languages.

Multi-tape FSTs are similar to multi-tape FSAs, having in addition a set of output tapes, one for each input tape.[2] (These devices have been used informally by Kay [29].)

1. An FSA with only one read head that scans all tapes simultaneously is just a multi-track FSA, which is equivalent to a standard FSA.
2. Actually, a multi-tape FST could have more or less output tapes than input tapes. We have chosen to make this a part of our definition. This choice is not without linguistic significance, and will be discussed later on in this chapter.

Their finite state control specifies for each transition whether the machine is scanning or printing, and on which tape. A multi-tape FST is said to transduce an n-tuple of words $W=<w_1, ..., w_n>$ to another n-tuple $X=<x_1, ..., x_n>$ if, beginning in a start state, it can completely scan each word of W on the input tapes, and end in a final state with X on the output tapes.

The basic definitions of multi-tape FSAs and multi-tape FSTs are non-deterministic. That is, at each step of execution, the automaton may have several possible valid transitions to choose from. Notice that a non-deterministic multi-tape FST may transduce an input to more than one output. Of course, we could also define deterministic versions of these machines, in which at each step only one choice is available. We have chosen to use the non-deterministic version of these machines because they can accept languages using fewer states than deterministic machines.

3.1.2. MSFAs

We will define a version of multi-tape FSAs whose states are labelled rather than their transitions. They are generalizations of the *state-labelled non-deterministic finite automata (SFAs)* of Bird & Ellison [8]. They are accordingly called *multi-tape SFAs (MSFAs)*. An example of a two-tape MSFA is given in Figure 3-1.

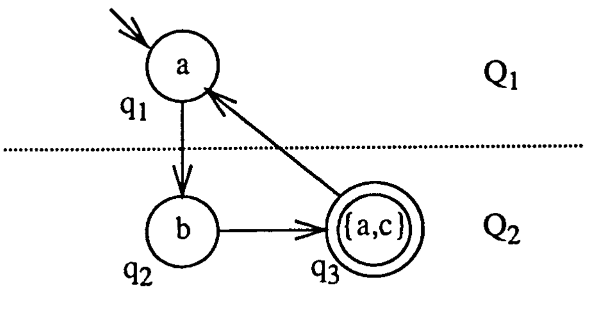


Figure 3-1: A simple MSFA

Most of the notation used above is familiar. Notice that the states are labelled with sets (braces are omitted for single element sets). The states are also named q_1, q_2, and q_3 simply for ease of reference. (Do not confuse a state's *label* with its *name*.) The dotted line divides the states of the automaton into two partitions, Q_1 and Q_2. Each partition corresponds to one of the input tapes of the automaton. A transition *into* a state q in partition Q_i is interpreted as scanning a single symbol on the i^{th} tape. The symbol scanned must be one of those given in the label of the state q.

Suppose this automaton is given the input <aa, bcba> (ie. "aa" on tape 1, "bcba" on tape 2). It begins by entering the initial state q_1 and advancing the read head on tape 1 past the first "a". It then makes a transition to q_2, scanning the first "b" on the second tape. This is followed by a transition to q_3, which allows for either an "a" or a "c" to be scanned on tape 2; the input symbol is a "c", so the automaton can proceed. In this way the transitions continue back to q_1, to q_2, and finally to q_3. At this point all the input on both tapes has been scanned, and the automaton is in a final state, so it terminates successfully, accepting the input <aa, bcba>.

We can describe this sequence of transitions as a sequence of pairs, each pair showing the current state and the portion of the input that has already been scanned:

$$\begin{aligned}
[q_1, \langle \text{a}, \varepsilon \rangle] &\rightarrow [q_2, \langle \text{a}, \text{b} \rangle] \\
&\rightarrow [q_3, \langle \text{a}, \text{bc} \rangle] \\
&\rightarrow [q_1, \langle \text{aa}, \text{bc} \rangle] \\
&\rightarrow [q_2, \langle \text{aa}, \text{bcb} \rangle] \\
&\rightarrow [q_3, \langle \text{aa}, \text{bcba} \rangle]
\end{aligned}$$

One can see the kind of 2-tuples this automaton will accept: <a, bc>, <aaa, bababa>, <aaaa, bcbcbabc>. This MSFA accepts exactly the set of 2-tuples in the relation $\langle \text{a}^n, (\text{b}(\text{a+c}))^n \rangle$, $n \geq 0$.[3]

3. Actually, there is nothing in the description of the automaton so far that tells us whether or not it accepts the empty input <ε, ε>. This will be remedied later.

3.1.3. MSFTs

The basic element of our model of autosegmental phonology will be *multi-tape state-labelled finite transducers* (*MSFTs*), a version of multi-tape FSTs. An example of an MSFT with two input tapes and two output tapes appears in Figure 3-1.

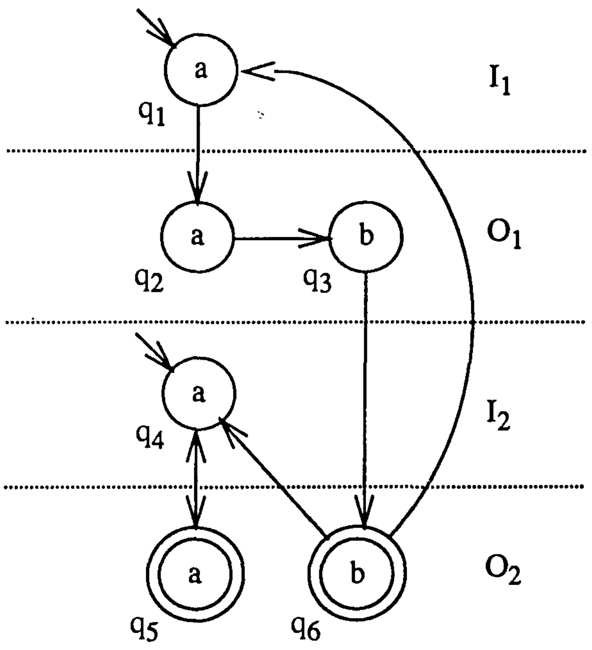


Figure 3-2: A simple MSFT

The dotted lines divide the states of the MSFT into four parts: two *input* partitions I_1 and I_2, and two *output* partitions O_1 and O_2. A transition into a state in I_i causes a symbol to be scanned from input tape *i*, and a transition into a state in O_i causes a symbol to be printed on output tape *i*.

Notice that there are two initial states. Both MSFTs and MSFAs can have multiple initial states, so that the machine can choose which state to begin execution in. It is necessary to allow this choice, because (unlike transition-labelled machines) the initial state specifies which symbol(s) must be read first, and on which tape, and we want to allow any possibility at the first step of execution, just like at any other step. Notice that this choice does not necessarily imply that the machine is non-deterministic. If the labels of the initial states are mutually disjoint, for example, then the machine will have only one choice at the first step of execution.

The double-headed arrow between states q_4 and q_5 is simply shorthand for two one-way arrows.

This machine transduces the input <aa, a> to the output <abab, bba>. The sequence of transitions that it follows could be described by a sequence of triples, each triple consisting of the current state, the input scanned so far, and the output printed so far:

$$
\begin{aligned}
[q1, \langle a, \varepsilon\rangle, \langle \varepsilon, \varepsilon\rangle] &\rightarrow [q2, \langle a, \varepsilon\rangle, \langle a, \varepsilon\rangle] \\
&\rightarrow [q3, \langle a, \varepsilon\rangle, \langle ab, \varepsilon\rangle] \\
&\rightarrow [q6, \langle a, \varepsilon\rangle, \langle ab, b\rangle] \\
&\rightarrow [q1, \langle aa, \varepsilon\rangle, \langle ab, b\rangle] \\
&\rightarrow [q2, \langle aa, \varepsilon\rangle, \langle aba, b\rangle] \\
&\rightarrow [q3, \langle aa, \varepsilon\rangle, \langle abab, b\rangle] \\
&\rightarrow [q6, \langle aa, \varepsilon\rangle, \langle abab, bb\rangle] \\
&\rightarrow [q4, \langle aa, a\rangle, \langle abab, bb\rangle] \\
&\rightarrow [q5, \langle aa, a\rangle, \langle abab, bba\rangle]
\end{aligned}
$$

This MSFT takes any input of the form $\langle a^n, a^m \rangle$ and transduces it to $\langle (ab)^n, b^n a^m \rangle$, $n, m \geq 0$.[4]

3.2. Formal Definitions

Let us now give a more formal definition of these devices.

4. Again, it is not clear what the MSFT would do if given the input <ε, ε>. This will be made clear.

3.2.1. MSFAs

Definition 3-1: A *multi-tape non-deterministic state-labelled finite automaton* with n tapes (n-tape *MSFA*) is a septuple $<Q, \Sigma, \lambda, \delta, I, F, e>$ where:

- Q is a finite set of *states*, partitioned into n subsets $<Q_1, ..., Q_n>$
- Σ is a finite set, the *alphabet*
- $\lambda \subseteq Q \times \Sigma$ is a *labelling relation*, assigning a subset of the alphabet to each state
- $\delta \subseteq Q \times Q$ is a *transition relation*, assigning a subset of states to each state
- $I \subseteq Q$ is a set of *initial states*
- $F \subseteq Q$ is a set of *final states*
- e is a boolean value, *true* if the automaton accepts the empty n-tuple $<\varepsilon, ..., \varepsilon>$, and *false* otherwise ❑

Each partition Q_i of Q corresponds to one of the tapes of the automaton. A transition *into* a state q in partition Q_i is interpreted as scanning a single symbol on the i^{th} tape. The state q must be *compatible* with the particular symbol on that tape.

Definition 3-2: A state q is *compatible* with a symbol $\sigma \in \Sigma$ on tape i iff $(q, \sigma) \in \lambda$ and $q \in Q_i$. ❑

An MSFA accepts an input n-tuple written on its tapes as follows: The read heads begin at the left edges of their respective input tapes. The MSFA begins execution by, as it were, making a "transition" into any initial state $q \in I$ that is compatible with the first symbol on some tape. The read head on that tape is advanced past the first symbol. On each transition, the MSFA may change states to any state q' such that $(q, q') \in \delta$ and q' is compatible with the next symbol on some tape i. This advances the read head on tape i. The MSFA *accepts* the input if there is some sequence of transitions that leads to a final state, with all the read heads at the end of their respective tapes. More formally:

Let $A = <Q, \Sigma, \lambda, \delta, I, F, e>$ be an n-tape MSFA.

Definition 3-3: A *situation* of A is a pair $[q, X]$, where q is the current state, and X is an n-tuple $<x_1, ..., x_n>$, each $x_i \in \Sigma^*$, representing the portion of the input that has already been scanned on the tapes. ❑

Definition 3-4: Define the relation $\rightarrow_A$ on situations of A as follows: $[q, X] \rightarrow_A [r, Y]$ iff:

- •$(q, r) \in \delta$ (there is a transition from q to r)
- •for some tape i, $\exists\ \sigma \in \Sigma$ such that $x_i\sigma = y_i$ and $x_j = y_j\ \forall\ j \neq i$; in other words, $X = <x_1, ..., x_i, ..., x_n>$ and $Y = <x_1, ..., x_i\sigma, ..., x_n>$ (a single symbol σ has been scanned on tape i)
- •r is compatible with σ on tape i

Define $\rightarrow_A^*$ to be the transitive closure of $\rightarrow_A$. ❑

In order to define formally what it means for an MSFA to accept an input, we need to consider a slight variation on an MSFA called a *modified MSFA*. An example of a modified MSFA that is equivalent to the MSFA in Figure 3-1 is shown in Figure 3-3.

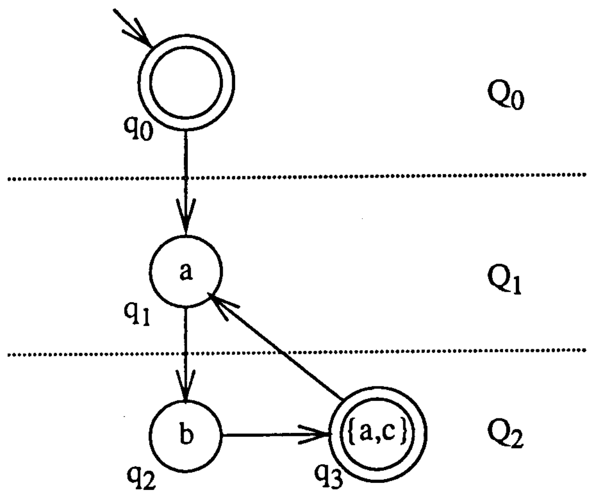


Figure 3-3: A simple modified MSFA

This machine is interpreted slightly differently. It begins execution already in the initial state q_0, with both of its read heads to the left of the input words on the tapes. If the input is $<\varepsilon, \varepsilon>$, then no transitions are taken, and the input is accepted because q_0 is a final state. If the input is not $<\varepsilon, \varepsilon>$, then execution proceeds as it does with an MSFA, each transition into a state resulting in a symbol being scanned on one of the tapes.

This modified MSFA accepts the same inputs as its counterpart in Figure 3-1. However, notice how Figure 3-3 makes explicit something that was not specified in Figure 3-1: namely, whether or not the empty input $<\varepsilon, \varepsilon>$ is accepted.

The state q_0 is a special state labelled with the empty set, and is in a partition by itself, a partition which does not correspond to either of the input tapes. The only purpose of q_0 is to specify (by means of being a final state or not) whether the empty input is accepted.

Definition 3-5: Let $A = <Q, \Sigma, \lambda, \delta, I, F, e>$ be an n-tape MSFA, where Q is partitioned into $<Q_1, ..., Q_n>$. Let $W = <w_1, ..., w_n>$ be an n-tuple, each $w_i \in \Sigma^*$. To more easily define what it means for A to accept W, we first define a *modified MSFA* $A' = <Q', \Sigma, \delta', \lambda, I', F'>$, a variation on A, as follows:

- $Q' = Q \cup \{q_0\}$ (add a distinguished state q_0, labelled with the empty set)
- Q' is partitioned into $<Q_0, Q_1, ..., Q_n>$, where $Q_0 = \{q_0\}$ (q_0 is in its own special partition Q_0, which does not correspond to any tape)
- $\delta' = \delta \cup \{(q_0, q) \mid \forall\, q \in I\}$ (add transitions from q_0 to all initial states in A)
- $I' = \{q_0\}$ (q_0 is the only initial state in A')
- $F = F \cup \{q_0\}$ if $e = true$, otherwise $F' = F$ (q_0 is a final state iff A accepts the empty input $<\varepsilon, ..., \varepsilon>$)

We say that A *accepts* W iff $[q_0, <\varepsilon, ..., \varepsilon>] \rightarrow_{A'}^{*} [q, W], \exists\, q \in F$. In other words, A accepts W iff A' can start in state q_0 and find a sequence of transitions that scan W and end in a final state. ❑

Modified MSFAs are simply a device to make the formal definition of *accept* easier to state. They will also be especially helpful later on when we define one of our operations on MSFAs. Usually, diagrams of MSFAs will be of the type in Figure 3-1. In order to indicate on a diagram whether or not an MSFA accepts the empty input, we can use two different types of arrow-heads on the arrows that mark the initial states: hollow arrow-heads (—⊳) will mean that that <ε, ..., ε> is accepted, and solid arrow-heads (—►) will mean that it is not.

Actually, MSFAs have more than one possible interpretation. We have been speaking of them as *acceptors*, interpreting the tapes as input tapes with read heads, each transition scanning an input symbol on a tape. But (as with FSAs) they can also be considered *generators*, interpreting the tapes as output tapes with write heads, each transition printing an output symbol on a tape. Furthermore, since MSFAs have multiple tapes, *some* of the tapes could be considered input tapes, and *some* output tapes; ie. MSFAs can be interpreted as *transducers*.

3.2.2. MSFTs

Definition 3-6: A *multi-tape non-deterministic state-labelled finite transducer* with n tapes (n-tape *MSFT*) is a $2n$-tape MSFA, where n tapes are designated as input tapes, and n as output tapes.[5] The states Q are partitioned into $<I_1, ..., I_n, O_1, ..., O_n>$, each I_i corresponding to the ith input tape, and each O_j corresponding to the jth output tape. The boolean value e is *true* iff the MSFT transduces <ε, ..., ε> to <ε, ..., ε> (ie. iff, considered as a $2n$-tape MSFA, it accepts the $2n$-tuple <ε, ..., ε, ε, ..., ε>). □

Definition 3-7: An n-tape MSFT is said to *transduce* input $W=<w_1, ..., w_n>$ to output $X=<x_1, ..., x_n>$ iff, considered as a $2n$-tape MSFA, it accepts $<w_1, ..., w_n, x_1, ..., x_n>$. □

There are also several possible interpretations of MSFTs — four, to be precise. A transducer can be interpreted as either reading or writing on its set of "input" tapes, and

5. As we have indicated before, a more general definition is possible in which an MSFT can have a different number of input tapes than output tapes. For our purposes in modelling autosegmental rules, we only need the more restricted definition given here.

likewise on its set of "output" tapes. If it reads from both, it is *accepting* input-output pairs. If it writes on both it is *generating* input-output pairs. If it reads from its input tapes and writes on its output tapes (the usual interpretation), it is performing *transduction.* If it reads from its output tapes and writes on its input tapes, it is performing *inverse transduction.* This latter interpretation will be important for our purposes in modelling autosegmental rules. We will be able to use the same device to model the application of rules in two directions: from underlying to surface forms via transduction, and from surface to underlying forms via inverse transduction.

3.3. Elsewhere in the Literature

Multi-tape FSAs were first introduced by Rabin and Scott in 1959 [34]. Fischer gives a good summary of their properties [17]. Fischer points out in his survey that the class of n-ary relations accepted by multi-tape FSAs (for $n > 1$) has different properties than the class of unary relations (ie. languages) accepted by FSAs. This implies that multi-tape FSAs cannot be equivalent to FSAs.

Our MSFAs and MSFTs are a generalization of the SFAs of Bird & Ellison [8]. They can also be considered a generalization of *generalized sequential transducers* (*GSTs*) [14].

GSTs are themselves a generalization of finite state transducers. In one sense they are no more powerful than FSTs, because they can only map regular languages to other regular languages. However, they do have a capability that FSTs do not have. Because they can, for each input symbol scanned, print zero or more output symbols, they can transduce input strings to output strings that are longer or shorter than the input string [14].

We could have defined a different version of multi-tape FST, one that could only write exactly one output symbol for each input symbol scanned. This could have been done by defining a multi-tape FST as a multi-tape FSA with an alphabet of pairs, by analogy with the way FSTs have often been defined as FSAs with an alphabet of pairs. However, given that we want to model autosegmental rules, we need to be able to perform transductions that do not preserve length. When an autosegmental rule adds a segment to, or deletes a segment from, a tier, it changes the length of that tier. For this reason we needed to define transducers that can perform transductions that do not preserve length.

State-labelled transducers were chosen over transition-labelled transducers because they have a couple of minor advantages when used to represent autosegmental rules. One is that they typically have less non-determinism; the other is that their diagrams are visually clearer.

There are several equivalences between our automata and other automata found in the literature that are straightforward to prove. Our MSFAs are equivalent to the multi-tape FSAs defined by Rabin & Scott [34]. This allows us to later make use of some of their results. Also, Bird & Ellison have noted the equivalence of their SFAs with (transition-labelled) FSAs [8], a result that follows from the equivalence of Mealy and Moore machines [23]; this result can be extended to MSFAs, defining a transition-labelled version of multi-tape FSAs and demonstrating equivalence. The proof would use similar techniques to those found in the proof of Mealy/Moore equivalence. Finally, this result can be used to show that GSTs are a special case of MSFTs.

3.4. Phonological Considerations

As has been mentioned before, using multiple tapes instead of a single tape on which to encode autosegmental representations is natural, given that the representations themselves are multi-tiered, and not linear. Kornai [32] has reduced autosegmental representations to a linear encoding so that they will fit on a single tape of an FST, but there are difficulties with this approach that we will discuss in Chapter 8. It is a hard problem to overcome these difficulties by finding an alternative linear encoding. One gets the feeling that autosegmental representations are essentially multi-linear, and cannot be forced into a linear string without requiring more than finite state power to process them. We offer no conclusive proof of this statement, but instead offer a solution that retains the multi-linear character of autosegmental representations, and uses more than finite state power to process them.

One question that may have occurred to phonologists reading the definition of MSFTs relates to the alphabet. An MSFT has a single alphabet for all of its tapes. Each tape will correspond to a different autosegmental tier, and each alphabet symbol will represent an autosegment (containing one or more features). This means that the same alphabet symbols could appear on different tapes, contrary to the requirement in autosegmental phonology that features cannot appear on more than one tier [22].

This is not a problem. If we want to have a different alphabet for each tape, we can just partition the single alphabet into *n* parts, one for each tape, much as we partitioned the state set. An MSFT with multiple disjoint alphabets is a special case of an MSFT with a single alphabet.

However, it may be advantageous to allow the *possibility* of having the same feature on different tiers. Goldsmith considers this possibility in an analysis of Sierra Miwok ([22], p. 89). At least we have this option open to us.

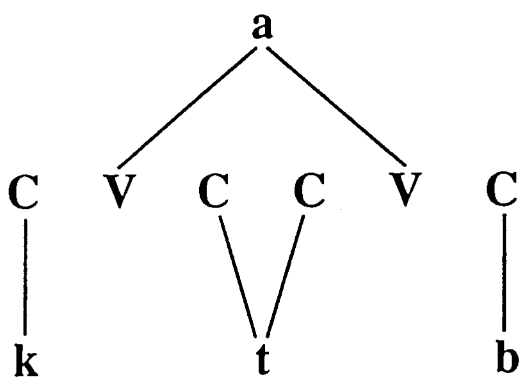
Chapter 4
Encoding Autosegmental Representations

Now that we have introduced MSFTs, the computational device we will use to model autosegmental rules, we will show how autosegmental representations can be encoded on the tapes of an MSFT.

4.1. Intuitive Description

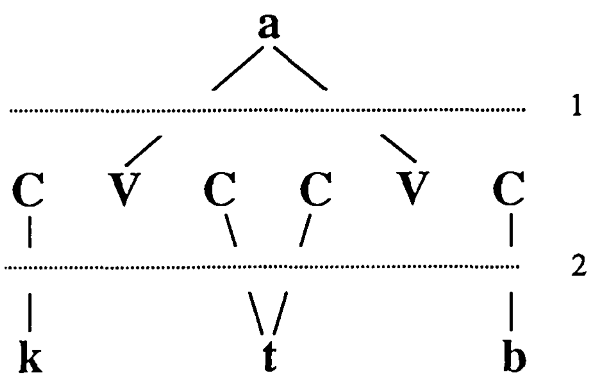
Consider the following representation of the Arabic verb stem **kattab** meaning "caused to write":

(20)



The verb stem contains three morphemes, one on each tier. This representation can be, in effect, cut into three parts like this:

(21)



Using this as a guide, the representation can easily be encoded as three strings on a 3-tape MSFT:

(22)
```
a11
C2V1C2C2V1C2
k2t22b2
```

Here, a digit following a segment is used to represent the incidence of an association line on that segment. The incidence of association lines from different charts is represented using distinct digits (here 1 or 2). So, for example, in the middle string (representing the middle tier) we have a C followed by a 2 representing an association line from the bottom chart; then a V followed by a 1 representing an association line from the top chart; etc. In the bottom string, there is a t followed by two 2's, representing the incidence on the segment t of two association lines from the bottom chart.

Although this example does not show it, this encoding is powerful enough to also represent associations between the top and bottom tier. We could use 3's to represent such association lines, if there were any.

It is important to realize why we do not lose any information content in the representation by dividing it up into parts like this. The reason is that we can always put the parts back together properly (ie. match up the halves of the association lines to their proper partners). We can do this because of the No Crossing Constraint (NCC), which says that well-formed autosegmental representations cannot contain association lines that cross[1]. The NCC ensures that within a chart, the association line halves will match up *in order.* It prohibits, for example, the first half-line on one tier matching up with the third half-line on the facing tier of the chart. So a person (or an algorithm) given the encoding in (22) could easily reconstruct the representation in (20) by pairing up the **1**'s in the order that they appear on the two tiers, and doing the same with the **2**'s.

4.2. Formal Definition

We will now define more precisely the encoding scheme we have described above. To do this, we must first precisely define what we mean when we speak of an autosegmental representation.[2]

4.2.1. Autosegmental Representations

We do this in terms of a generalization of the mathematical notion of a bipartite graph.

Definition 4-1: An *n-partite graph* G is a pair <V, E> where:

- V is a set of vertices, partitioned into *n* parts $<V_1, ..., V_n>$
- E is a set of edges — unordered pairs of vertices in V
- E satisfies the property that $u, v \in V_i \Rightarrow (u, v) \notin E$ (ie. there are no edges between vertices in the same partition) ❑

Definition 4-2: An *autosegmental representation A* with *n* tiers is a 4-tuple <G, S, L, O> where:

- G is an *n*-partite graph <V, E>, with V partitioned into $<V_1, ..., V_n>$ (these partitions correspond to the tiers of the representation)
- S is a set of symbols which represent segments[3]

1. We will define the NCC more precisely in a moment.
2. Our definitions are similar to those of Coleman and Local in [12].

- L: $V \rightarrow S$ is a labelling function, assigning to each vertex $v \in V$ a label $L(v)$ from the set S
- O is a collection of linear ordering relations $(\leq_1, ..., \leq_n)$, where each $\leq_i$ is a linear ordering of V_i[4]
- the edges E of G satisfy the NCC on each chart; that is: $\forall\, i, j, \forall\, u, v \in V_i, w, x \in V_j$, if $u \leq_i v$ and $w \leq_j x$, then $(u, x) \in E \Rightarrow (v, w) \notin E$. ❑

Definition 4-3: Define A_n as the set of all autosegmental representations with *n* tiers over a common set S of segments. ❑

Definition 4-4: Two autosegmental representations $A, B \in A_n$ are said to be *equal* iff there is an *isomorphism* between them (a one-to-one correspondence between their vertices that preserves all the structure that defines autosegmental representations — the edges, the vertex partitions, the vertex labels, and the linear ordering of the partitions). That is, given two *n*-tiered representations $A = <G_A, S, L_A, O_A>$ and $B = <G_B, S, L_B, O_B>$, we say $A = B$ iff there is a 1-1, onto function $\tau: V_A \rightarrow V_B$ such that $\forall\, u, v \in V_A$:

- $(u, v) \in E_A \Rightarrow (\tau(u), \tau(v)) \in E_B$ (edges are mapped to edges)
- $\forall\, i\, \exists\, j$ such that $u, v \in V_{A,i} \Rightarrow \tau(u), \tau(v) \in V_{B,j}$, and $u <_i v \Rightarrow \tau(u) <_j \tau(v)$ (partitions are mapped to partitions, and the linear order within them is preserved)
- $L_A(v) = L_B(\tau(v))$ (labels are preserved) ❑

4.2.2. The Multi-Linear Code

We want to encode elements of A_n as *n*-tuples of strings.

Definition 4-5: The *multi-linear code* is defined by an encoding function $E_n: A_n \rightarrow (\Sigma^*)^n$,

3. The elements of the set S are meant to represent phonological features or feature bundles (segments, in the sense that Goldsmith uses the term [22]); thus, they will have some internal structure. But we need not go into detail about this here, and will simply represent elements of S as single symbols, as is often done by phonologists.
4. We can allow these to be partial linear orderings if we want to represent tiers with partially ordered autosegments, or no linear ordering at all ([26], see [12]).

where $\Sigma = S \cup \{\alpha_{ij} \mid i,j = 1,\dots,n, i < j\}$, where the α_{ij} are unique symbols not contained in S. Each α_{ij} will represent the incidence of an association line on a segment: specifically, an association line from the chart containing the i^{th} and j^{th} tiers.

Let $A \in A_n$, as in Definition 4-2. We will define $E_n(A)$ by defining the various pieces that make it up. Let the *p(i)* vertices in each partition (tier) V_i be named, in order, $v_{i1} <_i v_{i2} <_i \dots <_i v_{ip(i)}$. Consider v_{ik}, the k^{th} vertex in the i^{th} partition (tier). It has a total of $q_{ijk} = |\{(v_{ik}, w) \in E \mid w \in V_j\}|$ association lines linking it to vertices in the j^{th} partition (tier). This is encoded as the string $g_{ijk} = \alpha_{ij}^{q_{ijk}}$. The vertex v_{ik} together with all its association lines is encoded as the string $f_{ik} = L(v_{ik})\, g_{i1k}\, g_{i2k} \dots g_{ink}$. The whole of the i^{th} tier is encoded by the string $e_i = f_{i1} f_{i2} \dots f_{ip(i)}$. So, finally, we can define $E_n(A) = \langle e_1, \dots, e_n \rangle$. ❑

In spite of the gory mathematical detail, this encoding function E_n is really quite simple, as we have already seen in the example above. In fact, it is so straightforward that it can be considered just a notational variant of autosegmental diagrams. Certainly it is not as graphic, but the correspondence between autosegmental representations and multi-linear encodings is obvious.

4.3. Linguistic Advantages of the Multi-Linear Code

We have already alluded to the simplicity of the multi-linear code. There are several other features of the code that will make it attractive to phonologists.

Kornai [32] has laid down four criteria of a good coding scheme for encoding autosegmental representations. Although these criteria were originally stated with respect to *linear* encodings, three of them — computability, invertibility, and iconicity — are just as applicable to non-linear encodings like the multi-linear code. As for the fourth criterion — compositionality — we shall see that a *linear* encoding *cannot* be fully compositional. Accordingly, we argue that a more appropriate criterion is an alternate version of compositionality that we call *tier-wise compositionality,* and show that the multi-linear code satisfies it fully. We also show that the multi-linear code is incapable of representing crossing association lines.

4.3.1. Computability

At the most basic level, this criterion means that there is an algorithm that can be used to effectively compute the encoding. This is certainly true for our encoding.

This basic definition can be refined in two ways. On the one hand, one can consider (as Kornai does) the formal amount of computing power required to compute the encoding. In other words, does it require the full power of a Turing machine, or only that of a finite state automaton, or something in between? However, in order for an automaton to process an autosegmental representation, it must be encoded in some way on the tape(s) of the automaton, so the job of computing an encoding needs to be done before the automaton can start. Kornai gets around this by defining a pseudo-automaton called an *autosegmental automaton* which can directly scan an autosegmental representation, but this is not the way automata are usually defined. The meaning of his result that an autosegmental automaton can compute his encoding is thus not clear.

The other possibility for refining the definition of computability is to consider the computational complexity of the algorithm. In other words, is the running time of the algorithm proportional to the size of the representation, or to a polynomial function of that size, or even an exponential function of that size? A code that can be computed by a linear time algorithm would be preferred over one that requires a quadratic or exponential time algorithm.

Having said all that, all we are really concerned with for now is that our code *can* be computed. We are more interested in the formal power required to model autosegmental rules.

4.3.2. Invertibility

As Kornai uses the term, the encoding function E_n is invertible (1-1) iff it always assigns different codes to different representations; ie. $A \neq B \Rightarrow E_n(A) \neq E_n(B)$, or, equivalently, $E_n(A) = E_n(B) \Rightarrow A = B$. As Kornai points out, an encoding function that assigns the same code to different autosegmental representations is relatively useless. An invertible code is useful precisely because given $E_n(A)$ one can find A.

The multi-linear code is certainly 1-1; given an encoding, it would be a simple matter to reconstruct the autosegmental representation that it encodes. The vertices and their labels, partitions, and linear ordering would be clear from an inspection of the encoding, and the edges can be reconstructed by a left-to-right matching of pairs of identical α_{ij} symbols from different strings in the encoding.

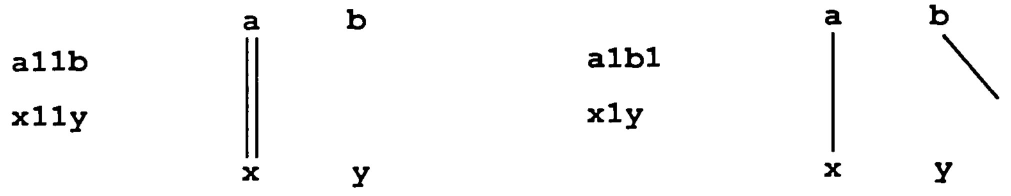
We can formally show that the multi-linear code satisfies this basic definition of invertibility, and at the same time explain in detail the method of decoding the code to find the original representation.

Theorem 4-1: The multi-linear encoding function E_n is invertible (1-1).

Proof: See Appendix A. ❑

There is a greater sense in which a function $f : X \rightarrow Y$ can be invertible. If it is onto as well as 1-1 (*fully invertible*), then for *every* possible element $y \in Y$ we can find a (unique) $x \in X$ such that $f(x) = y$. In terms of our encoding E_n, this would mean that every possible n-tuple in $(\Sigma^*)^n$ would correspond to a valid autosegmental representation. In fact, this is not the case: there are ill-formed encodings in $(\Sigma^*)^n$. There are two characteristics that make encodings ill-formed: "double" association lines and "dangling" association lines.

These two characteristics are illustrated in Figure 4-1.



"Double" association line "Dangling" association line

Figure 4-1: Ill-formed encodings

It could be argued that the criterion of full invertibility is really just a straw man. Any 1-1 function $f : X \to Y$ can be made fully invertible (1-1, onto) by restricting its codomain Y to be just the range f(X); ie. defining $f : X \to f(X)$. Our encoding function E_n can be made fully invertible by defining its codomain to be just the set $C_n \subseteq (\Sigma^*)^n$ of codes that are well-formed. Then we could define the inverse (decoding) function D_n: $C_n \to A_n$ as E_n^{-1}.

There is a problem with ignoring ill-formed codes like this if they actually turn up in practice. In that case, codes would have to be checked for well-formedness before they are decoded. But in fact, a computational tool could be implemented so that all autosegmental representations created by the user are checked for correctness, and all rules are guaranteed, given well-formed codes, to produce well-formed codes (we will say more about this later). Since valid representations would give rise to only well-formed codes, and valid rules would maintain well-formedness, ill-formed codes would never arise, unless there were bugs in the implementation.

In fact, it is easy to check the well-formedness of a multi-linear code by making sure all the α_{ij}'s match up properly (ie. that there are no "double" or "dangling" association lines). In fact this can be done using an MSFA. We will go into more detail in Section 6.1. about how this is done.

4.3.3. Iconicity

The intuitive idea behind iconicity is that minimal changes to a representation should correspond to minimal changes to its encoding. A small change to one part of the representation should not result in changes to the entire encoding. The multi-linear code certainly meets this intuitive criterion.

There are four basic types of changes that can be made to a representation by a rule:

- addition of an association line (edge), or "linking"
- deletion of an association line (edge), or "delinking"
- addition of a segment (vertex)
- deletion of a segment (vertex)

These are the elementary operations of autosegmental phonology. They are simple, and their simplicity is reflected in the corresponding operations that are performed on multi-linear encodings of autosegmental representations. These are:

- insertion of a symbol α_{ij} on each of two strings
- removal of a symbol α_{ij} from each of two strings
- insertion of a symbol $s \in S$ on a string
- removal of a symbol $s \in S$ from a string

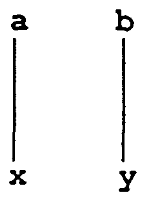
Examples of these operations on autosegmental representations and their encodings are shown in Figure 4-2.

Kornai does not precisely define his criterion of iconicity, but a careful reading of his comments about it reveals that it is closely tied to the information content of autosegmental representations. In information theory, the information content of an object is the number of bits required to represent it. (If there are N objects in a set, the information content of one of them is $\lceil \log_2 N \rceil$, assuming each element of the set has equal probability.[5]) On careful examination, it becomes clear that Kornai's definition of iconicity includes not only the intuitive idea above, but also requires that the length of the encoding be a function of the information content of the representation.

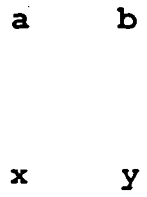
In this alternative sense, the multi-linear code is not iconic. Consider the two representations below, together with their encodings:

(23)

```
a1b1
x1y1
```



```
ab
xy
```



Representations with equal information content

There are a total of 12 valid autosegmental representations with two tiers and two vertices on each tier. Their association patterns all have the same information content: $\log_2 12$. However, their multi-linear codes are not the same length. The fundamental reason for this is that the multi-linear code is based on a unary notation for representing association patterns, rather than a binary notation. The presence of an association line is represented by the presence of two symbols in the code; the absence of an association line is likewise represented by the absence of such symbols. In an optimally iconic code, in Kornai's sense, each possible association pattern, including the two in (23), would be represented by 4 bits. Instead, we use varying numbers of bits to represent them.

5. This brings up two areas in which Kornai's information-theoretic criterion is unclear. First, A_n is an infinite set. Any member of A_n can be considered a member of various different subsets. Kornai picks one — the set of bistrings of length *n* — to talk about, but there are others. This choice will affect the measure of the information content of a particular representation. Second, it may or may not be reasonable to assign equal probabilities to phonological representations in a particular set. It is not clear how these probabilities would be computed. Thanks to Mark Ellison (personal communication) for pointing out this latter issue.

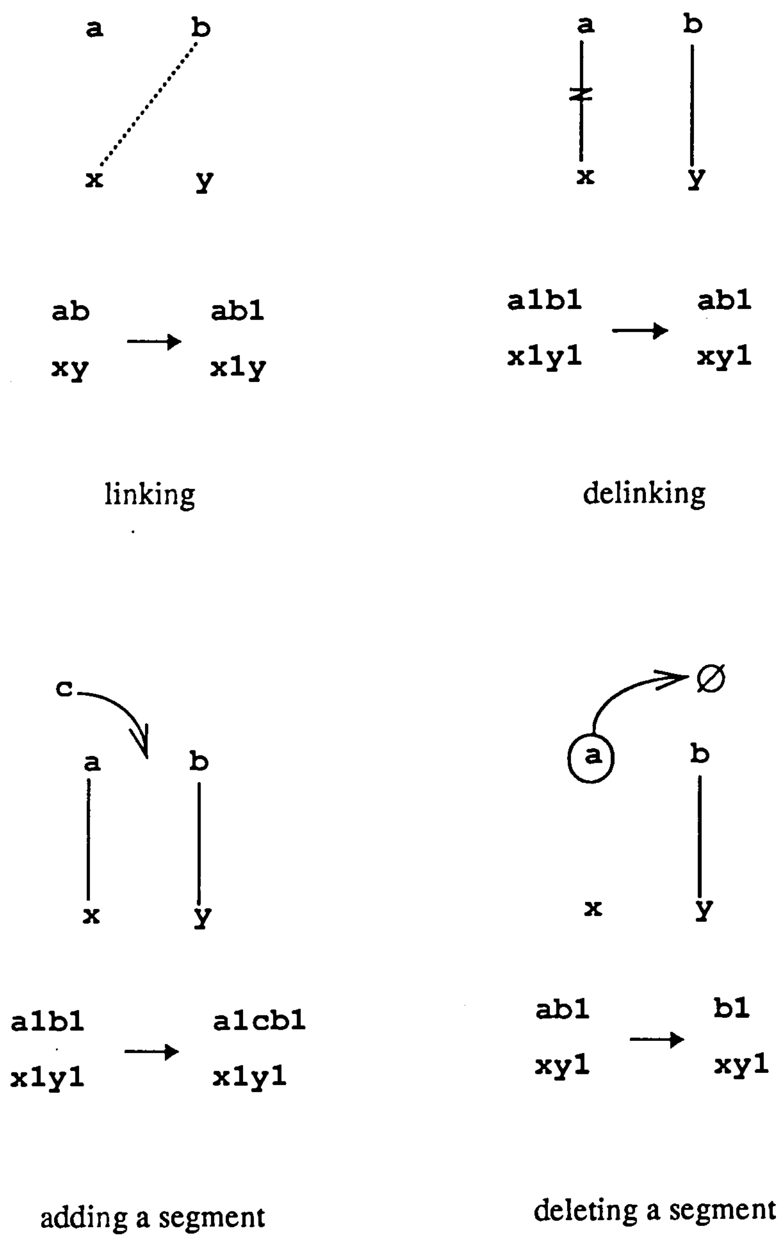


Figure 4-2: Elementary autosegmental operations

This information theoretic aspect of encodings is not as important to us as the perspicuity aspect. Our main concern is that the elementary operations of autosegmental phonology can be simply described in our multi-linear code.

4.3.4. Compositionality

In constructing a computational model of a linguistic theory, our goal will be to produce a model that is as *faithful* as possible to the original theory. Simple operations in one should correspond to simple operations in the other. (We have already seen indications that this is true of our model.) Ideally, every computational operation that is part of the model should be linguistically significant; the model should not contain linguistically irrelevant computational manipulations. This is a standard by which we can measure "goodness of fit" between model and theory. This is in accord with the "Occam's Razor" principle: given two models that adequately describe a theory, the simpler one is to be preferred.

This brings us to the operation of concatenation. This is an operation on autosegmental representations whose simplicity we want to preserve in our model. We want to be able to concatenate, and take apart, the encodings in much the same way as we do the representations. This is the idea behind the compositionality criterion.

The concatenation of representations and encodings is illustrated in Figure 4-2. The concatenation of representations involves concatenating their tiers individually and combining the results into a new representation that retains the old association lines.

Definition 4-6: If A, B $\in A_n$ are autosegmental representations then the *concatenation* of A and B is formed by taking the (set -theoretic) union of the corresponding parts of the representations: the graphs, the vertex partitions, the labelling functions, and the orderings, adding to the ordering the stipulation that the last vertex in each partition of A precedes the first vertex in the corresponding partition of B (together with all the implications that follow by the transitivity of an ordering relation).

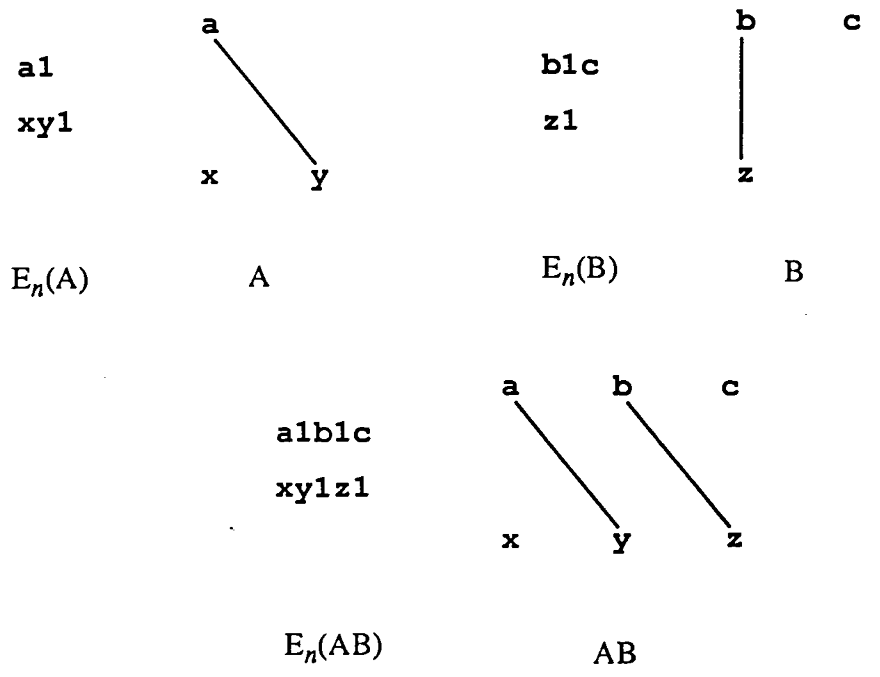


Figure 4-3: Concatenation

Definition 4-7: If E, F $\in (\Sigma^*)^n$ are multi-linear codes, then the concatenation of E and F is formed by concatenating the corresponding members of the n-tuples. So if E = $<e_1, ..., e_n>$ and F = $<f_1, ..., f_n>$, then the concatenation is EF = $<e_1f_1, ..., e_nf_n>$.

Definition 4-8: An encoding function E is *compositional* iff $\forall$ A, B, C $\in$ A_n, we have C = AB $\Rightarrow$ E(C) = E(A)E(B).

Theorem 4-2: The multi-linear encoding function E_n is compositional.

Proof: See Appendix A. ❑

We should point out that concatenating autosegmental representations is, strictly speaking, not a phonological operation. It is an operation of morphology (morphemes are concatenated in the formation of words) and of syntax (words are concatenated in the formation of sentences). In particular, the boundary between morphology and phonology has often been fuzzy and blurred in the literature because there is so much interaction between the two. This is one area where the dividing line needs to be drawn a little sharper.

Having said that, however, we should hasten to add that concatenation is an operation *on phonological representations*. Therefore, a model of phonology ought to facilitate this operation. If a computational model of phonology is built without regard for the models of morphology and syntax that must interface with it, it could unnecessarily complicate what should be a simple operation for those other models. That is why compositionality is an important criterion of a good encoding of autosegmental representations.

We conclude this section with an observation about *linear* encodings of autosegmental representations. Kornai has presented a linear encoding of A_n, in contrast to our non-linear encoding. His goal was to show that autosegmental phonology, as a transformational rule system, requires only the power of a finite state transducer to implement. Since finite state transducers have only a single input tape, this necessitated a linear encoding. We have reservations about Kornai's success in achieving his goal, which we will go into later. But there is another issue that is completely independent of the question of the computational power of autosegmental phonology. From the point of view of compositionality, *linear* encodings are inadequate.

As Kornai has well said, "the absolute minimum we should demand is that [a coding scheme] must be invertible" ([32], p. 23). Given that an encoding function must be 1-1, if it is also *linear* (encodes a representation as a single string), then it is *impossible* for it to be compositional.

Theorem 4-3: An invertible linear encoding of A_n cannot be compositional.

Proof: See Appendix A. ❑

What is the significance of this result? It ends the fruitless search for a compositional linear encoding (given that any encoding must be 1-1). Furthermore, if compositionality is taken to be an absolute requirement, it ends the search for a linear encoding altogether.

Kornai's code is, of course, not compositional. It is "partially" compositional — that is, "it is compositional for all those representations that are associated at both ends" (p. 26). Nevertheless, compositionality is a pass-fail criterion. If for some representations the encoding is not compositional, then every time the operation of concatenation is performed on encodings, the encodings must be checked to see if the operation will actually succeed. If not, some extra bit-twiddling must be done to make sure the outcome of the operation is a well-formed encoding, and that it is the correct encoding. This bit-twiddling would be a linguistically irrelevant operation — a trait that a good computational model of a linguistic theory should avoid (see comments on page 46 above). If you choose to encode autosegmental representations as *linear* strings, you are forced to include these irrelevant operations in your model.

As we said before, concatenation, though not actually not a phonological operation, should be facilitated by a phonological model so that morphology and syntax need not concern themselves with fixing up ill-formed encodings after they concatenate representations. This means choosing a compositional encoding. Therefore, since only a *non-linear* code can be compositional, our choice is made.

Theorem 4-3 is also an improvement on a result of Kornai's. After defining the four criteria of an ideal (linear) code (computability, invertibility, iconicity, and compositionality), he goes on to show that a code that fully meets all four of these criteria simply does not exist. Having done this, he notes that the proof does not depend on the computability criteria at all; hence, it is the other three criteria as a group that are impossible to meet fully. We have further shown that iconicity can be struck from this group, and the invertibility criteria can be weakened (since he defines full invertibiliy as not only 1-1 but also onto), and it is still impossible to find a code meeting these remaining two criteria. These criteria are basic in the sense that any linear encoding of autosegmental representations ought to satisfy them, as we have argued above. It is hard to see how the criteria could be weakened further.[6]

The representations A and B in the proof of Theorem 4-3 embody a fundamental idea: that of the independence of the tiers of autosegmental representations. The representation AAABBB could also be built up by concatenating in a different order: ABABAB or BBAABA or AABBBA, etc. Concatenations on one tier do not affect those on another tier; therefore they can be interspersed in any order. This is precisely the downfall of the compositional linear code. It is exactly this ability to concatenate in different orders from which we derived a contradiction in our proof. This is not surprising: a linear code cannot help but enforce a dependence between the tiers, since they are all encoded in the same string. The only way around this is to separate the tiers into different strings.

Not only should the encoding reflect the independence of the tiers, but so should the computational device that processes the encodings. It is therefore important that MSFAs have independent read heads, one on each tape, rather than a single read head that scans all the tapes. We shall see that this choice between a single read head vs. independent read heads makes a fundamental difference in the computational power of the device. The former gives the device only finite state power, but the latter gives it considerably more power than that.

4.3.5. Modularity

Closely related to the idea of independent tiers is that of independent charts. A pervasive phenomenon in autosegmental phonology is that different charts in a representation are separate and independent, having no phonological effect on one another (unless a rule is specifically constructed to refer to elements from two different charts). This implies that a good computational model should reflect this modularity, or independence of charts, in its encoding.

6. One way of weakening compositionality is to define it as $C = AB \Rightarrow E(C) = E(A)sE(B)$ where s is a kind of boundary marker that is independent of A and B (Kornai calls it a "syncategorematic element"). The proof of Theorem 4-3 would be trivial with this definition of compositionality.

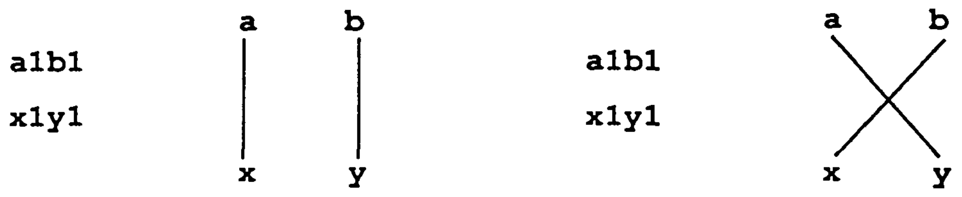
Suppose we wanted to add an association line to a chart. With a linear encoding, we would have to wade through all the material from the other charts to find the proper place to add the link, and the change may have repercussions throughout the whole encoding in the way the other charts are encoded, even though they have not changed. Again, Kornai's triple code (generalized to more than two tiers) has this problem. One needs a non-linear encoding to get around this kind of problem.

4.3.6. Multi-tiered Representations and Redundancy

One of the clearest advantages of the multi-linear code over Kornai's triple code is the ease with which it handles representations with more than two tiers. The triple code works for representations with two tiers, but as Kornai himself admits, the generalization of this code to more than two tiers is very impractical, and contains considerable redundancy (p. 72). Even with two-tiered representations, the triple code is redundant: a single segment may be represented by anywhere from 1 to k symbols, where k is the length of the longest tier. In contrast, the multi-linear code is practical and contains no redundancy at all, for arbitrary n. No matter how many tiers there are, each segment is still encoded as one symbol, and each association line as two symbols.

4.3.7. The No Crossing Constraint

The multi-linear code is incapable of encoding representations with crossing lines. Theoretically, one could apply Definition 4-5 to a representation with crossing lines, as in Figure 4-1, even though such an ill-formed representation is not in the domain of the encoding function E_n. The result would be the encoding of a well-formed representation.



Well-formed representation Representation with crossing lines

Figure 4-4: The No Crossing Constraint enforced

The procedure given in the proof of Theorem 4-1 for inverting the code will never produce a representation with crossing lines. This is because the α_{ij} symbols are matched up in pairs in a left-to-right manner.

This is certainly a desirable property of the multi-linear encoding, from a linguistic point of view. An association line is properly interpreted as indicating the temporal overlap of intervals corresponding to the linked segments [6, 39]. The NCC is then just a statement of the physical reality that if two intervals **b** and **x** overlap in time, it is impossible for any interval **a** preceding **b** to overlap with any interval **y** following **x**. If an encoding is incapable of representing such a situation, then this physical reality is embodied as an integral part of the encoding, rather than imposed as an external constraint. Just as it would be physically impossible for a person to pronounce the second representation in Figure 4-1, it is formally impossible for the multi-linear code to represent that situation.

Chapter 5
Operations on MSFAs

We have defined MSFTs, a special case of MSFAs, and shown how autosegmental representations can be encoded on the tapes of an MSFT. Our goal is to demonstrate that any conceivable autosegmental rule can be modelled by an MSFT. To this end, we will define various operations on MSFAs and show how these operations can be used to combine simple, elementary MSFAs into more complex, composite MSFAs. This approach will allow us to divide the complicated problem of modelling an autosegmental rule into small parts that are easily solved.

5.1. Theoretical Interest

FSAs (and thus FSTs, as a special case) can be combined by a *product* operation. If A_1 and A_2 are FSAs which accept the languages $L(A_1)$ and $L(A_2)$ respectively, then the product $A_1 \times A_2$ will accept the language $L(A_1 \times A_2) = L(A_1) \cap L(A_2)$. The product operation produces an automaton that simulates what happens when the two machines are executed in parallel, both scanning the same input word, and accepting that word only if it is accepted by both automata. Bird and Ellison have used this product operation to advantage in their model of autosegmental phonology [8].

Also, FSTs can be combined by a *composition* operation. If T_1 is an FST that transduces input s_0 to output s_1, and T_2 is an FST that transduces input s_1 to output s_2, then the composition $T_1 \circ T_2$ will transduce input s_0 to output s_2. Kaplan and Kay pointed out the usefulness of this operation for modelling the ordered rules of generative phonology [30]. Kornai followed this path [32], and Koskenniemi began on this path but took a detour that led to a different model [3].

MSFTs are a generalization of FSAs and FSTs. From a theoretical perspective, it is an interesting question in and of itself to ask whether or not these operations of product and composition generalize to MSFTs. We will find that, with a significant qualification, these operations (as well as others) do in fact generalize to MSFTs. We will also find that the product operation proves to be very useful for combining MSFTs like building blocks to yield more complex MSFTs that model autosegmental rules.

5.2. Product

5.2.1. Intuitive Description

Consider the two-tape MSFAs in Figure 5-1. Both A_1 and A_2 accept strings of **a**'s and **b**'s on their two tapes, and each places slightly different restrictions on those strings. Notice that in A_1 there are transitions between almost every pair of states, except that there is only one transition out of q_2, and only one transition into q_4. The effect of this is to ensure that a **b** is scanned on tape 1 iff a **b** is also scanned on tape 2. In other words, the strings on the two tapes must have equal numbers of **b**'s. Similarly, A_2 ensures that the strings it accepts have equal numbers of **a**'s.

Now consider executing A_1 and A_2 simultaneously (in parallel) on the same pair of strings, making transitions in synchronization with each other. Every time A_1 reads from tape 1, A_2 will also read from tape 1. Likewise on tape 2. In fact, we will consider that A_1 and A_2 *share the same read heads* (in the same way that the transducers in Koskenniemi's KIMMO system share read heads [33, 3, 29]). And since on each transition they will be reading the same symbol, the states that they simultaneously enter will both have to be compatible with that symbol. Given this manner of parallel execution of A_1 and A_2, we are interested in whether or not *both* A_1 and A_2 accept a given input.

Two MSFAs executing in parallel like this can be simulated by a single MSFA. For example, the parallel execution of A_1 and A_2 can be simulated by the MSFA $A_1 \times A_2$ in Fig-

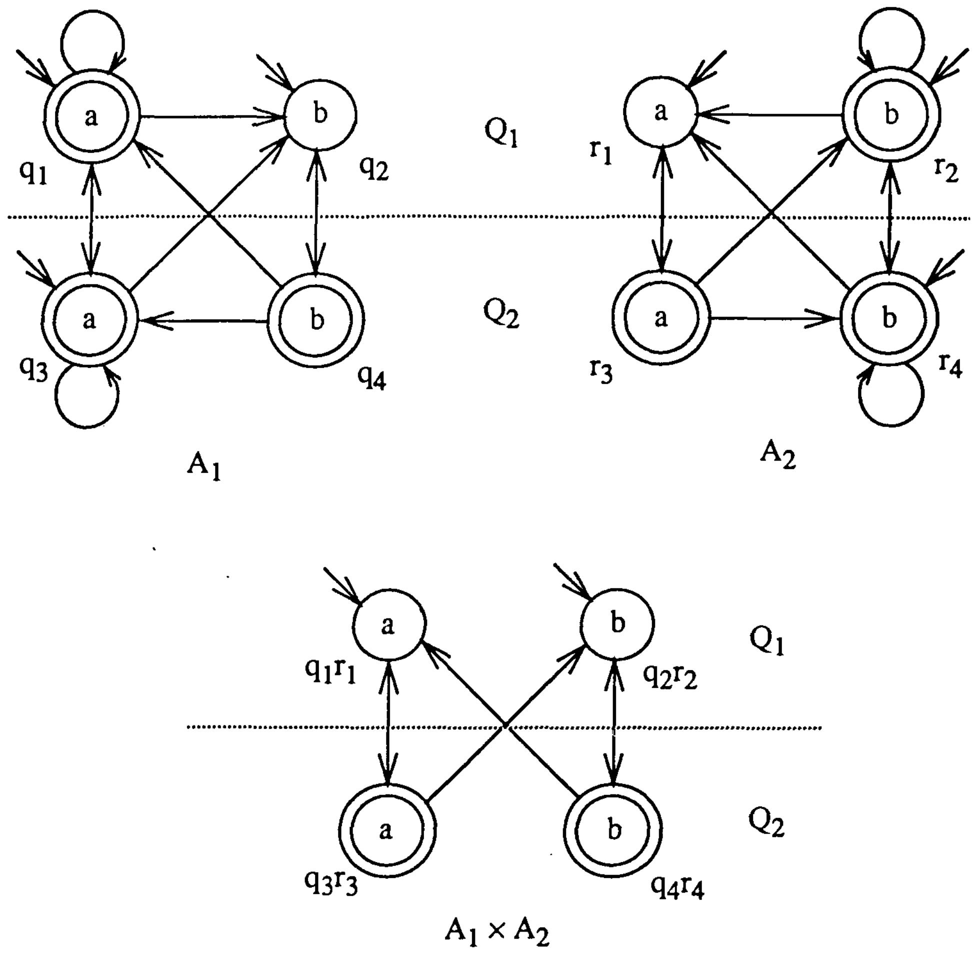


Figure 5-1: Combining MSFAs by the product operation

ure 5-1. (The $\times$ represents the *product* operation on MSFAs, as we will see later.) The technique is to have one state in $A_1 \times A_2$ for each pair of states that A_1 and A_2 could possibly be in at any given moment. For instance, it would be possible for A_1 to be in state q_3 and A_2 to be in state r_3, but not in state r_1 or r_4 (because q_3 and r_1 are in different partitions, and q_3 and r_4 don't have compatible labels). Then, each possible pair of simultaneous transitions in A_1 and A_2 can be simulated by a single transition in $A_1 \times A_2$.

The states of $A_1 \times A_2$ are pairs of states from A_1 and A_2. A transition from q_ir_j to q_kr_l in $A_1 \times A_2$ represents a simultaneous transition from q_i to q_k in A_1 and from r_j to r_l in A_2. A state q_ir_j in $A_1 \times A_2$ is final (initial) iff both q_i in A_1 and r_j in A_2 are final (initial). This means that $A_1 \times A_2$ will accept an input only if both A_1 and A_2 accept it. For example, all three automata accept the input <ab, ab>.

A significant point needs to be brought to attention here. Notice that both A_1 and A_2 accept the input <ab, ba>, but $A_1 \times A_2$ does not. Is something wrong here? It seems like we would want $A_1 \times A_2$ to accept all the inputs accepted by both A_1 and A_2. In other words, we would like our product operation to satisfy $L(A_1 \times A_2) = L(A_1) \cap L(A_2)$, as is the case when A_1 and A_2 are FSAs. However, it turns out that it is impossible, in general, to construct an MSFA that accepts the intersection of two languages accepted by MSFAs. In other words, the class of languages accepted by MSFAs is not closed under intersection [17, 16]. So we cannot hope to define a product operation , or any other operation, that satisfies the above equation. $A_1 \times A_2$ does not accept *all* the inputs accepted by both A_1 and A_2. What *is* true is the converse: Both A_1 and A_2 accept all the inputs accepted by $A_1 \times A_2$.

Underlying all of this discussion is the concept of a *scanning pattern* of an MSFA. Given a particular input, an MSFA scans its tapes in a particular order (eg. first a symbol from tape 2, then a symbol from tape 1, then another symbol from tape 2, etc.). This we refer to as a scanning pattern. It is precisely because A_1 and A_2 do not use the same scanning pattern when accepting <ab, ba> that $A_1 \times A_2$ does not accept <ab, ba>. By simulating A_1 and A_2 in the manner described above, $A_1 \times A_2$ requires not only that both A_1 and A_2 accept the given input, but also that they do it *using the same scanning pattern.*

A scanning pattern of a two-tape MSFA can be represented as a string of 1's and 2's, where a 1 represents scanning a symbol from tape 1, etc. A_1 accepts the input <ab, ba> using the scanning pattern 1122. On the other hand, A_2 accepts <ab, ba> using the scanning pattern 2121. A_1 and A_2 cannot accept <ab, ba> using the same scanning pattern; in fact, A_1 must begin by scanning a symbol on tape 1, whereas A_2 must begin on tape 2.

Interestingly enough, if we extend each input accepted by an MSFA A to include the scanning pattern used to accept it, then we can define the *extended language* $L_e(A)$ accepted by that MSFA, and these extended languages are closed under intersection; in fact, $L_e(A_1 \times A_2) = L_e(A_1) \cap L_e(A_2)$. As a partial example, A_1 accepts, among others, the extended inputs in the set {<ab, ba, 1122>, <ab, ab, 1212>, <ab, ab, 2112>} (notice that <ab, ab> can be accepted using two different scanning patterns). Similarly, A_2 accepts the extended inputs in {<ab, ba, 2121>, <ab, ab, 1212>, <ab, ab, 1221>}. If we intersect these sets, we obtain {<ab, ab, 1212>}. Not only does this intersection operation eliminate <ab, ba> because of the differing scanning patterns, but it also eliminates some <ab, ab> inputs for the same reason. Of all the extended inputs mentioned above, <ab, ab, 1212> is the only one accepted by $A_1 \times A_2$.

5.2.2. Formal Definition

Having considered these examples, let us now define the product of two MSFAs. Recall from Chapter 3 that an MSFA is a septuple $<Q, \Sigma, \lambda, \delta, I, F, e>$ where, in particular, the set Q of states is divided into *n* partitions corresponding to the input tapes, and λ is the labelling function, assigning sets of symbols to each state.

Definition 5-1: Two states q and r in different MSFAs are *compatible* iff each belongs to the i^{th} partition in its MSFA, and $\lambda(q) \cap \lambda(r) \neq \varnothing$.□

Definition 5-2: Let $A_1 = <Q_1, \Sigma, \lambda_1, \delta_1, I_1, F_1, e_1>$ and $A_2 = <Q_2, \Sigma, \lambda_2, \delta_2, I_2, F_2, e_2>$ be two *n*-tape MSFAs. Then the *product* of A_1 and A_2 is the *n*-tape automaton $A_1 \times A_2 = <Q, \Sigma, \lambda, \delta, I, F, e>$, where:

- $Q = \{<q, r> \in Q_1 \times Q_2 \mid$ q and r are compatible$\}$ (pairs of compatible states are states in $A_1 \times A_2$)
- Q is partitioned into $<T_1, ..., T_n>$ where $T_i = \{<q, r>$ in $Q \mid$ q and r are in the i^{th} partitions in A_1 and A_2 respectively$\}$
- $\lambda(<q, r>) = \lambda_1(q) \cap \lambda_2(r)$ (labels are formed by intersection)
- $(<q_1, r_1>, <q_2, r_2>) \in \delta$ iff $<q_1, q_2> \in \delta_1$, and $<r_1, r_2> \in \delta_2$ (transitions in $A_1 \times A_2$ represent simultaneous transitions in A_1 and A_2)

- $I = Q \cap (I_1 \times I_2)$ (pairs of initial states are initial)
- $F = Q \cap (F_1 \times F_2)$ (pairs of final states are final)
- $e = e_1 \wedge e_2$ ($A_1 \times A_2$ accepts the empty input iff both A_1 and A_2 do) ❑

5.3. Disjoint Product

We now turn our attention to another operation on MSFAs called *disjoint product.*

5.3.1. Intuitive Description

Consider the two MSFAs in Figure 5-2. For reasons that will become clear later on when we define the disjoint product operation, we have drawn A_1 and A_2 as *modified MS-FAs*, where the state of the machine before it reads any symbols is represented as an unlabelled state (see Definition 3-5). A_1 accepts strings on tape 1 of the form $(\mathbf{ab})^n$, and A_2 accepts strings on tape 2 of the form $\mathbf{a}^m\mathbf{b}^n$. Notice that A_1 has an empty 2nd partition, and A_2 has an empty 1st partition. They can never read from the same tape; therefore, $A_1 \times A_2 = \varnothing$, the empty automaton (which accepts only the empty input).

Now suppose we wanted to execute A_1 and A_2 in parallel as follows. They would scan the same input, but A_1 would read only tape 1, and A_2 would read only tape 2. They would make transitions completely independently of one another. If both MSFAs accepted the strings on their respective tapes, then the input as a whole would be accepted. If A_1 and A_2 were executed in *disjoint parallel* like this, they would accept inputs of the form $<(\mathbf{ab})^l, \mathbf{a}^m\mathbf{b}^n>$.

Two MSFAs executing in disjoint parallel can be simulated by a single MSFA. For example, the disjoint parallel execution of A_1 and A_2 can be simulated by the MSFA $A_1 \otimes A_2$ in Figure 5-2 (where the $\otimes$ stands for *disjoint product*). The technique is to have several copies of A_1 and A_2 in the disjoint product machine, and to be able to make transitions back and forth between these copies in a way that simulates the simultaneous execution of the two machines.

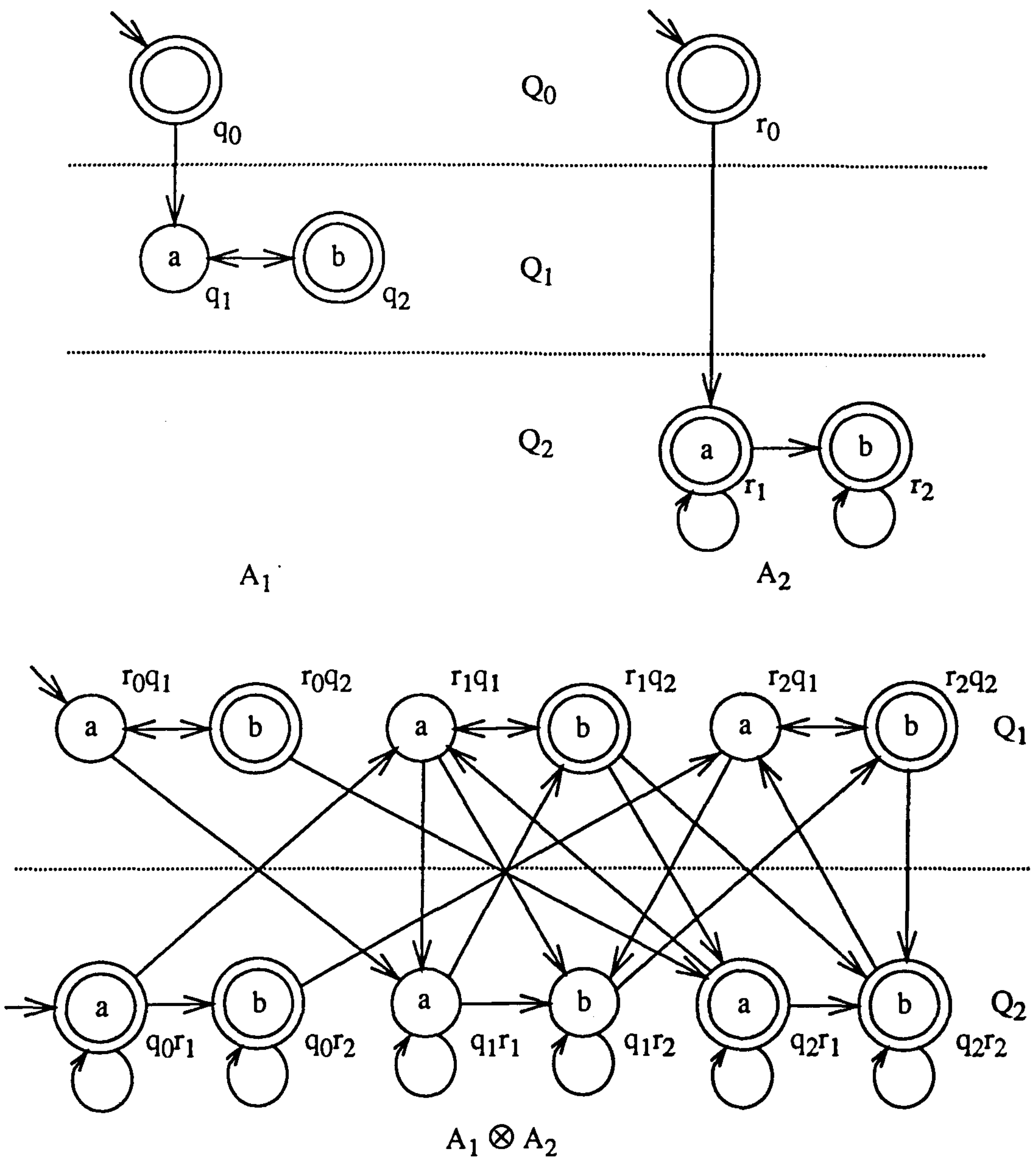


Figure 5-2: Combining MSFAs by the disjoint product operation

In $A_1 \otimes A_2$ there are three copies of A_1 in partition Q_1 and three copies of A_2 in partition Q_2. There is one copy of A_1 for each state in A_2, and one copy of A_2 for each state in A_1. The copies are exact except that states which were initial and/or final in the original may not be the same in the copy. We will see later how this is determined. As well, there are a lot of transitions between the two partitions.

A state r_iq_j in partition Q_1 is a copy of state q_j in A_1. When $A_1 \otimes A_2$ is in state r_iq_j, it is simulating the fact that A_1 is in state q_j and A_2 is in state r_i. Similarly, a state q_ir_j in partition Q_2 is a copy of state r_j in A_2 and simulates the fact that A_1 is in state q_i.

Whenever a transition is made from Q_1 to Q_2, it simulates the fact that A_1 remains in the state it was in, in "suspended animation", while A_2 makes a transition. Similarly, a transition from Q_2 to Q_1 simulates A_2 being "suspended" and A_1 making a transition. To take a concrete example, the transition from q_0r_1 to r_1q_1 represents A_2 being suspended in state r_1 while A_1 makes a transition from q_0 to q_1.

A careful examination of the transitions in Figure 5-2 will convince the reader that they agree with this description. Thus, $A_1 \otimes A_2$ accepts exactly inputs of the form $<(ab)^l, a^mb^n>$.

5.3.2. Formal Definition

Let us make this informal description more rigorous.

Definition 5-3: Two n-tape MSFAs A_1 and A_2 with partitions $<Q_1, ..., Q_n>$ and $<R_1, ..., R_n>$ are *tape-disjoint* iff $\forall\ i = 1, ..., n$, at least one of Q_i and R_i are empty. Two *modified* n-tape MSFAs A_1 and A_2 with partitions $<Q_0, ..., Q_n>$ and $<R_0, ..., R_n>$ are *tape-disjoint* iff they satisfy the same condition (which excludes the special partitions Q_0 and R_0). ❑

We will state the definition of disjoint product on modified MSFAs rather than standard MSFAs, simply because this makes it easier to state.

Definition 5-4: Let $A_1 = <Q_1, \Sigma, \lambda_1, \delta_1, I_1, F_1>$ and $A_2 = <Q_2, \Sigma, \lambda_2, \delta_2, I_2, F_2>$ be two tape-disjoint modified n-tape MSFAs. Then the *disjoint product* of A_1 and A_2 is the n-tape MSFA $A_1 \otimes A_2 = <Q, \Sigma, \lambda, \delta, I, F, e>$ defined as follows:

- Let $Q_1 = \{q_0, ..., q_s\}$ and $Q_2 = \{r_0, ..., r_t\}$ (including the initial unlabelled states q_0 and r_0). Then $Q = Q_1 \times (Q_2 - \{r_0\}) \cup Q_2 \times (Q_1 - \{q_0\})$.

- λ is defined by $\lambda(r_i q_j) = \lambda(q_j)$ and $\lambda(q_i r_j) = \lambda(r_j)$ $\forall$ $r_i q_j, q_i r_j \in Q$.
- δ is defined by the following, $\forall$ i,j,k where $i \geq 0, j,k > 0$:
  - $<r_i q_j, r_i q_k> \in \delta$ iff $<q_j, q_k> \in \delta_1$ (transitions within A_1)
  - $<q_i r_j, q_i r_k> \in \delta$ iff $<r_j, r_k> \in \delta_2$ (transitions within A_2)
  - $<r_i q_j, q_j r_k> \in \delta$ iff $<r_i, r_k> \in \delta_2$ (transitions from A_1 to A_2)
  - $<q_i r_j, r_j q_k> \in \delta$ iff $<q_i, q_k> \in \delta_1$ (transitions from A_2 to A_1)
- $I = \{<r_0, q_i> \mid <q_0, q_i> \in \delta_1\} \cup \{<q_0, r_i> \mid <r_0, r_i> \in \delta_2\}$ (initial states in the 0th copies of A_1 and A_2 are initial)
- $F = Q \cap (F_1 \times F_2 \cup F_2 \times F_1)$ (pairs of final states are final)
- e = true iff both q_0 and r_0 are final ($A_1 \otimes A_2$ accepts the empty input iff both A_1 and A_2 do)

The operations of product and disjoint product are both associative, so we will be able to meaningfully write expressions like $A \times B \times C$ and $A \otimes B \otimes C$. However, we will not assume that the two operations are associative over one another, so in expressions which use both operations we will use brackets to indicate operator precedence.

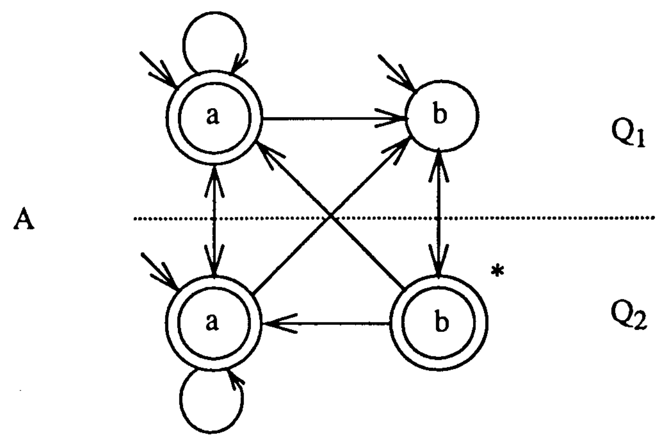
5.4. Other Operations

We need to define two other operations on MSFAs and MSFTs that will be useful in the next chapter.

5.4.1. Collapsing States

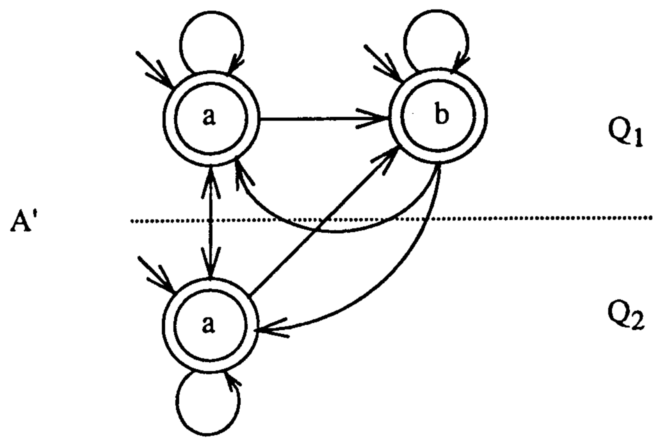
We will informally define the operation of *collapsing* a state in an MSFA. This operation will prove to be useful later on when modelling autosegmental rules with MSFTs — particularly, for example, modelling the deletion of an association line. Consider the following MSFA:

(24)



A accepts pairs of words with equal numbers of b's, interspersed between a's. Suppose we wanted to construct from A another MSFA that accepted pairs of words in which only the first word was allowed to have any b's. We could accomplish this by removing the starred state from A and reconnecting the transition arrows like this:

(25)



We determine where to add transitions by thinking of the starred state as a bus stop at which the bus no longer stops. It now passes right by the former stop and continues along any one of the routes indicated by the transition lines that leave the starred state. Also, because the bus stop used to be an exchange point (the starred state used to be a final state) where passengers could transfer to other lines, the transit company has decided to make all immediately previous bus stop(s) into exchange points (all states with transitions into the starred state become final states).

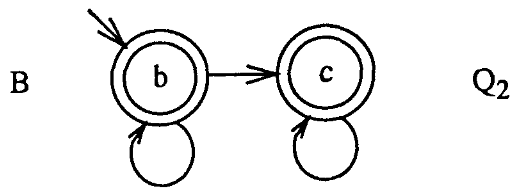
One can *think* of this operation as equivalent to changing the label of the collapsed state to the empty string ε (although MSFAs cannot have strings in their labels, only symbols), so that each time that state is entered, no symbol is read from any tape. The collapse operation produces the MSFA that accepts exactly the inputs that would be accepted if this was the case.

5.4.2. Expanding States

We can also (informally) define the operation of *expanding* a state into an MSFA. This operation is in some sense the opposite of collapsing a state. It can be thought of as equivalent to replacing the label of the expanded state with the set of inputs accepted by a given MSFA, so that when the expanded state is entered, one of those inputs is scanned on the tape(s).

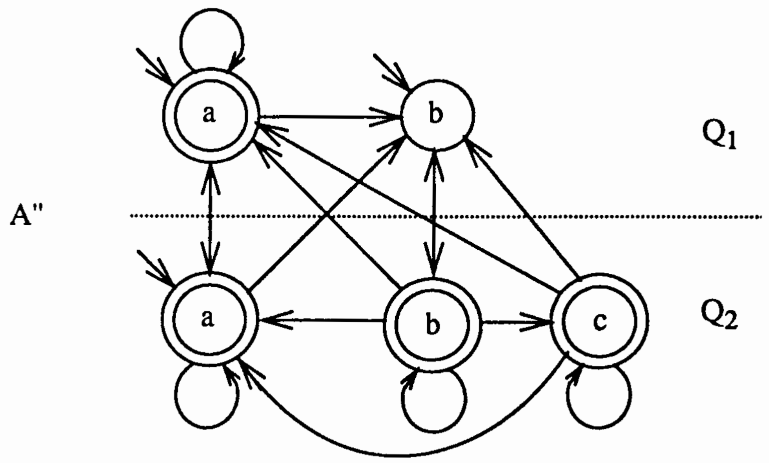
Suppose we start again with the machine A in (24), and expand the starred state into the following MSFA:

(26)



(This can be thought of as improving the bus service in a rapidly developing suburb.) This would produce a new MSFA:

(27)



Each initial state in B (there is only one) receives incoming transitions from all the states that had transitions into the starred state. Each final state in B (both are final) has outgoing transitions to all the states that the starred state did. Finally, the final states of B remain final states in A" iff the starred state was final.

Chapter 6
Modelling Autosegmental Rules

Having defined on MSFAs and MSFTs the operations of product, disjoint product, and collapsing and expanding of states, we now need to show their usefulness in modelling autosegmental rules. We start by showing how to check the well-formedness of a multi-linear encoding with an MSFA. We then take a closer look at some issues in interpreting autosegmental rules. Finally, we take a step by step walk through the procedure for modelling a simple rule.

6.1. Checking Well-formedness

We will begin by considering how MSFAs can be used to check the well-formedness of an encoded autosegmental representation.

Recall that the alphabet of our MSFAs is $\Sigma = S \cup \{\alpha_{ij} \mid i,j = 1, ..., n\}$, where S contains the symbols that represent segments, and the α_{ij} symbol is used to represent association lines between tier *i* and tier *j*. For convenience, let us define the following subsets of Σ:

- $A = \{\alpha_{ij} \mid i,j = 1, ..., n\}$
- for each *i,j*, $\overline{\alpha}_{ij} = \Sigma - \{\alpha_{ij}\}$
- $\overline{S} = \Sigma - S = A$

Also, let us introduce some convenient terminology for charts: chart (*i*, *j*) will refer to the chart containing tiers *i* and *j* (and the association lines between those two tiers).

The three automata in Figure 6-1 contain the essence of all the well-formedness crite-

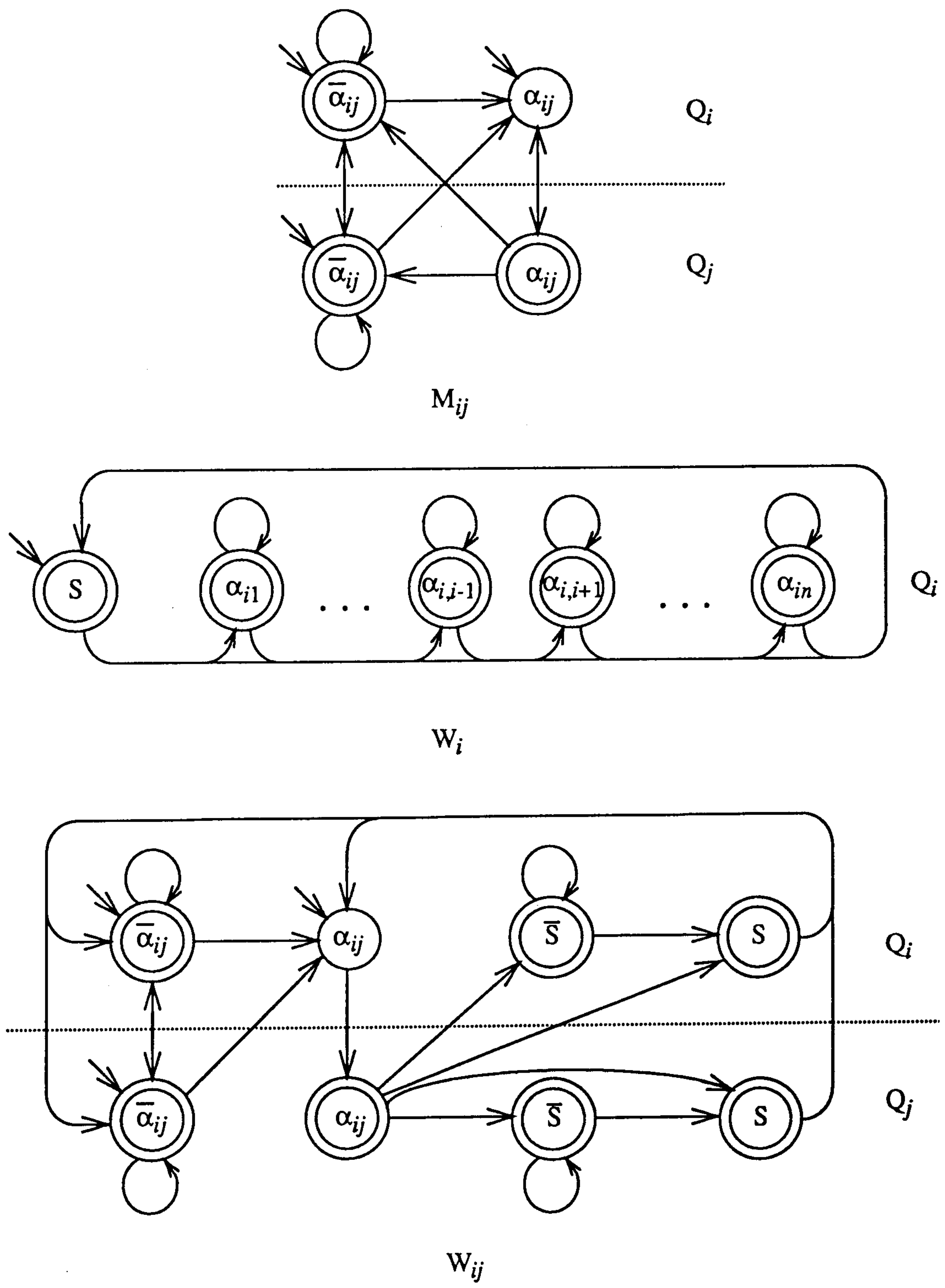


Figure 6-1: Checking well-formedness with MSFAs

ria that a multi-linear code must satisfy. M_{ij} ensures that there are no dangling association lines in chart (i, j) (ie. that the α_{ij} match up in pairs). W_i makes certain that on tier i, the segments and the α_{ij} are in the proper order relative to one another (ie. each tier must begin with a segment symbol, and the α_{ij}'s that follow each segment must be in a certain order). And W_{ij} verifies that there are no double association lines in chart (i, j) (ie. that between every matching pair of α_{ij}'s, there is at least one intervening segment on one of the tiers).

For example, suppose $S = \{W, X, Y, Z\}$. Then M_{ij} would accept the encoding $<W\alpha_{ij}\alpha_{ij}X\alpha_{ij}, Y\alpha_{ij}Z\alpha_{ij}\alpha_{ij}>$ because each tier contains three association symbols. But it would reject the encoding $<W\alpha_{ij}\alpha_{ij}X\alpha_{ij}, Y\alpha_{ij}Z\alpha_{ij}>$ because the top tier has more association symbols than the bottom. W_i would accept $<W\alpha_{ij}X\alpha_{ij}, Y\alpha_{ij}\alpha_{ij}>$ but reject $<\alpha_{ij}WX\alpha_{ij}, Y\alpha_{ij}\alpha_{ij}>$ because an encoding cannot begin with an α_{ij} symbol. Finally, W_{ij} would accept $<W\alpha_{ij}X\alpha_{ij}, Y\alpha_{ij}\alpha_{ij}>$ but reject $<W\alpha_{ij}\alpha_{ij}X, Y\alpha_{ij}\alpha_{ij}>$, because W and Y are double associated.

Actually, M_{ij} is unnecessary, because W_{ij}, in the process of checking for double association lines, also checks for dangling association lines. So W_i and W_{ij} are all we need.

The key observation to make here is that each automaton does a small part of the whole task of checking the well-formedness of an multi-linear code. We are working with n-tiered representations, in general, so even though these automata only have states in one or two partitions, there are n input tapes, and thus n partitions in each automaton. The checks performed by these automata need to be done on each and every chart, or tier, as the case may be. In other words, what we really need is several copies of each of these automata running in parallel on the different charts (or tiers) of the encoding. If they all accept the encoding, it is well-formed. If any one of them does not accept it, it is ill-formed. Thus, what we need to do is combine these automata, using the operations of product and disjoint product, into a single MSFA that simulates all of them executing in parallel.

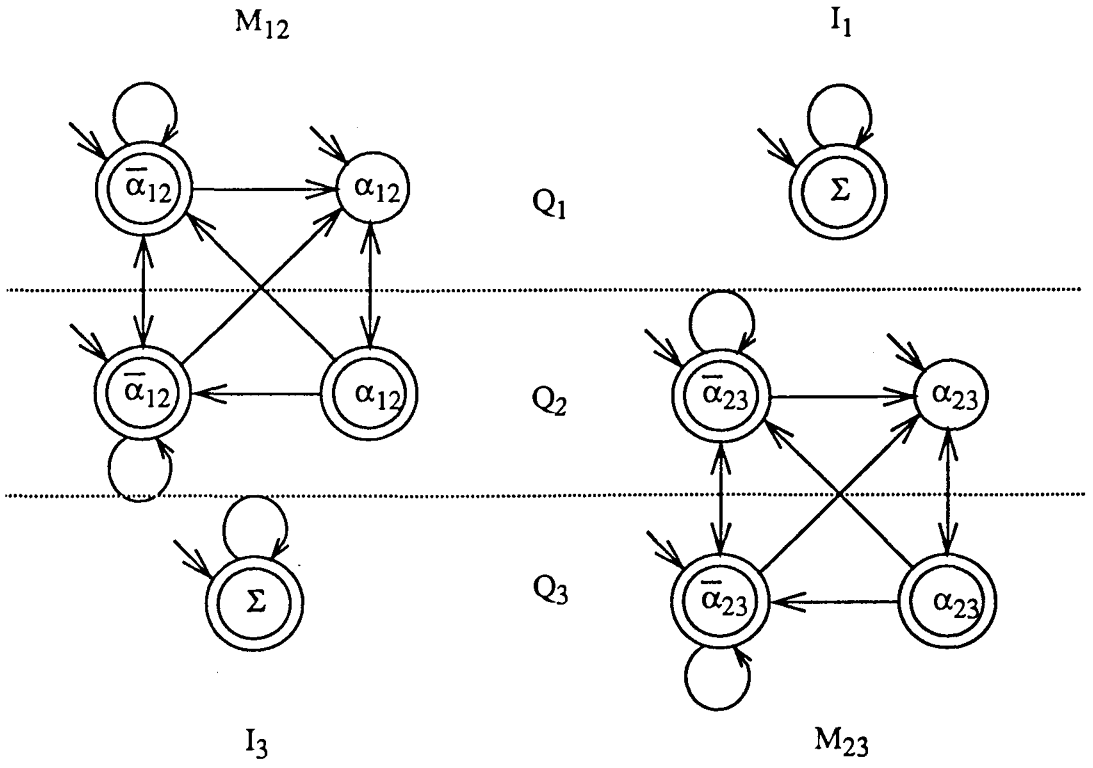
How can we do this? First of all, it is a simple matter to combine the W_i using disjoint product. The MSFA $W_1 \otimes W_2 \otimes \ldots \otimes W_n$ will check all n tiers of an encoding for proper ordering of symbols. However, combining the W_{ij} is not so straightforward. The disjoint product operation is not defined on the W_{ij} because they are not tape-disjoint. And if we take their product, we will end up with the empty automaton.

The reason for this is that each W_{ij} actually has n-2 empty partitions. In the product of the W_{ij}, if a partition is empty in *any* W_{ij}, that partition will also be empty in the product. Since for each partition we can find a W_{ij} which has that partition empty (assuming $n > 2$), the product of the W_{ij} will be the empty automaton.

If an MSFA has an empty partition, it cannot scan any symbols on the corresponding tape. This is not really what we want in the case of W_{ij}. Instead, we need to specify that those tiers can contain *anything* (rather than *nothing*). This is easily done by taking disjoint products with MSFA(s) that accept anything on those tiers. As a simple example, let us suppose we are working with 3-tiered representations ($n = 3$), and take M_{12} and M_{23} from Figure 6-1, as well as two other machines I_1 and I_3[1]:

1. We are abusing notation here by drawing several machines on the same diagram, using names to distinguish them. We will do this in other places as well, for convenience. This could cause confusion with the other interpretation of such a diagram as a single machine. We rely on the assumption (which will be true for the examples we give) that the diagram of each individual machine will be a completely connected graph; this will enable the reader to distinguish separate machines even without the names.

(28)



$M_{12} \otimes I_3$ will accept exactly those encodings with no dangling association lines on chart (1,2), placing no restrictions on tier 3. In other words, if a 3-tiered encoding has equal numbers of α_{12} symbols on tiers 1 and 2, then $M_{12} \otimes I_3$ will accept it, and vice versa. Similarly $I_1 \otimes M_{23}$ will accept exactly those encodings with no dangling association lines on chart (2,3), placing no restrictions on tier 1.

The product, $(M_{12} \otimes I_3) \times (I_1 \otimes M_{23})$, will accept exactly those encodings that are accepted by both $(M_{12} \otimes I_3)$ and $(I_1 \otimes M_{23})$ using the same scanning pattern. Now for multi-tiered representations, there will always be a scanning pattern involving all the tiers by which all the association lines can be traversed in the proper order; this was shown by Kornai ([32], §2.4.3, pp. 72-73). From this it follows that the product $(M_{12} \otimes I_3) \times (I_1 \otimes M_{23})$ can accept all (as well as only) those 3-tiered encodings that have no dangling association lines on charts (1,2) or (2,3). To complete the picture, we would have to add

M_{13}, which checks chart (1,3), yielding the MSFA $(M_{12} \otimes I_3) \times (M_{23} \otimes I_1) \times (M_{13} \otimes I_2)$.

These ideas can of course be extended to $n > 3$. For $n = 4$, the result would be $(M_{12} \otimes I_3 \otimes I_4) \times (M_{13} \otimes I_2 \otimes I_4) \times (M_{14} \otimes I_2 \otimes I_3) \times (M_{23} \otimes I_1 \otimes I_4) \times (M_{24} \otimes I_1 \otimes I_3) \times (M_{34} \otimes I_1 \otimes I_2)$.

It is convenient to state MSFAs without making reference to tiers that are irrelevant to the purpose at hand (as in Figure 6-1). Therefore we will do so, and use the method above to extend the MSFAs as necessary.

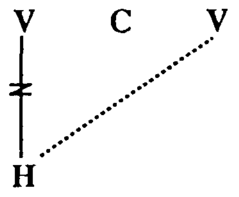
6.2. Understanding Autosegmental Rules

Constructing a rigorous formal model always forces one to examine more closely the thing being modelled to see if it is properly understood. This often leads to the unearthing of issues that were not apparent before. It is so in our case as well: before we can model autosegmental rules, we must understand precisely what they are, what they do, and how they do it.

6.2.1. The NCC

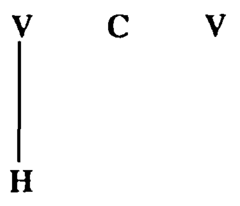
An autosegmental rule is made up of two parts: the structural description (SD) and the structural change (SC). The rule applies to a particular representation only if that representation contains the SD of the rule as a subpart. As an illustration consider the High Tone Shift rule from the Kikuyu language ([22], p. 17):

(29)



The SD of rule (29) is (30):

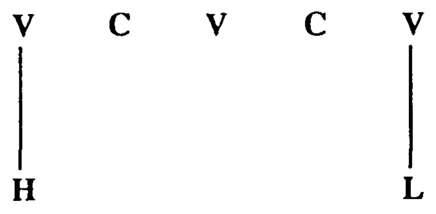
(30)



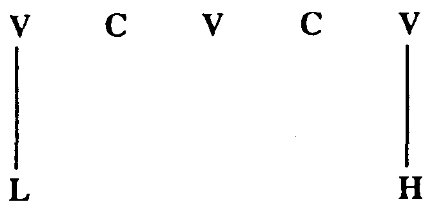
The SC is everything else in the rule; ie. the small 'z' that indicates the deletion of an association line, and the dotted line that indicates the addition of an association line.

It is usually easy to determine whether or not a rule applies to a representation. For example, it is clear that rule (29) applies to (31) but not to (32):

(31)

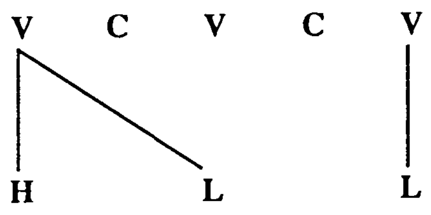


(32)



However, there are some things that are not so clear. Does the rule apply to (33)?

(33)



Although this representation contains the SD of the rule, notice that if the rule were applied, it would add an association line that crosses a line already present in the representation. This brings up the general question of whether or not a rule should apply if it creates a line-crossing situation.

As Bird and Ladd point out [7], Goldsmith is not consistent on this issue in *Autosegmental and Metrical Phonology*; he treats the NCC differently for different types of rules. In the case of the Association Convention and rules of unbounded spreading, the NCC is interpreted as blocking the addition of associations that would create line-crossing situations. In describing a rule of unbounded spreading on p. 30, he says:

> ...the autosegment will be associated with all unassociated accessible segments on the opposite tier...and so will be associated to all those unassociated segments to which it can link without crossing any association lines.

Thus, a segment on the opposite tier is only accessible to another segment if they can be associated without causing lines to cross [7]. And on page 47, Goldsmith says:

> The Association Convention and rules of unbounded spreading will not create line-crossing situations, since they are absolutely conditioned to affect only pairs of unassociated segments on tiers of a chart.

Actually, this is not quite correct; he should have said "pairs of unassociated *and accessible* segments".

The importance of this interpretation of the NCC to the correctness of his analysis of Sukuma verbs on p. 16,17 is duly noted:

> The notable result is that of (21d), where a High tone remains unassociated because there is no unassociated [*and accessible*] vowel for it to associate with.

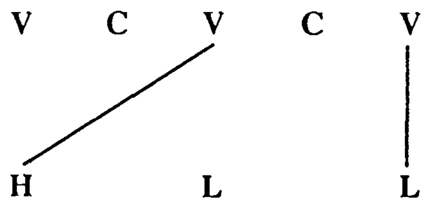
In contrast, Goldsmith says of language-particular rules on p. 47:

...If a rule is formulated to add a single association line, it can, in principle, cause a line-crossing situation. In this case, ...the line that the rule adds remains, but the line that formerly existed is taken to be the offending line and is automatically erased.

In this case, then, the NCC does not block the addition of crossing lines, but rather repairs ill-formed representations by "erasing offending lines".

Under these assumptions, the rule (29) *would* in fact apply to (33), and produce (34).

(34)



If we take a moment to reflect on the implications of Goldsmith's assumptions, we realize that they add to autosegmental rules a powerful mechanism — the ability to repair ill-formed representations. Suppose we have a rule that adds an association line between two segments A and B, and consider a representation that matches the SD of this rule. If A and B are a considerable distance apart in the representation, then this rule has the potential to wipe out all association lines in its path between A and B. This kind of extra power could get out of hand, allowing rules to do things they were never intended to do, which would at the very least force the phonologist to be cautious when writing rules.

In order to interpret the NCC consistently, and to avoid the possible danger of phonological rules with runaway power, we have chosen to block a rule from applying if it would create a line-crossing situation. The next section lends support to this conclusion.

6.2.2. Limits on the Number of Associations Per Segment

The NCC is one well-formedness constraint[2] to which Goldsmith grants the power to repair ill-formed representations. He extends this kind of thinking to another well-formedness constraint. Each tier has a minimum and maximum number of associations allowed per segment. Possible violations of this condition are discussed on p. 18, 19:

> ...it is incumbent on us to explain, therefore, what happens if the number of tones on a vowel should exceed the maximum, or not reach the minimum.

If the maximum is exceeded during application of a rule:

> ...the tone assigned by the rule is maintained, but the earlier tone is dissociated.

This repair strategy is well-defined if the maximum is 1, but suppose the maximum is 2. In this case, if a rule associates a tone to a vowel that already has two tones, one of these two tones must be dissociated, but which one? The phrase "the earlier tone" is ambiguous. Even if we were to make it more precise, what criterion should we use to choose between the two earlier tones? The only reasonable criterion seems to be "the outermost tone on the side where the new tone is associating". Whether this criterion is supported by real language data has yet to be determined.

Furthermore, what repair strategy are we to use if a rule causes a vowel to have less than the minimum allowable number of tones? Suppose the minimum is 1, and that a rule dissociates the last vowel from a tone. Then we need to immediately find a tone to associate to this vowel. Should we reassociate the one that was just dissociated? This would be equivalent to blocking the rule, if it does nothing else besides the dissociation. The choice of any other tone would be arbitrary at best, and capricious at worst.

6.2.3. Summary

Granting well-formedness constraints the power to repair ill-formed representations is

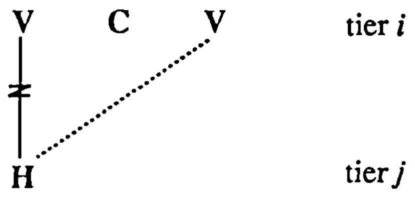
2. Our use of the term "well-formedness constraint" is not to be confused with Goldsmith's Well-Formedness Condition (WFC).

clearly problematic. It necessitates more clearly defined repair strategies than have yet been proposed (we offered a few crude first attempts above). Also, it forces an inconsistent interpretation of the No Crossing Constraint. Finally, it is more difficult computationally. The alternative, a consistent interpretation of all well-formedness constraints as blocking rules that would create well-formedness violations, is computationally straightforward within the model we have chosen.

6.3. Modelling an Autosegmental Rule

There are several steps in the process of constructing an MSFA that models an autosegmental rule. We will demonstrate this process for the High Tone Shift rule, repeated here for convenience:

(35)



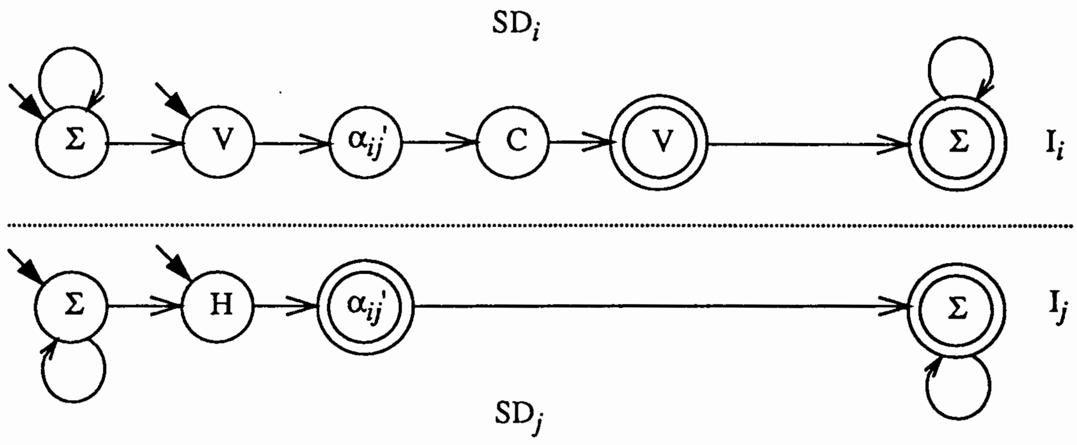
Up to this point, we have represented all association lines using α_{ij} symbols, one for each chart. But there are (at least) three *different types* of association lines relative to an autosegmental rule: (1) those that are part of the input representation but are *not* explicitly mentioned in the rule, (2) those that are part of the input representation and *are* explicitly mentioned by the rule (including those that will be deleted by the rule), and (3) those that are added by the rule. In what follows, we will differentiate between these three types of links (which we briefly refer to as *background*, *foreground* and *added* lines), using the symbols α_{ij}, α_{ij}', and α_{ij}'' respectively in the labels of MSFA states (the alphabet Σ will be replaced by the larger alphabet $\Sigma' = \Sigma \cup \{\alpha_{ij}', \alpha_{ij}''\}$). This distinction will later be done away with. The reason for it will become clear as we go along.

Think of all three of these symbols as being the same when it comes to matching encodings (ie. they all match an α_{ij} symbol in an encoding), but different as far as MSFAs are concerned (the distinction will be maintained as we combine various MSFAs in the process of constructing an MSFA that models this rule). This dual interpretation will be justified in the end, because in the last step of the process, all state labels containing α_{ij}' and α_{ij}'' will be modified so that they only contain α_{ij}.

6.3.1. Matching a Structural Description

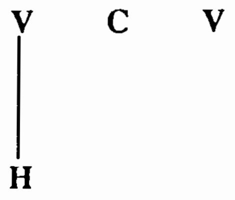
We start with the following two automata:

(36)



The basic purpose of these automata is to verify that the encoding on the input tapes contains the SD of the High Tone Shift rule:

(37)

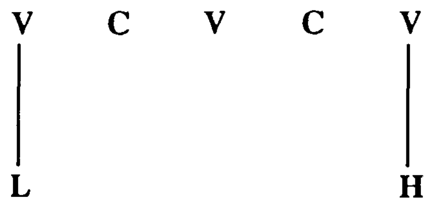


This SD can be expressed as a pair of regular expressions $R = \langle\Sigma^* V\alpha_{ij}'CV\Sigma^*, \Sigma^* H\alpha_{ij}'\Sigma^*\rangle$, together with the stipulation that the two α_{ij}' symbols in R are a pair; ie. they refer to the *same association line.*

$SD_i \otimes SD_j$ accepts all encodings that match R. As it accepts an encoding, it makes a distinction between foreground and background association symbols by virtue of which state it uses to scan an association symbol. If it uses a state labelled α_{ij}', it is interpreting the α_{ij} symbol being scanned as part of a foreground line. If it uses a state labelled Σ (which contains α_{ij} but not α_{ij}'), the α_{ij} symbol being scanned is being interpreted as part of a background line.

But there is a problem. Consider representation (32), which we repeat here for convenience:

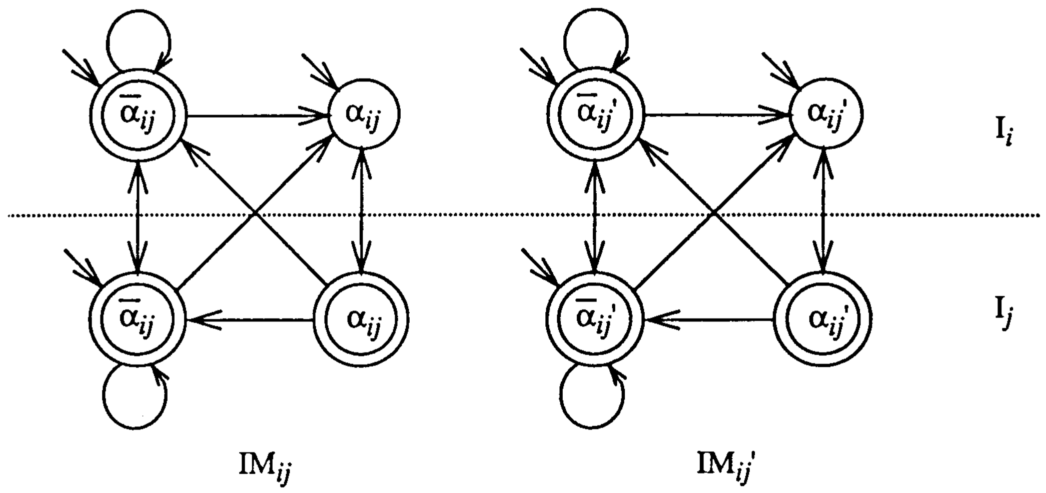
(38)

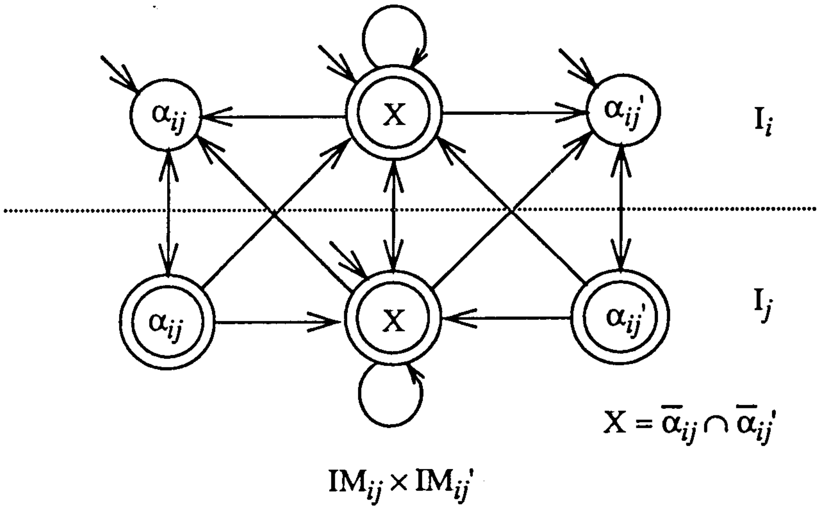


This representation would be encoded as $<V\alpha_{ij}CVCV\alpha_{ij}, L\alpha_{ij}H\alpha_{ij}>$. $SD_i \otimes SD_j$ accepts this encoding, because it does match R. But rule (35) does not apply to representation (32). The problem is that the α_{ij} symbols in the encoding that are interpreted as parts of the foreground line in the rule do *not* correspond to the *same association line* in the encoding.

This problem is easily solved using the product operation and the machines IM_{ij} and IM_{ij}' below, taken from Figure 6-1 (recall that the bar notation means set complement):

(39)




α_{ij}
X
α_{ij}'
I_i
α_{ij}
X
α_{ij}'
I_j
$X = \bar{\alpha}_{ij} \cap \bar{\alpha}_{ij}'$
$\mathrm{IM}_{ij} \times \mathrm{IM}_{ij}'$

IM_{ij} and IM_{ij}' scan association symbols in pairs. IM_{ij}' scans foreground association symbols, and IM_{ij} scans background association symbols. Their product will scan in pairs *all* the association symbols in an input encoding, ensuring that each pair corresponds to an association line. In other words, once an α_{ij} or α_{ij}' is encountered on one tape, it must be encountered on the other. The states labelled X will accept any number of symbols which are neither α_{ij} nor α_{ij}'.

Thus, in order to verify that an input encoding matches R and that the association symbols that are interpreted as parts of foreground lines actually do form pairs in the encoding, we need the automaton $SD_{ij} = (SD_i \otimes SD_j) \times (IM_{ij} \times IM_{ij}')$.

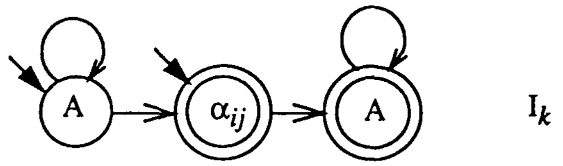
A second point needs to be made about the α_{ij} symbols. Consider again the SD in (37). What precisely does it say about the representation to which the rule can apply? It says that certain segments must be present on the two tiers, and that some of those segments must be associated in certain ways. Notice, however, that the segments may have *other* associations (within this chart, not to mention associations to other tiers) that are not mentioned by the SD. As Goldsmith says on p. 39:

> If an autosegment is multiply associated while a rule mentions only one of the lines in its structural description, will the rule apply? ...If the rule changes or deletes the autosegment(s) in question, then the rule will not apply; ...If the function of the rule is to add or delete an association line, then the rule will apply in any event.

(This condition on the application of rules is referred to as the Conjunctivity Condition, and will come up again in this chapter.) We will come back to the question of operations that change or delete autosegments, but the High Tone Shift rule has no such operations (it only operates on association lines). So any of the segments mentioned in rule (35) may have other association lines besides those mentioned explicitly in the SD. This includes segments like the second V which have *no* associations mentioned by the SD. (If a rule needs to specify that a segment has no associations at all (within the chart(s) explicitly mentioned by the rule), then the segment must have a circle around it.) This, then, means that our automaton SD_{ij} is incomplete. It does not allow for other association lines.

The following MSFA accepts an α_{ij} symbol preceded and followed by any number of other association symbols:

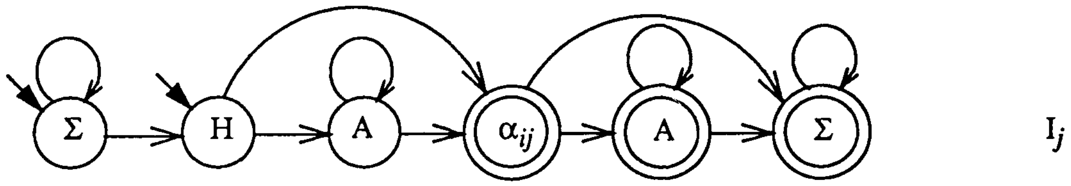
(40)



A_{ij}

Each state in SD_i and SD_j labelled with an α_{ij}, needs to be expanded into the MSFA A_{ij} so that these other association lines can be scanned (recall the definition of the expansion operation in Section 5.4.2.) For example, SD_j would become:

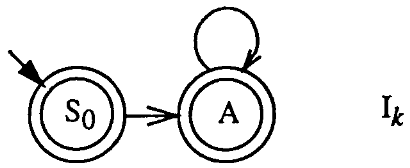
(41)



SD_j

Also, the following MSFA accepts a segment S_0 followed by any number of other association symbols:

(42)



S_0

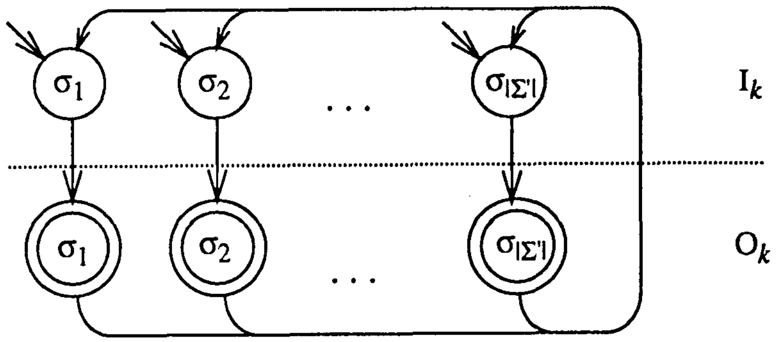
In the same manner, each state in SD_i and SD_j labelled with a segment symbol (in this case, V, C, or H) must be expanded into the MSFA S_0, where the label S_0 is replaced by the appropriate segment symbol. If, in the SD, the particular segment had a circle around it, indicating that this segment had no associations on chart (i,j), then we could also change the label A in S_0 to A - $\{\alpha_{ij}\}$.

Given that we have this mechanical procedure for expanding automata to account for extra association lines, we will proceed to draw all our diagrams unexpanded, since they are much easier to read that way.

6.3.2. Inserting and Deleting Segments and Association Lines

Recall our automaton $SD_{ij} = (SD_i \otimes SD_j) \times (IM_{ij} \times IM_{ij}')$. Now so far SD_{ij} only deals with the input encoding. We must also produce an output encoding. The first step towards this is the identity transduction MSFA:

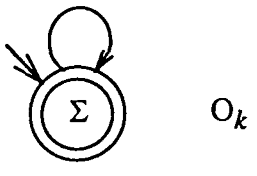
(43)



ID_k

This is the identity transduction machine for tier k, with two states for each symbol in the alphabet $\Sigma' = \{\sigma_1, \sigma_2, ..., \sigma_{|\Sigma'|}\}$. It reads any string on the k^{th} input tape and writes it on the k^{th} output tape. Now SD_{ij} so far only reads from input tapes (i and j), so we start by factoring in a machine that scans/prints anything on an output tape:

(44)



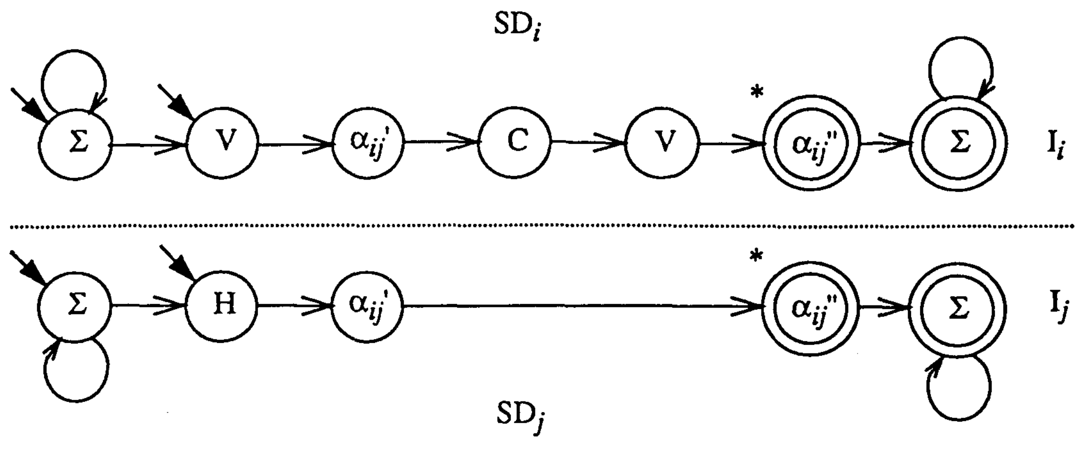
$O\Sigma_k$

$SD_{ij} \otimes O\Sigma_i \otimes O\Sigma_j$ reads an input encoding that satisfies the SD of the High Tone Shift rule, and allows anything at all on the output tapes. Now that we have a machine whose ith and jth input *and* output partitions are non-empty, we can factor in the identity transduction on each tier. Then we have our first MSFA approximation to the High-tone Shift rule, $HS_0 = (SD_{ij} \otimes O\Sigma_i \otimes O\Sigma_j) \times (ID_i \otimes ID_j)$. This machine reproduces on the output tapes any input encoding that satisfies the SD. If the input encoding does not satisfy the SD, the machine fails.[3]

Now HS_0 is closer to what we want, but of course we have done nothing yet about deleting and inserting association lines. How do we do this? The beginning of the answer is to go back to square one and change SD_i and SD_j (which we first introduced in (36)) so that the association line that is added becomes part of the SD, as shown in (45). This may seem a little strange, but bear with us for the moment :

3. We will say more about the machine's failure once we have finished showing how it simulates a rule.

(45)



The basic idea is this: HS_0 will reproduce this extra association line on the output tapes. To get a machine that will insert and delete association lines (or segments), we collapse states in HS_0 (recall the definition of the collapse operation in Section 5.4.1.). If we want a segment or an association line to be inserted (as in our example), we collapse the state(s) that scan it on the input tape. If we want a segment or association line to be deleted, we collapse the state(s) that write it on the output tape.

We have introduced the third type of association line, the *added* line, into our automaton SD_{ij}. Accordingly, we will factor in another machine IM_{ij}'', which scans α_{ij}'' symbols in pairs. It is easy to extrapolate from (39) and see what $IM_{ij} \times IM_{ij}' \times IM_{ij}''$ would look like: it would scan all three types of association symbols in pairs. We redefine SD_{ij} (and therefore HS_0) as $SD_{ij} = (SD_i \otimes SD_j) \times (IM_{ij} \times IM_{ij}' \times IM_{ij}'')$. This will be important when we discuss the NCC below.

In our example, there are two starred states in SD_i and SD_j that correspond to the added association line. We have said that in order to simulate the insertion of that association line, we must collapse the states in HS_0 that scan it on the input tapes. But HS_0 has many more and different states than SD_i and SD_j. The question is, which states in HS_0 correspond to the added association line?

The answer is that one must keep track of which states correspond to the added association line (ie. are "starred") as the product and disjoint product operations are performed. Both of these operations produce machines whose states are pairs of states from the factor machines. So the answer is quite straightforward: in a *product* machine, state-pairs which contain a starred state in *either* position will also be starred states; in a *disjoint product* machine, state-pairs which contain a starred state as the *second* element of the pair will also be starred states (see Definition 5-4). This means that there will be many starred states in HS_0 corresponding to a single starred state in SD_i or SD_j.

We need to notice two things about the way we have simulated a phonological rule with an MSFT. The automaton searches (non-deterministically) in the input encoding for exactly one occurrence of the SD of the rule, and either changes the encoding at the point where the SD is matched, or fails if it cannot match the SD. This is not consistent with the usual interpretation of phonological rules on two counts: (1) A rule ought to apply at *all* points where it can match the SD, not just one, and (2) if the SD cannot be matched, the rule should have no effect (ie. output the input encoding unchanged) instead of rejecting the input.

Let us briefly mention two possible strategies for solving these problems. One strategy would be to add a meta-level mechanism above the rule transducers which feeds input encodings into them and executes them. This mechanism could detect transducer failure and act accordingly, and could also reapply a rule to its own output until all possible applications of the rule are exhausted. This, however, adds extra computational power to the overall model. An alternative strategy is to construct, beginning with the automata we have presented, more complicated rule automata which can perform the structural changes specified by the rule in each place that the SD can be matched, which means performing the identity transduction if the SD cannot be matched.

6.3.3. The NCC Revisited

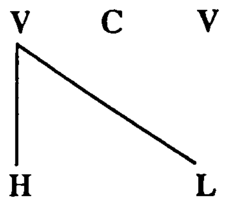
HS_0 enforces the NCC. To see why this is so, consider the following observations.

First, the factor ($ID_i \otimes ID_j$) imposes the scanning pattern of the input tapes onto the output tapes. In other words, the output tapes must be scanned with the same pattern as the input tapes. This is because each symbol that is read on the input tape is immediately written on the corresponding output tape before the next input symbol is read.

Second, when any pair of association symbols is scanned, the two symbols are scanned *in immediate succession.* No other input symbol can be scanned in between. This means that there will be a point during the execution of HS_0 when the input tape heads will *simultaneously* rest on both of the association symbols in that pair.

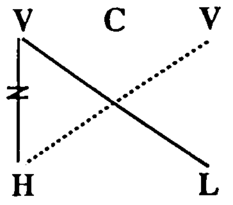
Because this is true on the output tapes as well as the input tapes, it would be impossible for HS_0 to add an association line that created a line-crossing situation. For example, given the following input representation:

(46)



HS_0 could not produce the following output:

(47)



This output representation would have the encoding $<V\alpha_{ij}CV\alpha_{ij}", H\alpha_{ij}"L\alpha_{ij}>$, where we have indicated the interpretations that HS_0 would have to place on the various association symbols. According to our previous arguments, the output tape heads would at one point have to rest simultaneously on the α_{ij} pair, and at another (earlier or later) point, rest simultaneously on the α_{ij}'' pair. This is impossible, because MSFAs cannot scan backwards, which is what HS_0 would have to do to perform this feat.

Therefore, HS_0 would fail when trying to apply the High Tone Shift rule to representation (46). This is equivalent to the blocking of the rule, which, as we have already argued, is the most reasonable course of action when a rule is faced with the option of creating a line-crossing situation.

6.3.4. Well-Formedness Constraints Revisited

When an autosegmental rule applies to a representation, its output must conform to certain well-formedness constraints. There are three of these that could be violated:

- no "double" association lines
- the Conjunctivity Condition
- upper and lower limits on the number of associations per freely associating segment

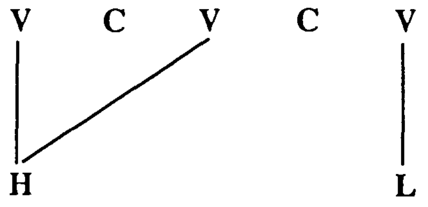
Accordingly, we need to make sure that HS_0 enforces these well-formedness constraints on its output. We will present two automata, OW_{ij}, and LA_{ij}, which scan the output tapes and enforce the first and last constraints. We also describe a different technique for enforcing the Conjunctivity Condition. We can then define $HS_1 = HS_0 \times (I\Sigma_i \otimes I\Sigma_j \otimes (OW_{ij} \times LA_{ij}))$, which will incorporate these constraints into our MSFT model of the High Tone Shift rule.

Some well-formedness constraints do not need to be enforced. These include 'no "dangling" association lines' and 'proper ordering of symbols on the tiers'. The reason these don't need to be checked is that that they will never be violated. If we assume that we start with well-formed encodings, our rule automata are constructed in such a way as to maintain these kinds of well-formedness; they never attempt to write symbols on the output tapes in an improper order, or to write unequal numbers of association symbols on facing tiers. We only need to check well-formedness constraints that stand in danger of being violated by the application of a rule.

6.3.4.1. Double Association Lines

Consider the following representation:

(48)



This representation contains the SD of the High Tone Shift rule at its leftmost end. But does the rule apply here? The association line that the rule would add at this point is already present. Again, this is an issue that Goldsmith does not consider, so we can only offer suggestions. The alternatives are (1) the rule does not apply here because the association line is already present, (2) the rule applies in any case, and deletes the first association line, but does not add an association line.[4]

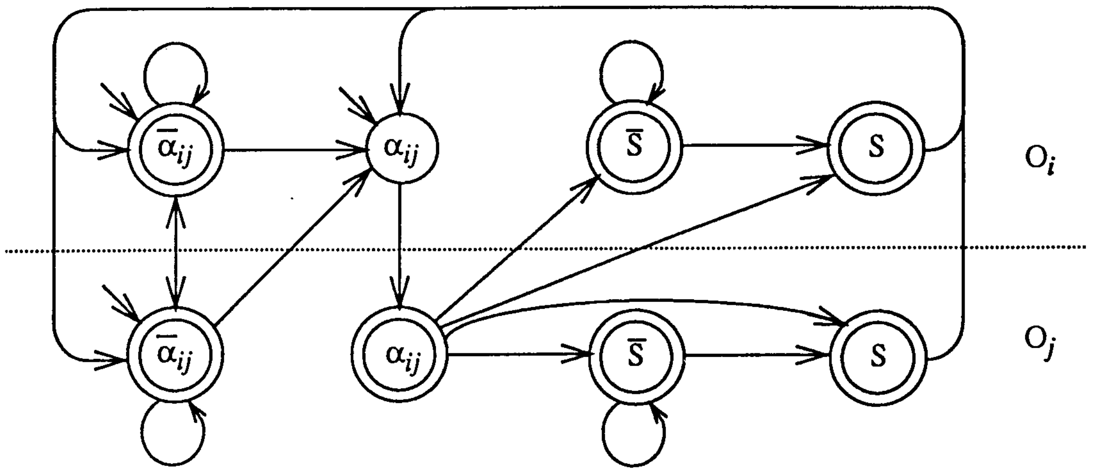
The first of these alternatives seems the most natural for two reasons. First, option (2) involves partial application of a rule: part of the rule takes effect and part does not. This concept is not found in the literature; a rule either applies completely or it doesn't apply at all. Also, partial application of the High Tone Shift rule does not accomplish the purpose for which the rule was written, namely, to shift the association of the high tone from one vowel to the next. In other words, the deletion and the insertion are an integral package that cannot be divided up without defeating the purpose of the rule. In general, it seems

4. We assume that autosegmental representations cannot have double links, which eliminates the possibility of a third alternative, namely, adding the association line anyway, resulting in a double association line. Double association lines are only possible in the *encoding* of an autosegmental representation. Multiple association lines in the representations themselves would have undesirable empirical consequences such as having some segments more strongly associated than others (ie. requiring more than one delinking to dissociate them).

that for any rule, the possibility of partial application would allow the rule to apply in situations where it does not accomplish its purpose. Therefore we will choose option (1) for our model.

We have already seen in Figure 6-1 the automaton W_{ij} that prevents double association lines. OW_{ij} is just the version of this machine that operates on the output tapes:

(49)



OW_{ij}

This factor OW_{ij} ensures that the output encoding will have no double association lines. Therefore, the rule will be blocked from applying in situations where it would add an association line between segments that are already associated.

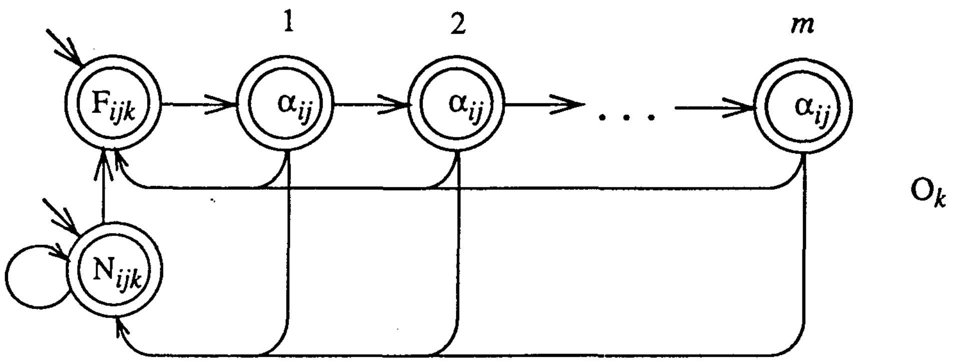
6.3.4.2. Limits on the Number of Associations Per Segment

For each chart (i, j), and each tier k within that chart, there is a subset $F_{ijk} \subseteq S$ designated *freely associating segments*. Only these segments are allowed to have association lines in chart (i, j). And there are upper and lower limits on how many association lines they may have (although these limits may be ∞ and 0 respectively).

Correspondingly, on each tier we could define $N_{ijk} = S - F_{ijk}$, the set of all non-associating segments on tier k in chart (i, j).

LA_{ij} will be the automaton that scans the output tapes and enforces limits on the number of associations per segment. Here is the version of LA_{ij} in the case where the minimum is 1 and the maximum is m:

(50)



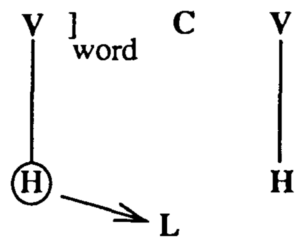
If there was no upper limit on the number of associations, then we could delete the states numbered 2 through m, and put a self-loop on state 1. If the lower limit was higher (say $l > 1$), then there would be no transitions from states 1 through l-1 going back to the left. If there was no lower limit (ie. it was 0), there would be a self-loop on the state labelled F_{ijk} and a transition from it to the state labelled N_{ijk}.

Keep in mind that all the states in this machine need to be expanded as discussed earlier to allow for association lines from this tier to other charts besides (i, j).

6.3.4.3. The Conjunctivity Condition

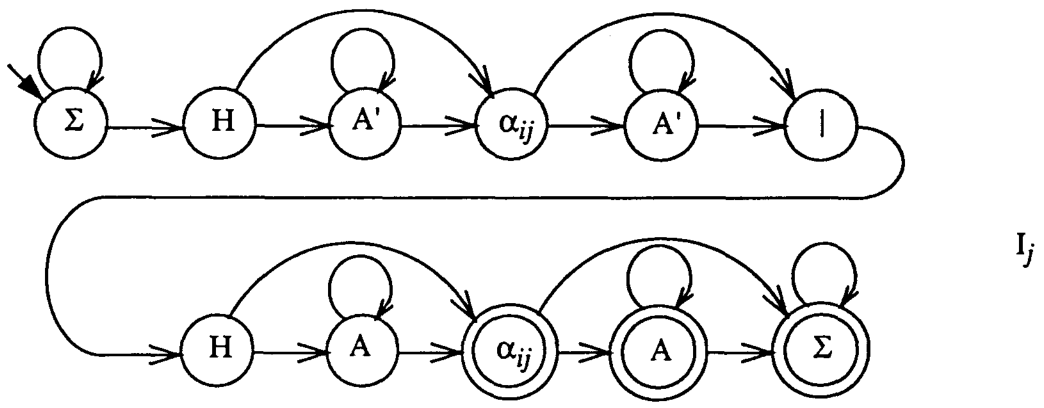
Recall from page 79 that the Conjunctivity Condition states that a segment cannot be modified or deleted unless all its association lines on the appropriate chart(s) are mentioned in the SD of the rule. This condition makes things convenient: it means among other things that we will not create dangling association lines[5]; we will never have to worry about the choice between deleting them or reattaching them or blocking the rule -- the choice is already made for us -- block the rule. This is a simple matter of not allowing any association symbols (on the appropriate chart(s)) besides those explicitly mentioned by the rule. For example, take the rule of Sandhi Lowering in KiHunde ([22], pp. 37-39):

(51)



Because the first High tone is being modified, the rule must explicitly mention all of its associations to the CV tier. If there are any other associations, the rule will not apply (ie. will block). Therefore, when constructing the MSFT for this rule, we would have to begin with the following expanded automaton SD_j for the lower tier ('|' represents a word boundary):

(52)



5. Except that we may create dangling lines if the deleted segment has associations to charts other than the ones mentioned in the rule.

where A' = A - $\{\alpha_{ij}\}$. In other words, when we are expanding the α_{ij} states, we expand the one next to the first H with a slightly different automaton.

6.3.5. The Final Step

The last step in the process of modelling the High Tone Shift rule is to change the state labels of HS_1 that contain α_{ij}' or α_{ij}''. For every state q_k in HS_1, if $\lambda(q_k) \cap \{\alpha_{ij}', \alpha_{ij}''\} \neq \varnothing$, then let $\lambda(q_k) = \lambda(q_k) - \{\alpha_{ij}', \alpha_{ij}''\} \cup \{\alpha_{ij}\}$. This means that every state that before accepted α_{ij}' or α_{ij}'' will now instead accept α_{ij}. This eliminates the distinction between background, foreground, and added lines. The distinction is no longer needed — it has served its purpose in affecting the structure of the MSFT we created to model the High Tone Shift rule. The structure of the MSFT is such that it will enforce the NCC, as we described above.

6.4. Summary

We have shown above that the basic autosegmental mechanisms of addition and deletion of association lines and segments can be performed by MSFTs. There are some other mechanisms that we have not dealt with, but which can also be performed by MSFTs. These include modification of segments; recognizing boundaries between morphemes, words, and phrases; unbounded spreading, and the Association Convention.

The user of a computational tool based on this model would not have to concern himself with the details of MSFTs. He could simply specify autosegmental representations and rules in autosegmental notation (by means of a graphical user interface, perhaps). Representations could then be converted to multi-linear codes. A rule would be converted into regular expressions, and from these regular expressions, the automata shown above could be automatically constructed. These automata would be combined into a single MSFT, which would then be executed on the multi-linear codes to simulate the action of the rule.

If a graphical interface was too tedious, or a way of specifying representations and rules in a text file was needed, then autosegmental representations could be specified as multi-linear codes, and rules could be described using regular expressions, along with some other notation.

This chapter, then provides a proof of concept for the idea that MSFTs can implement autosegmental rules. By going through a specific example, we have exemplified the general principles of combining MSFAs and MSFTs into rule automata.

Chapter 7
Computational Power of MSFAs

We have already made allusions to the fact that MSFAs have more theoretical computing power than FSAs. In this chapter we substantiate that claim and make some preliminary investigations into the class of formal languages accepted by MSFAs.

7.1. Interpreting MSFAs as Language Acceptors

MSFAs, strictly speaking, do not accept languages; rather, they accept *n*-ary relations [34]. They do not accept single words, but *n*-tuples of words, one word on each tape. Therefore, in order to talk about the class of languages accepted by MSFAs, we need to interpret an *n*-tuple of words in some way as a single word. In our case, this means interpreting multi-linear codes as linear strings.

The main reason for interpreting MSFAs as language acceptors and multi-linear codes as single words (linear strings) is so that we can investigate the computational power of our model in a way that is comparable with Kornai's model [32]. Kornai uses transducers to process linear strings; we can conceptually view our model as doing the same. Once we have done this, we will find that one of the fundamental differences between our model and Kornai's is a difference in computational power.

There are two obvious possibilities: the words of an *n*-tuple can be combined into one word either in *series* or in *parallel*. In the former case, the *n*-tuple $<w_1, w_2, ..., w_n>$ is interpreted as the word $w_1w_2...w_n$. In the latter case, it is interpreted as a single word over an alphabet of *n*-tuples of characters. Individual letters of the words are combined into *n*-tuples in parallel. For example, the 3-tuple <ab, cd, ef> would be interpreted as the two-letter word <a,c,e><b,d,f>, consisting of 3-tuple characters.

Which method should we choose? With a series interpretation, many n-tuples would be interpreted as the same word. For example, <aa, bb, cc>, <a, abbc, c> and <aab, bc, c> would all be interpreted as the word aabbcc. In our application, many encodings of autosegmental representations could be interpreted as the same formal language word. This interpretation would mean losing the 1-1 property of our linear encoding when moving to the formal language domain.[1] On the other hand, the parallel interpretation yields an invertible mapping of encodings.

More importantly, the series interpretation loses all intuition that there is a temporal connection between segments on the different tapes.[2] This intuition is also lost in the parallel interpretation, but not to as great an extent, because there is some sense that segments in different co-ordinate positions within the n-tuples can overlap in time.

Finally, under the series interpretation, an MSFA can accept strictly context sensitive languages. In particular, it is easy to see that an n-tape MSFA can accept the n-tuple <w, w, ..., w>, which under a series interpretation is the language $\{w^n \mid w \in \Sigma^*\}$. For a 2-tape MSFA, this is the language ww, which is strictly context sensitive. We will see below that under the parallel interpretation, an n-tape MSFA cannot accept ww for any n. Therefore, the series interpretation leads to a more powerful kind of automata.

For all these reasons, we have chosen the parallel interpretation of input words on the tapes of an MSFA.

Consider representation (21) from Chapter 4. Its encoding was the 3-tuple:

1. There would be no way to invert this mapping to go from the formal language word back to the encoding, unless each tier had a special end-of-tier marker, or unless segments could not appear on more than one tier.
2. Jo Calder (personal communication).

(53)
```
all
C2V1C2C2V1C2
k2t22b2
```

Under the parallel interpretation, this would be interpreted as the following word, where we have written the 3-tuple letters vertically:

(54) $$\begin{bmatrix}\texttt{a}\\\texttt{C}\\\texttt{k}\end{bmatrix}\begin{bmatrix}\texttt{1}\\\texttt{2}\\\texttt{2}\end{bmatrix}\begin{bmatrix}\texttt{1}\\\texttt{V}\\\texttt{t}\end{bmatrix}\begin{bmatrix}\texttt{-}\\\texttt{1}\\\texttt{2}\end{bmatrix}\begin{bmatrix}\texttt{-}\\\texttt{C}\\\texttt{2}\end{bmatrix}\begin{bmatrix}\texttt{-}\\\texttt{2}\\\texttt{b}\end{bmatrix}\begin{bmatrix}\texttt{-}\\\texttt{C}\\\texttt{2}\end{bmatrix}\begin{bmatrix}\texttt{-}\\\texttt{2}\\\texttt{-}\end{bmatrix}\begin{bmatrix}\texttt{-}\\\texttt{V}\\\texttt{-}\end{bmatrix}\begin{bmatrix}\texttt{-}\\\texttt{1}\\\texttt{-}\end{bmatrix}\begin{bmatrix}\texttt{-}\\\texttt{C}\\\texttt{-}\end{bmatrix}\begin{bmatrix}\texttt{-}\\\texttt{2}\\\texttt{-}\end{bmatrix}$$

Since all the words in an n-tuple will not necessarily be the same length, we must introduce a new symbol "-" into our alphabet Σ as a kind of place-holder, to explicitly represent the blank symbol.

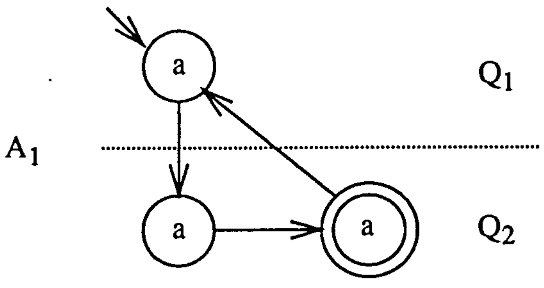
7.2. Beyond Regular Languages

Clearly MSFAs can accept all regular languages, since one-tape MSFAs are equivalent to FSAs. But MSFAs can accept non-regular languages as well. The intuitive reason for this is that an MSFA scanning a word encoded in parallel on its input tapes in the manner described above can scan *different parts* of the word *at the same time.*

Let us make an analogy to FSAs. It is easy to see that an FSA with two independent read heads scanning the same input tape can accept the (non-regular) context free language $a^m b^m$, because one of the read heads can skip over the a's, and then the two heads can compare the a's and b's, symbol for symbol. In this way the FSA can take advantage of the ability to scan different parts of the word at the same time. MSFAs have a similar kind of ability. Their read heads can scan n-tuples separated by arbitrary distances, and each head reads one co-ordinate of the n-tuple under it. Although they cannot compare whole n-tuples, the ability to compare widely separated co-ordinates like this is enough to give them extra computational power.

It is precisely this ability to scan different parts of an input word at the same time that is so important in modelling autosegmental rules. Association lines can associate segments in any part of one tier to segments in any part of the facing tier. In order for any computational device to efficiently process autosegmental representations, it must be able to scan two associated segments from widely separated parts of the representation at the same time. Kornai's model is based on standard FSAs and FSTs, which do not have this ability. This is the basic source of some problems with his model, as we will argue later.

MSFAs, by taking advantage of this ability, can accept non-regular languages. Specifically, they can scan 2 symbols on one tape for every one symbol scanned on another tape; or scan 3 symbols for 1; or *k* symbols for 1, for any fixed *k*. For example, Figure 7-1 and



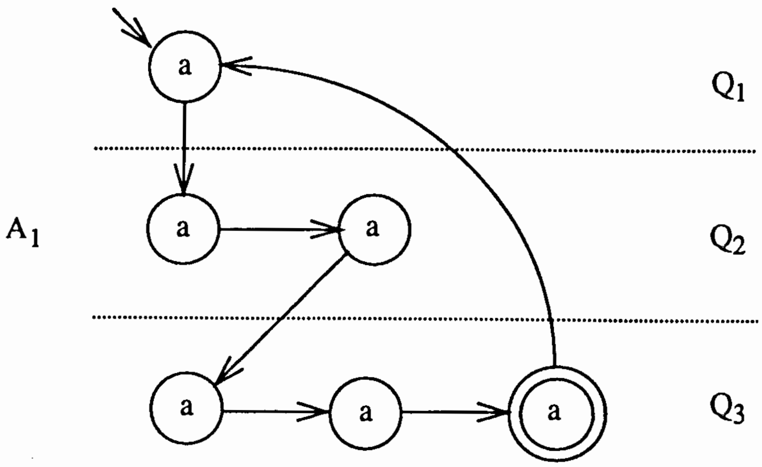
$$L(A_1) = \langle a^m, a^{2m} \rangle = \begin{bmatrix} a \\ a \end{bmatrix}^m \begin{bmatrix} - \\ a \end{bmatrix}^m$$

Figure 7-1: Context free languages accepted by MSFAs

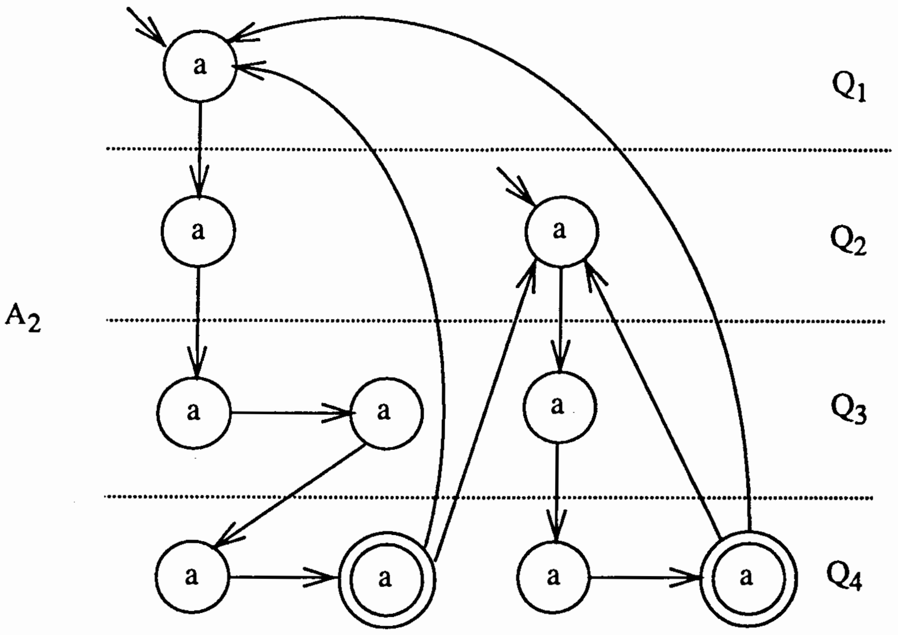
Figure 7-2 show how MSFAs can accept languages that are homomorphic to $x^m y^m$, $x^m y^m z^m$, and $w^l x^m y^l z^m$. In this context, the homomorphisms map *n*-tuples to symbols; eg. in Figure 7-1, we can map <a, a> to x and <-, a> to y to show that $L(A_1)$ is homomorphic to $x^m y^m$.

7.3. The Chomsky Hierarchy

However, MSFAs cannot recognize *all* context sensitive languages, or even all context free languages. To show this, we appeal to a theorem of Rabin and Scott ([34], p. 124),



$$L(A_1) = \langle a^m, a^{2m}, a^{3m} \rangle = \begin{bmatrix} a \\ a \\ a \end{bmatrix}^m \begin{bmatrix} - \\ a \\ a \end{bmatrix}^m \begin{bmatrix} - \\ - \\ a \end{bmatrix}^m$$



$$L(A_2) = \langle a^l, a^{l+m}, a^{2l+m}, a^{2l+2m} \rangle = \begin{bmatrix} a \\ a \\ a \\ a \end{bmatrix}^l \begin{bmatrix} - \\ a \\ a \\ a \end{bmatrix}^m \begin{bmatrix} - \\ - \\ a \\ a \end{bmatrix}^l \begin{bmatrix} - \\ - \\ - \\ a \end{bmatrix}^m$$

Figure 7-2: Context sensitive languages accepted by MSFAs

which we restate below in terms of MSFAs:[3]

Theorem 7-1: Let A be a deterministic two-tape MSFA. The set of all words w_1 for which there exists some word w_2 such that $<w_1, w_2>$ is in L(A) (ie. the domain of the [binary] relation defined by A) is definable by a state-labelled finite automaton (SFA). An SFA defining this set can in fact be constructed effectively from A.[4]

This theorem can be generalized to MSFAs with more than two tapes, as Rabin and Scott claim. It can also be generalized to non-deterministic MSFAs, a question which Rabin and Scott did not consider.

This theorem means that the languages accepted on each individual tape of an MSFA must be regular. This is a useful tool for proving that certain languages do not belong to the class of languages accepted by MSFAs (henceforth referred to as MSFA languages). For example, we know that the following language cannot be an MSFA language:

(55) $<a^m b^m, a^{2m}> = \begin{bmatrix} a \\ a \end{bmatrix}^m \begin{bmatrix} b \\ a \end{bmatrix}^m$

for the simple reason that the language accepted on the first tape is $a^m b^m$, a non-regular language. So this is an example of a context free language that is not an MSFA language.

Notice that language (55), like the language in Figure 7-1, is homomorphic to $a^m b^m$. Yet one is an MSFA language and one is not. This is counter-intuitive. Most of the language classes in the Chomsky hierarchy that have been investigated so far are closed under homomorphism. This is not the case with MSFA languages.

We also have an example of a (strictly) context sensitive language ([14], p. 366) that is not an MSFA language. We can again use Theorem 7-1 to show that:

3. It is easily proved that the deterministic version of our MSFAs is equivalent to Rabin and Scott's multi-tape automata.
4. It would be interesting to answer the following question: Is this effective procedure of Rabin and Scott's equivalent to collapsing all the states in the other partitions?

Theorem 7-2: No language $\{ww \mid w \in A^*\}$ ($|A| > 1$) is accepted by an n-tape MSFA for any n.

Proof: Suppose that $L = \{ww \mid w \in A^*\}$ is accepted by an n-tape MSFA M (A is an alphabet of n-tuples, with $|A| > 1$). We will show that this is impossible by contradiction.

Since $|A| > 1$, we can find two n-tuples in A which differ in some i^{th} co-ordinate. Let A_i be the alphabet of symbols that appear in the i^{th} co-ordinate of n-tuples in A ($|A_i| > 1$). Note that equality of n-tuples is based on co-ordinate-wise comparison. From this it follows that since M accepts *only* words of the form ww, it accepts only words of that form on tape i. Also, for every word w_iw_i, $w_i \in A_i^*$, there is a word ww, $w \in A^*$, accepted by M, whose projection on the i^{th} co-ordinate is w_iw_i. Thus M accepts *all* words of the form ww on tape i.

So M accepts a (strictly) context sensitive language on its i^{th} tape. This contradicts Theorem 7-1 above. ❑

7.4. An Upper Bound

To summarize so far, we have shown that MSFAs can accept all regular languages, some (but not all) strictly context free languages, and some (but not all) strictly context sensitive languages. To cap off our investigation of the formal computational power of MSFAs, we will show that they cannot accept non-context-sensitive languages. We will do this by simulating an MSFA with a linear bounded automata (LBA), which is the machine equivalent of a context sensitive grammar [23].

Theorem 7-3: An LBA can simulate an n-tape MSFA.

Proof: We will describe the ideas of the proof intuitively. The proof is easier using the transition-labelled version of multi-tape FSAs, so we will simulate a multi-tape transition-labelled finite automata (MTFA) and invoke their equivalence to MSFAs.

The technique is to use an LBA with an n-track tape (which is equivalent to an LBA with an alphabet of n-tuples). Since this LBA has only one read head (scanning all n tracks), it will have to remember the position of the n simulated read heads of the MTFA using special symbols, and make up for its lack of read heads by its ability to scan in both directions instead of just one.

An LBA can use special symbols to mark positions on the tracks as follows. For each symbol s in the alphabet of the MTFA, the LBA will have two symbols: s and s'. If the LBA scans s' on the i^{th} track, it will know that the simulated read head on the i^{th} track is positioned there, about to scan the symbol s.

At each execution step, an MTFA is in a state that dictates which tape is to be read from. It reads from this tape, advancing the read head over one symbol, and depending on what that symbol is, makes a transition into another state. This can be simulated as follows. Because there are a fixed finite number (n) of input tapes on the MTFA, the LBA can use its states to remember what tape (track) it is currently reading from. Suppose this is tape i. It reads a symbol from track i by resetting its read head back to the beginning of the tape, and scanning forward until it sees a special symbol s' on track i. It then rewrites s' as s, and makes a transition based on having read s on tape i (simulating the corresponding transition in the MTFA). This transition would include recording (in the state information) which tape is now to be read from, based on what partition the MTFA transition entered. Then the symbol t to the immediate right of s on the tape would be rewritten as t', to simulate the movement of the head on that tape.

This is all straightforward. We only need to observe that this can all be done in space bounded by the length of the input; in fact, it is done in exactly the space taken up by the input. The final states of the LBA will be based on the final states of the MTFA, so that the two machines will accept exactly the same inputs. This completes the proof. ❑

This shows that the class of MSFA languages is contained in the class of context sensitive languages. The containment is proper, since ww is not an MSFA language..

We summarize our findings with Figure 7-3, a diagram of the Chomsky hierarchy. The

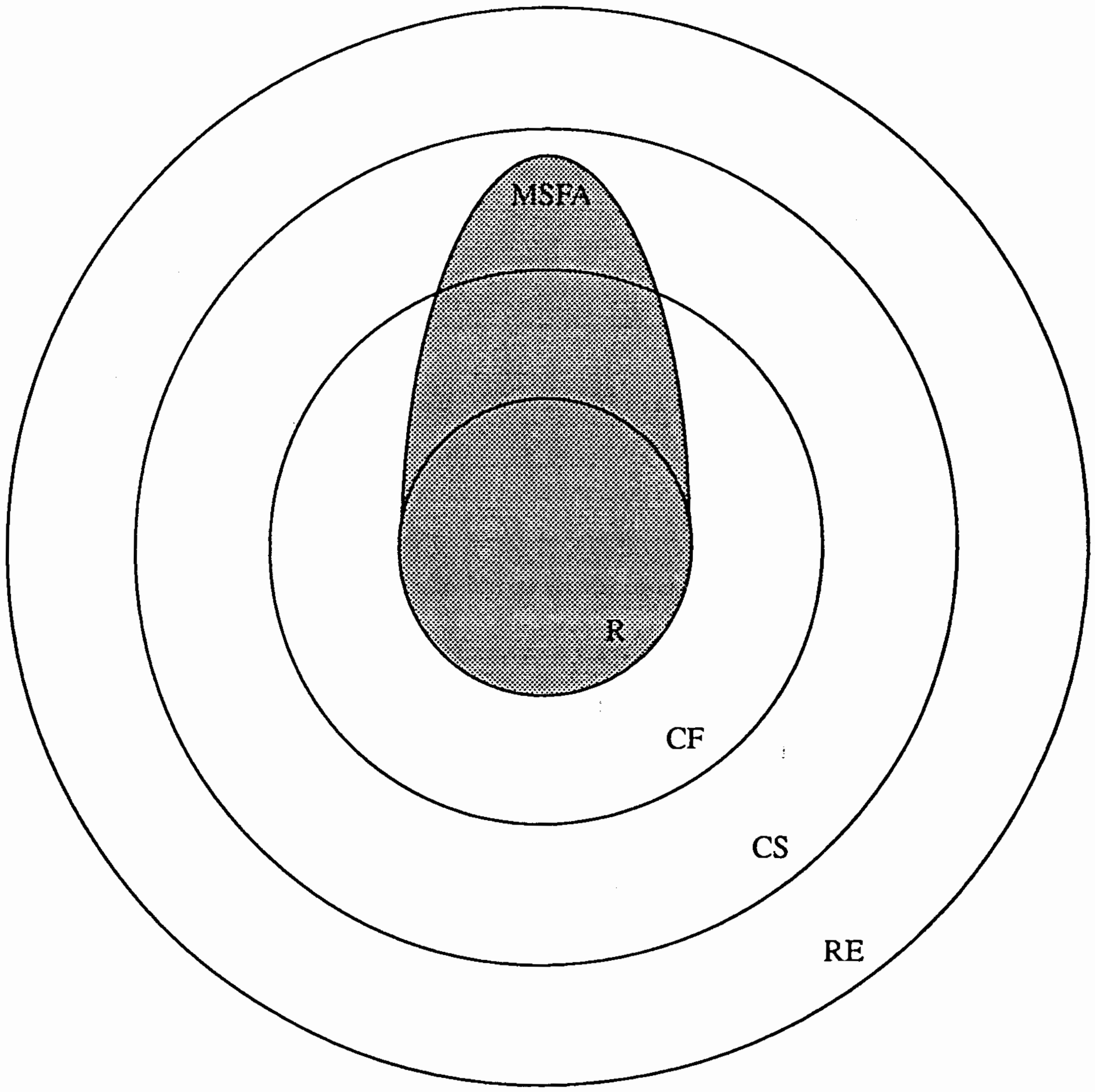


Figure 7-3: MSFAs and the Chomsky Hierarchy

four circles, recall, represent the regular (R), context free (CF), context sensitive (CS) and recursively enumerable (RE) languages. The shaded region indicates the MSFA languages and their relationship to this hierarchy.

Chapter 8
Evaluation

In this final chapter, we compare and contrast our model with various other models that make use of finite state automata. In doing so, we evaluate the model based on the three criteria of adequate expressive power, faithfulness to autosegmental theory, and suitability as a basis for a computational tool. We also suggest possible directions for future research.

We will consider four quite different approaches to modelling a theory of phonology using finite state automata. The earliest work on finite state autosegmental phonology was by Kay [29]. The KIMMO system is based on the two-level phonology of Kimmo Koskenniemi [33]. Two other approaches, proposed by Kornai [32] and Bird & Ellison [8], are based on autosegmental theory.

8.1. Kay and Multi-Tape Transducers

8.1.1. Computational Power

A well-known application of FSTs to Arabic morphology and phonology was made by Kay [29]. In this work, Kay made use of multi-tape FSTs, claiming that they had no more than finite state power.

He begins by describing a transducer with four tapes, three of which are interpreted as input tapes, and one as an output tape. It has only one read/write head: "...when the automaton moves from one state to another, each of the four tapes will advance..." Here he is safely within the bounds of finite state power, because this is nothing more than a four-track FSA, which is equivalent to a single tape FSA with an alphabet of 4-tuples.

However, he goes on to say (p. 5):

> I shall allow myself some extensions to this basic scheme which will enhance the perspicuity and economy of the formalism without changing its essential character. In particular, these extensions will leave us clearly within the domain of finite state devices.

These extensions of which he speaks include notational extensions which allow the transducer to advance (or not advance) each tape *independently.*

As Bird & Ellison point out ([8], p. 41), "the claim that his model *is* a [finite state] transducer is unsubstantiated." In fact, his transducer is *not* finite state; it is a 4-tape MSFA, with 3 tapes considered as input tapes and 1 considered as an output tape.[1]

That this device is not finite state can be seen from results reported by Fischer [17]. He defines D_n and N_n as the classes of sets of n-tuples accepted by the deterministic and non-deterministic versions of n-tape FSAs, respectively. He then notes that D_n is not closed under any of the operations that regular languages are closed under, except set complementation (it is not closed under union, intersection, input reversal, concatenation, or Kleene closure). Also, although N_n is closed under many of these operations, it is not closed under complementation or intersection. If multi-tape FSAs were finite state devices, the sets that they accept would be closed under all these operations (see Section 2.3.2.). We have seen in Chapter 7 that multi-tape FSAs have somewhat more than finite state power.

The problem arose when Kay gave his transducers the ability to advance each input tape independently, which is as much as to say that each input tape has an independent read head. Now, with an FST (one input and one output tape), the ability to advance the *output* tape independently from the *input* tape does not increase the computational power;

1. This is a generalized kind of MSFT, with *unequal* numbers of input and output tapes (see Definition 3-6 and footnote).

this is generalized sequential transduction, which still specifies regular relations, although it allows relations between words of unequal length [14]. But to allow one *input* tape to advance independently of another *input* tape is quite another matter. This allows the transducer to scan different parts of the input at the same time, which gives it extra computational power (see comments above in Section 7.2.).

It is revealing that in an attempt to process non-linear representations with finite state automata, Kay unknowingly resorts to something more powerful to get the job done. This can be taken at least as circumstantial evidence that more power is needed. In any case, this thesis contributes a proper understanding of the multi-tape automata that Kay used to deal with Arabic morphology and phonology.

8.1.2. Association

Another point of comparison between Kay's work and ours has to do with representing association lines. Kay has morphemes written on the tapes of his transducers without any indication of the associations between the segments in those morphemes. He takes it as the task of his transducers to determine what those associations should be. For example, from the three morphemes `a`, `CVCCVC`, and `ktb`, his transducer deduces that the `a` should be associated to the two `V` slots, the `t` to the two center `C` slots, etc. (see (20) in Chapter 4); thus it writes out the correct form `kattab` on its output tape.

We could divide the action of Kay's transducer into two conceptual tasks — a phonological one and a morphological one. First, it properly associates segments, a task that would be done by a set of autosegmental rules plus some other conventions like the Association Convention. Second, it combines the three tiers into one according to those associations, producing a linear representation.

The association lines that the transducer adds are not explicitly represented on the tapes, but are implicit in the the way the transducer scans the tapes. In a sense, the associations are encoded in the transition function of the transducer. If a segment is scanned simultaneously with a C or V slot, and is written to the output tape, this represents an association between the segment and the C or V slot.

By contrast, we encode the associations explicitly on the tapes of our automata using special association symbols. This is because the scope of our work is broader than Kay's. He is not modelling the whole of autosegmental theory, but just its application to a particular set of Arabic data. In fact, he is more concerned with morphographemics (the interaction of morphology and orthography) than with phonology. It so happens that for the data he considers, there are no underlying forms with lexically specified associations. His model could not deal with data like this. Since ours must, we have moved the encoding of associations out of the transducer into the encoding of the representations. In our transducers we encode the more general principles of autosegmental rules which add and manipulate association lines.

8.2. Koskeniemmi and Two-Level Phonology

8.2.1. Background

Antworth [3] gives a summary of the history leading up to Koskenniemi's work on two-level phonology, and the KIMMO system which resulted from it. In an unpublished conference paper ([30]; see [28]), Kaplan and Kay suggested the possibility of implementing the ordered rules of classical generative phonology as a series of cascaded FSTs, the output from each transducer feeding into the input of the next in sequence. Automata theory tells us that it is possible to combine a series of FSTs like this into a single equivalent FST which would produce the surface form directly from the underlying form.

The difficulty with this, as Koskenniemi found, was that the number of states in the combined FST got prohibitively large, and made the computational model inefficient. So he chose a different model: instead of having the FSTs operate in series, he had them operate in parallel. A bank of FSTs would execute in parallel, together generating the surface form from the underlying form. These FSTs would in fact share the same read/write heads.

Again, it is possible to combine such parallel FSTs into a single equivalent FST, but this time more efficiently; that is, the number of states in the combined FST is much smaller.

These parallel FSTs could also be viewed as accepting pairs of underlying/surface forms. A pair is accepted only if it is accepted by *all* the parallel FSTs. As Kay points out [29], it is only because the FSTs share read/write heads that they can be combined into one. Although *FSAs* are closed under intersection, it is not possible *in general* to find an *FST* that accepts the intersection of the regular relations accepted by two FSTs, if empty transitions are allowed. For example, the intersection of the regular relations $<a^m b^*, c^m>$ and $<a^* b^n, c^n>$ is the non-regular relation $<a^n b^n, c^n>$ [29, 36]. But it *is* possible, when the FSTs are required to share the same tape heads, and thus use the same scanning pattern when accepting their inputs, to find an FST that simulates both executing in parallel.

We generalized this idea of sharing the same read/write heads when we defined our product operation on MSFAs. The generalization was to more than one input tape or more than one output tape. The point of contact is with 2-tape MSFAs (1-tape MSFTs), which can likewise accept $<a^m b^*, c^m>$ and $<a^* b^n, c^n>$, but not $<a^n b^n, c^n>$.

The implementation of two-level phonology was dubbed KIMMO. It actually incorporates two finite state models--one for phonology as described above, and another one for morphology. It has served widely as a useful tool for morphological and phonological analysis [37, 3].

8.2.2. Non-linear Representations

However, from a phonological perspective, two-level phonology has an important limitation: its representations are linear. The linear phonological representations of classical generative phonology have now been superseded by non-linear representations. Antworth ([3], p. 12) notes this limitation, saying, "[KIMMO] does not support non-linear representation, and it is not clear how finite state phonology could be modified to do so."

We can now offer an answer to this question. First, if we take "finite state phonology" to mean a model with no more than finite state power, then we conjecture that finite state phonology cannot adequately support non-linear representation. We argue below that Kornai's model does not adequately support non-linear representations with only finite state power. We suspect that other attempts at finite state processing of non-linear representations will also be inadequate.

Currently, two-level phonology computes regular relations between linear representations using FSTs.[2] However, we now have a new type of transducer, one with more than finite state power. We have shown in Chapter 6 that MSFTs can compute a wide range of relations defined by autosegmental rules between non-linear representations. If the FSTs in the KIMMO system were replaced by MSFTs[3], the result would be a model that could adequately support non-linear representations. This would be a fruitful line of further research, especially in light of the wide acceptance and use that the KIMMO system has seen since its introduction about 10 years ago.

8.2.3. Reversibility

The KIMMO system can recognize words as easily as generate them. This follows from the fact that FSTs are reversible, in the sense that they can be directly interpreted as producing input words from output words, as well as the other way around. MSFTs are also reversible in this sense; that is, they can be interpreted as reading an encoding from the output tapes, and writing a corresponding encoding on the input tapes. In this case, the MSFT is performing inverse transduction, producing the input encoding(s) that are related to a given output encoding by the rule the MSFT models.

If a single MSFT is used to generate surface forms from underlying forms (in the style of either KIMMO or Kaplan and Kay), this means that given the surface form, the MSFT could produce its corresponding underlying form. This ability to perform generation and recognition by a single mechanism is a significant advantage in a practical computational

2. Ritchie shows that the two-level model does not fully utilize the generative power of the parallel FSTs that are used to implement it [36].
3. Our thanks goes to Jo Calder (personal communication) for first pointing out this possibility.

tool.

8.3. Kornai and the Triple Code

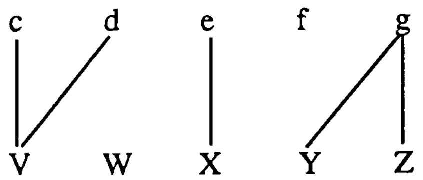
8.3.1. Background

One of Kornai's goals with his model [32] was to fully implement autosegmental phonology using FSTs. The strategy was to encode non-linear autosegmental representations as linear strings so that FSTs could process them in the manner proposed by Kaplan and Kay.

This thesis follows in the footsteps of Kornai's dissertation, and owes much to it. Our model was constructed in the same transducer-based, procedural spirit as his. One of the primary motivations for this work was the perception that Kornai's model had inadequate expressive power. Being based on FSTs, it computes regular relations between phonological representations. But there are some phonological relations which are arguably not regular, at least under his linear encoding of autosegmental representations.

To support this claim, we will have to take a closer look at Kornai's linear encoding of two-tiered autosegmental representations (ie. charts), called the *triple code*.[4] Here is a chart and its encoding:

(56)



```
c1V.t.d1V.e0W.b.e1X.f0Y.t.g1Y.b.g1Z
```

4. Strictly speaking, Kornai's triple code is an improved version of the code we present here. We use his preliminary version of it for ease of presentation. The difference is minor and does not affect our subsequent comments.

The dots are not part of the code; they are just delimiters to make the code easier to read. The code is based on an algorithm which scans the chart from left to right and writes out the code. The two tiers are scanned by, as it were, two independent read heads. At each step, the segments under the read heads, and their association status (0 = unassociated, 1 = associated), are written out as a triple. Then one or both of the read heads may move to the next segment; these possibilities are referred to as *top move* (t), *bottom move* (b), or *full move*. Top and bottom moves are explicitly recorded in the code; full moves are implied by the juxtaposition of two triples without any move recorded between them. The choice of move depends on the association pattern in the chart, because neither read head can advance past an association line until it has been recorded. The choice of move is deterministic, because a full move is always made if possible.

Kornai discusses how this linear encoding can be processed by FSTs which implement the operations of autosegmental phonology, such as the Association Convention, and the insertion and deletion of segments and association lines. He contends that these operations, as well as autosegmental rules, iterated rules, and entire derivations can be simulated with a single FST. Thus he claims that only finite state power is needed to implement a computational model of autosegmental phonology.

In passing, we should mention that, in contrast, we have only taken our model to the level of autosegmental rules; we have not discussed iteration or entire derivations. We claim that all autosegmental rules can be implemented as MSFAs, but for the time being, in order to iterate rules, or combine them via various rule ordering strategies into derivations, we must rely on a meta-level mechanism above the transducers. This mechanism would order the transducers, feed the output of one transducer into the input of the next, detect when a transducer fails, etc.

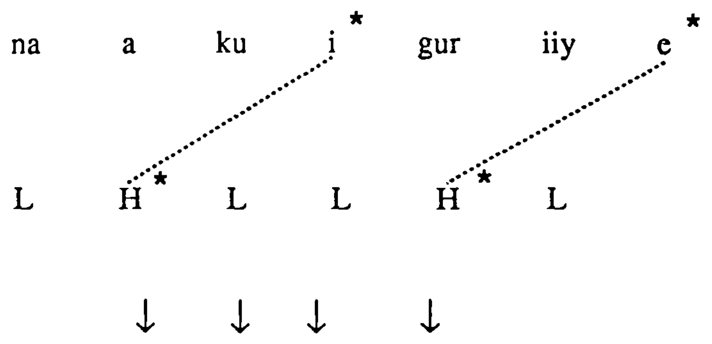
We say "for the time being" because it may very well be possible to simulate an entire derivation using a single MSFA. It is possible to extend our rule transducers so that when they cannot match the SD of the rule they simulate, they will perform the identity transduction instead of failing. It is also possible to define a composition operation on MSFTs, by analogy with composition of FSTs, so that a single MSFT can simulate two MSFTs operating in series, subject to the condition that the scanning pattern with which the first transducer writes its output is the same as the scanning pattern with which the second transducer reads its input. With these and other tools, it may be possible to give a procedure for combining an entire set of rules and a rule ordering strategy into a single MSFT. This is an area for further work.

8.3.2. Non-regular Phonological Relations

Now, to return to Kornai's model, there are some reasons why we have difficulty accepting his claim that autosegmental phonology can be modelled with finite state power. The fundamental issue here is that there are certain operations in autosegmental phonology that cannot be done by a finite state transducer because they require an unbounded amount of memory. That is, there is no principled upper bound on the amount of memory required.

For example, the elementary operation of adding an association line suffers from this problem. This was independently discovered by Bird & Ellison ([8], p. 45), and by the author [41]. An example from Goldsmith's discussion of accent systems ([21], p. 55), referred to in [8], is of tone association in Ci-Ruri, a Bantu language of Tanzania:

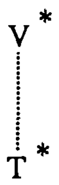
(57) "I bought it for you"



```
naOL.aOH.kuOL.iOL.gurOH.iiyOL.t.eOL
                  ↓
naOL.aOH.t.kuOH.t.i1H.gurOL.iiyOL.e1H.b.eOL
```

Here two association lines are being added according to the Accent Association Rule ([21], p. 49)[5]:

(58)



The triple codes of the representation before and after the application of this rule are shown in (57). (Each of the syllables of the word are treated as a single segment/symbol in the code, and as a V slot for the purposes of the rule.) Notice that an FST, as it scans the input encoding and writes the output encoding, would have to store some of the tones, because they are scanned well before they are written. The transducer needs to have sufficient memory to store two tones at a time, because each of the tones marked with an arrow

5. If it is argued that Goldsmith's accent system is an extra-autosegmental one (ie. goes beyond the bounds of standard autosegmental theory), we would counter by saying that the accents can be modelled within standard autosegmental theory by adding an accent tier to the feature geometry. An accented vowel or tone would be one that is associated to an accent feature on the that tier, and Goldsmith's Pre- and Post-accents [21] can be recast as standard rules that reassociate accents.

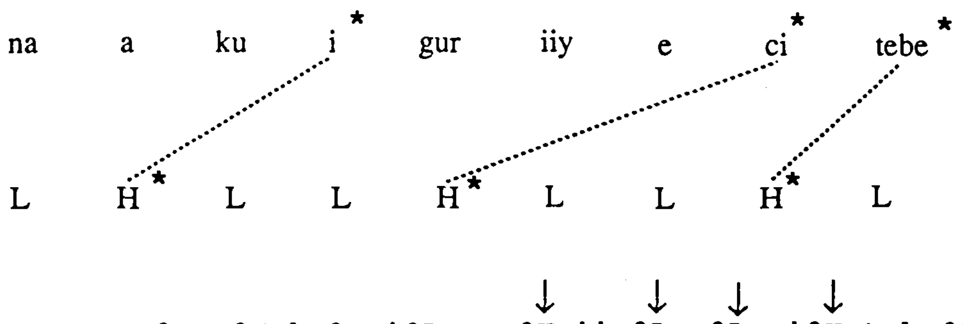
in the triple code in (57) must be stored, and output two triples later. For example, the first H tone is scanned in the second triple and output in the (second, third, and) fourth triple. (The last L tone does not need to be stored, because it is already part of the last triple in the first code.)

In discussing insertion of segments (p. 52), Kornai says:

> ...the changes [in the code] always involve adjacent triples, so they can still be handled by finite transducers, in spite of the fact that these are commonly thought of as being 'memoryless'.

However, in the case of adding an association line, the changes do not always involve adjacent triples, as we have seen. In a related example ([21], p. 54), the problem is even more pronounced:

(59) "I bought (it) for you the chair"



```
naOL.aOH.kuOL.iOL.gurOH.iiyOL.eOL.ciOH.tebeOL
```

↓

```
naOL.aOH.t.kuOH.t.i1H.gurOL.iiyOL.eOH.t.ci1H.tebeOL.b.
tebeOL.b.tebe1H.b.tebeOL
```

Here we have a phrase with three accented syllables. After some accent rules apply, including one that causes the second accent to "hop" from the first word to the second, the initial tone association shown above is performed. The tones indicated by the arrows must be shifted over by three triples in the output. This means that the finite transducer that processes these codes must be able to store three tones.

We could likely go on to find examples of this type that require an automaton to store four or five tones. However, it is clear already that an autosegmental rule must be able to associate two widely separated segments. In principle, there is no bound on the distance between segments that can be associated by a rule.

Bird & Ellison give another example of a phonological operation which is not regular under Kornai's encoding: the Association Convention ([8], p. 45). They discuss the possibility of dividing up the task into individual acts of association, and performing each by a finite state transducer. If we took this approach in our example above, this would reduce the maximum memory required by an individual transducer, but the overall picture would be the same (p. 45):

> It is true that each individual act of association...can be performed by a finite-state transducer. But no principled bound on the length of the derivation can be made, and the quantity of memory required is an increasing function of the length of the derivation. Consequently, the step from unassociated to associated form cannot be made by a single FST.

Therefore, under Kornai's triple code, processes such as association and the Association Convention are not finite state.

8.3.3. Response and Rebuttal

Kornai rightly identifies what he calls the *memorylessness fallacy.* This refers to the misconception that finite state automata have no memory. The fact is that they do have memory — just a fixed finite amount of it. Therefore they cannot do tasks that we think of as requiring (an unbounded amount of) memory, such as parenthesis matching, which requires an (infinite) stack. With this in mind, our claim is that autosegmental operations require an *unbounded* amount of memory.

Kornai responds to this by pointing out that his automata do not need to solve the general problem of applying autosegmental operations to arbitrary encodings, but only to a finite number of encodings — the data that is actually met in practice. As he says on pages 60-61:

> ...the problem is to express complex phonological operations...We do not have a general-purpose finite state solution to the problem...But if we know that we need only a finite class of...strings recognized, we can of course devise a finite automaton for that class of strings...

It is true that the phonologist will always be working with a finite amount of data, and that therefore we can always put a bound on the amount of memory required to implement phonological operations. But this bound is an arbitrary, empirical one, and not a principled theoretical one. We contend that Kornai's claim that FSTs have sufficient power to implement autosegmental phonology is similar to the claim that a regular grammar is sufficent to parse any finite subset of a context free language. Both are true, but both make it difficult to express generalizations that are present in the data, even if these generalizations do not find full expression.

Consider constructing a regular grammar for $L = \{ab, a^2b^2, ..., a^{100}\ b^{100}\}$. It would take a hundred rules, and these rules would essentially list each and every word. This would completely miss the obvious generalization that is efficiently and concisely captured by the two rules of the context-free grammar $\{S \rightarrow aSb, S \rightarrow ab\}$. This gives an intuitive idea how Kornai's approach leads to a model that is inefficient and that is unable to capture significant linguistic generalizations.

For example, a finite transducer could not implement the addition of an association line in a unified way, but would essentially list all the possible cases, depending on whether the segments being associated were 0, 1, 2, ... up to some finite number *k* triples apart. Similarly, if the Association Convention were implemented as a transducer, it would have to treat a finite number of special cases, depending on how many association lines it added. In each case, we are missing a *single* general description of a what is, after all, a *single* phonological operation.

Also, the appeal to only having a finite amount of data seems a too easy answer. Even if we have a strictly recursively enumerable language, we can still parse a finite subset of it with an FSA! After all, a finite state machine can simulate a Turing machine up to a certain finite point[6]. But this approach gives us no grasp on the structure of the languages we are parsing if they are truly non-regular.

Therefore, it is profitable to investigate models with more power than an FST. Although this extra power may not be empirically necessary, such models will be more efficient and will be able to describe linguistic generalizations that they otherwise could not describe. This is what we have done. Our MSFTs can be viewed as transducers processing linear encodings, but they are transducers with more than finite state power. This is a fundamental difference between our model and Kornai's.

As we saw in Chapter 7, MSFTs are transducers with finite state power plus a little bit of context free and context sensitive power. Exactly how much extra power this is needs to be determined. Further investigations need to be made into the properties of MSFAs, and the possibility of a grammar equivalent to MSFAs. Along with a grammar would come the question of efficient parsing algorithms. This is another area for future work.

We have abandoned FSTs for a more powerful device that is able to capture useful generalizations elegantly. Why not use an even more powerful device? Our main reason for abandoning FSTs was their inability to generally describe a phenomenon at the very core of autosegmental phonology: that of adding association lines.[7] Until such time as an important limitation like this becomes apparent with MSFTs, there is no reason to abandon them for a more powerful device.

6. The computer is a classic example of this.
7. This stands in contrast to the way context free grammars are still widely used to describe syntax, because their limitations are not at the core of syntax, but at the fringe, in a few rare phenomena.

8.3.4. Towards Extra Computational Power

In reaching beyond finite state power, we are in accord with recent work. Bird & Ellison's model of autosegmental phonology [8], although based on SFAs, uses more than finite state power, because the process of applying rules (which are viewed as constraints and modelled as SFAs) involves combining automata using operations of product and co-product. We must point out that their use of a product operation is different than ours. This is because for them, the product operation is itself the means of *applying* a rule, whereas for us, the product operation is the means of *constructing* a rule, a transducer, which is then *applied* by being executed.

Thus, for Bird & Ellison, the process of applying a rule requires indexed-grammar power (see Figure 2-1 for the place of indexed grammar languages in the Chomsky hierarchy). In their own words (p. 48):

> The question arises how we can construct the associations if the same operation for Kornai is not regular. The answer is simple. The operation which we have applied here, the product operation, is a non-regular operation on any linearisation of automata...The task of making the necessary associations corresponds to taking the product of SFAs. Specifying the product operation on regular expressions, for instance, has at least indexed-grammar power.

So they too have gone beyond the limits of finite state power to model autosegmental phonology.[8]

8.3.5. The Triple Code

Apart from questions of computational power, we can compare the encoding schemes used by the two models to encode autosegmental representations. We have already dis-

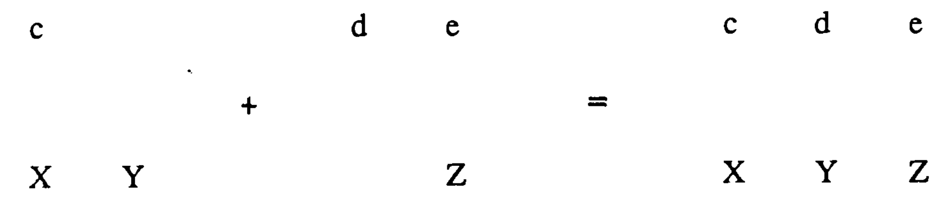
8. We should mention also that Jones has done an implementation of autosegmental phonology [26], but he did not investigate its formal computational power.

cussed the advantages of the multi-linear code in Section 4.3.; here we examine the triple code more closely.

The triple code does not meet some of Kornai's own criteria for a good linear encoding [8]. For example, the encoding is not compositional (Theorem 4-3 indicates that this *must* be the case): the concatenation of two encodings is not necessarily the same as the encoding of the concatenation. Also, it is not iconic: local changes to a representation can correspond to global changes in its encoding. These are illustrated below:

(60)

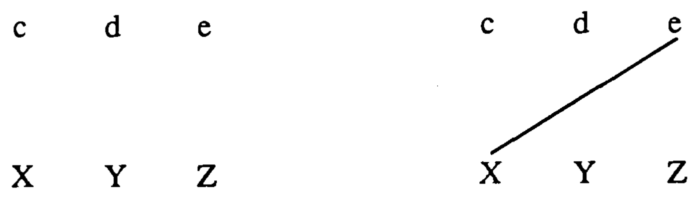
(a)



```
c0X.b.c0Y    +    d0Z.t.e0Z    ≠    c0X.d0Y.e0Z
```

Triple code not compositional

(b)



```
c0X.d0Y.e0Z    c0X.t.d0X.t.e1X.b.e0Y.b.e0Z
```

Triple code not iconic

There are other desirable criteria that are not met. The triple code fails to maintain the inherent simplicity of the elementary operations of autosegmental phonology, such as insertion and deletion of segments and association lines. Instead of being simply and uniformly described, the complexity of these operations varies depending on the environment of the affected segment or association line. For example, for representation (60)(b), it takes more computation to add a line between **e** and **X** than between **c** and **X**. The elementary operations of autosegmental phonology ought to be implemented computationally in a way that preserves their inherent simplicity; this is an important factor to the theoretical linguist.

Kornai's encoding is not easily extended to autosegmental representations with more than two tiers ([32], p. 70; [8], p. 45-46). Kornai discusses various possibilities for generalizing the code to multi-tiered representations. He is forced to either introduce an impractical amount of redundancy to the code, or devise a more complicated algorithm to compute the code. By contrast, our multi-linear code is designed from the start to handle multi-tiered representations, and introduces no redundancy.

The triple code linearizes a representation so that the segments and associations from different charts end up intermingled in the linear code. Thus, an FST that performs an operation on a single chart must wade through phonological material that is unrelated to the process at hand. Ideally, a computational model of a linguistic theory should not include computations that are linguistically irrelevant, but rather each computation should embody a linguistically significant process. Also, a good encoding should reflect the independence of the charts of a representation.

Another linear encoding of autosegmental representations is proposed in [41]. It overcomes some of the problems discussed above. However, this new code still does not result in a finite state model. Adding an association line still requires more than finite state power, as with Kornai's code. The difficulty of forcing a multi-linear representation into a linear encoding in such a way as to be able to process it with finite state power raised our suspicions that this may be impossible to do. This led us to propose the multi-linear code, which overcomes all of the problems above, as discussed in Section 4.3.

8.4. Bird & Ellison and One-level phonology

8.4.1. Background

Recently, a new model of autosegmental phonology based on FSAs has been proposed by Bird & Ellison [8, 9]. This approach differs from that of Kornai in several fundamental ways.

Rather than modelling representations as linear strings and rules as FSTs which operate on them, both representations and rules are modelled using FSAs. This is based on a view of representations as the fundamental objects in autosegmental theory, and of the tiers of a representation and the rules as (partial) descriptions of those objects. For example, we might have the Etsako word for house, ówõwà (see Section 2.2.). This is an object, various aspects of which are described by, for example: (1) the segmental tier of its representation, which contains the segments o w a o w a, with the a's being optional, (2) the tonal tier of its representation, which specifies that the tone of this word goes from high to low to high to low, on certain vowels, and (3) a rule that states that an optional vowel followed by another vowel is not pronounced, but its tone *is* pronounced. These three descriptions join forces to completely describe the word ówõwà.

This is a declarative view of phonology, where rules are viewed (and implemented) as constraints rather than as procedures which transform representations. The representations and rules are constructed in such a way that no information is ever lost in the process of applying a rule; information can only be added. An example is given where the word *electric* is underlyingly represented with *both* a *k* and an *s* segment at the end of the word. If the suffix *-al* is added, the *s* is "disabled" and the word is pronounced *electrikal.* If the suffix *-ity* is added, the *k* is disabled and the word is pronounced *electrisity.* Notice that no information is lost (neither segment is ever deleted). Instead, information is added, indicating that certain segments are disabled. Because rule application is a monotonic process of accumulating information, the process can easily be reversed. Word generation is no easier than word recognition.

8.4.2. Declarative vs. Procedural

This declarative view is quite different from the procedural view adopted by ourselves and Kornai. As Bird and Ellison state (p. 11):

> ...our model enables us to give a direct procedural semantics to a *declarative reformulation* of non-linear phonology [italics mine].

Autosegmental phonology has been traditionally presented from a generative, procedural standpoint (eg. [22], which is mainly procedural throughout, and offers an alternative view in the last chapter). Rules are seen as processes that manipulate representations. Our goal was to faithfully implement this theory *as is*, so that the analyses that currently exist can be machine-tested.

The declarative, constraint-based approach has its advantages, as well as growing support in the work of several phonologists (eg. McCarthy). But it requires re-analysis of the data. For example, the above quote is followed up with this statement:

> This means that if a non-linear phonological analysis does not avail itself of extrinsic rule ordering devices and destructive operations then it is possible to mechanically convert that analysis into a verification system. Such a system is able to automatically test the empirical claims of theoretical statements, and promises to bring untold benefits to phonological analysis.

If, however, an analysis does depend on rule ordering or destructive operations like the deletion of association lines, one either has to re-analyze the data in a declarative style, or find a computational model that allows these mechanism. Our model fills this gap.

Kornai has summed up well our basic point. Speaking of formal systems of phonology and morphology, including recent work on autosegmental phonology at Edinburgh, he says ([32], p. 8):

> However, these systems do not offer a formal *reconstruction* of mainstream

generative phonology, they offer formal *alternatives*. Because they explicitly reject one or more of the fundamental assumptions underlying the sequential mode of rule application used in the vast majority of generative phonological analyses, they do not make it possible to restate the linguists' work in a formal setting — in order to enjoy the benefits of the formal rigor offered by these systems one must reanalyze the data.

8.4.3. Complexity of Automata

One feature that our model shares with Bird & Ellison's is complicated automata. In both cases it is easy to give small examples, but the number of states and transitions in an automata quickly grows as the examples get bigger. For example, they give a (simplified) four-state automata on page 27 that describes a two-tiered representation with four segments and two association lines. As the number of tiers, segments, and association lines increases, however, the corresponding automaton gets considerably larger. As is said on page 24, "Clearly, such automata have the potential for getting complicated rather quickly."

If we were to multiply out all the factor machines in our model of an autosegmental rule, we would end up with a very large automata which would be hard to understand. These automata are best understood by analyzing them into their components, as we have done.

In general, our automata tend to be more complicated than the automata that do an equivalent job for Bird & Ellison. One gets a feel for this from a reading of the example in Chapter 6 of modelling the High Tone Shift rule. From this, one could imagine constructing a number of rules (including the Association Convention) that would be required to describe the Arabic phonology that they model in Section 5.3 of their paper with three very simple automata.

The number of states in a disjoint product can grow proportionally with the product of the number of states in each machine. In the case of the product operation, the number of states in each partition can be as large as the product of the number of states in those partitions in the factor machines. However, the total number of states and transitions in each case also depends on the compatibility of the states in the machines being multiplied. A product machine can be smaller (in terms of number of states and transitions) than its factor machines in many cases, if few states are compatible.

There are some techniques that can be used to simplify automata by eliminating unnecessary states and transitions. This is another area for further investigation. The application of such techniques to any model that takes products of automata would clearly be beneficial.

8.5. Model and Theory

As always, constructing a formal model of a theory has led to clarifications of (or at least to clarification questions about) the theory. In our case, these had to do with the precise working of autosegmental rules. They include:

- Should a well-formedness condition be interpreted consistently as a constraint that blocks the application of a rule, or should it be interpreted sometimes (in language-specific rules) as a mechanism that repairs violations of the condition? We chose the former.
- In the case of the well-formedness condition that limits the number of associations per freely associating segment, if automatic repairs are allowed, how should they be done? This is unclear in some cases.
- Should a rule be blocked from applying in a situation where it would add an association line where there is one already? This seems reasonable.
- If a segment is deleted which has associations to other tiers not mentioned in a rule, should the rule be blocked, or should those associations be deleted along with the segment?
- How exactly does the Association Convention work? When there are more tones

than vowels, should all the tones be associated, or should multiple associations only be done by language-specific rules? When there is more than one possible way to add association lines, how does one decide between the options?

These questions must be answered before autosegmental rules can be given a computational interpretation. They ought also to be answered for the sake of a clearer definition of the phonological theory. If there are competing answers, these at least should be made clear.

Answers to these questions, especially the last three, will require co-operative work from theoretical phonologists and working phonologists, because the answers have empirical consequences. Answers need to be agreed upon on the basis of sound theoretical principles as well as on the basis of what works over a wide range of data.

8.6. Conclusions

The main result of this thesis is the definition of a new kind of transducer, more powerful than an FST, and a demonstration of its adequacy to computationally implement autosegmental rules. We have argued that this extra computational power is needed to capture significant generalizations that are central to autosegmental theory, something that FSTs processing linearized autosegmental representations cannot do.

There are at least two possible directions to go from here. It may be that MSFTs are enough to implement the entire standard theory of autosegmental phonology, by simulating the effect of ordered sequences of rules in a single transducer. This was Kornai's goal, and we also set out in this direction.

On the other hand, we could replace the finite state transducers in the KIMMO system with MSFTs, yielding a modified, more powerful form of two-level phonology. This would enable KIMMO to process non-linear representations, bringing it up to date with current phonological practice.

The primary motivation behind this work was the desire to provide a practical computational tool for phonological analysis. MSFTs, now 33 years old, have turned their hand to non-linear phonology and proved themselves quite capable. They have earned a central place in a computational phonological tool.

Appendix A
Properties of Encodings

In this appendix we prove some theorems that were stated in the main text (Chapter 4), theorems about various properties of encodings of autosegmental representations.

Theorem 4-1: The multi-linear encoding function E_n is invertible (1-1).

Proof: Let $A \in \mathbf{A}_n$. Suppose we are given the encoding $E_n(A) = <e_1, ..., e_n>$. From this encoding we can define $B \in \mathbf{A}_n$ such that $E_n(B) = E_n(A)$ and A is isomorphic (equal) to B. This shows that any A with multi-linear code $<e_1, ..., e_n>$ is equal to B, so E_n is invertible.

First, let us define B. Notice that the alphabet Σ in the definition of the multi-linear code (Definition 4-5) is the disjoint union of two sets S and $\{\alpha_{ij} \mid i,j = 1,...,n\}$. By distinguishing between symbols in these two sets[1], we can define B as follows:

- subdivide each string e_i into $p(i)$ substrings $f_{i1}, f_{i2}, ..., f_{ip(i)}$, where $p(i)$ is the number symbols from S in e_i, in such a way that only the first symbol in each f_{ik} (call it s_{ik}) is from S, and $f_{i1}f_{i2}...f_{ip(i)} = e_i$[2]

1. Notice that even if we were not given these sets *a priori*, we could, by examining how the symbols are used in the encoding, determine not only which symbols belong to S and which to $\{\alpha_{ij}\}$, but also, for the α_{ij}, which are tiers i and j. This is because the symbols in S are never used on more than one tier (assuming we uphold Goldsmith's requirement that features cannot appear on more than one tier), while the α_{ij} are each used on exactly two tiers — the ones indicated by their subscripts.
2. In terms of our example, the elements from S were letters, and the $\{\alpha_{ij} \mid i,j = 1,...,n\}$ were numbers, so we are dividing ei into substrings that start with a letter and end with only numbers.

- define the vertex set of B as $V = \{v_{ik} \mid i=1,\ldots,n;\ k=1,\ldots,p(i)\}$
- define the labelling function of B by $L(v_{ik}) = s_{ik}$
- define the partitions of V as $<V_1, \ldots, V_n>$ where $V_i = \{v_{ik} \mid k=1,\ldots,p(i)\}$
- define the orderings of the partitions in the obvious way: $\forall\ i,\ v_{i1} <_i v_{i2} <_i \ldots <_i v_{ip(i)}$ (this ordering keeps the labels in the same order that they appear in the encoding)

Construct the edge set E of B as follows: for each pair of tiers i and j $(i \neq j)$, begin from the left edge of e_i and e_j, scanning rightward simultaneously in each string, looking for the symbol α_{ij} in both strings, matching up pairs of them in left to right order. For each pair, if an α_{ij} symbol is contained in f_{ik} in e_i, and in f_{jl} in e_j, then add the edge (v_{ik}, v_{jl}) to E. Such edges are all the edges in E.

The reader can verify that $E_n(B) = E_n(A) = <e_1, \ldots, e_n>$, using Definition 4-5. It is also straightforward to construct the isomorphism between A and B (Definition 4-4) that shows that A = B. ❑

Theorem 4-2: The multi-linear encoding function E_n is compositional.

Proof: Let A, B, C $\in A_n$, and suppose C = AB. Then, by Definition 4-6, we can write the vertices of C in each partition i as $v_{i1} < v_{i2} < \ldots < v_{ip(i)}$, where A's ith partition contains exactly the vertices $v_{i1} < \ldots < v_{iq(i)}$ and B's ith partition contains exactly the vertices $v_{iq(i)+1} < \ldots < v_{ip(i)}$, and where the vertex labels and the edges of C are those of A and B combined. Let $E_n(A) = <e_1, \ldots, e_n>$ and $E_n(B) = <f_1, \ldots, f_n>$. Then $E_n(A)E_n(B) = <e_1f_1, \ldots, e_nf_n>$. But also $E_n(C) = <e_1f_1, \ldots, e_nf_n>$, because: (a) this encoding contains all the vertex labels in the proper order in the appropriate strings, and (b) for each pair of strings e_if_i and e_jf_j in the encoding (corresponding to a chart), the symbols α_{ij} in those strings still match up in pairs from left to right in the same way they did in $E_n(A)$ and $E_n(B)$, meaning that they will still represent the same edges that they did before. This latter statement (b) rests on the simple observation that in well-formed encodings like $E_n(A)$ and $E_n(B)$, the number of α_{ij}'s in the i^{th} and j^{th} strings are always equal, so that concatenating well-formed encodings will not change the way the α_{ij}'s match up in pairs. Thus the encoding $E_n(A)E_n(B)$ will represent exactly all the association lines of A and B combined. ❑

Lemma 4-1: Let w and x be non-empty strings in Σ^*. Then wx = xw iff there exists a "common factor" that "divides" both strings; that is, there is a string f such that $w = f^i$ for some i and $x = f^j$ for some j.

Proof: A moment's consideration will convince the reader that wx = xw is a very strong constraint on w and x. Not just any non-empty strings can be concatenated in either order and produce the same result. This idea of having a "common factor" exactly characterizes those strings that satisfy this constraint. The "if" direction of the proof is easy: if $w = f^i$ and $x = f^j$, then certainly wx = xw, for $f^i f^j = f^j f^i$. The "only if" direction is more interesting, and is the one that will be needed to prove Theorem 4-3 below.

If $|w| = |x|$ then $wx = xw \Rightarrow w = x$, and our common factor is f = w = x. Assume without loss of generality that $|w| > |x|$. We will prove the lemma by strong induction on the length of the shortest string (ie. x). The base case will be $|x| = 1$; we start with the induction step, where $|x| > 1$.

Let $d = |w|$ div $|x|$ [3] and $r = |w| \bmod |x|$. Then $|w| = d|x| + r$.

(61)
$$\begin{aligned} wx = xw &\Rightarrow \text{w starts with x} &&\Rightarrow wx = x(xw_1) \ \exists\ w_1 \\ &\Rightarrow \text{w starts with xx} &&\Rightarrow wx = x(xxw_2) \ \exists\ w_2 \\ &\Rightarrow \ldots \\ &\Rightarrow \text{w starts with } x^{d-1} &&\Rightarrow wx = x(x^{d-1}w_{d-1}) = x^d w_{d-1} \ \exists\ w_{d-1} \end{aligned}$$

3. $|w|$ div $|x|$ is the integer part of $|w| / |x|$.

This means $w_{d-1} = vx$ for some v, where $|v| = r$, and $w = x^d v$. Now if $r = 0$, then $v = \varepsilon$, $w = x^d$, and $f = x$ is our common factor. If $r \neq 0$, then $v \neq \varepsilon$, and:

$$wx = xw \Rightarrow x^d vx = xx^d v \Rightarrow vx = xv$$

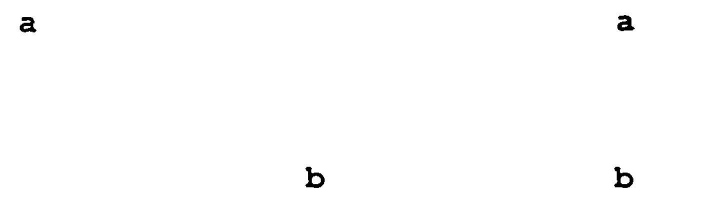
We are almost back where we started ($wx = xw$), but now we have smaller strings, since $|v| = r < |x| < |w|$. By our strong inductive hypothesis, x and v have a common factor f. Since f "divides" both v and x, it will "divide" $w = x^d v$ as well, so f is the common factor of w and x that we are looking for.

For the base case of the induction, where $|x| = 1$, the line of reasoning used in (61) will show that x is the common factor f, since r must necessarily be 0. ❑

Theorem 4-3: An invertible linear encoding of A_n cannot be compositional.

Proof: Let E: $A_n \rightarrow \Sigma^*$ be a 1-1 linear encoding of A_n. Assume that E is also compositional; we will derive a contradiction from this. Consider the following two representations A and B (each having two tiers, with one tier empty), together with their concatenation:[4]

(62)



A B AB = BA

4. This is not just a theoretical pathological case. One would find representations like this being concatenated in, for example, tonal languages that have morphemes consisting of just a single tone feature, and no other phonological material. A model of autosegmental phonology must be able to account for phenomena like this.

Let E(A) = w and E(B) = x, where w and x are strings in Σ^*. Note that since E is compositional and invertible, only the empty representation $\varnothing$ is encoded by the empty string ε,[5] and thus $w \neq \varepsilon$ and $x \neq \varepsilon$. Also, since E is compositional and AB = BA, we have wx = E(A)E(B) = E(AB) = E(BA) = E(B)E(A) = xw.

By Lemma 4-1, w and x have a common factor f. In fact, by the same lemma, any two representations that can be concatenated in either order with the same result (as A and B can), will have encodings under E that share a common factor. Furthermore, we will go on to show that *all* the representations that have an empty bottom tier have encodings that share a *single* common factor. (The same is true for all representations with an empty top tier).

Let T be the set of representations with an empty bottom tier. $A \in T$. Let $A' \in T$. Then E(A) and E(B) share a common factor f, and E(A') and E(B) share a common factor g. Thus, $\exists\ i, j$ such that $E(B) = f^i = g^j$. We will show below that f and g themselves share a common factor h, which will of course be a common factor of E(A) and E(A'). Then we can pick another $A'' \in T$, and in the same way show that E(A) and E(A'') have a common factor which is a factor of h. This argument can be repeated ad infinitum to each element of T in turn, yielding a common factor of all the representations in T.

Given that $f^i = g^j$, it follows that f and g share a common factor. Consider placing the two strings f^i and g^j on top of each other and observing how the various copies of the factor f overlap the copies of the the factor g. Overlapping characters, of course, must be equal. If we number the characters of g from 0 through $|g|-1$, then the first character of f will overlap at the following positions within various copies of g:

5. $E(A) = \varepsilon \Rightarrow E(AB) = E(A)E(B) = E(B) \Rightarrow AB = B \Rightarrow$ A is the empty representation $\varnothing$.

0|f| mod |g|

1|f| mod |g|

2|f| mod |g|

...

$(i\text{-}1)$ |f| mod |g|

It is a simple result of number theory that this set of numbers (eliminating duplicates) is {0, *d*, 2*d*, 3*d*, ..., |g|-*d*}, where *d* = gcd(|f|, |g|). This means that a copy of f overlaps a copy of g at each of these positions. Let h be the substring consisiting of the first *d* characters of f, or equivalently, of g. Then h is a common factor of f and g.

We have now shown that every representation in T shares some single common factor, call it ϕ. Finally now, we can arrive at a contradiction relating to this alleged invertible, compositional, linear encoding E. Consider the representations below.

(63)

c	**d**	**cd**	**dc**
C	D	CD	DC

We know that E(C) = ϕ^k for some *k*, and E(D) = ϕ^l for some *l*. But then E(CD) = E(C)E(D) = $\phi^k\phi^l = \phi^l\phi^k$ = E(D)E(C) = E(DC), even though CD ≠ DC; ie. E is not invertible. This is a contradiction; hence, there is no such thing as a 1-1, compositional, linear encoding of autosegmental representations. ❑

Bibliography

[1] Aho, A.V. (1968). Indexed Grammars — an Extension of Context Free Grammars, *Journal of the Association for Computing Machinery* 15: 647-671.

[2] Aho, A.V. (1969). Nested Stack Automata. *Journal of the Association for Computing Machinery* 16: 383-406.

[3] Antworth, Evan L. (1990). *PC-KIMMO: A Two-level Processor for Morphological Analysis.* Summer Institute of Linguistics, Dallas, Texas. Occasional Publications in Academic Computing, Number 16.

[4] Archangelli, Diana and Pulleyblank, Douglas. *Grounded Phonology.* To appear.

[5] Bird, M. (1973). The equivalence problem for deterministic 2-tape automata. *Journal of Comput. and Systems Science* 7: 218-236.

[6] Bird, Steven and Klein, Ewan (1990). Phonological Events. *Journal of Linguistics* 26: 33-56.

[7] Bird, Steven and Ladd, D.Robert (1991). Presenting Autosegmental Phonology. *Journal of Linguistics* 27(1): 193-210.

[8] Bird, Steven and Ellison, T. Mark (1992). One Level Phonology: Autosegmental Representations and Rules as Finite-State Automata. Edinburgh Research Papers in Cognitive Science, EUCCS/RP-51, Edinburgh University, April1992.

[9] Bird, Steven (1992). Finite-State Phonology in HPSG. *Proceedings of Coling '92*, Nantes, July 1992.

[10] Chomsky, Noam and Halle, Morris (1968). *The Sound Pattern of English.* New York: Harper and Row.

[11] Church, Kenneth (1983). A Finite-State Parser for Use in Speech Recognition. *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, 91-97.

[12] Coleman, John and Local, John (1991). The 'No Crossing Constraint' in Autosegmental Phonology. *Linguistics and Philosophy* 14: 295-338.

[13] Culik, K. and Karhumaki, J. (1989). HDTOL Matching of Computations of Multitape Automata. *Acta Informatica*, Vol. 27, No. 2: 179ff.

[14] Denning, Peter; Dennis, Jack; and Qualitz, Joseph (1978). *Machines, Languages, and Computation.* Prentice-Hall: Englewood Cliffs, New Jersey.

[15] Durand, Jacques (1990). *Generative and Non-Linear Phonology.* New York: Longman.

[16] Elgot, C.C. and Mezei, J.E. (1965). On Finite Relations Defined by Generalized Automata. *IBM Journal of Research and Development* 9: 47-68.

[17] Fischer, Patrick C. (1965). Multi-Tape and Infinite-State Automata — A Survey. *Communications of the ACM* 8: 799-805.

[18] Gibbon, Dafydd (1987). Finite State Processing of Tone Systems. *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*: 291-297.

[19] Goldsmith, John (1976). An Overview of Autosegmental Phonology. *Linguistic Analysis* 2: 23-68.

[20] Goldsmith, John (1979). *Autosegmental Phonology.* PhD dissertation, MIT, 1976. Distributed by IULC. New York: Garland Press.

[21] Goldsmith, John (1982). Accent Systems. *The Structure of Phonological Representations (Part I)*, eds. Harry van der Hulst and Norval Smith: 47-63. Dordrecht: Foris Publications.

[22] Goldsmith, John (1990). *Autosegmental and Metrical Phonology.* Cambridge, MA: Blackwell.

[23] Hopcroft, John E. and Ullman, Jeffrey D. (1979). *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley.

[24] Hulst, Harry van der and Smith, Norval (1982). An Overview of Autosegmental and Metrical Phonology. *The Structure of Phonological Representations (Part I)*: 1-45. Dordrecht: Foris Publications.

[25] Johnson, C. D. (1972). *Formal Aspects of Phonological Description.*Mouton.

[26] Jones, Doug (1988). *Models of Phonological Grammars.* M.A. Thesis, Stanford University.

[27] Katamba, Francis (1989). *An Introduction to Phonology.* Learning about Language Series. New York: Longman.

[28] Kay, Martin (1983). When Meta-Rules Are Not Meta-Rules. *Automatic Natural Language Parsing*, eds. Karen Sparck Jones and Yorick Wilks: 94-116. Chichester: Ellis Horwood Ltd. See pages 100-104.

[29] Kay, Martin (1987). Nonconcatenative Finite-State Morphology. *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*: 2-10.

[30] Kay, Martin and Kaplan, Ronald (1981). Phonological Rules and Finite-State Transducers. Unpublished conference paper.

[31] Kinber, E. (1983). The inclusion problem for some classes of deterministic multitape automata. *Theoretical Computing Science* 26: 1-24.

[32] Kornai, Andras (1991). *Formal Phonology.* PhD dissertation, Stanford University, CA.

[33] Koskenniemi, Kimmo (1983). *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production.* PhD dissertation, University of Helsinki.

[34] Rabin, M.O. and Scott, D. (1959). Finite Automata and Their Decision Problems. *IBM Journal of Research and Development 3*: 114-125.

[35] Ritchie, Graeme (1989). On the Generative Power of Two-Level Morphological Rules. *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*: 51-57.

[36] Ritchie, Graeme (1992). Languages Generated by Two-Level Morphological Rules. *Computational Linguistics* 18(1): 41-59

[37] Ritchie, Graeme D., Black, Alan W., Russell, Graham J., and Pulman, Stephen G. (1992). *Computational Morphology: Practical Mechanisms for the English Lexicon.* ACL-MIT Press Series in Natural Language Processing, eds. Aravind K. Joshi, Karen Sparck Jones, and Mark Y. Liberman. MIT Press, Cambridge, Massachusetts.

[38] Rosenberg, A.L. (1964). On *n*-tape Finite State Acceptors. *Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, Princeton, 1964: 76-81.

[39] Sagey, Elizabeth (1988). On the Ill-Formedness of Crossing Association Lines. *Linguistic Inquiry* 19:109-118.

[40] Sproat, Richard (1991). PC-KIMMO: A Two-level Processor for Morphological Analysis (Review). *Computational Linguistics* 17(2): 229-231.

[41] Wiebe, Bruce (1992). An Efficient Linear Encoding of Autosegmental Representations. Abstract presented at the *DIMACS Workshop on Human Language*, Princeton University, Princeton, New Jersey, March 20-22.