OXFORD SURVEYS IN SYNTAX AND MORPHOLOGY

PUBLISHED

1. *Grammatical Relations*
Patrick Farrell
2. *Morphosyntactic Change*
Olga Fischer
3. *Information Structure: the Syntax-Discourse Interface*
Nomi Erteschik- Shir
4. *Computational Approaches to Morphology and Syntax*
Brian Roark and Richard Sproat
5. *Constituent Structure*
Andrew Carnie

IN PREPARATION

The Acquisition of Syntax and Morphology
Shanley Allen and Heike Behrens

The Processing of Syntax and Morphology
Ina Bornkessel-Schlesewsky and Matthias Schlesewesky

Morphology and the Lexicon
Daniel Everett

Grammatical Relations
Revised second edition
Patrick Farrell

The Phonology–Morphology Interface
Sharon Inkelas

The Syntax–Semantics Interface
Jean-Pierre Koenig

Complex Sentences
Toshio Ohori

Syntactic Categories
by Gisa Rauh

Language Universals and Universal Grammar
Anna Siewierska

Argument Structure: The Syntax–Lexicon Interface
Stephen Weschler

Computational Approaches to Morphology and Syntax

BRIAN ROARK AND RICHARD SPROAT

OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS
Great Clarendon Street, Oxford OX2 6DP
Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship,
and education by publishing worldwide in
Oxford New York
Auckland Cape Town Dar es Salaam Hong Kong Karachi
Kuala Lumpur Madrid Melbourne Mexico City Nairobi
New Delhi Shanghai Taipei Toronto
With offices in
Argentina Austria Brazil Chile Czech Republic France Greece
Guatemala Hungary Italy Japan Poland Portugal Singapore
South Korea Switzerland Thailand Turkey Ukraine Vietnam
Oxford is a registered trademark of Oxford University Press
in the UK and in certain other countries
Published in the United States
by Oxford University Press Inc., New York

First published 2007

British Library Cataloguing in Publication Data
Data available
Library of Congress Cataloging in Publication Data
Data available
Typeset by SPI Publisher Services, Pondicherry, India
Printed in Great Britain
on acid-free paper by
Biddles Ltd., King's Lynn, Norfolk
ISBN 978–0–19–927477–2 (Hbk.)
978–0–19–927478–9 (Pbk.)
1 3 5 7 9 10 8 6 4 2

Contents

General Preface

Oxford Surveys in Syntax and Morphology provides overviews of the major approaches to subjects and questions at the centre of linguistic research in morphology and syntax. The volumes are accessible, critical, and up-to-date. Individually and collectively they aim to reveal the field's intellectual history and theoretical diversity. Each book published in the series will characteristically contain: (1) a brief historical overview of relevant research in the subject; (2) a critical presentation of approaches from relevant (but usually seen as competing) theoretical perspectives to the phenomena and issues at hand, including an objective evaluation of the strengths and weaknesses of each approach to the central problems and issues; (3) a balanced account of the current issues, problems, and opportunities relating to the topic, showing the degree of consensus or otherwise in each case. The volumes will thus provide researchers and graduate students concerned with syntax, morphology, and related aspects of semantics with a vital source of information and reference.

The current volume, *Computational Approaches to Morphology and Syntax* by Brian Roark and Richard Sproat, is the first volume in the series to examine analyses of morphology and syntax from outside of linguistics proper, in this case from computer science. The discussion presupposes a basic background in computational linguistics and not only surveys various computational procedures but also draws out their implications for morphological and syntactic theories.

Robert D. Van Valin, Jr
General Editor

Heinrich Heine University, Düsseldorf

University at Buffalo, The State University of New York

Preface

This book provides an introduction to the current state-of-the-art in computational modeling of syntax and morphology, with a particular focus on the computational models that are used for these problems. As such it is not intended as a "cookbook". The reader should not in general expect to come away knowing how a particular system is implemented; rather he or she should expect to understand the general algorithms that are involved in implementing such a system. So, for example, in our treatment of computational models of morphology, we shall focus largely on the regular operations that are involved in describing morphological phenomena. In principle there are a number of different ways that one might implement a morphological analyzer for a particular language, so rather than focus on such particulars, we prefer to give the reader an understanding of the formal operations that any such system would either implement directly or be formally equivalent to.

Where possible we shall attempt to show the relevance of computational models to theoretical questions in linguistics. There has traditionally been a bit of a disconnect between theoretical and computational linguistics. Computational linguists have often been uninterested in the subtler questions that trouble theoretical linguists, while theoretical linguists have often been unimpressed by computational arguments for one or another position. This state of affairs is poised to change as larger and larger sources of linguistic data become available for linguists to test theoretical claims, and more and more well-designed linguistic features are needed for ever more sophisticated machine learning algorithms.

Needless to say we have not been able to cover all topics that are relevant to the theme of the book. In most cases, we hope, this is not because we are ignorant of said topics, but merely because we have chosen to emphasize certain aspects of the problem over others. Where possible and relevant, we have justified such omissions.

Computational linguistics is an interdisciplinary field. As anyone who has taught an introductory course on this topic can attest, it is a real challenge to uniformly engage students from different disciplinary

backgrounds. Every attempt has been made to make the material in this book broadly accessible, but many readers will find unfamiliar notation of one form or another. For some, linguistic glosses for an obscure language may be intimidating; for others, pseudocode algorithms with operations on sets or vectors. In many cases, apparently complex notation is actually relatively simple, and is used to facilitate presentation rather than complicate it. We encourage readers to spend the time to understand unfamiliar notation and work through examples.

There are a number of people who have not been involved in the production of this book, but who have had a profound influence on our understanding of the issues that we discuss, and we would like to acknowledge their influence here. First and foremost we would like to thank the former AT&T Labs researchers–Michael Riley, Mehryar Mohri, Fernando Pereira, and Cyril Allauzen–who were instrumental in the creation of the AT&T weighted finite-state tools upon which much of our own work has been based. We would also like to acknowledge Mark Johnson, Eugene Charniak, and Michael Collins for insightful discussions of issues related to syntactic processing, language modeling, and machine learning. Ken Church has had an important influence on the field of computational linguistics as a whole, and directly or indirectly on our own work.

On the material presented here, we have received helpful feedback from Lauri Karttunen, Dale Gerdemann, audiences at the Workshop on Word-Formation Theories (25–26 June, 2005, Prešov, Slovakia) and the Université du Québec à Montréal, and classes at the University of Illinois at Urbana-Champaign and Oregon Health & Science University. Thanks to Emily Tucker, Meg Mitchell, and Kristy Hollingshead for detailed comments on early drafts. We have also benefited from comments from two anonymous reviewers for Oxford University Press.

Finally, we would like to thank our editors, Robert Van Valin at the University at Buffalo and John Davey at Oxford University Press, for their support and patience with our continual delays in bringing this work to completion.

List of Figures

List of Tables

Abbreviations

ABL	ablative
ACC	accusative
Adj	Adjective
Adv	Adverb
argmax	the value resulting in the highest score
argmin	the value resulting in the lowest score
ASR	Automatic Speech Recognition
BITG	Bracketing ITG
CCG	Combinatory Categorial Grammar
CFG	Context-free Grammar
CKY	variant of CYK
CLR	function tag in Penn Treebank (close-related)
CNF	Chomsky Normal Form
COMP	complement
CRF	Conditional Random Fields
CS	Context Similarity
CYK	Cocke-Younger-Kasami dynamic programming parsing algorithm
DAT	dative
DATR	a language for lexical knowledge representation
DECOMP	a module of the MITalk text-to-speech synthesis system
DEF	definite
Det	Determiner
DFA	Deterministic Finite Automaton
DOP	Data Oriented Parsing
DP	Determiner phrase
DT	Determiner POS-tag (Penn Treebank)
EM	Expectation Maximization
exp	exponential
FEM	feminine
FSA	Finite-state Automaton
FSM	Finite-state Machine
FSRA	Finite-state Registered Automaton
FST	Finite-state Transducer
fut	future
FV	Final vowel

GCFG	Generalized Context-free Grammar
GEN	Candidate generation mechanism in OT
GEN	genitive
HMM	Hidden Markov Model
HPSG	Head-driven Phrase Structure Grammar
iff	if and only if
INF	infinitive
IPA	International Phonetic Alphabet
ITG	Inversion Transduction Grammars
JJ	Adjective POS-tag (Penn Treebank)
KATR	Kentucky version of DATR
KIMMO	Koskenniemi's first name, used first by Lauri Karttunen to refer to the two-level morphology system that Koskenniemi invented
LC	left context
LCFRS	Linear context-free rewriting systems
LFG	Lexical Functional Grammar
LHS	Left-hand side of a context-free production
LL(k)	Left-to-right, Leftmost derivation with k lookahead items
log	logarithm
LP	Labeled precision
LR	Labeled recall
LR(k)	Left-to-right, Rightmost derivation with k lookahead items
LS	Levenshtein similarity
MAS	masculine
max	maximum
MaxEnt	Maximum Entropy
MCTAG	Multicomponent TAG
MD	Modal verb POS-tag (Penn Treebank)
MDL	Minimum description length
min	minimum
ML	Maximum Likelihood
MT	Machine Translation
N	Noun
NCS	Normalized cosine score
NER	Named entity recognition
NLP	Natural Language Processing
NN	common noun POS-tag (Penn Treebank)
NNP	proper noun POS-tag
NNS	plural common noun POS-tag (Penn Treebank)
NOM	nominative

NP	noun phrase
NUM	number
OT	Optimality Theory
PA	Pushdown Automaton
PARC	Palo Alto Research Center
PARSEVAL	Parse evaluation metrics
PC	Personal computer
PCFG	Probabilistic CFG
PF	paradigm function
PL	plural
POS	Part-of-speech
PP	Prepositional phrase
PPMV	pair of potential morphological variants
RB	Adverb POS-tag (Penn Treebank)
RC	right context
RHS	Right-hand side of a context-free production
RR	realization rule
<s>	beginning of string
</s>	end of string
S	sentence
SBAR	subordinate clause
SG	Singular
SLM	Structured language model
SPE	Sound Pattern of English
SUBJ	Subject
SuperARV	a finite-state class-based language modeling approach
TAG	Tree-adjoining Grammar
TDP	Top-down parsing
TMP	temporal
V	Verb
VB	Infinitival verb POS-tag (Penn Treebank)
VBD	Past tense verb POS-tag (Penn Treebank)
VBG	Gerund verb POS-tag (Penn Treebank)
VBN	Past participle verb POS-tag (Penn Treebank)
VBP	Non-third person singular present verb POS-tag (Penn Treebank)
VBZ	Third person singular present verb POS-tag (Penn Treebank)
VP	Verb phrase
WCFG	Weighted CFG
WFSA	Weighted FSA
WFST	Weighted FST

WSJ	Wall St. Journal
XFST	Xerox FST tools
XLE	Xerox Linguistic Environment
XTAG	TAG grammar from University of Pennsylvania

1

Introduction and Preliminaries

1.1 Introduction

Computational approaches to morphology and syntax are generally concerned with formal devices, such as grammars and stochastic models, and algorithms, such as tagging or parsing. They can range from primarily theoretical work, looking at, say, the computational complexity of algorithms for using a certain class of grammars, to mainly applied work, such as establishing best practices for statistical language modeling in the context of automatic speech recognition. Our intention in this volume is to provide a critical overview of the key computational issues in these domains along with some (though certainly not all) of the most effective approaches taken to address these issues. Some approaches have been known for many decades; others continue to be actively researched.

In many cases, whole classes of problems can be addressed using general techniques and algorithms. For example, finite-state automata and transducers can be used as formal devices for encoding many models, from morphological grammars to statistical part-of-speech taggers. Algorithms that apply to finite-state automata in general apply to these models. As much as possible, we will present specific computational approaches to syntax and morphology within the general class to which they belong. Much work in these areas can be thought of as variations on certain themes, such as finite-state composition or dynamic programming.

The book is organized into two parts: approaches to morphology and approaches to syntax. Since finite-state automata and transducers will figure prominently in much of the discussion in this book, in this chapter we introduce the basic properties of these devices as well as some of the algorithms and applications. For reasons of space we will only provide a high-level overview, but we will give enough references to

recent work on finite automata and their applications so that interested readers can follow up on the details elsewhere.

One thing we hope to convey here is how to think of what automata compute in *algebraic* and *set-theoretic* terms. It is easy to get lost in the details of the algorithms and the machine-level computations. But what is really critical in understanding how finite automata are used in speech and language processing is to understand that they compute relations on sets. One of the critical insights of the early work by Kaplan and Kay from the 1970s (reported finally in Kaplan and Kay, 1994) was that in order to deal with complex problems such as the compilation of context-sensitive rewrite rules into transducers, one has to abandon thinking of the problem at the machine level and move instead to thinking of it at the level of what relations are being computed. Just as nobody can understand the wiring diagram of an integrated circuit, neither can one really understand a finite automaton of any complexity by simply looking at the machine. However, one can understand them easily at the algebraic level, and let algorithms worry about the details of how to compile that algebraic description into a working machine.

We assume that readers will be at least partly familiar with basic finite-state automata so we will only briefly review these. One can find reviews of the basics of automata in any introduction to the theory of computation such as Harrison (1978), Hopcroft and Ullman (1979), and Lewis and Papadimitriou (1981).

1.2 Finite-State Automata and Transducers

The study of finite-state automata (FSA) starts with the notion of a *language*. A language is simply a set of expressions, each of which is built from a set of symbols from an *alphabet*, where an alphabet is itself a set: typical alphabets in speech and language processing are sets of letters (or other symbols from a writing system), phones, or words.

The languages of interest here are *regular languages*, which are languages that can be constructed out of a finite alphabet – conventionally denoted Σ – using one or more of the following operations:

- set union denoted "$\cup$" e.g., $\{a, b\} \cup \{c, d\} = \{a, b, c, d\}$
- concatenation denoted "$\cdot$" e.g., $abc \cdot def = abcdef$
- transitive closure (Kleene star) denoted "*" e.g., a^* denotes the set of sequences consisting of 0 or more a's

TABLE 1.1 Phrasal reduplication in Bambara

wulu	*o*	*wulu*	"whichever dog"
dog	MARKER	dog	
wulunyinina	*o*	*wulunyinina*	"whichever dog
dog searcher	MARKER	dog searcher	searcher"
malonyininafilèla	*o*	*malonyininafilèla*	"whichever rice
rice searcher watcher	MARKER	rice searcher watcher	searcher watcher"

Any finite set of strings from a finite alphabet is necessarily a regular language, and using the above operations one can construct another regular language by taking the union of two sets A and B; the concatenation of two or more such sets (i.e. the concatenation of each string in A with each string B); or by taking the transitive closure (i.e. zero or more concatenations of strings from set A).

Despite their simplicity, regular languages can be used to describe a large number of phenomena in natural language including, as we shall see, many morphological operations and a large set of syntactic structures. But there are still linguistic constructions that cannot be described using regular languages. One well-known case from morphology is phrasal reduplication in Bambara, a language of West Africa (Culy, 1985), some examples of which are given in Table 1.1. Bambara phrasal reduplication constructions are of the form *X-o-X*, where *-o-* is a marker of the construction and X is a nominal phrase. The problem is that the nominal phrase is in theory unbounded, and so the construction involves unbounded copying. Unbounded copying cannot be described in terms of regular languages; indeed it cannot even be described in terms of context-free languages (which we will return to later in the book).

A couple of important regular languages are the *universal language* (denoted Σ^*) which consists of all strings that can be constructed out of the alphabet Σ, including the empty string, which is denoted ϵ; and the *empty language* (denoted $\emptyset$) consisting of no strings.

The definition given above defines some of the *closure properties* for regular languages but regular languages are also closed under the following operations:

- intersection denoted "$\cap$" e.g., $\{a, b, c\} \cap \{c, d\} = \{c\}$
- difference denoted "$-$" e.g., $\{a, b, c\} - \{c\} = \{a, b\}$
- complementation denoted "$\overline{X}$" e.g., $\overline{\mathrm{A}} = \Sigma^* - \mathrm{A}$
- string reversal denoted "X^R" e.g., $(abc)^R = cba$

Regular languages are commonly denoted via *regular expressions*, which involve the use of a set of reserved symbols as notation. Some of these reserved symbols we have already seen, such as "*", which denotes "zero or more" of the symbol that it follows: recall that a^* denotes the (infinite) set of strings consisting of zero or more a's in sequence. We can denote the repetition of multi-symbol sequences by using a parenthesis delimiter: $(abc)^*$ denotes the set of strings with zero or more repetitions of abc, that is, $\{\emptyset, abc, abcabc, abcabcabc, \ldots\}$. The following summarizes several additional reserved symbols used in regular expressions:

- "zero or one" denoted "?" e.g.,(abc)? denotes $\{\emptyset, abc\}$
- disjunction denoted "|" or $\cup$ e.g., $(a \mid b)$? denotes the set of strings with zero or one occurrence of either a or b, i.e., $\{\emptyset, a, b\}$
- negation denoted "$\neg$" e.g., $(\neg a)^*$ denotes the set of strings with zero or more occurrences of anything other than a

This is a relatively abbreviated list, but sufficient to understand the regular expressions used in this book to denote regular languages.

Finite-state automata are computational devices that compute regular languages. Formally defined:

Definition 1 *A finite-state automaton is a quintuple* $M = (Q, s, F, \Sigma, \delta)$ *where:*

1. Q *is a finite set of states*
2. s *is a designated initial state*
3. F *is a designated set of final states*
4. Σ *is an alphabet of symbols*
5. δ *is a transition relation from* $Q \times (\Sigma \cup \epsilon)$ *to* Q, *where* $A \times B$ *denotes the cross-product*[1] *of sets A and B*

Kleene's theorem states that every regular language can be recognized by a finite-state automaton; similarly every finite-state automaton recognizes a regular language.

[1] The cross-product of two sets creates a set of pairs, with each member of the first set paired with each member of the second set. For example, $\{a, b\} \times \{c, d\} = \{\langle a, c\rangle, \langle a, d\rangle, \langle b, c\rangle, \langle b, d\rangle\}$. Thus the transition relation δ is from state/symbol pairs to states.

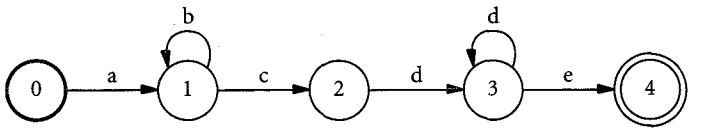

FIGURE 1.1 A simple finite-state automaton accepting the language ab^*cdd^*e

A diagram of a simple finite-state automaton, which accepts the language ab^*cdd^*e, is given in Figure 1.1. A string, say *abbcddde*, that is in the language of the automaton is matched against the automaton as follows: starting in the initial state (here, state 0), the match proceeds by reading a symbol of the input string and matching it against a transition (or arc) that leaves the current state. If a match is found, one moves to the destination state of the arc, and tries to match the next symbol of the input string with an arc leaving that state. If one can follow a path through the automaton in such a manner and end in a final state (denoted here with a double circle) with all symbols of the input read, then the string is in the language of the automaton; otherwise it is not. Note that the operation of *intersection* of two automata (see Section 1.5) follows essentially the same algorithm as just sketched, except that one is matching paths in one automaton against another, instead of a string. Note also that one can represent a string as a single-path automaton, so that the string-matching method we just described can be implemented as automata intersection.

We turn from regular languages and finite-state automata to *regular relations* and *finite-state transducers* (FST). A regular relation can be thought of as a regular language over n-tuples of symbols, but it is more usefully thought of as expressing relations between sets of strings. The definition of a regular n-relation is as follows:

1. $\emptyset$ is a regular n-relation
2. For all symbols $a \in [(\Sigma \cup \epsilon) \times \ldots \times (\Sigma \cup \epsilon)]$, $\{a\}$ is a regular n-relation
3. If R_1, R_2, and R are regular n-relations, then so are
   (a) $R_1 \cdot R_2$, the *(n-way) concatenation* of R_1 and R_2: for every $r_1 \in R_1$ and $r_2 \in R_2$, $r_1 r_2 \in R_1 \cdot R_2$
   (b) $R_1 \cup R_2$
   (c) R^*, the *n-way transitive (Kleene) closure* of R.

For most applications in speech and language processing $n = 2$, so that we are interested in relations between pairs of strings.[2] In what follows we will be dealing only with 2-relations.

[2] An exception is Kiraz (2000), who uses n-relations, $n > 2$ for expressing non-concatenative Semitic morphology; see Section 2.2.9.

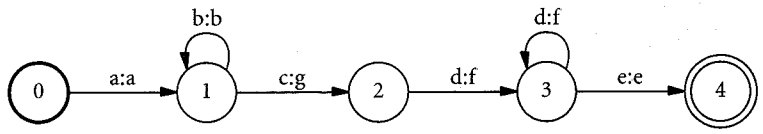

FIGURE 1.2 A simple finite-state transducer that computes the relation $(a{:}a)(b{:}b)^*(c{:}g)(d{:}f)(d{:}f)^*(e{:}e)$

Analogous to finite-state automata are *finite-state transducers*, defined as follows:

Definition 2 *A (2-way) finite-state transducer is a quintuple* $M = (Q, s, F, \Sigma \times \Sigma, \delta)$ *where:*

1. Q *is a finite set of states*
2. s *is a designated initial state*
3. F *is a designated set of final states*
4. Σ *is an alphabet of symbols*
5. δ *is a transition relation from* $Q \times (\Sigma \cup \epsilon \times \Sigma \cup \epsilon)$ *to* Q

A simple finite-state transducer is shown in Figure 1.2. With a transducer, a string matches against the input symbols on the arcs, while at the same time the machine is outputting the corresponding output symbols. Thus, for the input string *abbcddde*, the transducer in Figure 1.2 would produce *abbgfffe*. A transducer determines if the input string is in the domain of the relation, and if it is, computes the corresponding string, or set of strings, that are in the range of the relation.

The closure properties of regular relations are somewhat different from those of regular languages, and the differences are outlined in Table 1.2. The major differences are that relations are not closed under intersection, a point that will be important in Chapter 4 when we discuss the KIMMO morphological analyzer (see Section 4.2.1); and that relations are closed under a new property, namely *composition*. Composition – denoted $\circ$ – is to be understood in the sense of composition of functions. If f and g are two regular relations and x a string, then $[f \circ g](x) = f(g(x))$. In other words, the output of the composition of f and g on a string x is the output that would be obtained by first applying g to x and then applying f to the output of that first operation.

Composition is a very useful property of regular relations. There are many applications in speech and language processing where one wants to factor a system into a set of operations that are cascaded together using composition. A case in point is in the implementation

TABLE 1.2 Closure properties for regular languages and regular relations

Property	Languages	Relations
concatenation	yes	yes
Kleene closure	yes	yes
union	yes	yes
intersection	yes	no
difference	yes	no
composition	—	yes
inversion	—	yes

of phonological rule systems. Phonological rewrite rules of the kind used in early Generative Grammar can be implemented using regular relations and finite-state transducers.[3] Traditionally such rule systems have involved applying a set of rules in sequence, each rule taking as input the output of the previous rule. This operation is implemented computationally by composing the transducers corresponding to the individual rules.

Table 1.2 also lists *inversion* as one of the operations under which regular relations are closed. Inversion consists of swapping the domain of the relation with the range; in terms of finite-state transducers, one simply swaps the input and output labels on the arcs. The closure of regular relations under composition and inversion leads to the following nice property: one can develop a rule system that compiles into a transducer that maps from one set of strings to another set of strings, and then invert the result so that the relation goes the other way. An example of this is, again, generative phonological rewrite rules. It is generally easier for a linguist to think of starting with a more abstract representation and using rules to derive a surface representation. Yet in a morphological analyzer, one generally wants the computation to be performed in the other direction. Thus one takes the linguist's description, compiles it into finite-state transducers, composes these together and then inverts the result.

Of course, regular relations resulting from such descriptions are likely to be many-to-one, as in many input strings mapping to one output string; for example, many underlying forms mapping to the

[3] We will not discuss these compilation algorithms here as this would take us too far afield; the interested reader is referred to Kaplan and Kay (1994) and Mohri and Sproat (1996).

same surface form. In such a case, the inversion yields a one-to-many relation, resulting in the need for disambiguation between the many underlying forms that could be associated with a particular surface form.

1.3 Weights and Probabilities

Disambiguation in morphological and syntactic processing is often done by way of stochastic models, in which weights can encode preferences for one analysis versus another. In this section, we will briefly review notation for weights and probabilities that will be used throughout the book.

Calculating the sum or product over a large number of values is very common, and a common shorthand is to use the sum ($\sum$) or product ($\prod$) symbols ranging over variables. For example, to sum the numbers one through nine, we can write:

$$\sum_{i=1}^{i<10} i = 1+2+3+4+5+6+7+8+9 = 45 \tag{1.1}$$

Similarly, to multiply them:

$$\prod_{i=1}^{i<10} i = 1*2*3*4*5*6*7*8*9 = 362880 \tag{1.2}$$

Logarithms (log) and exponentials (exp) are also very common, and we will by convention use natural logarithm,[4] base e. Recall the basic relationship between them: for any x, y

$$\log(\exp(x)) = \log(e^x) = x \tag{1.3}$$

$$\exp(\log(y)) = e^{\log(y)} = y \tag{1.4}$$

One of the nicest properties of logs is that the log of a product is a sum:

$$\log\left(\prod_{i=1}^{i<10} i\right) = \sum_{i=1}^{i<10} \log(i) \tag{1.5}$$

This is nice because the log of each i can be taken separately, prior to combination, rather than having to combine them before taking the log. This is critical when the product leads to extremely large or

[4] When multiple bases are used, natural log is sometimes denoted *ln*, but here we will just use log.

extremely small floating point numbers. Another nice property is that log is order preserving. That is, if $x > y$, then $\log(x) > \log(y)$.

Briefly, let us introduce simple empirically estimated probabilities of the sort we will mainly be considering in this book. All of the probabilistic models that we will be discussing are discrete distributions, where there are k discrete outcomes (such as different words from a vocabulary Σ of size k) each with its own parameter. When $k = 2$, this is known as a binomial distribution; when $k > 2$, this is a multinomial distribution. For example, we can assign a probability to each word w in a vocabulary Σ; this is a multinomial distribution with $|\Sigma|$ parameters (one parameter $\mathrm{P}(w)$ for each word $w \in \Sigma$) where $\sum_{w\in\Sigma} \mathrm{P}(w) = 1$.

If we have a corpus of N words taken from a vocabulary Σ, we can calculate the probability of any observed word $w \in \Sigma$ in that corpus using *relative frequency estimation*:

$$\mathrm{P}(w) = \frac{f(w)}{N} \tag{1.6}$$

where $f(w)$ is the frequency of the word (its count). Note that using relative frequency estimation leads us to give zero probability to words that have not occurred in our corpus, a problem that is discussed later in the book.

We might want to find the most probable word in the corpus, that is, the word with the maximum probability. The maximum probability, here denoted $\hat{p}$, is

$$\hat{p} = \max_w \mathrm{P}(w) \tag{1.7}$$

If we want to know the word that provides us with this maximum probability, we use "argmax":

$$\hat{w} = \operatorname*{argmax}_w \mathrm{P}(w) \tag{1.8}$$

Then, by definition, $\mathrm{P}(\hat{w}) = \hat{p}$.

Of course, if we are using *negative log probabilities*, the order reverses, hence we will be more interested in the "min" and "argmin", which are defined similarly.

1.4 Weighted Finite-State Automata and Transducers

Finite-state automata and transducers can be extended to include *weights* or *costs* on the arcs. Such machines are termed *weighted finite-state automata* (WFSA) and *weighted finite-state transducers* (WFST).

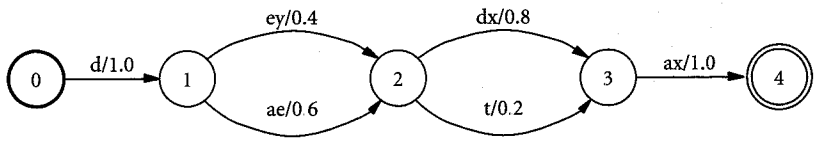

FIGURE 1.3 Representations of several pronunciations of the word *data*, with transcriptions in ARPAbet

The weights can serve a number of functions, but the most frequent use in speech and language processing is to represent probabilities, or more commonly, negative log probabilities, of different analyses.

An example is shown in Figure 1.3. In this example, several plausible pronunciations are shown for the word *data*, with associated probabilities; note that the probabilities for all arcs leaving a given state must sum to one. (The four pronunciations correspond to the IPA transcriptions /deɪtə/, /dætə/, /deɪrə/, and /dærə/.) The probability of a particular path is given by multiplying the individual arc probabilities along the path. In this example, for instance, the pronunciation /d ey t ax/ has the probability $1 * 0.4 * 0.2 * 1 = 0.08$.

In a toy example like the one we have just examined, it is reasonable to represent probabilities as themselves, but in any realistic scenario this presents a computational problem since the probabilities along any given path can be very small and will generally lead to difficulties in floating point represention of the values. Thus it is more common to represent probabilities in the log domain and, more specifically, to represent them in terms of negative logs. Note that in this representation, smaller numbers correspond to more probable events. Recall that, if we use negative log probabilities, we must *sum* the weights along the path rather than multiply them.

When automata and transducers are given weights, the interpretation of those weights must be provided. In addition to specifying how weights are combined *along* a path (which for probabilities is by multiplication), one must also specify how weights are combined *between* paths. For example, suppose there are two paths in an automaton with the same symbols on the arc labels of the paths, and we want to collapse those into a single path. How are the weights combined? When dealing with straight probabilities, in order to ensure proper normalization, the weights (probabilities) of two paths are added together. Different kinds of weights – e.g., logs or probabilities – will have different ways of combining the weights along the path (which we will generally term

the *times* operation) and between paths (which we will term the *plus* operation).

The different interpretations of weights are usefully unified in terms of *semirings*. Before we can introduce the notion of a semiring, we first need the definition of a *monoid*:

Definition 3 *A monoid is a pair $(M, \bullet)$, where M is a set and $\bullet$ is a binary operation on M, obeying the following rules:*

1. ***closure:** for all a, b in M, $a \bullet b$ is in M*
2. ***identity:** there exists an element e in M, such that for all a in M, $a \bullet e = e \bullet a = a$. This is termed the neutral element.*
3. ***associativity:** $\bullet$ is an associative operation; that is, for all a, b, c in M, $(a \bullet b) \bullet c = a \bullet (b \bullet c)$*

A monoid $(M, \bullet)$ is *commutative* if $a \bullet b = b \bullet a$ for all a, b in M. We can take this notion of monoid and define semirings in terms of notional summation and multiplication as follows:

Definition 4 *A semiring is a triple $(\mathbb{K}, \oplus, \otimes)$, where $\mathbb{K}$ is a set and $\oplus$ and $\otimes$ are binary operations on $\mathbb{K}$, obeying the following rules:*

1. *$(\mathbb{K}, \oplus)$ is a commutative monoid with neutral element denoted by 0*
2. *$(\mathbb{K}, \otimes)$ is a monoid with neutral element denoted by 1*
3. *The product $(\otimes)$ distributes with respect to the sum $(\oplus)$, i.e., $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$*
4. *For all a in $\mathbb{K}$, $a \otimes 0 = 0 \otimes a = 0$*

Since most applications use real numbers as the semiring set, one typically denotes a particular semiring with a pair $(\oplus, \otimes)$ that specifies the actual instantiation of the notional plus and times operations. Common semirings used in speech and language processing are the $(+, \times)$ or "real" semiring, and the $(min, +)$ or "tropical" semiring.[5] The $(+, \times)$ semiring is appropriate for use with probabilities: to get the probability of a path, one *multiplies* along the path; to get the probability of a set of paths, one *sums* the probabilities of those paths. The $(min, +)$ semiring is appropriate for use with negative log probabilities: one *sums* the weights along the path, and one computes the minimum of a set of paths – which is useful if one is looking for the best scoring path, since lower scores are better with negative logs.

[5] So called because the mathematician who pioneered this semiring, Imre Simon, was from Brazil.

Weighted finite-state automata (and the corresponding weighted languages) are closed under intersection. When one combines two paths via intersection, the resulting cost is obtained by using the semiring $\otimes$ operation. Note that $\otimes$ is often referred to as the *extend* operation, since it is the operation that one uses when one extends a path by an additional arc.

A formal definition of a weighted finite automaton is as follows:

Definition 5 *A weighted finite-state automaton is an octuple* $A = (Q, s, F, \Sigma, \delta, \lambda, \sigma, \rho)$, *where:*

1. $(Q, s, F, \Sigma, \delta)$ *is a finite-state automaton*
2. *An initial output function* λ: $s \rightarrow \mathbb{K}$ *assigns a weight to entering the automaton*
3. *An output function* σ: $\delta \rightarrow \mathbb{K}$ *assigns a weight to transitions in the automaton*
4. *A final output function* ρ: $F \rightarrow \mathbb{K}$ *assigns a weight to leaving the automaton*

For any transition $d \in \delta$, let $i[d] \in (\Sigma \cup \epsilon)$ be its label; $p[d] \in Q$ its origin state; and $n[d] \in Q$ its destination state. A path $\pi = d_1 \ldots d_k$ consists of k transitions $d_1, \ldots, d_k \in \delta$, where $n[d_j] = p[d_{j+1}]$ for all j, i.e., the destination state of transition d_j is the origin state of transition d_{j+1}. We can extend the definitions of label, origin and destination to paths: let $i[\pi] = i[d_1] \ldots i[d_k]$; $p[\pi] = p[d_1]$; and $n[\pi] = n[d_k]$. A *cycle* is a path π such that $p[\pi] = n[\pi]$, i.e., a path that starts and ends at the same state. An *acyclic* automaton or transducer has no cycles.

We can also extend the definition of the σ function of Definition 5 to paths: $\sigma[\pi] = \sigma[d_1] \otimes \cdots \otimes \sigma[d_k]$. Let $P(q, x, q')$ be the set of paths π such that $p[\pi] = q$, $i[\pi] = x$, and $n[\pi] = q'$. Given a semiring $(\mathbb{K}, \oplus, \otimes)$, the weight associated by A to a string $x \in \Sigma^*$ can be defined as follows:

$$[[A]](x) = \bigoplus_{f \in F} \bigoplus_{\pi \in P(s,x,f)} \lambda(s) \otimes \sigma(\pi) \otimes \rho(f)$$

Weighted finite-state transducers are an obvious extension of finite-state transducers and weighted automata. A WFST computes a regular relation, but in addition it associates each mapping with a weight. For example, in a transducer encoding a rewrite rule system, the weights might represent the probabilities of a particular rule application.

1.5 A Synopsis of Algorithmic Issues

The basic texts on automata theory that we have already cited give algorithms for various finite-state operations including concatenation, Kleene closure, union, intersection, complementation, determinization, and minimization. While these algorithms obviously produce correct results and work fine for small automata and transducers, they are often not efficient enough to handle the very large machines that are typical of serious speech- and language-processing applications. Furthermore, the textbook algorithms do not deal with weighted automata, and the correct treatment of weights turns out to be critical for efficient processing. Some algorithmic issues that have been very important in the application to speech and language processing include efficient algorithms for composition, minimization, determinization, and epsilon removal. In this section we will provide a brief high-level overview of some of these algorithmic issues, with pointers to some papers that deal with them in depth.

One of the most fundamental algorithms that we will be assuming for much of our discussion of finite-state methods in this book is *composition*, so it is useful to have a basic understanding of how this works. At its core, composition is essentially the same as automata intersection as defined in standard texts. We start by reminding the reader how intersection works. The basic algorithm for intersection is as follows. Given two automata $M = (Q, s, F, \Sigma, \delta)$ and $M' = (Q', s', F', \Sigma', \delta')$, construct a new automaton M'' such that:

- Its set of states $Q'' = Q \times Q'$ is the cross-product of the states of the individual machines.
- $s'' = (s, s')$
- $F'' = F \times F'$
- $\Sigma'' = \Sigma \cap \Sigma'$
- $\delta''((p, p'), x) = (q, q')$ just in case $\delta(p, x) = q$ is in M and $\delta'(p', x) = q'$ is in M'.

The basic algorithm for transducer composition is essentially the same, with the difference that with transducers one is matching the *output* label of one transducer with the *input* label of the other. The resulting arc has as its input label the input label of the arc from the first machine and as its output label the output label of the arc from the second machine. Automata can be seen as a special case of transducers, where the input and output symbols are always identical.

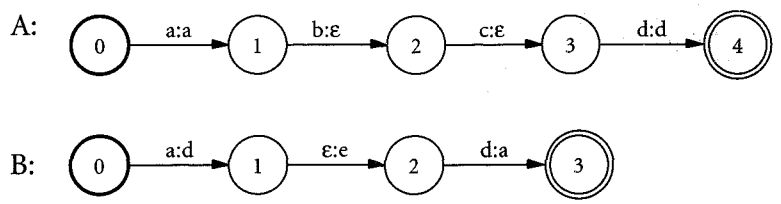

FIGURE 1.4 Two transducers with epsilons. Example taken from Pereira and Riley (1997)

When one is dealing with *weighted* intersection or composition, as we noted above, one computes the weights of the resulting path as the extend ($\otimes$) of the weights of the two input paths.

Even with the basic algorithms there are various efficiency issues. Computation of the transition function δ'' for a new state (p, p') and label x requires efficient search. For example, suppose we are looking at transducer M, at state p, and at an arc with an output label x. We wish to find in M', state p', the set of arcs, if any, that have input labels x. If the arcs of the second machine are arranged in no particular order, then one has no choice but to search linearly through the arcs in M' to find any that have input label x, and this will be inefficient if there are a large number of arcs exiting p'. A solution is to index M' on the input side, so that for any state the arcs are sorted by input label, allowing for a more efficient search method, such as a binary search. Even more efficient methods are possible.

The complication with transducers involves epsilons: arcs where the output side (if on the first transducer) or the input side (if on the second transducer) are labeled with the empty-string symbol ϵ. Epsilons allow one to implement *different-length* relations (as opposed to *same-length* relations) so that, for example, one can implement a rule that deletes symbols in certain contexts. In such a case the transducer would contain arcs labeled with the symbols in question on the input side and ϵ on the output side. The problem with epsilons is that they introduce non-determinism over and above the non-determinism that one would have due to one or both of the input machines being non-deterministic, and hence inefficiency.[6] With weighted transducers the situation is worse: one will actually get the wrong result.

[6] Note that the same issue arises with epsilons in the intersection of acceptors, and the same solution applies. However, note also that acceptors can always have their epsilons removed before intersection, whereas with transducers it is not generally possible to remove epsilons when the epsilon is only on one side of the arc label pair.

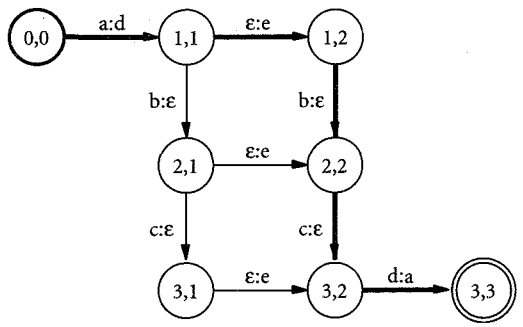

FIGURE 1.5 Naive composition of the transducers from Figure 1.4. Example taken from Pereira and Riley (1997)

To see this, consider the two transducers in Figure 1.4, from Pereira and Riley (1997). In composing A with B, what is at issue is how to move from state 1 through 2 and 3 in machine A and at the same time move from state 1 to 2 in machine B. Since these operations consume no output in A or input in B, there are in principle a number of ways one could do this. One could, for example traverse both arcs from states 1 to 3 in A before traversing any arc in B; or one could traverse the arc between 1 and 2 in B before traversing any arcs in A; or one could choose to move from 1 to 2 in A, then stay in 2 in A while moving from 1 to 2 in B, and then complete the transition to 3 in A. These various options are diagrammed in Figure 1.5. These multiple paths lead to inefficiency in unweighted transducers but are otherwise correct. In weighted transducers, however, they yield the wrong result for the simple reason that the weights from the two original paths will be combined in each of the alternatives (via $\otimes$); these multiple alternatives will then be combined (via $\oplus$), meaning that in many semirings the resulting cost of the intersection of the two original paths will be wrong. The solution to this problem is to insert an *epsilon filter* F between the transducers A and B so that in effect one is composing $A \circ F \circ B$; the transducer F forces the result to have just one of the paths, specifically the bold-marked path in Figure 1.5.[7]

Beyond composition, other algorithmic issues that arise relate to determinization, minimization and epsilon removal. Weighted automata and transducers (whether weighted or not) cannot in general be determinized, but certain types of machines, including acyclic

[7] The actual algorithm is somewhat more complicated than what we have sketched here, and the epsilon filter transducer is simulated rather than actually constructed; see Pereira and Riley (1997) for further details.

machines, can be. (See Mohri, 1997, for a rigorous characterization of the class of determinizable machines.) Since machine minimization requires a determinized machine (Harrison, 1978; Hopcroft and Ullman, 1979; Lewis and Papadimitriou, 1981), this also implies that not all weighted acceptors or transducers can be minimized, though, again, some classes of machine can be. Transducers and weighted acceptors that fall into the class of determinizable and minimizable machines include machines that are useful in speech and language processing. For example, a dictionary can be modeled as an acyclic transducer, mapping input words to some other property such as their part of speech or pronunciation; and a *lattice* of possible analyses output by a speech recognizer can be modeled as an acyclic weighted acceptor. Determinizing and minimizing such machines can provide large efficiency gains (Mohri and Riley, 1999).[8] Epsilon removal with weighted automata is an interesting algorithmic issue in particular because epsilon arcs may have weights, and one must therefore be careful to distribute the weights correctly once the epsilon arcs are removed (Mohri, 2002).

1.6 Computational Approaches to Morphology and Syntax

One might wonder why so much attention is paid to finite-state methods in this book, even in the sections that are devoted to syntax. Weren't finite-state techniques mostly relegated to the dustheap during the 1970s? That has certainly been one common view. In the mid 1990s one of the authors gave a talk at a major industrial research lab in the Seattle, Washington area. He presented some work on applications of finite-state methods to text analysis for text-to-speech synthesis. Several natural language researchers in the audience reacted negatively to his presentation, claiming that finite-state methods were outdated and belonged in the 1970s not the 1990s.

Developments over the past decade have proved this view to be ill-founded: there has been a veritable explosion of research in finite-state methods with applications in a number of areas of speech and language processing including morphology and phonology (in which there was already substantial work by the mid 1990s and as we shall discuss further below), the computational analysis of syntax (e.g., Voutilainen, 1994; Mohri, 1994, and see Chapter 6), language modeling for speech recognition (Pereira and Riley, 1997; Mohri et al., 2002), text

[8] Even for machines that cannot be determinized, it is often possible to *locally* determinize them (Mohri, 1997).

normalization systems for speech synthesis (Sproat, 1997a), pronunciation modeling (Mohri et al., 2002), the analysis of document structure (Sproat et al., 1998), inter alia. Outside speech and language processing, finite-state methods have found applications in other fields, such as computational biology (Durbin et al., 1998). Thus, if we seem to dwell too much on finite-state methods in this book, it is for a reason: such methods have a broad range of applications and students of speech and language processing would do well to master them.

Despite the broad applicability of finite-state methods, however, there is a fundamental difference between computational approaches to morphology and computational approaches to syntax, in that the former (we shall argue) can be accomplished entirely with finite-state methods, while the latter cannot. Finite-state approaches to syntax can be extremely efficient and useful for many applications requiring some amount of syntactic processing, but it has been widely known since Chomsky (1957) that many syntactic phenomena simply cannot be described without context-free or even context-sensitive grammars. Grammars built for computational syntactic processing must typically trade-off the richness of syntactic description provided by the grammar with the computational cost of using it. Often the utility of a syntactic annotation will not justify – within the context of a particular application – the cost of annotating it. This is much less of an issue for morphological processing, since finite-state models and algorithms are generally sufficient for morphological description.

Computationally, a grammar may be used for syntactic processing in several ways. First, it may be used to generate word strings in the language described by the grammar. It may also be used to recognize (or accept) strings in the language and reject strings that are not in the language. The grammar may additionally be used to provide some useful annotation to the accepted strings, such as, labels, delimiters, or perhaps a numerical score. Very often it is this annotation, and not just acceptance or rejection, that makes grammars useful computationally.

The quality of a grammar (or syntactic model) is usually inversely related to the efficiency with which the model can be built and used. Very rich syntactic formalisms that find favor with syntacticians because they do a good job of accepting just those sentences that are grammatical in a language are often not used because explicit, detailed grammars require significant expertise and a relatively long time to write, and because the most efficient recognition algorithms that make use of such grammars are simply not efficient enough for particular

applications. The most commonly implemented syntactic models fall far short of what linguists would expect from a grammar in terms of describing languages, yet they provide useful information and are both easy to build and efficient to use. These latter considerations will carry the day, unless a compelling difference in application performance can be demonstrated.

A very important consideration is robustness in the face of noise. Unless the application is severely constrained, for example, machine translation of official weather reports, the language usage will be varied and noisy. In more natural, less constrained settings, any grammar will be presented with false starts, disfluencies, sentence fragments, out-of-vocabulary words, misspellings, run-on sentences, or just plain ungrammaticalities. A parser is usually expected to yield some useful information to an application beyond rejection, even in the face of these phenomena.

Issues of efficiency and robustness have made simple, weighted finite-state methods very popular. However, much research is currently focused on enriching robust syntactic models, and making richer syntactic formalisms more efficient and robust. Part II of this book (chapters 6–9) will look at various computational approaches to syntax, starting with the simplest and most efficient techniques before moving on to richer ones. The inclusion of scores among syntactic annotations will be particularly emphasized, since this particular annotation, as shall be seen, can make the difference between a useless model and a very useful one. It is in computational linguistics, more than any other subdiscipline of linguistics, where statistical and probabilistic approaches to disambiguation have been investigated, and it is this most of all that distinguishes computational approaches to syntax from other perspectives.

Many of the most successful computational models of syntax share much in common with constraint-based linguistic formalisms, such as Optimality Theory (OT), with some simple differences. What is shared is the notion of a feature or constraint that encodes some informative linguistic distinction. General, effective automatic feature induction methods are beyond the current state-of-the-art in Natural Language Processing, so that effective manual feature selection – often based on linguist-documented generalizations – is a key part of building effective syntactic analyzers. Computational approaches typically differ from OT approaches primarily in terms of how the evidence of various features/constraints is combined, but also in terms of the

explicitness of the candidate generation mechanism, known as GEN in OT. Chapters 6–9 will present just how serious a problem efficient candidate generation can be, and a range of computational solutions. Disambiguation through candidate ranking will be presented in Chapter 9.

In Chapter 2, we will show that models of morphology can be implemented in a unified framework of finite-state transducers. That is not the case for syntax, which brings efficiency to center stage. As a result, Part II (Computational Approaches to Syntax) will have much more of a focus on the accuracy/efficiency trade-off than Part I (Computational Approaches to Morphology), with efficient approximations that provide useful annotations receiving much attention. Efficient algorithms will be explicitly presented, to clearly illustrate why computational linguists are forced to make the choices they do. Differences in focus between the two parts of the book reflect differences in the key issues driving the two topic areas.

Part I

Computational Approaches to Morphology

2

The Formal Characterization of Morphological Operations

In this chapter we focus on the ways in which different languages encode information morphologically, with particular attention paid to the formal devices that are used. The point of this discussion is not merely to present a taxonomy but in addition to argue that all morphology can be understood in terms of a single regular operation, namely *composition*.

This is a somewhat different view from the one normally presented in work on computational morphology. For example, Beesley and Karttunen (2000) observe that concatenation is sufficient for straightforward concatenative morphology (see Section 2.2.1). For non-concatenative morphology such as that found in Semitic languages (Section 2.2.9) or reduplication (Section 2.4), they propose the operation *compile-replace* (which we will describe in more detail in Section 2.2.9).

Indeed when we look at straightforward cases of concatenative morphology, we are naturally seduced by the idea that regular concatenation is the operation of choice. While this is perfectly viable, as we shall argue below, not only can concatenative morphology be handled by composition, but this is in fact the preferred mechanism, since many cases of affixation also have either restrictions on what kinds of bases they can attach to or impose modifications on their bases. As we shall see, such additional stipulations fall out naturally from treating affixation in terms of composition.

But we are getting ahead of ourselves. Let us start out with some background on morphological analysis and what it is about.

2.1 Introduction

Suppose you are studying Latin. You come across an unfamiliar form: *scrīpsērunt*. Naturally enough, you want to know what it means. You already know the verb *scrībō* "I write", and you guess that there is enough similarity between the forms that perhaps *scrīpsērunt* is actually a form of the verb "write". Sure enough, with a little delving into your grammar, you find that this is correct and that the form *scrīpsērunt* is in fact the THIRD PERSON, PLURAL, PERFECT, ACTIVE, INDICATIVE form of the verb "write": "they wrote". You also learn that *scrīpsērunt* can be decomposed into the stem *scrīb* "write" (which becomes *scrīp* before /s/), the perfect stem-forming affix *-s-*, which characterizes third conjugation verbs, and the (perfect) third plural suffix *-ērunt*.

Relating word forms and detecting the structure of word forms is what morphological analysis is all about. The task of relating a given form to a canonical form is called *lemmatization*. Thus in the Latin example, we would lemmatize *scrīpsērunt* to *scrībō*: note that the lemma is typically the form of the word that we find in a dictionary, which in the case of Latin is the FIRST PERSON, SINGULAR, PRESENT, ACTIVE, INDICATIVE form.

Both lemmatization and the decomposition into parts have their uses. For tasks that involve meaning (such as information retrieval or machine translation), we would typically be interested in lemmatizing words to some canonical form.

For applications such as text-to-speech synthesis, we may also be interested in the analysis of a word into its parts. Clearly in a speech synthesizer we do not want the system to speak the lemma – e.g. to say *scrībō* when the text has *scrīpsērunt*. However, we may want the system to know something about the decomposition of the word, since such knowledge can affect its pronunciation. In Standard German, for example, knowing that a word-internal *s* is a compound linking morpheme is important for determining whether it is pronounced /s/ or /ʃ/. Thus the middle /s/ in *Staatsprotokoll* "state record" is a linking morpheme, and therefore does not belong to the onset of the following syllable; thus it is pronounced /s/, and not /ʃ/, which is what we would expect if it were part of the same syllable as the following /p/.

In this discussion we will be neutral as to whether we are considering decomposition or lemmatization. In fact, one is formally derivable from the other. To see this, consider that lemmatization can be viewed as a combination of decomposition followed by normalization to the

lemmatized form. Anticipating somewhat, if you have a function $\mathcal{D}$ that transmutes a word to its morphological decomposition, we can *compose* that function with another function $\mathcal{L}$ that determines, for each decomposed stem, what the appropriate lemma would be. Thus we can lemmatize a given form by applying the function $\mathcal{D} \circ \mathcal{L}$ to that form.

In the case of our Latin example, $\mathcal{D}$ would map *scrīpsērunt* to *scrīb+s+ērunt*, tagging the form as THIRD PERSON, PLURAL, PERFECT, ACTIVE, with the stem *scrīb*:

scrīb+s_{PERFECT}+$\bar{e}run t_{\text{THIRD PERSON, PLURAL, ACTIVE, INDICATIVE}}$.

$\mathcal{L}$ would then map the stem *scrīb* to the canonical first person present active form, and remove other non-featural material, yielding:

$scr\bar{i}b\bar{o}_{\text{PERFECT,THIRD PERSON, PLURAL, ACTIVE, INDICATIVE}}$.

The task of morphological analysis then is to take word forms and relate them to other word forms; while at the same time deriving featural information about the form.

It is customary in discussions of morphology to talk about *inflectional* versus *derivational* morphology, in terms of the kinds of features each of these encodes. We will not have much to say about this distinction here. Rather we will concentrate purely on the computational mechanisms for performing morphological analysis and how these mechanisms can represent two kinds of linguistic information:

- The *formal* properties of morphological operations – i.e. the *syntagmatic* combinations of morphological elements.
- The *paradigmatic* relation between forms.

It will be argued that the most general operation that allows us to describe nearly all of morphology with purely finite-state devices is *composition*. This demonstration will occupy the bulk of the next two sections of this chapter. The third section deals with paradigms and how these can be represented computationally. We end in the fourth section with a discussion of reduplication, the one kind of morphology that seemingly requires special treatment in any finite-state approach. This computational approach to morphology has some important implications for debates in theoretical morphology; we will address those matters in the next chapter.

Throughout this discussion we will assume that we can notate the features of complex morphological forms as a string of feature values, as in the above Latin examples. We will consider this as a shorthand for a more articulated feature-structure-style representation. Thus we

will assume that $scr\bar{\imath}b\bar{o}_{\text{PERFECT, THIRD PERSON, PLURAL, ACTIVE, INDICATIVE}}$ is a shorthand for something like the following:

$$\begin{bmatrix} \text{LEX} & scr\bar{\imath}b \\ \text{ASPECT} & \text{PERFECT} \\ \text{PERSON} & \text{THIRD} \\ \text{NUMBER} & \text{PLURAL} \\ \text{VOICE} & \text{ACTIVE} \\ \text{MOOD} & \text{INDICATIVE} \end{bmatrix}$$

It is not obvious to us that there is any loss in descriptive power.[1] Of course, the sequence does have to encode the features so that, for instance, PLURAL is a feature of NUMBER and not of, say, VOICE. But this is trivially arranged by making sure that each feature value is uniquely associated with a given feature.

One note on notation: There is not a lot that different theories of morphology agree on, but one thing they do seem to agree on is that morphological pieces come in two flavors. One flavor is affixes or affixation processes. Generally these have some set of restrictions on their attachments and can thus be said to select for a particular kind of base. The other is the things they attach to, which are words, stems or roots, depending upon the affix (or process) involved, but in any event things that are more "word-like" in the sense that they lack selectional requirements for attachment. Simply put, in a word like *dogs*, it is the *s* that selects for *dog*, not the other way around. In the formal expressions in what follows, we will indicate affix-like entities with lower-case Greek letters and word-like entities with upper-case Greek letters.

Second, we will per normal convention assume an alphabet Σ that comprises the alphabet of all symbols (e.g., phonemes, letters) that make up the morphological pieces we are examining. For clarity, we will never use Σ to represent a word-like entity. We will also use ϵ for the empty string. Again for clarity, we will never use ϵ to represent an affix-like entity.

There are two issues that we wish to make clear at the outset. The first is that while the analyses presented in the ensuing discussion are toys, it should be understood that there have been many large-scale implementations of morphology using finite-state techniques. Much of that work has been done using Two Level Morphology (Koskenniemi,

[1] On a related noted, Kornai (1991) presents a linearization of multidimensional autosegmental representations in phonology.

1983) or the Xerox finite-state tools (Beesley and Karttunen, 2003), but there have also been systems developed using many other toolkits, such as the systems for German, French, Spanish, Russian, and Italian, which were part of the Bell Labs multilingual text-to-speech system (Sproat, 1997a, b), and were developed using the *lextools* toolkit.

Second, there has been a significant amount of work on inflectional morphology using the nonmonotonic inheritance network language DATR (Corbett and Fraser, 1993; Andry et al., 1993; Barg, 1994, inter alia), and extensions such as KATR (Finkel and Stump, 2002). DATR is particularly well-suited to handling the kinds of partial inheritances common in inflectional morphology. For example, it is very easy in DATR to handle cases where a word inflects mostly according to a particular pattern, except in a few cases where there is a local override. It is easy to indicate, for example, that all English verbs form their present participles in *-ing*, that a subset of verbs form their passive participles in *-en*, and that the particular verb *rise* has its past tense form as *rose*. DATR is a perfectly reasonable high-level language for describing these kinds of phenomena. But as with, for example, context-sensitive rewrite rules (Kaplan and Kay, 1994; Mohri and Sproat, 1996), DATR descriptions of morphology can be compiled down into finite-state automata (see Evans and Gazdar, 1989b). The fact that we do not discuss DATR-based approaches in this book says nothing about the utility of such approaches. Rather, what it reflects is the fact that DATR does not add to the formal power of the finite-state approaches that we will be describing.

2.2 Syntagmatic Variation

2.2.1 Simple Concatenation

At first glance, the most natural implementation of concatenative morphology would make use of the regular operation of concatenation. For example, if we have a stem A and a suffix β, we can combine them into a form Γ using regular concatenation as in 2.1:

$$\Gamma = A \cdot \beta \tag{2.1}$$

An obvious example of this operation is regular plural formation in English, where a singular noun is suffixed (orthographically) with *-s* or *-es*.

Note, however, that it is possible to analyze it instead as follows. Suppose, instead of treating β as a string on a par with A, we instead

think of it as a function β' that takes as input a string, and produces as output that string concatenated with β. β' would be defined as in 2.2 and the whole construction as in 2.3:

$$\beta' = \Sigma^*[\epsilon : \beta] \tag{2.2}$$

$$\Gamma = A \circ \beta' \tag{2.3}$$

(Note that here and henceforth we will use Σ^* – a specification of a regular *language* – to represent a regular *relation* that maps strings into themselves. In the notation of Kaplan and Kay (1994), this would be represented as $\mathrm{Id}(\Sigma^*)$, but we will dispense with this more correct notation as long as no ambiguity arises.)

What are the advantages of doing this? From a computational point of view this is not immediately clear, since while concatenation as in 2.1 is a constant time operation, the composition in 2.3 is linear in the length of A, because minimally each symbol in A must be read to match with Σ^*.

But it is frequently the case that when affixes combine they also either trigger some phonological, spelling, or morphological change that affects the stem, or the affix, or both; or else they select for some prosodic property of the base. And in these cases composition is necessarily involved at some level since phonological rules are implementable using composition (C. D. Johnson, 1972; Kaplan and Kay, 1994; Mohri and Sproat, 1996), and composition also affords a natural way of implementing prosodic selection, as we shall see. Further examples will be given below, but for now note the obvious case of English plural 's', which is /ɨz/ after apical fricatives and affricates, /z/ after voiced sounds, and /s/ elsewhere. This rule can be implemented with a transducer T. If we assume that the plural suffix σ is attached to the stem S with concatenation, for a plural form Π we have:

$$\Pi = [S \cdot \sigma] \circ T \tag{2.4}$$

Now, we can refactor this as:

$$\Pi = S \circ [\Sigma^*[\epsilon : \sigma]] \circ T \tag{2.5}$$

And if we then define σ' to be:

$$\sigma' = [\Sigma^*[\epsilon : \sigma]] \circ T \tag{2.6}$$

this yields

$$\Pi = S \circ \sigma' \tag{2.7}$$

Our new affix σ' adds the suffix and makes the appropriate modifications in one fell swoop.

We return below to other cases where affixes have a more drastic effect on their bases, thus further motivating the use of composition. But before that we need to introduce the concept of *Prosodic Circumscription*, which will be useful in our later discussions.

2.2.2 Interlude: Prosodic Circumscription

In their work on *Prosodic Morphology*, McCarthy and Prince (1986; 1990) present the notion of *prosodic circumscription*:

> **Prosodic Circumscription of Domains.** The domain to which morphological operations apply may be circumscribed by prosodic criteria as well as by the more familiar morphological ones. In particular, the minimal word within a domain may be selected as the locus of morphological transformation in lieu of the whole domain.
> (McCarthy and Prince, 1990)

Prosodic circumscription starts with factoring a base into a prosodically defined unit and its *residue*. For example, we might define a syllable at the right edge of a stem as a prosodically defined unit, in which case the whole stem to the left of the final syllable is the residue. Or we might define the initial onset of the stem as the prosodically defined unit, in which case the remainder of the stem after the onset would be the residue. In McCarthy and Prince's notation, a base B would be factored into a prosodically defined unit "B:" concatenated ("*"), in some order, with a residue "$B/$":[2]

$$B = B{:} \ * \ B/ \tag{2.8}$$

Prosodic morphological operations can then be defined to apply either to the prosodically defined unit B: or to the residue $B/$. Thus, given an operation O, we can define operations O: and $O/$ as follows:

$$O{:} = O(B{:}) \ * \ B/ \tag{2.9}$$

$$O/ = B{:} \ * \ O(B/) \tag{2.10}$$

[2] The ":" and "*" operators are McCarthy and Prince's notation and are not to be confused with the use of those symbols in regular expressions.

So, with O: we first factor the base into B: and $B/$, apply O to B:, and then reconstitute $O(B:)$ with $B/$. Contrariwise, with $O/$ we factor the base, apply O to $B/$ and then reconstitute the result.

$O/$ is the case of *extrametricality*: we define B: as a prosodic domain to ignore, apply an operation to the residue, and then reconstitute the whole. O: is the case of positive circumscription: we define B: as the prosodic domain to which operations apply.

Both of these phenomena are common in morphology. Extrametricality is exemplified by infixation in many Philippine languages, where one ignores the first onset of a word, and attaches the infix as a prefix to the remainder. Thus in Tagalog we have *tawag* "call" but *tumawag* "call (perfective)"; see Section 2.2.7.

The converse situation is illustrated by infixation in Ulwa where, as we shall see in Section 2.2.8, the infix is placed after an initial iambic foot.

Specifics aside, we can completely characterize prosodic circumscription in terms of the finite-state operation of composition. The definition of the prosodic unit can be implemented as a transducer that inserts a marker after (or before) the prosodic unit of interest. For concreteness, consider the example of infixation in Tagalog, where the prosodic unit to be identified is the initial onset of the stem, if any. We wish to insert a marker, say >, after this constituent C and then leave the rest of the base alone. The marker transducer M that accomplishes this can be defined as follows:

$$M = C?[\epsilon :>]V\Sigma^* \tag{2.11}$$

One can then use the marker > as an anchor to define the domain of other operations. For example, in Tagalog, one would use > to define where to place the infix *-um-*. One can then implement the actual infixation as a transducer that simply rewrites > as *-um-*: again, see Section 2.2.7.

In one sense the description just given is simpler than that of McCarthy and Prince: note that once one has defined the placement of the marker, subsequent operations merely need to reference that marker and have no need of notions like "prosodically defined unit" or "residue". Thus, if we handle Tagalog infixation as sketched above, we could characterize *-um-* as "prefixing" to the residue (e.g., *-awag*), but we could equally well characterize it as suffixing to the prosodically defined unit (e.g., *t-*). The former may make more sense as a linguistic

TABLE 2.1 Comparative affixation in English

fat	fatter	fattest
dumb	dumber	dumbest
sìlly	sillier	silliest
yéllow	yellower	yellowest
timid	timider	timidest
curious	*curiouser	*curiousest

conception, but at the level of computational implementation, the distinction is immaterial.

This computational understanding of prosodic circumscription is important for what follows since when we get beyond the most straightforward concatenation into increasingly complex modes of morphological expression, we find that the bases of the morphological processes involved are restricted prosodically in various ways. In such cases the Σ^* of expression 2.2 is replaced by a more complex regular relation.

2.2.3 *Prosodically Governed Concatenation*

The English comparative affix *-er* and the superlative affix *-est* are good examples of affixes that have prosodic restrictions on their attachment. The affixes are restricted to bases that are monosyllabic or disyllabic adjectives; see Table 2.1.[3]

Assuming this is the correct characterization of English comparative formation, we can characterize the base to which the comparative affix

[3] One class of adjectives that do not admit of comparative affixation are participles. Thus *more known*, but **knowner*.

The other class of prima facie exceptions to the generalization are adjectives prefixed with *un-* such as *unhappier*. Note that the interpretation of *unhappier* is the same as that of *more unhappy*, suggesting that *-er* is attached outside *unhappy*. Some previous literature – e.g., Pesetsky (1985) and Sproat (1985) – has analyzed these cases as *bracketing paradoxes*, with mismatching syntactic and phonological structures. On this view, the syntactic and hence the semantic interpretation has *-er* outside *unhappy*, but the phonological structure has *-er* attaching only to *happy*, in line with expectations.

More recently, Stump (2001) has characterized cases like *unhappier* as *head operations*, whereby the operation in question, in this case the operation of comparative formation, operates on the head of the word (*happy*) rather than the whole word. Thus, again, the prosodic conditions of the comparative and superlative affixes are met.

TABLE 2.2 Template- and non-template-providing affixes in Yowlumne

Root	Neutral Affixes		Template Affixes[a]	
	-al "dubitative"	*-t* "passive aorist"	*-inay* "gerundial" CVC(C)	*-ʔaa* "durative" CVCVV(C)
caw "shout"	caw-al	caw-t	caw-inay	cawaa-ʔaa-n
cuum "destroy"	cuum-al	cuum-t	cum-inay	cumuu-ʔaa-n
hoyoo "name"	hoyoo-al	hoyoo-t	hoy-inay	hoyoo-ʔaa-n
diiyl "guard"	diiyl-al	diiyl-t	diyl-inay	diyiil-ʔaa-n
ʔilk "sing"	ʔilk-al	ʔilk-t	ʔilk-inay	ʔiliik-ʔaa-n
hiwiit "walk"	hiwiit-al	hiwiit-t	hiwt-inay	hiwiit-ʔaa-n

[a] For template-providing affixes, the form of the provided template is given in boldface.

attaches as in Equation 2.12:[4]

$$B = C^*VC^*(VC^*)? \tag{2.12}$$

The comparative affix κ would then be characterized as follows:

$$\kappa = B[\epsilon : \text{er}[+\text{COMP}]] \tag{2.13}$$

Composing a base adjective A with κ as in Equation 2.14 would yield a non-null output Γ just in case the base A matches B.

$$\Gamma = A \circ \kappa \tag{2.14}$$

More dramatic are cases where the affix provides the template for the stem, rather than merely selecting for stems that have certain prosodic forms. A well-known example is provided by Yowlumne (Yawelmani), as discussed in Newman (1944) and Archangeli (1984); see Table 2.2, which is copied from Bat-El (2001). The two template-providing affixes shown here are *-inay*, which requires that the stem be reconfigured to

[4] As a reviewer has pointed out, Pullum and Zwicky (1984) argued against a phonological account, suggesting instead that comparative affixation was a case of "arbitrary lexical conditioning." This conclusion was motivated by a particular theoretical position on the phonology/syntax interface, and their evidence consisted in the observation that many examples of adjectives that should allow comparative affixation were nonetheless unacceptable. The problem is that many of the adjectives they list as unacceptable – among them *wronger*, *iller*, *afraider*, *unkemptest*, *aloner* – are readily found via a Google search. Of course we cannot tell how many of these uses are intentionally jocular (such as Lewis Carroll's use of *curiouser*) but it is a fair bet that not all are. In any case it is clear that English speakers readily accept comparatives of nonce adjectives that fit the prosodic criteria (*John is a lot* ***dribber*** *than Mary*), and just as readily reject those that do not (?*John is a lot* ***dramooliger*** *than Mary*). Given these considerations, it does not seem that we can so easily dismiss a prosodic account of the phenomenon.

match the template CVC(C); and *-ʔaa*, which requires the template CVCVV(C). Thus a stem such as *diiyl* "guard", must shorten its vowel to conform to CVC(C), when attaching to *-inay* (*diylinay*). Similarly, it must shorten the stem vowel, but then insert a long copy of the stem vowel between the second and third consonants in order to attach to *-ʔaa* (*diyiilʔaan*).

The template $T_{cvc(c)}$ for CVC(C) is easily characterized as follows:

$$T_{cvc(c)} = CV[V : \epsilon]^*C[V : \epsilon]^*C? \tag{2.15}$$

This ensures that only the first vowel of the root is preserved, and in particular deletes any vowels after the second consonant. Thus the result of composing $T_{cvc(c)}$ with particular stems is as follows:

$$\text{caw} \circ T_{cvc(c)} = caw \tag{2.16}$$

$$\text{diiyl} \circ T_{cvc(c)} = diyl \tag{2.17}$$

$$\text{hiwiit} \circ T_{cvc(c)} = hiwt \tag{2.18}$$

The expression for $T_{cvcvv(c)}$ (CVCVV(C)) is somewhat more complicated since it involves copying vowel material in the root. In order to see how to do this, it is somewhat easier to start by considering deriving possible roots from the durative form. Roots must have at least one V, matching the first vowel of the durative stem, but may have as many as three Vs as in cases like *hoyoo* and *hiwiit*. There may or may not be a final (third) consonant. This range of possibilities can be derived from the template CVCVV(C), if we assume a transducer specified as follows:

$$\Xi = CV[\epsilon : V]?C(V \cup [V : \epsilon])(V \cup [V : \epsilon])C? \tag{2.19}$$

This transducer will force the first V to match the vowel of the root, will allow an optional second vowel in the root's first syllable, then will allow either zero, one or two vowels, followed optionally by a consonant following this first syllable. This is not quite enough, however, since it does not guarantee that the Vs are all identical. To get that, we can implement a simple filter F specified as follows:

$$F = (C \cup i)^* \cup (C \cup a)^* \cup (C \cup o)^* \cup (C \cup u)^* \tag{2.20}$$

See Figure 2.1. The desired transducer $T_{cvcvv(c)}$ is then simply the inverse of the composition of F and Ξ; since we want the vowels to be identical

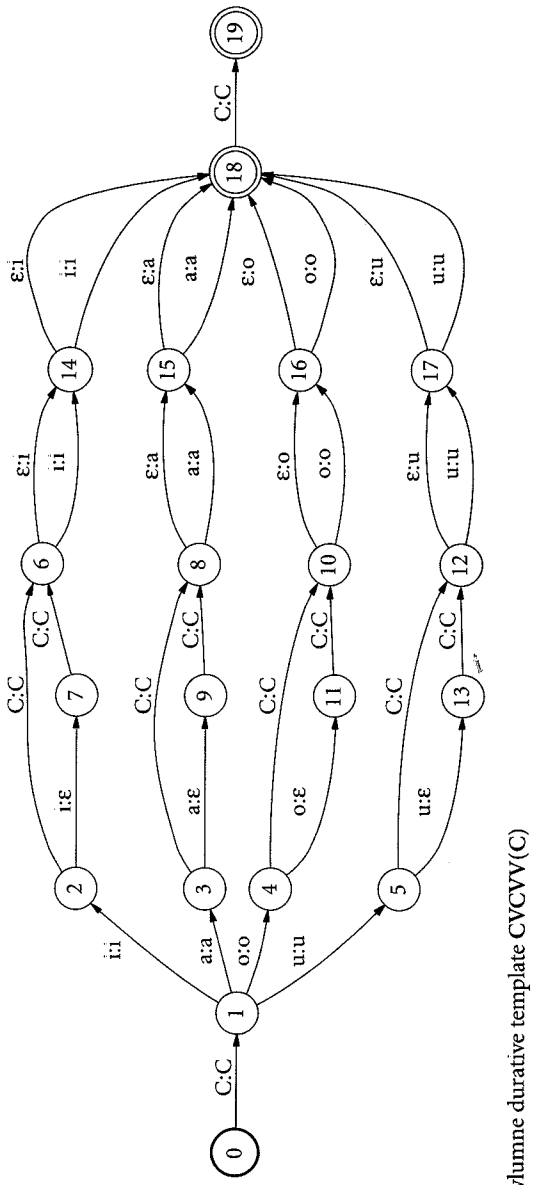

FIGURE 2.1 Yowlumne durative template CVCVV(C)

on both sides of the transducer, we impose the filter on both sides:

$$T_{cvcvv(c)} = [F \circ \Xi \circ F]^{-1} \tag{2.21}$$

Given $T_{cvc(c)}$ and $T_{cvcvv(c)}$, we can now represent the morphemes *-inay* and *-ʔaa* as follows:

$$\kappa_1 = T_{cvc(c)}[\epsilon : \text{inay}[+\text{GER}]] \tag{2.22}$$

$$\kappa_2 = T_{cvcvv(c)}[\epsilon : \text{ʔaa}[+\text{DUR}]] \tag{2.23}$$

Again, composition of these affixes with the roots yields the observed forms.

2.2.4 Phonological Changes Induced by Affixation

In the case of Yowlumne, affixes impose a particular prosodic shape on the base. A milder variety of this phenomenon involves phonological changes that are induced by affixes, which do not involve major prosodic rearrangements of the base material. An example is the diminutive suffixes *-chen*, and *-lein* in Standard German. Generally, these affixes induce umlaut (fronting) on the vowel of the stem to which they attach. Some examples are given in Table 2.3. Assuming this rule is productive, then we can implement it in a way that is straightforwardly parallel to the Yowlumne case. That is, we assume a transducer T_{uml} that will change vowels into their appropriate umlauted forms. (Only /a/, /o/ and /u/, and the diphthong /au/, undergo umlaut.) Then the entry for *-chen* would be as follows:

$$\chi = T_{uml}[\epsilon : \text{chen}] \tag{2.24}$$

Composing a stem with χ produces the affixed form with umlaut applied to the stem.

TABLE 2.3 Diminutive suffixation in German

Hund	Hündchen	"dog"
Haus	Häuschen	"house"
Blatt	Blätchen	"leaf"
Maus	Mäuschen	"mouse"
Frau	Fräulein	"woman" (diminutive = "young woman")
Rose	Röslein	"rose"

TABLE 2.4 Irish first declension genitive formation[a]

Nominative		Genitive		Gloss
bád	/d/	báid	/dʸ/	"boat"
cat	/t/	cait	/tʸ/	"cat"
eolas	/s/	eolais	/ʃ/	"knowledge"
leabhar	/ɾ/	leabhair	/ɾʸ/	"book"
mac	/k/	mic	/kʸ/	"son"
naomh	/v/	naoimh	/vʸ/	"saint"
páipéar	/ɾ/	páipéir	/ɾʸ/	"paper"
sagart	/ɾt/	sagairt	/ɾʸtʸ/	"priest"

[a] Shown are the nominative (base) form, the final consonant cluster of that base form, the genitive and the final consonant of the genitive.

2.2.5 Subsegmental Morphology

Morphological alternants can also be indicated with subsegmental information, such as a change of a single feature. A straightforward example of this is provided by genitive formation in Irish, exemplified in Table 2.4. In these forms, the genitive is marked by palatalizing the final consonant (sequence) of the base (nominative) stem. (Note that the palatal variant of /s/ in Irish is regularly /ʃ/.) A standard linguistic analysis of this alternation would be to assume that the feature [+high] is linked to the final consonant (Lieber, 1992). The genitive can be represented by a transducer γ that simply changes the final consonant (sequence) in the described way; γ can be compiled from a context-dependent rewrite rule, using algorithms described in Kaplan and Kay (1994) and Mohri and Sproat (1996). The genitive form Γ is then produced from the nominative form N by composition:

$$\Gamma = N \circ \gamma \tag{2.25}$$

A more complex instance of subsegmental morphology is illustrated by initial consonant mutation in the related Celtic language Welsh.[5] Initial mutations are of three basic types: "soft" mutation, or lenition, which involves voicing unvoiced consonants, spirantizing or deleting voiced consonants, and otherwise leniting others; "aspirate" mutation, which involves spirantizing voiceless stops; and nasal mutation, which causes nasalization of stops. The mutations are induced by various

[5] All modern Celtic languages, including Irish, have consonant mutation, but the mutation system of the Brythonic languages – Welsh, Cornish, and Breton – is more complex than that of the Goidelic languages – Irish, Scots Gaelic, and Manx.

TABLE 2.5 Initial consonant mutation in Welsh[a]

Base	Gloss	Soft – *a* 'his X'	Asp. – *a* 'her X'	Nas. – *fy* 'my'
pen	"head"	a ben	a phen (/f/)	fy mhen (/mh/)
tad	"father"	a dad	a thad (/θ/)	fy nhad (/nh/)
cath	"cat"	a gath	a chath (/x/)	fy nghath (/ŋh/)
bachgen	"boy"	a fachgen (/v/)		fy machgen
damwain	"accident"	a ddamwain (/ð/)		fy namwain
gwraig	"wife"	a wraig		fy ngwraig
mam	"mother"	a fam (/v/)		
rhosyn	"rose"	a rosyn		
llais (/ɬ/)	"voice"	a lais		

[a] Illustrated are (one variant of) "soft" mutation (lenition), nasal mutation, and aspirate mutation, along with example-triggering morphemes. Entries left blank have no change in the relevant cell. Phonetic values of the initial consonants are indicated where this may not be obvious.

morphosyntactic triggers including many function words (prepositions, possessive markers, articles with following feminine nouns), word-internally with various prefixes and in compounds, and (in the case of lenition) by various syntactic environments such as the initial consonant of an NP following another NP (Harlow, 1981).

Many treatments and descriptions of the mutation system of Welsh have been published – Lieber (1987), Harlow (1989), and Thorne (1993) are just three examples, though the most complete analysis is probably still Morgan (1952). Some examples of the mutations are given in Table 2.5.

If we assume three transducers – λ for lenition, ν for nasalization and ϕ for aspirate mutation – then morphemes that select for the different mutations would be indicated as inducing the relevant mutation on the base. Thus to take a concrete example, the possessive marker *dy* "your (sg.)" would be specified as follows:

$$\delta = [\epsilon : \mathrm{dy}]\lambda \tag{2.26}$$

The second person singular possessive marking of a noun B would thus be constructed as follows:

$$\Gamma = B \circ \delta \tag{2.27}$$

2.2.6 Subtractive Morphology

One case of subtractive morphology, also called truncation, involves plural stem formation in Koasati, exemplified in Table 2.6 (Lombardi and McCarthy, 1991). The generalization is that the final rime of the

TABLE 2.6 Koasati plural stem truncation

Singular	Plural	Gloss
pitáf-fi-n /pitáf-li-n/	**pít**-li-n	"to slice up the middle"
latáf-ka-n	**lat**-ka-n	"to kick something"
tiwáp-li-n	**tiw**-wi-n /tiw-li-n/	"to open something"
atakáa-li-n	**aták**-li-n	"to hang something"
icoktakáa-li-n	**icokták**-li-n	"to open one's mouth"
albitíi-li-n	**albít**-li-n	"to place on top of"
cił́íp-ka-n	**cíł́**-ka-n	"to spear something"
facóo-ka-n	**fas**-ka-n /fac-ka-n/	"to flake off"
onasanáy-li-n	**onasan**-níici-n	"to twist something on"
iyyakohóp-ka-n	**iyyakóf**-ka-n /iyyakóh-ka-n/	"to trip"
koyóf-fi-n /koyóf-li-n/	**kóy**-li-n	"to cut something"

Data from Lombardi and McCarthy (1991)

singular stem, consisting of a final vowel and any following consonant, is deleted in the formation of the plural stem. The onset of the final syllable of the singular stem is retained.

Lombardi and McCarthy, who offer an analysis in terms of Prosodic Circumscription theory argue that the retention of the onset follows from phonotactic considerations due to a generalization of root-final heaviness. Be that as it may, the analysis in terms of finite-state operations is clear. We assume a truncation transducer τ' that deletes the final rime of the base. The whole plural transducer is as in Equation 2.28, and is depicted in Figure 2.2.

$$\tau = \tau'[\epsilon : [+\text{PL}]] \tag{2.28}$$

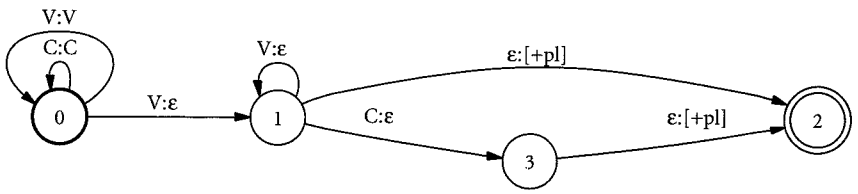

FIGURE 2.2 Koasati plural truncation transducer

TABLE 2.7 Infixation of *-um-* in Bontoc

antj´ŏak	"tall"	*umantj´ŏak*	"I am getting taller"
k´ăwĭsat	"good"	*kum´ăwĭsat*	"I am getting better"
p´ūsiak	"poor"	*pum´ūsiak*	"I am getting poorer"

A singular stem A, composed with τ, will result in a truncated form tagged as morphologically plural:

$$\Gamma = A \circ \tau \tag{2.29}$$

2.2.7 Extrametrical Infixation

An example of extrametrical infixation can be found in Bontoc (Seidenadel, 1907), as well as many other languages of the Philippines. In Bontoc, the infix *-um-* marks a variety of different kinds of semantic information. In the examples in Table 2.7 the alternation involves what Seidenadel terms *progressive quality*. The infix is prefixed to the word, ignoring the first consonant, if any.

It is perhaps simplest to think of this infixation as involving two components computationally. The first component inserts a marker $>$ ($> \notin \Sigma$) in the appropriate location in the word, and the second converts the marker to the infix *-um-*. Note that the other infix in the language, *-in-* (which among other things marks parts of the body as wounded), also attaches in the same position as *-um-*, so it makes sense to factor out placement of the infix from the actual spellout of the infix.

Marker insertion can be accomplished by the simple transducer M in Figure 2.3. The infixation transducer ι will map $>$ to *-um-* and will simultaneously add a morphosyntactic feature marking the construction – call it [+be] – to the end of the word. This is shown in Figure 2.4. Infixation of *-um-* is then accomplished by composing the input word A with M and ι:

$$\Gamma = A \circ M \circ \iota \tag{2.30}$$

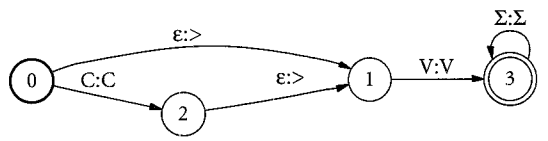

FIGURE 2.3 Marker insertion transducer M for Bontoc infixation. "V" represents a vowel, and "C" a consonant

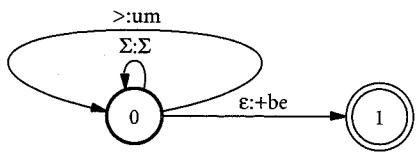

FIGURE 2.4 Transducer ι, which converts > to *-um-* and inserts [+be] at the end of the word

Again, we can precompose M and ι:

$$\mu = M \circ \iota \tag{2.31}$$

so that now

$$\Gamma = A \circ \mu \tag{2.32}$$

2.2.8 Positively Circumscribed Infixation

Ulwa (Nicaragua) (CODIUL, 1989) provides a good example of infixation that attaches to a prosodically defined portion of the base. Some examples are given in Table 2.8. In this case the prosodic unit is an iambic foot, and the generalization is that the infixes in question – in Ulwa, the set of personal possessive markers – suffix to the first foot of the word. Iambic feet include disyllables with a short vowel followed by either a long or short vowel (*bilam*, *anaa*), and monosyllables with either a long vowel (*dii*) or a closed syllable with VC (*sik*). If the word is a single foot, as in the case of *bilam* "fish" or *dii* "snake" the affix is merely a suffix; otherwise the affix attaches to the first foot, as in *liima* "lemon".

As with Bontoc, Ulwa infixation can be factored into two components, the first of which introduces a marker into the relevant post-foot position, the second of which transduces that marker to the relevant infixes. The marker machine M can be described using a set of

TABLE 2.8 Infixation in Ulwa

bilam	"fish"	*bilamki*	"my fish"
dii	"snake"	*diimamuih*	"your (sg.) snake"
liima	"lemon"	*liikama*	"his lemon"
sikbilh	"horsefly"	*siknibilh*	"our (incl.) horsefly"
onyan	"onion"	*onkinayan*	"our (excl.) onion"
baa	"excrement"	*baamana*	"your (pl.) excrement"
mistu	"cat"	*miskanatu*	"their cat"
anaalaka	"chin"	*anaakanalaka*	"their chin"

context-dependent rewrite rules:

$$\epsilon \rightarrow > \; /\hat{}C?(VC|VV|VCVV?)\_CV \tag{2.33}$$

$$\epsilon \rightarrow > \; /\hat{}C?(VC|VV|VVC|VCVV?C)\_\$ \tag{2.34}$$

Here, "^" and "$" indicate the beginning and end of string, respectively. The first rule introduces > after initial (C)VC, (C)VV and (C)VCVV? before a following CV. The second introduces > after initial (C)VC, (C)VVC, (C)VV, and (C)VCVV?C before a following end-of-string.

Once again, a machine ι will convert the marker to one of the infixes so that the infixed form Γ of base A is again defined as follows, where once again we can reassociate:

$$\Gamma = A \circ M \circ \iota = A \circ [M \circ \iota] = A \circ \mu \tag{2.35}$$

2.2.9 Root-and-Pattern Morphology

The best-known example of root-and-pattern morphology is the derivational morphology of the verbal system of Arabic, which was given the first formal generative treatment by McCarthy (1979). As is well known, Semitic languages derive verb stems – actual verbs with specific meanings – from consonantal roots, where the overall prosodic "shape" of the derivative is given by a prosodic template (in McCarthy's original analysis a CV template), and the particular vowels chosen depend upon the intended aspect (perfect or imperfect) and voice (active or passive). Some examples for the active forms with the ubiquitous root *ktb* "notion of writing" (Kiraz, 2000) are given in Table 2.9.[6]

For the sake of the present discussion we will assume that we are combining two elements, the root and the vocalized stem; the analysis would only be marginally more complicated if we chose to separate out the vowel as a separate morpheme as in most autosegmental (and some computational – see Kiraz, 2000) treatments of the phenomenon.

We will assume that the root morpheme, such as *ktb*, is represented as a sequence of consonants as it would be represented in traditional

[6] While tables such as Table 2.9 are often referred to as "paradigms" it is important to understand that the relations expressed here are not inflectional as in a standard verb paradigm for a language like Spanish, but *derivational*: each line in Table 2.9 represents a different verb, much as *stand*, *understand*, and *withstand* are different verbs in English.

TABLE 2.9 Arabic active verb stems derived from the root *ktb* "notion of writing". Note that the use of the vowel /a/ indicates that the stem is active

Pattern	Template	Verb stem	Gloss
I	$C_1aC_2aC_3$	*katab*	"wrote"
II	$C_1aC_2C_2aC_3$	*kattab*	"caused to write"
III	$C_1aaC_2aC_3$	*kaatab*	"corresponded"
IV	$aC_1C_2aC_3$	*aktab*	"caused to write"
VI	$taC_1aaC_2aC_3$	*takaatab*	"wrote to each other"
VII	$nC_1aC_2aC_3$	*nkatab*	"subscribed"
VIII	$C_1taC_2aC_3$	*ktatab*	"copied"
X	$staC_1C_2aC_3$	*staktab*	"caused to write"

analyses. Thus we could define the root P as follows:

$$P = ktb \tag{2.36}$$

Similarly, we assume that the templates are represented more or less as in the standard analyses, except that the additional affixes that one finds in some of the patterns – the *n-* and *sta-* prefixes in VII and X or the *-t-* infix in VIII – will be lexically specified as being inserted. This serves the dual purpose of making the linking transducer (below) simpler to formulate and underscoring the fact that these devices look like additional affixes to the core CV templates (and presumably historically were):

$$\tau_{\mathrm{I}} = CaCaC \tag{2.37}$$

$$\tau_{\mathrm{II}} = CaCCaC \tag{2.38}$$

$$\tau_{\mathrm{III}} = CaaCaC \tag{2.39}$$

$$\tau_{\mathrm{IV}} = [\epsilon : a]CCaC \tag{2.40}$$

$$\tau_{\mathrm{VI}} = [\epsilon : ta]CaaCaC \tag{2.41}$$

$$\tau_{\mathrm{VII}} = [\epsilon : n]CaCaC \tag{2.42}$$

$$\tau_{\mathrm{VIII}} = C[\epsilon : t]aCaC \tag{2.43}$$

$$\tau_{\mathrm{X}} = [\epsilon : sta]CaCaC \tag{2.44}$$

To get a transducer corresponding to all of the above templates, one simply takes the union:

$$\tau = \bigcup_{p \in patterns} \tau_p \tag{2.45}$$

Finally we need a transducer to link the root to the templates; this transducer will serve as the computational implementation of the association lines in the standard autosegmental analysis. The linking transducer must do two things. First it must allow for optional vowels between the three consonants of the root. Second, it must allow for doubling of the center consonant to match the doubled consonant slot in pattern II. The first part can be accomplished by the following transducer:

$$\lambda_1 = C[\epsilon : V]^*C[\epsilon : V]^*C \tag{2.46}$$

The second portion – the consonant doubling – requires rewrite rules (Kaplan and Kay, 1994; Mohri and Sproat, 1996) of the general form:

$$C_i \rightarrow C_iC_i \tag{2.47}$$

This optional rule (in a brute force fashion) allows a copy of the center consonant; call this transducer λ_2. Then the full linking transducer λ can be constructed as:

$$\lambda = \lambda_1 \circ \lambda_2 \tag{2.48}$$

The whole set of templates for *ktb* can then be constructed as follows:

$$\Gamma = P \circ \lambda \circ \tau \tag{2.49}$$

Given that composition is closed under association, we can factor this problem differently. For example, if we prefer to view Arabic verbal morphology as involving just two components – a root and a pattern component, sans the linking component – then we could analyze the pattern as, say, consisting of $\lambda \circ \tau$, and then this new transducer (call it τ') would be composed with the root machine. This would bring Arabic root-and-pattern morphology more in line with a number of other examples we have considered in that it would involve the composition of two elements. But this would be a purely theoretical move, not computationally motivated.

It should be noted that while what we have described works for Arabic, as a practical matter, most large-scale working systems for Arabic,

such as Buckwalter (2002), sidestep the issue of constructing verb stems, and effectively compile out the various forms that verbs take. This is not such an unreasonable move, given that the particular forms that are associated with a verbal root are lexically specified for that root, and the semantics of the derived forms are not entirely predictable.

Another approach taken is that of Beesley and Karttunen (2000) who propose new mechanisms for handling non-concatenative morphology including an operation called *compile-replace*. The basic idea behind this operation is to represent a regular expression as part of the finite-state network, and then to compile this regular expression on demand. To see how this works, consider a case of total reduplication (see Section 2.4) such as that found in Malay: thus a form like *bagi* "bag" becomes *bagibagi* "bags". In Beesley and Karttunen's implementation, a lexical-level form *bagi +Noun +Plural* would map to an intermediate surface form *bagi^2*. This itself is a regular expression indicating the duplication of the string *bagi*, which when compiled out will yield the actual surface form *bagi-bagi*. Thus for any input string *w*, the reduplication operation transforms it into the intermediate surface form *w^2*, which compile-replace then compiles out and replaces with the actual surface form.

For Semitic templatic morphology, Beesley and Karttunen propose an additional device that they call *merge*. The operation of *merge* is simple to understand by example. Consider a root *drs* "read" and a template *CVVCVC*. The root is viewed as a *filler* for the template. The merge operation walks down the filler and the template in parallel, attempting to find a match between the template slot and what is available in the current position of the filler; note that this presumes that we have a table that indicates that, say, *C* will match consonants such as *d*, *r* or *s*. As described by Beesley and Karttunen (2000: 196):[7]

1. If a successful match is found, a new arc is added to the current result state. The arc is labeled with the filler arc symbol; its destination is the result state that corresponds to the two original destinations.
2. If no successful match is found for a given template arc, the arc is copied into the current result state. Its destination is the result state that corresponds to the destination of the template arc and the current filler state.

[7] The operation described here is in many ways similar to the mechanism described in Beesley (1989) and discussed in Sproat (1992).


```
Lexical:    d r s =Root C V V C V C =Template u * i =Voc
Surface: ^[ d r s  .m>. C V V C V C .<m.      u * i      ^]
```

FIGURE 2.5 Representation of the Arabic *duuris*, following Beesley and Karttunen (2000, Figure 16). The surface form is a regular expression involving two mergers between the root *drs*, the pattern *u*i* and the template *CVVCVC*. The surface form is derived via compile–replace

Beesley and Karttunen distinguish leftwards and rightwards merge depending upon whether the filler is on the right or left, respectively. These are expressed as *.m>.* and *.<m..* in the Xerox regular expression formalism (Beesley and Karttunen, 2003). An example is shown in Figure 2.5, where the surface form *duuris* is constructed out of a root *drs*, a vocalism *u*i* (where "*" has the normal Kleene closure interpretation) and a root *CVVCVC*. Once again, the intermediate surface form derives the surface form via compile-replace.

While this is a clean model of Arabic morphology, it is important to remember that the system has no more computational power than a model based solely upon composition. In particular, the merge operation is really a specialized version of intersection or composition. The second option in the algorithm sketched above is straightforwardly simulated by adding to the filler a loop over a label that will match the current template position. Consider the case illustrated in Figure 2.6, consisting of the root *drs* and the template *CVVCVC*. Suppose we have read the first *d* of the root and have matched it with the first *C* of the template. Thus we are in state 1 in the root machine and 1 in the template machine, or state {1,1} in the merged machine. Now we go to read the *V* in the template machine, and find that there is nothing to match it in the root machine. In Beesley and Karttunen's algorithm we advance the template machine into state 2, leaving the root machine in state 1, putting us in state {2,1} in the merged machine. But the equivalent resulting state is achieved through standard intersection if we replace the root machine at the top of Figure 2.6 with the augmented root machine at the bottom. In that case we will follow the loop on *V* in the root machine, matching it against the *V* linking states 1 and 2 in the template machine, putting us into state {2,1} in the resulting intersected machine. This is equivalent to the description of Semitic morphology that we presented earlier in this section. Therefore, not surprisingly, Beesley and Karttunen's merge operation is equivalent to standard intersection with a filler machine augmented with loops that will match the template elements not otherwise matched by the elements in the

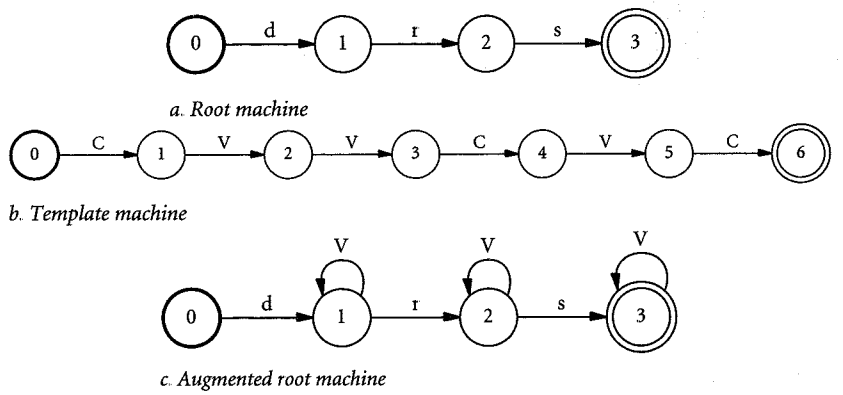

FIGURE 2.6 Filler acceptors representing the root *drs* (a) and the template acceptor *CVVCVC* (b), following Beesley and Karttunen (2000). The third acceptor (c) is the root augmented with loops on *V*

root. Indeed, there should be no difference in efficiency between the standard intersection algorithm and the merge operation since in both cases we have to check the current state of the filler machine for arcs that match arcs leaving the current state of the template machine and move each machine into the appropriate next state. Thus, while the merge operation allows us to represent the roots and vowel templates in a somewhat simpler way (without the explicit self loops), this amounts to syntactic sugar. The operation of Beesley and Karttunen's whole system using merge and compile-replace is entirely equivalent to the model based on composition that we had previously sketched.[8]

2.2.10 Morphomic Components

Aronoff (1994) discusses morphological functions that he terms *morphomic*, which are purely morphological constructs. One example is the English passive participle and past participle (*eaten, fried, wrung*, etc.), which are always identical in form, yet have clearly different morphosyntactic functions. The forms also differ morphophonologically: some verbs mark the form with *-en* (*eaten*), others with *-ed* (*fried*) or *-t* (*dealt*), still others with a vowel change (*wrung*). Aronoff therefore argues that the forms are identical at a purely morphological level and makes the general substantive claim that "the mapping from morphosyntax to phonological realization is not direct but rather

[8] We realize that the description just presented violates the maxim we introduced in Chapter 1 to avoid thinking at the machine level. However, it is necessary to delve down into the machine-level computations when comparing the efficiency of two algorithms.

TABLE 2.10 Latin third stem derivatives

	Verb	3rd Stem	Derived Form
Perfect Participle	laudāre "praise"	laudāt-	laudātus
	ducere "lead"	duct-	ductus
	vehere "carry"	vect-	vectus
	premere "press"	press-	pressus
	ferre "bear"	lat-	latus
Future Participle	laudāre "praise"	laudāt-	laudātūrus
	ducere "lead"	duct-	ductūrus
	vehere "carry"	vect-	vectūrus
	premere "press"	press-	pressūrus
	ferre "bear"	lat-	latūrus
Supine	piscāre "fish"	piscāt-	piscātum
	dicere "say"	dict-	dictum
Agentive Noun	vincere "defeat"	vict-	victor "winner"
	tondēre "shear"	tons-	tonsor "barber"
-iō Noun	cogitāre "think"	cogitāt-	cogitātiō "thought"
	convenīre "meet"	convent-	conventiō "meeting"
-ūr Noun	scrībere "write"	script-	scriptūra "writing"
	pingere "paint"	pict-	pictūra "painting"
Desiderative Verb	edere "eat"	ēs-	ēsurīre "be hungry"
	emere "buy"	empt-	empturīre "want to buy"
Intensive Verb	iacere "throw"	iact-	iactāre "fling"
	trahere "drag"	tract-	tractāre "drag"
Iterative Verb	vidēre "see"	vīs-	vīsitāre "see often, visit"
	scrībere "write"	script-	scriptitāre "write often"

Participles are given in their masculine, singular, nominative form; supines in their accusative form; agentive and other nouns in their nominative singular

passes through an intermediate level" (page 25). So in English two morphosyntactic functions, namely past participle and passive participle, are mapped to a single *morphome*, which Aronoff labels F_{en}, and thence to various surface morphophonological forms, depending upon the verb.

Another more complex example discussed by Aronoff is the *third stem* of Latin verbs, which forms the basis for the further derivation of nine distinct deverbal forms. The third stem is the base of the perfect participle, but also forms the basis for the future participle, the supine, agentive nouns in *-or*/*-rix*, abstract denominals in *-iō* and *-ūr*, desiderative verbs, intensive verbs and iterative verbs. Some examples are given in Table 2.10.

Aronoff argues that there is no basis for regarding any of the nine forms as semantically basic, hence there is no reason to believe that any of the forms are derived from one another. However, they all share a common morphological base, the third stem, which thus, in Aronoff's analysis, has a morphomic status.

Clearly in any analysis of these data, we must assume that the nine morphological processes outlined above have the property that they select for the third stem: this is true whether we believe, contra Aronoff, that one of the forms is systematically basic, or with Aronoff that there is no basis for such a belief. Thus any computational model of the formation of each of these nine deverbal forms must start from the assumption that there are two processes: one that maps a verb to its third stem form, and another that selects that third stem form, and (possibly) affixes additional material. For concreteness, let us consider the formation of agentive nominals in *-or*.

Assume for the sake of argument that we wish to derive the third stem of a verb from its first stem – i.e., the one that we find in the present tense, or generally with the infinitive.[9] For the verb *scrībō* "write", the first stem is *scrīb-*. For many verbs, the derivation of the third stem from the first stem is predictable, though a quick examination of Table 2.10 will suggest that this is not always the case. For first conjugation verbs (infinitive in *-āre*), third stems are regularly formed by suffixing *-t* to the first stem. Thus *laudā-* "praise" forms its third stem as *laudāt-*. Third conjugation verbs such as *scrīb-* frequently also suffix *t* (with regular devoicing of the /b/ to /p/): *script-*.

We assume in any case that the third stem can be formed by a transducer that makes appropriate changes to the first stem and tags them with a marker $>_{3st}$, where $>_{3st} \notin \Sigma$. Call this transducer T.

Morphological processes that select for third stem forms will then be specified so as to combine only with stems that are marked with $>_{3st}$. Agentive *-or*, for example, will look as follows:

$$\beta = \Sigma^*[>_{3st}: \epsilon]\Sigma^*[\epsilon : or] \tag{2.50}$$

Given a first stem F, then we will derive agentive forms Γ as follows:

$$\Gamma = F \circ T \circ \beta \tag{2.51}$$

[9] As we argued in Section 2.1, it is immaterial whether we relate derived forms to underlying stems or citation forms of words.

Formally, this is just an instance of prosodic selection of the kind we saw for Yowlumne in Section 2.2.3. In this analysis, then, morphomic elements are purely morphological in the sense that they serve as an intermediate step in the composition of a full form. In this regard they are no different from several other kinds of morphology that we have seen.

2.3 Paradigmatic Variation

We have talked in this chapter about syntagmatic aspects of morphology, or in other words, how the pieces of morphology are put together. We have said nothing about another important aspect, namely how morphologically complex forms are related to one another. One kind of relationship is *paradigmatic*, and we turn now to a brief discussion of this.

Traditional grammars of languages like Latin present inflected forms of words in a type of table called a *paradigm*. A paradigm is a (usually) two dimensional, though in principle n-dimensional (where n is the number of features expressed by the morphology), array where each cell in the array corresponds to a particular combination of features: for example, in Latin nouns, a combination of one case setting from the set {NOMINATIVE, GENITIVE, DATIVE, ACCUSATIVE, ABLATIVE} and one number setting from the set {SINGULAR, PLURAL}.

A word form that occupies a given cell in a paradigm is understood as bearing the features associated with that cell. In languages of the type exemplified by highly inflected Indo-European languages, words of a particular part-of-speech may be divided into a number of different classes. Each of these classes, by and large, shares the same paradigm structure as the other classes, but the forms are different. A very typical example, involving the five major Latin noun declensions, is given in Table 2.11.

The derivation of the different forms exemplified in Table 2.11 is at one level trivial in that the forms consist simply of stems that bear the meaning of the word and endings that mark the inflectional features. Thus the genitive plural of "barn" can be derived by taking the stem *horre-* and concatenating it with the suffix *-ōrum*. If this were all there were to it, then paradigms would have little status beyond being convenient ways of presenting grammatical data.

But various scholars, including Matthews (1972), Carstairs (1984), Stump (2001), and others have argued that paradigms have more

TABLE 2.11 The five major Latin noun declensions

	Singular	Plural	Gloss
Declension 1, F			
NOMINATIVE	fēmina	fēminae	"woman"
GENITIVE	fēminae	fēmināru m	
DATIVE	fēminae	fēminīs	
ACCUSATIVE	fēminam	fēminās	
ABLATIVE	fēminā	fēminīs	
Declension 2, M			
NOMINATIVE	asinus	asinī	"ass"
GENITIVE	asinī	asinōrum	
DATIVE	asinō	asinīs	
ACCUSATIVE	asinum	asinōs	
ABLATIVE	asinō	asinīs	
Declension 2, N			
NOMINATIVE	horreum	horrea	"barn"
GENITIVE	horreī	horreōrum	
DATIVE	horreō	horreīs	
ACCUSATIVE	horreum	horrea	
ABLATIVE	horreō	horreīs	
Declension 3, F			
NOMINATIVE	fax	facēs	"torch"
GENITIVE	facis	facum	
DATIVE	facī	facibus	
ACCUSATIVE	facem	facēs	
ABLATIVE	face	facibus	
Declension 3, N			
NOMINATIVE	opus	opera	"work"
GENITIVE	operis	operum	
DATIVE	operī	operibus	
ACCUSATIVE	opus	opera	
ABLATIVE	opere	operibus	
Declension 4, F			
NOMINATIVE	manus	manūs	"hand"
GENITIVE	manūs	manuum	
DATIVE	manuī	manibus	
ACCUSATIVE	manum	manūs	
ABLATIVE	manū	manibus	
Declension 5, F			
NOMINATIVE	rēs	rēs	"thing"
GENITIVE	reī	rērum	
DATIVE	reī	rēbus	
ACCUSATIVE	rem	rēs	
ABLATIVE	rē	rēbus	

"F" = FEMININE, "M" = MASCULINE, "N" = NEUTER.

significance than that and actually have a first-class status in any theory of morphology. Part of the motivation for this belief is the existence of regularities that transcend particular forms and seem to characterize the paradigms themselves. For example, a cursory examination of the paradigms in Table 2.11 will reveal that although the dative and ablative are distinct cases in that they typically have different forms in the singular, they are uniformly identical in the plural. This is true for all nouns in all genders and all declensions: no matter what suffix is used in the dative plural, the ablative plural will always be the same. A more circumscribed but nonetheless equally universal generalization is that the nominative and accusative forms of neuter nouns are always identical (and indeed, always marked by *-a* in the plural). Once again, whatever the form of the nominative, the accusative (for the same number) is always the same. These generalizations are distinct from, say, the identity in the first and second declension between the genitive singular and the nominative plural; this generalization does not extend beyond these two declensions, and does not even hold of neuter nouns of the second declension. Rather, the identity of dative and ablative plurals, and nominative and accusative neuters, are a general fact of the language, transcending any particularities of form: knowing these identities is part of what it means to know Latin.

What is an appropriate computational characterization of these regularities? We can divide the problem into two components, the first of which relates morphosyntactic features to abstract morphomic features, and the second of which relates those forms to particular surface forms for a particular word class. We will illustrate this with a small set drawn from the first three declensions, which are the most widespread.

The abstract representation can be derived by a set of rewrite rules, compiled as a transducer; see Table 2.12. The first two rules handle the fact that for any neuter noun, the nominative and accusative are identical. The rules third and second from the end handle the fact that the dative and ablative plurals are always the same and are gender independent, since the forms are identical in the first (mostly feminine) and second (masculine and neuter) declensions. The final rule handles the fact that the forms in the third declension are gender independent. The remaining rules map case/number combinations to abstract features.[10]

[10] Note that we could abstract away from gender further than in just the third declension, in that many of the endings of the first and second declensions are identical: thus the genitive plural is identical – viz. *equōrum*, *fēminārum* – once we discount the stem vowel.

TABLE 2.12 Rewrite rules mapping concrete morphosyntactic features to an abstract morphomic representation

NEUT NOMUACC SG	→	NEUTNASG
NEUT NOMUACC PL	→	NEUTNAPL
NOM SG	→	NOMSG
ACC SG	→	ACCSG
GEN SG	→	GENSG
DAT SG	→	DATSG
ABL SG	→	ABLSG
NOM PL	→	NOMPL
ACC PL	→	ACCPL
GEN PL	→	GENPL
GENDER DAT PL	→	DATABLPL
GENDER ABL PL	→	DATABLPL
GENDER	→	ϵ / III[a]

[a] III denotes third declension.

A second transducer maps from the abstract endings to actual forms. We can express this in terms of affixes that select for stems with particular abstract morphomic features, as in Table 2.13. Note that this description is a fragment; in particular it does not account for first declension masculine nouns (*nauta* "sailor") or second declension nouns in *-r* (*puer* "boy"), and it assumes that all non-neuter third declension nouns end in *-s* in the nominative singular which, on the surface at least, is not true. Call the transducer defined in Table 2.12 α and the transducer defined in Table 2.13 σ. Then given a set of bases B annotated with morphosyntactic features, we can define the set of inflected forms of B as follows:

$$\Gamma = B \circ \alpha \circ \sigma \tag{2.52}$$

The above analysis accords with a number of treatments of paradigmatic variation in that it assumes an abstract level at which morphomic features like DATABLPL have a first-class status. On the other hand, as with the treatment of Arabic root-and-pattern morphology in Section 2.2.9, we can also use the observation that composition is associative to precompile α and σ into a new transducer $\sigma' = \alpha \circ \sigma$. This new σ' will now produce endings based on the surface morphosyntactic features of B; the abstraction α is now hidden in the combined transducer. We will develop this point much further in Chapter 3.

TABLE 2.13 Fragment for Latin nominal endings

Σ* [I-II[a] DATABLPL : īs]
Σ* [NEUTNAPL : a]
Σ* [I-II NEUTNASG : um]
Σ* [I-II FEM ABLSG : ā]
Σ* [I-II FEM ACCPL : ās]
Σ* [I-II FEM ACCSG : am]
Σ* [I-II FEM DATSG : ae]
Σ* [I-II FEM GENPL : ārum]
Σ* [I-II FEM GENSG : ae]
Σ* [I-II FEM NOMPL : ae]
Σ* [I-II FEM NOMSG : a]
Σ* [I-II MAS ABLSG : ō]
Σ* [I-II MAS ACCPL : ōs]
Σ* [I-II MAS ACCSG : um]
Σ* [I-II MAS DATSG : ō]
Σ* [I-II MAS GENPL : ōrum]
Σ* [I-II MAS GENSG : ī]
Σ* [I-II MAS NOMPL : ī]
Σ* [I-II MAS NOMSG : us]
Σ* [I-II NEUT ABLSG : ō]
Σ* [I-II NEUT DATSG : ō]
Σ* [I-II NEUT GENPL : ōrum]
Σ* [I-II NEUT GENSG : ī]
Σ* [III ABLSG : e]
Σ* [III ACCPL : ēs]
Σ* [III ACCSG : em]
Σ* [III DATABLPL : ibus]
Σ* [III DATSG : ī]
Σ* [III GENPL : um]
Σ* [III GENSG : is]
Σ* [III NEUTNASG : ϵ]
Σ* [III NOMPL : ēs]
Σ* [III NOMSG : s]

[a] I-II denotes nouns of declensions I and II.

2.4 The Remaining Problem: Reduplication

The one apparent significant exception to the generalization that all morphological operations can be cast in terms of composition is reduplication. The basic reason why reduplication is problematic is that it involves copying, and finite-state devices do not handle unbounded copying. A key point is contained in this last statement: finite-state devices are in principle capable, albeit inelegantly and non-compactly,

TABLE 2.14 Gothic Class VII preterites after Wright (1910)

Infinitive	Gloss	Preterite
falþan	"fold"	faífalþ
haldan	"hold"	haíhald
ga-staldan	"possess"	ga-staístald
af-áikan	"deny"	af-aíáik
máitan	"cut"	maímáit
skáidan	"divide"	skaískáiþ
slēpan	"sleep"	saíslēp
grētan	"greet"	gaígrōt
ga-rēdan	"reflect upon"	ga-raírōþ
tēkan	"touch"	taítōk
saian	"sow"	saísō

of dealing with any *bounded* copying. Basically we need as many paths as we have strings that are in the domain of the copying operation.

For example, consider the reduplication found in the past tense of Class VII verbs in Gothic in Table 2.14 (Wright, 1910). In forming the preterite, some verbs undergo stem changes, as in the alternation between *grēt* and *grōt*, in addition to reduplication; prefixes such as *ga-* or *af-* are not part of the stem and are prefixed outside the reduplicated stem. Putting these issues to one side, the rule for reduplication itself is simple:

- Prefix a syllable of the form (A)Caí to the stem, where C is a consonant position and (A) an optional appendix position.
- Copy the onset of the stem, if there is one, to the C position. If there is a pre-onset appendix /s/ (i.e., /s/ before an obstruent such as /p,t,k/), copy this to the (A) position.

A fragment of a transducer that introduces the appropriate reduplicated prefix for Gothic Class VII preterite reduplication is shown in Figure 2.7. The model is naive in that it simply remembers what was inserted, and then imposes the requirement that the base match appropriately. Thus there are as many paths as there are distinct reduplicated onsets. Given that Gothic reduplication is limited, this is not an overly burdensome model.[11] It is, however, clearly inelegant.

[11] Indeed, Gothic reduplication is even more limited than we have implied since it only applies to Class VII verbs, and this is a closed class.

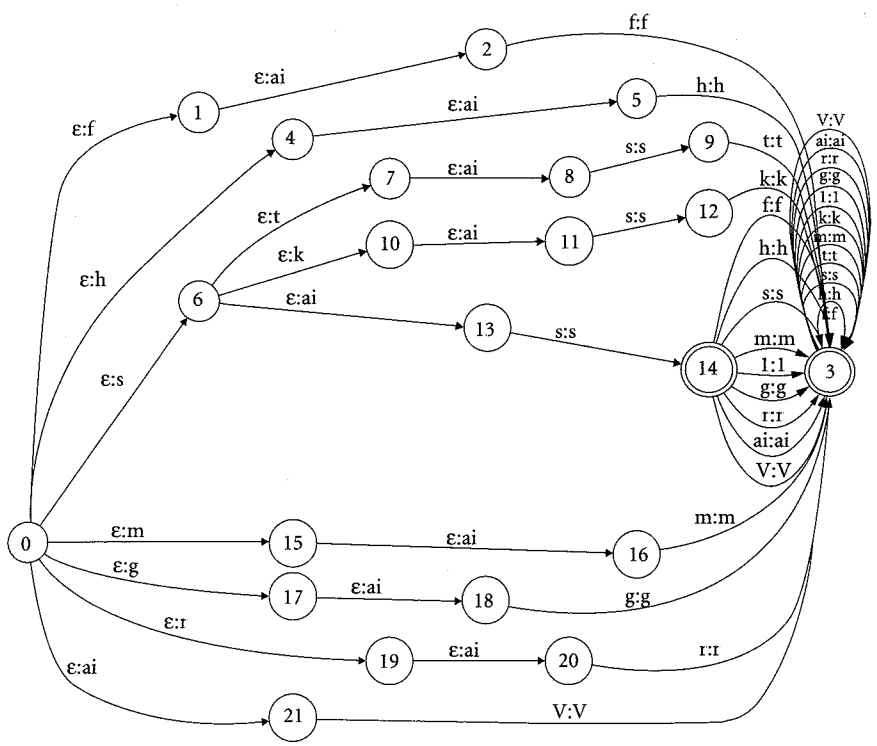

FIGURE 2.7 A transducer for Gothic Class VII preterite reduplication. Here *ai* represents the vowel *aí*, and *V* represents all other vowels

Inelegance converts into impossibility when we turn to *unbounded* reduplication, a well-known example of which is Bambara noun reduplication, discussed in Culy (1985). The construction takes a noun *X* and forms a construction *X-o-X*, where *-o-* is a semantically empty marker, and the meaning of the whole construction is "whichever X"; see Table 2.15. The key issue, as shown in these examples, is that *X* is in principle unbounded. So, compounds such as *wulu-nyinina* "dog searcher" or *malo-nyinina-filèla* "rice searcher watcher" can readily serve as input to the process. If the input to the reduplication process is

TABLE 2.15 Unbounded reduplication in Bambara after Culy (1985)

wulu dog	*o* MARKER	*wulu* dog	"whichever dog"
wulu-nyinina dog searcher	*o* MARKER	*wulu-nyinina* dog searcher	"whichever dog searcher"
malo-nyinina-filèla rice searcher watcher	*o* MARKER	*malo-nyinina-filèla* rice searcher watcher	"whichever rice searcher watcher"

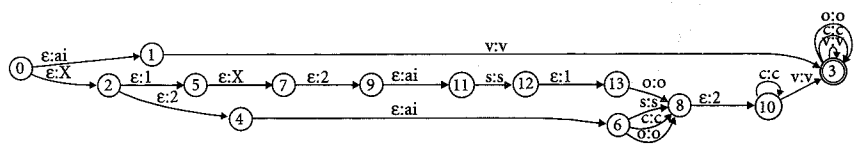

FIGURE 2.8 Schematic form for a transducer that maps a Gothic stem into a form that is indexed for copy checking. Here *V* represents any vowel, *ai* is the vowel /aí/, *s* is /s/, *O* is an obstruent stop, *C* is any other consonant, and *X* stands for any consonant in the reduplicated portion, which has to be checked for identity per the introduced indices

unbounded, then simply precompiling out all of the copies as we did in the case of Gothic is not feasible.

To handle bounded reduplication elegantly, and unbounded reduplication at all, we have to add an additional memory device that can remember what has been seen already in the copy, and then match that to what comes afterwards in the base. (In the case of suffixing reduplication it is the copy that follows the base, but the mechanism is the same.)

It is useful to think of reduplication in terms of two separate components. The first component models the prosodic constraints, either on the base, the reduplicated portion, or both. In the case of Gothic, for example, the constraint is that the reduplicated portion is of the form (A)Caí, where C is a consonant and A is an optional preonset appendix. The second component – the copying component – verifies that the prefix appropriately matches the base. It is useful to separate these two components since the prosodic check can be handled by purely finite-state operations, and it is only the copying component that requires special treatment.

Breaking down the problem as just described, we can implement Gothic reduplication as follows. First, assume a transducer R, which when composed with a base β, produces a prefixed version of β, and in addition to the prefix, adds indices to elements in the prefix and base that must match each other:

$$\alpha = \beta \circ R = (A_1)C_2 a\acute{i}\beta' \tag{2.53}$$

Here, β' is an appropriately indexed version of β. The transducer R is shown schematically in Figure 2.8.

For example, if we have the stem *skáiþ* "divide" and compose it with the transducer R it will produce the output $X_1X_2a\acute{i}s_1k_2\acute{a}i$þ, where X here ranges over possible segments.

A number of approaches could be taken to check matches of the indexed arcs. One way is to impose a set of finite-state filters, one for each index. For example the following filter imposes the constraint that arcs indexed with "1" must be identical; it consists of union of paths where for each indexed segment s_1, there must be a matching indexed segment. The overbar represents complementation and we use $\overline{s_i}$ as shorthand for any single segment that is not s_i.

$$\bigcup_{s \in segments} \overline{[\Sigma^* s_1 \Sigma^* \overline{s_1} \Sigma^*]} \tag{2.54}$$

For a finite number of indices – a viable assumption for bounded reduplication – we can build a filter that handles all matches as follows:

$$\bigcup_{i \in indices} \bigcup_{s \in segments} \overline{[\Sigma^* s_i \Sigma^* \overline{s_i} \Sigma^*]} \tag{2.55}$$

There have been various proposals in the literature for handling reduplication that are formally equivalent to the mechanisms we have just sketched. For example, Walther (2000a; 2000b) adds two new types of arcs to finite-state automata to handle reduplication:

- **Repeat** arcs, which allow moving backwards within a string, allowing for repetition of part of the string;
- **Skip** arcs, which allow us to skip forward over portions of the string.

A more general solution that handles reduplication and other types of non-concatenative morphology is *finite-state registered automata* (FSRAs), introduced in Cohen-Sygal and Wintner (2006). FSRAs extend finite-state automata with the addition of finite registers; note that since the registers are finite, the resulting machine is computationally equivalent to a potentially much larger FSA. The transition function of the automaton is extended to include operations that read or write designated elements of the register, and, depending upon the success or failure of those operations, allow the machine to move to a designated next state. Cohen-Sygal and Wintner demonstrate a substantial reduction in machine size for non-concatenative morphology such as Hebrew templatic morphology over a naive finite-state implementation, such as the one that we described in Section 2.2.9.[12] For reduplication they introduce a particular subclass of FSRA – FSRA* – where

[12] Gains in size are offset, in general, by losses in runtime performance, however.

register-write operations record what was seen in the copy, and register-read operations verify subsequently that the base matches.[13]

There are two limitations on Cohen-Sygal and Wintner's approach. First, FSRA*s are defined for finite-length copies, which covers nearly all cases of reduplication but fails to cover the Bambara case described above. The second limitation is that the register-read operations presume exact matches between the copy and the base. While this is frequently the case, there are also many cases where the copied material undergoes modification due to phonological rules or for other reasons.

A number of such cases are discussed in Inkelas and Zoll (1999, 2005). A simple example is Dakota *kičaxčaya* "he made it for them quickly", where the second syllable of the base *kičax* is reduplicated, but shows up as *čay* for phonetic reasons.[14] But more dramatic examples can be found, such as in the case of Sye. In Sye, verb stems have, in addition to their basic form, a modified form that shows up in a variety of environments. Some examples, from Inkelas and Zoll (1999) are given in Table 2.16. In verbal reduplication, which has an intensifying meaning in Sye, we find two copies of the entire stem, but if the environment requires a modified stem, only the first copy shows up in the modified form. Thus, from *omol*, we find *cw-amol-omol* (3pl.fut-fall_{mod}-fall_{bas}) "they will fall all over".

Another example comes from Kinande where a variety of constraints conspire to yield situations in which the copy in a reduplicative construction does not match exactly what follows. One constraint is that the copy – a prefix – must be disyllabic. This is straightforwardly met in an example like *huma+huma* "cultivate here and there" from *huma* "cultivate", where the latter is itself morphologically complex, consisting of a base *hum* and a final vowel (FV) marker *-a*. With a base like *gend-esy-a* (go-CAUSE-FV) "make go", we find *gend-a+gend-esy-a*, where the stem *gend* in the copy is augmented with the FV *-a* to make it disyllabic. However, we can also find an alternative form involving a shortened causative morpheme *-y* (cf. the full form *-esy*): *gend-y-a+gend-esy-a*, where again the copy is augmented with *-a* to make it

[13] Dale Gerdemann, p.c., argues that the XFST extension of *flag diacritics* (Beesley and Karttunen, 2003) is more effective than FSRAs. Flag diacritics amount to a form of "lazy composition", where we can limit the size of the automaton, and check correspondences between portions of the input at runtime.

[14] Note that this particular feature was not what interested Inkelas and Zoll in this particular example, but rather the palatalization of the onset of the second syllable of the base from /k/ (*kikax*) to /č/ (*kičax*) in both the stem and the copy.

TABLE 2.16 Basic and modified stems in Sye

Basic	Modified	Gloss
evcah	ampcah	"defecate"
evinte	avinte	"look after"
evsor	amsor	"wake up"
evtit	avtit	"meet"
ocep	agkep	"fly"
ochi	aghi	"see it"
omol	amol	"fall"
oruc	anduc	"bathe"
ovoli	ampoli	"turn it"
ovyu-	avyu-	(causative prefix)
owi	awi	"leave"
pat	ampat	"blocked"
vag	ampag	"eat"

Source: Inkelas and Zoll (1999)

disyllabic. Note that in neither alternative is the prefixed material identical to what follows. Another constraint in Kinande is a Morpheme Integrity constraint that states that no morpheme may be truncated in the copy; thus *hum-ir-e* (cultivate-APPLICATIVE-SUBJ) is reduplicated as *hum-a+hum-ir-e*. The form *hum-i+hum-ir-e* is ruled out since this form would result in a violation of the Morpheme Integrity constraint because the integrity of *-ir* would be violated.

Following Cohen-Sygal and Wintner's approach, we could of course design a machine that required identity on only some of the elements between the reduplicant and the stem. But such machines would have to be quite lexically specific in their application. In Sye, the modified stem is only partly and unpredictably related to the basic stem, so we would have to construct our copy machines on a stem by stem basis. Similarly, in Kinande, with cases like *gendya+gendesya* involving a truncated version of the causative morpheme in the copy, we would have to have the reduplication machine be sensitive to the particular stem involved: as it turns out, not all causative verbs allow the truncated alternate, and in those cases where this alternate is not available, a reduplicated form like *gendya+gendesya* is not possible. While we could certainly develop an analysis à la Cohen-Sygal and Wintner along these lines, it would somewhat defeat the purpose since if we require a set of lexical specifications on how the copy machines operate, then we may as well simply precompile out the reduplicated forms beforehand.

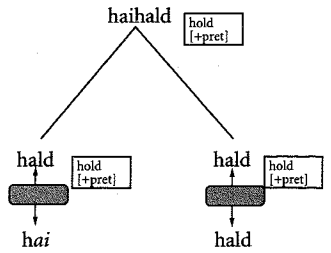

FIGURE 2.9 The Gothic Class VII preterite form *haíhald* "held" under Morphological Doubling theory. Each copy is derived via a purely regular relation mapping (depicted by the shaded boxes) from the underlying form *hald* to either *haí* or *hald*. Restrictions on matching between the two halves reside at the level of morphosyntactic features, so that the final form *haíhald* is licensed by the match between the two [+pret] forms

Interestingly, Inkelas and Zoll's point in discussing such examples is to argue for an alternative theory of reduplication to the "Correspondence Theory" that has dominated nearly all work on reduplication in theoretical morphology – and all work in computational morphology. Under their analysis, reduplication does not involve phonological copying at all, but rather is the result of *morphological* doubling, effectively meaning that we have two copies of the same (possibly complex) lexical form. Phonological similarity is thus expected in many cases (we are dealing with two copies of the same basic form) but not required (since different environments can require different actual forms to surface). Put in another way, under Inkelas and Zoll's Morphological Doubling theory, there are constraints on how each copy is spelled out phonologically, but no constraints relating the reduplicated phonological form to the base, as we find in Correspondence Theory. This analysis has an interesting computational implication: if it is correct, it effectively removes the only exception to the generalization that formal morphological operations can be handled by purely finite-state mechanisms. If there is no phonological copying, there is no need to resort to special mechanisms to extend the finite-state apparatus beyond what is needed for the remainder of morphological operations.

To see this, consider again the case of Gothic preterite reduplication. Under Morphological Doubling theory, each component of a form like *haíhald* "held" is generated separately. In each case, we can model the mapping – either from *hald* to *haí* or from *hald* to itself – as a regular relation. It is up to the morphosyntax to validate the form as involving two identical morphosyntactic feature bundles. See Figure 2.9.

2.5 Summary

Composition of regular relations is the single most general computational operation that can handle the formal devices found in natural language morphology. The only exception to this is reduplication, which seems, at least for an elegant account, to require a non-regular operation, namely copying. However, even reduplication can be reduced to purely regular mechanisms if Inkelas and Zoll's Morphological Doubling theory is correct.

The fact that composition is so general has interesting implications for theoretical views of morphology. In particular, morphologists have argued for decades about whether morphology should primarily be viewed as the construction of words out of small lexical pieces – morphemes – or whether instead it is better viewed as involving the modification of stems or roots via rules. The argument over "Item-and-Arrangement" versus "Item-and-Process" approaches has effectively divided the field. In the next chapter we shall argue that at least from a computational point of view, there is really very little difference between the two approaches.

3

The Relevance of Computational Issues for Morphological Theory

3.1 Introduction: Realizational versus Incremental Morphology

For many decades a central debate in the theory of morphology has been whether morphology is best thought of in terms of constructing a complex form out of small pieces, usually called *morphemes*, which are roughly on a par with each other; or, alternatively, as involving base forms (stems, or roots) modified by rules. Hockett (1954) termed these two approaches "item-and-arrangement" versus "item-and-process".

Item-and-process theories are older; they are the theories that were implicitly assumed by, for example, traditional grammatical descriptions. In a traditional grammar of Latin, for example, one would be presented with paradigms listing the various inflected forms of words and their functions, as well as rules for deriving the inflected forms. But item-and-process approaches have also characterized much work in recent theories of morphology, including Matthews (1972), Aronoff (1976), Anderson (1992), Aronoff again (1994), and most recently Stump (2001).

Item-and-arrangement theories are particularly associated with the American structuralists, but they have been adopted by at least some generative morphologists including Lieber (1980), Sproat (1985), and Lieber again (1992).

To some extent these different approaches were motivated by the properties of different languages. For traditional grammarians the core languages of interest were highly inflected Indo-European languages such as Latin, Classical Greek, and Sanskrit. For these languages, it seems to make sense to think in terms of morphological

rules rather than morphemes. For one thing, it is typical in such languages that many morphosyntactic features are expressed in a single affix, so that in Latin, for instance, the verbal ending *-ō* does not only represent 1ST SINGULAR but in fact something more like 1ST SINGULAR PRESENT ACTIVE INDICATIVE; change any one of those features and the form changes. Second, one often finds the kind of morphomic stem alternations that we have already discussed in the previous chapter, so that it is typically not an issue of merely affixing a particular affix to a stem, but rather affixing to a particular variant of the stem. Finally, the affixes themselves may change, depending upon the particular paradigm the base word belongs to. So first and second declension Latin nominals have the suffix *-īs* for the DATIVE/ABLATIVE plural; but third declension nominals use *-ibus* to encode the same morphosyntactic features. So all around it seems more obvious for these kinds of languages to think in terms of rules that introduce affixes according to whatever features one wishes to encode, rather than assuming separate morphemes that encode the features. In addition to being sensitive to morphosyntactic features, the rules can also be sensitive to features such as the paradigm affiliation of the word, and furthermore they can effect the particular stem form required for the affix in question.

In contrast, if one looks at agglutinative languages like Finnish, one finds that morphosyntactic features are encoded fairly systematically by individual morphemes that are arranged in particular linear orders. Consider, for example, the partial paradigm for Finnish listed in Table 3.1.[1] As will be readily seen, each word form consists of either one, two or three pieces: the base *talo*, a possible affix marking number, and a possible case affix. The case affixes are consistent across singular and plural, so that, for instance, NOMINATIVE is always marked with *-Ø*, ELATIVE with *-sta* and GENITIVE with *-(e)n*. With a single exception, the alternations in form of the affixes are due to phonological rules, as in *-en* versus *-n* for the GENITIVE, and the spelling of the plural affix *-i* as a glide *-j* in intervocalic position.[2] The single exception is the appearance of the plural affix as *-t* in the NOMINATIVE PLURAL.

[1] Adapted from Andrew Spencer's course notes for a morphology course at Essex, L372, available at http://privatewww.essex.ac.uk/~spena/372/372_ch7.pdf. Curiously, Spencer uses the Finnish example to illustrate the *difficulties* of morpheme-based approaches to inflection.

[2] Some of the suffixes do change form in another way, namely due to vowel harmony. This is, however, a completely regular phonological process in Finnish, and does not target particular morphological operations.

TABLE 3.1 Partial paradigm of Finnish noun *talo* "house", after Spencer

	sg	pl
Nominative	talo	talo-t
Genitive	talo-n	talo-j-en
Partitive	talo-a	talo-j-a
Inessive	talo-ssa	talo-i-ssa
Elative	talo-sta	talo-i-sta
Adessive	talo-lla	talo-i-lla
Ablative	talo-lta	talo-i-lta
Allative	talo-lle	talo-i-lle

With this one exception, the system looks as if it is most straightforwardly accounted for by assuming that there are in each case three morphemes, namely the stem morpheme which introduces the basic semantics of the word ("house", or whatever) and the number and case affixes. As is typical with such morphemic approaches, one would assume empty morphemes in the case of SINGULAR number and NOMINATIVE case. The number affix introduces the morphosyntactic feature SINGULAR or PLURAL; the case affix introduces the appropriate case feature. The representation of the number affixes would be encoded so as to select for the base stem (i.e., without a case ending). Similarly, the case affixes would be marked so as to select for forms that are already marked for number. One could in principle treat the alternation in the plural morpheme (*-i/-j* vs. *-t*) as a case of selection: one would have two allomorphs of the plural affix, with the *-t* variant being selected by the NOMINATIVE affix, and the *-i/-j* variant by all others.

The item-and-arrangement versus item-and-process debate has often been cast in terms of this simple binary choice, with possible sub-choices under each main choice. But more recently Stump (2001) has presented a two-dimensional classification. The first dimension is *lexical* versus *inferential*. Lexical theories are those theories in which all morphemes are given lexical entries, so that the third singular verbal affix *-s* in English has an entry associated with the features 3RD SINGULAR, PRESENT, and INDICATIVE. In contrast, *inferential* theories posit rules which are sensitive to such morphosyntactic features; a form like *likes* is *inferred* from *like* due to a rule that associates a suffix *-s* with a particular set of morphosyntactic features (3RD SINGULAR).

TABLE 3.2 The four logically possible morphological theory types, after Stump (2001)

	INCREMENTAL	REALIZATIONAL
Lexical	Lieber	Distributed Morphology
Inferential	Articulated Morphology	Matthews, Anderson, Stump

The second dimension is *incremental* versus *realizational*. Incremental theories assume that rules or morphemes always add information when they are applied. Thus *likes* has its meaning by virtue of the addition of *-s* to *like*. Under *realizational* theories, in contrast, the introduction of form is licensed by particular morphosyntactic features.

All four logically possible combinations of these contrasts are found; see Table 3.2.

Thus a classic lexical–incremental theory is that of Lieber (1992); in that theory, affixes are lexical entries with features just like roots and stems. Combinations of affixes are controlled by subcategorization and features are combined by *percolation*.[3] A lexical–realizational theory is that of Halle and Marantz (1993). Here, morphosyntactic features are combined in the syntax, but the expression of these features in the morphology is controlled by insertion rules that pick affixes that are compatible (given their lexical entries) with the syntax-derived features.

Inferential–incremental theories are exemplified by Articulated Morphology (Steele, 1995), wherein affixal material is introduced by rule, rather than affixes having separate lexical entries. Here, rules not only introduce the phonological changes to the word, but also effect changes in the morphosyntactic feature matrices. Finally, inferential–realizational theories include those of Matthews (1972), Anderson (1992), Aronoff (1994), and Stump himself. Needless to say, Stump's major objective is to argue that the facts of morphology dictate an inferential–realizational theory rather than any of the other three logical possibilities.

In what follows we will consider only the two extremes of Stump's four possible theories, namely lexical–incremental versus inferential–realizational, and argue that at least at a computational level, there is no difference between the two approaches. To the extent that this argument

[3] In more sophisticated lexical–incremental theories, feature combination would involve unification of attribute-value matrices.

is convincing to the reader, it will follow that the other two types of theories are also amenable to the same reduction.

Before proceeding, we note that Karttunen (2003) reached much the same conclusion as we will reach here. In that paper, he argued that Stump's theory was in fact reducible to finite-state operations. From this observation it follows straightforwardly that Stump's theory is formally equivalent to any other morphological models that can be reduced to finite-state mechanisms.

3.2 Stump's Theory

Stump's inferential–realizational theory, which he calls *Paradigm Function Morphology*, is arguably the most carefully articulated theory of morphology to have appeared in recent years, and so we shall use his theory as the example of recent theoretical work in morphology. Stump's main goal, as we have noted already, is to argue that the data from morphology support an inferential–realizational theory of the kind that he presents. On the other hand, when one considers this issue from a computational point of view – i.e., from the point of view of someone who is trying to build a system that actually implements morphological alternations – the distinctions that Stump draws between his approach and a lexical–incremental theory are less clear. But we are getting ahead of ourselves.

Stump presents two fundamental arguments in favor of realizational over incremental approaches to inflectional morphology. First, inflectional morphology often exhibits *extended exponence*, by which is meant that a given morphosyntactic feature bundle may be spelled out by material scattered across the word. An example is Breton double noun plurals such as *bagoùigoù* "little boats" (sg. *bagig*), where the boldface suffix *-où* occurs twice, once after the stem *bag* and once after the diminutive suffix *-ig*. This is, on the face of it, problematic for incremental theories since the expectation is that a single morphosyntactic feature bundle should be expressed by a single morphological object. In contrast, realizational theories make weaker claims in this regard, and thus are fully compatible with extended exponence.

A second argument for realizational theories over incremental ones comes from cases where there is no overt marker for a particular morphosyntactic feature bundle, and the form of the word thus underdetermines the actual morphosyntactic features of that word. Stump gives the example of Bulgarian verb inflection where the first singular form

in both the imperfect and aorist consists of a verb stem, followed by a vowel indicating the aspect/tense, followed by a stem-forming suffix; thus *krad-'á-x* ("steal" imperfect 1sg) and *krád-o-x* ("steal" aorist 1sg) consist of the stem *krad*, the thematic vowel (*'a*, *o*) and the stem-forming suffix *-x*. Crucially there is no overt mark for first person singular. The only way to handle this case under an incremental theory is to assume a zero morpheme. Again, such phenomena are fully compatible with the weaker assumptions of realizational theories.

In a similar vein, cases such as the Breton example given above would seem to mitigate against lexical theories, whereas they are fully compatible with inferential theories. As Stump notes, in a lexical theory such as Lieber's (1980), a morphosyntactic feature bundle should be traceable to one and only one lexical item, consisting of a single affix. In contrast, in an inferential theory, since the only requirement is that a rule be sensitive to a particular set of morphosyntactic features, one expects to find cases where a given set triggers more than one rule: since morphosyntactic features are a property of a given word and are always present, there is nothing to stop more than one rule firing – or, as in the case of Breton, the same rule firing more than once – at different stages of the construction of a word form.

Thus, it seems that there are large differences between lexical–incremental theories on the one hand, and inferential-realizational theories on the other, and furthermore there is evidence favoring the latter types of theory over the former. It is therefore perhaps surprising that at least from a computational point of view, there turns out to be essentially no difference between the two types of theory. We turn directly to a discussion of this topic.

3.3 Computational Implementation of Fragments

In what follows, we present a computational analysis of data discussed in Stump, with a view to showing that, at least from a computational point of view, there is no fundamental difference between lexical–incremental theories and inferential–realizational theories. In each case the example is a fragment of the morphology in question: we do not attempt to offer as complete a treatment as Stump (although note that his descriptions are fragments also). However, we believe that we have provided proof-of-concept implementations of the selected phenomena.

3.3.1 Stem Alternations in Sanskrit

In highly inflected languages, it is common to find alternations in the shape of stems depending upon the particular position in a paradigm. Often these alternations are quite systematic, in that words might have, say, two distinct stems, and each of these two stems will be associated with particular paradigm slots. For example, one might find that Stem I is used in the nominative singular, accusative singular and genitive plural; and that Stem II is used in all other cases. This pattern will likely be found for any lexeme that belongs to the inflectional class in question. While the stem alternations will be predictable at an abstract level, it will frequently not be the case that the *form* of the alternation is predictable in any general way. Thus, different lexemes may have different mechanisms for forming different stems so that if one knows how a lexeme λ_i forms Stem I and Stem II, it will not necessarily tell us how λ_j forms those two stems. The alternations are thus morphomic, in Aronoff's (1994) sense. To complicate matters further, a language may have a fixed set of mechanisms for forming stem alternants, but for a particular alternation for a particular paradigm, the mechanisms may distribute differently across different lexemes. Stump discusses a case of this kind from Sanskrit.

Tables 3.3 and 3.4 give a flavor of the issues. Table 3.3 shows the masculine declension of the adjective *bhagavant* "fortunate". This adjective has two stems, a strong stem, which is used in the nominative and accusative, singular and dual, as well as in the nominative plural; and a middle stem, which is used elsewhere.

Some forms have three stems. So Table 3.4 shows the masculine declension of *tasthivans*. The strong stem for *tasthivans* is distributed

TABLE 3.3 Stem alternation in the masculine paradigm of Sanskrit *bhagavant* "fortunate", with strong stem **bhágavant** and middle stem bhágavat

	SG	DU	PL
NOM	**bhágavān**	**bhágavant-āu**	**bhágavant-as**
ACC	**bhágavant-am**	**bhágavant-āu**	bhágavat-as
INSTR	bhágavat-ā	bhágavad-bhyām	bhágavad-bhis
DAT	bhágavat-e	bhágavad-bhyām	bhágavad-bhyas
ABL	bhágavat-as	bhágavad-bhyām	bhágavad-bhyas
GEN	bhágavat-as	bhágavat-os	bhágavat-ām
LOC	bhágavat-i	bhágavat-os	bhágavat-su

(= Stump's Table 6.1, page 170)

TABLE 3.4 Stem alternation in the masculine paradigm of Sanskrit *tasthivans* "having stood", with strong stem **tasthivā́ṃs**, middle stem tasthivát, and weakest stem *tasthúṣ*

	SG	DU	PL
NOM	**tasthivā́n**	**tasthivā́ṃs-āu**	**tasthivā́ṃs-as**
ACC	**tasthivā́ṃs-am**	**tasthivā́ṃs-āu**	*tasthúṣ*-as
INSTR	*tasthúṣ*-ā	tasthivád-bhyām	tasthivád-bhis
DAT	*tasthúṣ*-e	tasthivád-bhyām	tasthivád-bhyas
ABL	*tasthúṣ*-as	tasthivád-bhyām	tasthivád-bhyas
GEN	*tasthúṣ*-as	*tasthúṣ*-os	*tasthúṣ*-ām
LOC	*tasthúṣ*-i	*tasthúṣ*-os	tasthivát-su

(= Stump's Table 6.3, page 174)

the same way as the strong stem for *bhagavant*, but in addition this lexeme has both a middle and weakest stem. The weakest stem is used throughout the singular (except for the nominative and accusative), in the dual genitive and locative and in the plural accusative and genitive.

Cross-cutting these stem alternations are another class of alternations, namely the grade alternations of *vṛddhi*, *guṇa* and *zero* grade. For the stem *pād-* "foot", for example, the *vṛddhi* form is *pād-*, the *guṇa* is *pad-* and the *zero* is *pd-*. The stem alternations cannot be reduced to these grade alternations. In particular the strong stem may be either *vṛddhi* or *guṇa*, or in some cases a stem form that is not one of the standard grades, depending upon the lexeme. Similarly, the middle stem may be either *zero*, or one of a number of lexeme or lexeme-class specific stems, again depending upon the lexeme. Finally, for those lexemes that have a weakest stem, this may occasionally be a *zero* stem, but it is more often one of a number of lexeme or lexeme-class specific stems. For *bhagavant* the strong stem is *guṇa* grade and the middle stem is *zero* grade. For *tasthivans* the strong stem is *vṛddhi* grade, and the middle and weakest stems are derived by mechanisms that are particular to the class to which *tasthivans* belongs.

Stump uses these facts to argue for the *Indexing Autonomy Hypothesis*, whereby a stem's index, which indicates which particular stem (e.g., strong, middle, weakest) will be used, is independent of the form assigned to that stem. Stem alternants are thus highly abstract objects. They seem to serve no particular function other than a purely morphological one. Neither can they in general be reduced to other alternations such as the *vṛddhi/guṇa/zero* grade alternation. They are, in Aronoff's terms, purely morphomic.

In Figures 3.1–3.5 we give a *lextools* implementation of a tiny fragment of Sanskrit, following Stump's description. Included are the following basic components:[4]

- Symbol file: Lextools file of symbols — Fig. 3.1
- DummyLexicon: Template for lexical forms — Fig. 3.2
- Features: Rules introducing predictable morphosyntactic feature sets given the specification of the lexical class of the lexeme — Fig. 3.2
- Endings: Realizational rules that spell out endings appropriate to the features — Fig. 3.2
- Referral plus Filter: Rules of referral to handle syncretic forms, plus a filter to filter out forms that did not undergo the rules of referral — Fig. 3.3
- Stem: Stem selection rules — Fig. 3.4
- Grademap: Lexeme-specific map between the morphomic stems and the grade — Fig. 3.4
- Grade: Phonological forms of the grade — Fig. 3.4
- Phonology: Phonological alternations — Fig. 3.5
- Lexicon: Toy minilexicon containing entries for *bhagavant* and *tasthivans* — Fig. 3.5

The fragment of Sanskrit above can be implemented by composing the transducers 1–9 together as follows: [5]

$$\begin{aligned} \textit{Suffixes} = \textit{DummyLexicon} \circ \textit{Features} \circ \textit{Endings} \circ \\ \textit{Referral} \circ \textit{Filter} \circ \textit{Stem} \circ \\ \textit{Grademap} \circ \textit{Grade} \circ \textit{Phonology} \end{aligned} \tag{3.1}$$

The entire morphology for this fragment can then be implemented as follows:

$$\textit{Morphology} = \textit{Lexicon} \circ \textit{Suffixes} \tag{3.2}$$

But note now that we can factor this problem in another way. In particular, we can easily select out portions of the Suffixes FST that match

[4] This and other fragments discussed in this chapter are downloadable from the web site for this book, <http://compling.ai.uiuc.edu/catms>. Information on *lextools* can be found in Appendix 3A. See Appendix 3B for an XFST implementation of the Sanskrit data due to Dale Gerdemann, p.c.

[5] The function of the DummyLexicon is in part to keep this composition tractable, by limiting the left-hand side of the composition to a reasonable subset of Σ^*.


```
VShort  a e i o u au ai
VLong   a= e= i= o= u= a=u a=i
Vowel   VShort VLong
UCons   k c t  t p
VCons   g j d  d b
UCons   kh ch t.h th ph
VCons   gh jh d.h dh bh
SCons     n~ n  n m
UCons     sh s  s f
SCons   v y
Cons    UCons VCons SCons
Seg Cons Vowel
gend    masc fem neut
case    nom acc instr dat abl gen loc
num sg du pl
Category:  noun case gend num
Grade       Guna Vrddhi Zero ivat us
GradeClass  GZN VIUs
Diacritics  Ending MascNoun
Diacritics  Strong Middle Weakest
Diacritics  Grade GradeClass
Class       MascNoun
```

FIGURE 3.1 Lextools symbol file for implementation of a mini-fragment of Sanskrit, following Stump (2001). Note that the labels *ivat* and *us* are used to mark stem alternations needed for *bhagavant* and *tasthivans*

certain feature specifications. Thus, for instance, we can select the set of instrumental dual suffixes as follows:

$$\begin{aligned} \textit{Suffix}_{instr,du} = \textit{Suffixes} \circ (\Sigma^*[\text{noun gen} = \text{masc} \\ \text{case} = \text{instr num} = \text{du}]\Sigma^*) \end{aligned} \tag{3.3}$$

And then one can affix this particular suffix to the base as follows:

$$[\text{bh}]\text{agavant}[\text{GZN}][\text{MascNoun}] \circ \textit{Suffix}_{instr,du} \tag{3.4}$$

If this looks like an ordinary case of lexical, incremental affixation, this is no accident. Note in particular that the full form is being constructed using composition, which we argued in the previous chapter to be the most general single operation that can handle all forms of morphological expression. Furthermore, the feature specifications are introduced with the suffix, for example, $\textit{Suffix}_{instr,du}$. In no sense is it the case, in the above factorization, that the particular form of the affix (in this case *-bhyām*), is introduced by a rule that is sensitive to features already present, and thus the operation would appear to be incremental. In a similar fashion it would appear to be lexical: the suffix is a separate entity that includes the case and number features and which combines with the base to form the fully inflected noun.

Now there is no question that an approach such as Stump's has some benefit at elucidating generalizations about Sanskrit paradigms.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
## Transducer 1. DummyLexicon

## Template for lexical entries: they consist of
## a string of segments followed by a grade class
## such as GZN (Guna-Zero-Nothing), or VIUs (Vrddhi-ivat-us);
## followed by a class, such as MascNoun

[Seg]* [GradeClass][Class]

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
## Transducer 2. Features

## This rule introduces the actual feature matrices, given the lexical
## class of the stem. Note that nouns (actually nominals, including
## adjectives) are specified for case and number in addition to
## gender. The following rule thus produces a lattice of possible
## feature specifications, with only the gender feature specified. The
## Ending diacritic is just a placeholder for where the endings will
## go.

[MascNoun] -> [noun gend=masc][Ending]

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
## Transducer 3. Endings

## This spells out the endings given the feature specifications of the
## base. These are thus straight realizational rules. These are just
## the endings applicable to masculine adjectives such as
## bhagavant and tasthivans

[Ending] -> [<epsilon>] /  [noun case=nom gend=masc num=sg] __
[Ending] -> am / [noun case=acc gend=masc num=sg] __
[Ending] -> [a=] / [noun case=instr gend=masc num=sg] __
[Ending] -> e / [noun case=dat gend=masc num=sg] __
[Ending] -> as / [noun case=abl gend=masc num=sg] __
[Ending] -> i / [noun case=loc gend=masc num=sg] __

[Ending] -> [a=]u / [noun case=nom gend=masc num=du] __
[Ending] -> [bh]y[a=]m / [noun case=instr gend=masc num=du] __
[Ending] -> os / [noun case=gen gend=masc num=du] __

[Ending] -> as / [noun case=nom gend=masc num=pl] __
[Ending] -> [bh]is / [noun case=instr gend=masc num=pl] __
[Ending] -> [bh]yas / [noun case=dat gend=masc num=pl] __
[Ending] -> [a=]m / [noun case=gen gend=masc num=pl] __
[Ending] -> su / [noun case=loc gend=masc num=pl] __
```

FIGURE 3.2 Transducers 1–3 for implementation of a mini-fragment of Sanskrit, following Stump (2001)

Take the rules of referral, for example. The rules, as we have presented them, crosscut many paradigms and are not just properties of particular lexical items. As such, it is useful to have a notion of "paradigm" to which such rules can refer (though in actual fact, they refer to morphosyntactic feature combinations, rather than paradigm cells per se). But no matter: what is clear is that whatever the descriptive merits of Stump's approach, it is simply a mistake to assume that this forces


```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
## Transducer 4. Referral

## These are the rules of referral that specify the syncretic
## relations in the paradigm

Optional

[noun case=abl gend=masc num=sg] -> [noun case=gen gend=masc num=sg]

[noun case=nom gend=masc num=du] -> [noun case=acc gend=masc num=du]
[noun case=instr gend=masc num=du] -> [noun case=dat gend=masc num=du]
[noun case=instr gend=masc num=du] -> [noun case=abl gend=masc num=du]
[noun case=gen gend=masc num=du] -> [noun case=loc gend=masc num=du]

[noun case=nom gend=masc num=pl] -> [noun case=acc gend=masc num=pl]
[noun case=dat gend=masc num=pl] -> [noun case=abl gend=masc num=pl]

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
## Transducer 5. Filter

## This filters out all strings ending in the diacritic Ending.
## This simply eliminates all forms that have not had an ending
## added. Note that the realizational ending rules above do not have
## specifications for feature combinations that are created by the
## rules of referral. This means that if we have a combination such as
## [noun case=abl gend=masc num=pl], and it is generated by the base
## rule in "Features" it will not get an ending; it will thus end in the
## Ending diacritic and be filtered out. The only case of
## [noun case=abl gend=masc num=pl] that will pass the filter is that
## created by the rules of referral.

[<sigma>]* ([<sigma>] - [Ending])
```

FIGURE 3.3 Transducers 4–5 for implementation of a mini-fragment of Sanskrit, following Stump (2001)

one to view morphology as realizational–inferential rather than, say, lexical–incremental, at a mechanistic level. The two are no more than refactorizations of each other.

3.3.2 *Position Classes in Swahili*

In many languages, complex words are built up by concatenating classes of morphemes together in a fixed order. In Finnish, for example, as we saw above, inflected nouns consist of a stem, a number affix and a case affix. In Swahili, inflected verbs have a fixed set of prefixal position classes so that, for example, many verbs have the structure: (NEG)-SUBJECTAGREEMENT-TENSE-STEM.

On the face of it such facts would appear to call most naturally for an item-and-arrangement, or in Stump's terms, lexical–incremental view. After all, the complex words would seem to be composed out of distinct pieces, and each of these pieces would appear to be identifiable with a clear set of morphosyntactic features. But realizational theories

```
##########################################################################
## Transducer 6. Stem

## These are the stem selection rules. By default introduce the
## Weakest stem, but modify it to Strong for a particular set of
## features, to Middle for everything in the GZN grade class, and to
## Middle everywhere else.

[<epsilon>] -> [Weakest] / __ [GradeClass]

[Weakest] -> [Strong] / __ [GradeClass] [noun gend=masc case=nom]
[Weakest] -> [Strong] / __ [GradeClass] [noun gend=masc case=acc num=sg]
[Weakest] -> [Strong] / __ [GradeClass] [noun gend=masc case=acc num=du]

[Weakest] -> [Middle] / __ [GZN]

[Weakest] -> [Middle] / __ [GradeClass] [noun gend=masc case=instr num=du]
[Weakest] -> [Middle] / __ [GradeClass] [noun gend=masc case=dat num=du]
[Weakest] -> [Middle] / __ [GradeClass] [noun gend=masc case=abl num=du]
[Weakest] -> [Middle] / __ [GradeClass] [noun gend=masc case=instr num=pl]
[Weakest] -> [Middle] / __ [GradeClass] [noun gend=masc case=dat num=pl]
[Weakest] -> [Middle] / __ [GradeClass] [noun gend=masc case=abl num=pl]
[Weakest] -> [Middle] / __ [GradeClass] [noun gend=masc case=loc num=pl]

##########################################################################
## Transducer 7. Grademap

## This maps the selected stem to the particular grade given the grade
## class of the lexeme

[Middle] [GZN] -> [Zero]
[Strong] [GZN] -> [Guna]
[Strong] [VIUs] -> [Vrddhi]
[Middle] [VIUs] -> [ivat]
[Weakest] [VIUs] -> [us]

##########################################################################
## Transducer 8. Grade

## This implements the phonological expression of the grade. Thus
## "avant" becomes "avat" in Zero grade.

avant -> avat / __ [Zero]

iv[a=]ms -> u[s.] / __ [us]
iv[a=]ms -> ivat / __ [ivat]

[Grade] -> [<epsilon>]
```

FIGURE 3.4 Transducers 6–8 for implementation of a mini-fragment of Sanskrit, following Stump (2001)

are committed to the idea that affix material is introduced by rule, and inferential theories are committed to the idea that such rules are triggered by morphosyntactic features.

Within a lexical–incremental theory, the structure of complex words such as those in Swahili would be described in terms of a "word syntax", which would either specify slots into which morphemes of particular


```
##########################################################################
## Transducer 9. Phonology

## Finally, this implements some phonological rules such as the
## expression of "-vant" or "-v[a=]ms" as "-v[a=]n" in the nominative
## singular and the voicing of "t" to "d" before a following voiced
## consonant. Note that [<bos>] and [<eos>] denote the beginning and
## end of string, respectively.

(vant|v[a=]ms) -> v[a=]n / __ [noun] [<eos>]
vat -> vad / __ [noun] [VCons]

##########################################################################
## 10. Lexicon

## A minilexicon for the two adjectives "bhagavant" 'fortunate' and
## "tasthivans" 'having stood'.

[bh]agavant [GZN] [MascNoun]
tas[th]iv[a=]ms [VIUs] [MascNoun]
```

FIGURE 3.5 Transducer 9 and lexicon for an implementation of a mini-fragment of Sanskrit, following Stump (2001)

classes could be placed, or else would specify subcategorization requirements that would guarantee that affixes would attach in a particular order; another alternative is to assume a principle such as Baker's *Mirror Principle* (Baker, 1985), whereby the construction of morphologically complex words is controlled by aspects of phrasal syntax. But no matter: within lexical–incremental theories the construction of complex words is stated in terms of syntactic conditions on the arrangement of discrete morphemes, each of which bears morphosyntactic features.

Within realizational–inferential theories, such phenomena are handled by positing blocks of rules, with ordering among the blocks. So for Swahili, in Stump's analysis there is a block of rules that applies first to add tense affixes to the stem; then a block of rules adds prefixes that spell out subject agreement morphology; and finally, if the verb is marked as negative, a rule block introduces the negative morpheme.

Stump uses Swahili as an example of *portmanteau* rule blocks, by which he means cases where a single morph apparently occupies the position of two or more contiguous rule blocks. Consider the partial verbal paradigms for *taka* "want" in Table 3.5. The forms in these paradigms exemplify up to three position classes. The innermost block, Block III, comprises tense morphemes, which may also include indications of polarity, namely separate negative forms. Block IV, the next block, comprises subject agreement affixes. Finally, the optional Block V comprises the negative affix *ha-*. Each block is exemplified by one affix in a given paradigm entry, and vice versa, so that there

TABLE 3.5 Positional classes in Swahili, for *taka* "want"

	V	IV	III	Stem	
Past					
1SG		ni-	li-	taka	
2SG		u-	li-	taka	
3SG (CLASS 1)		a-	li-	taka	
1PL		tu-	li-	taka	
2PL		m-	li-	taka	
3PL (CLASS 2)		wa-	li-	taka	
Negative Past					
1SG		si-	ku-	taka	
2SG	ha-	u-	ku-	taka	(→ hukutaka)
3SG (CLASS 1)	ha-	a-	ku-	taka	(→ hakutaka)
1PL	ha-	tu-	ku-	taka	
2PL	ha-	m-	ku-	taka	
3PL (CLASS 2)	ha-	wa-	ku-	taka	
Future					
1SG		ni-	ta-	taka	
2SG		u-	ta-	taka	
3SG (CLASS 1)		a-	ta-	taka	
1PL		tu-	ta-	taka	
2PL		m-	ta-	taka	
3PL (CLASS 2)		wa-	ta-	taka	
Negative Future					
1SG		si-	ta-	taka	
2SG	ha-	u-	ta-	taka	(→ hutataka)
3SG (CLASS 1)	ha-	a-	ta-	taka	(→ hatataka)
1PL	ha-	tu-	ta-	taka	
2PL	ha-	m-	ta-	taka	
3PL (CLASS 2)	ha-	wa-	ta-	taka	

Source: Stump (2001), Table 5.1, p 140.

is a one-to-one relation between affixes and blocks, with one notable exception: in the case of the first singular, whereas we would expect to get the Block V + Block IV sequence *ha-ni-*, what we find instead is *si-*. This is an example of a portmanteau rule block: *si-* spans both Block IV and Block V.

Stump accounts for the use of *si-* by a rather intricate mechanism involving a default multi-block rule of referral. This defaults to the modes of expression of the individual blocks, but is available for Paninian override by a more specific rule that refers to those particular blocks. This analysis thus reifies a superblock – in this case


```
Vowel   a e i o u
UCons   k t p
VCons   g d b
SCons     n m
UCons     sh s f h
SCons   w y l
Cons    UCons VCons SCons
Seg Cons Vowel
num sg pl
per 1 2 3
pol pos neg
tns pst fut
gen 1,2
Category:  verb num per pol tns
Label   Verb
```

FIGURE 3.6 Lextools symbol file for an implementation of a mini-fragment of Swahili, following Stump (2001)

BlockIV + V – which will be expressed as a single morph under the condition that a specific rule referring to that block exists (as in the case of *si-* expressing first person and negative polarity features), and will be expressed as the sequence of the individual blocks otherwise.

Putting aside the merits of this particular approach over other conceivable approaches, let us consider a computational implementation along the same lines. Figures 3.6 and 3.7 show a lextools implementation of the Swahili fragment that Stump discusses. As with the Sanskrit example, we assume a lexicon that generates all possible feature expressions for a given stem. Then, affixation rules are introduced block by block to spell out morphosyntactic features on the stem.

To construct the complex forms, one simply composes the stem with each of the blocks in turn:

$$Dummy \circ BlockIII \circ BlockIV \circ BlockV \tag{3.5}$$

So far so good, but what about the case of *si-*? In the analysis in Figure 3.7, this is accomplished by doing a slight refactorization of the above composition and then combining the *si-* morpheme in a slightly different way. In particular the system is combined by composing the lexicon with Block III. Blocks IV and V are composed together, but we need to take care of the case where the expected *ha-ni* sequence is replaced by *si-*. This can be handled straightforwardly by *priority union*; see Definition 7, Section 4.3. The desired operation is:

$$si \cup_P [BlockIV \circ BlockV] \tag{3.6}$$

The range of the relation for *si-* is all entries with the specifications [verb per=1 num=sg pol=neg]. In such cases, the string prefix *si-* is inserted at the beginning of the string. The priority union ensures that

```
##########################################################################
## Transducer definitions:

## 1. Lexicon
## Note that the feature [verb] expands into all possible ways of
## expressing number, person, polarity and tense, as defined in the
## symbol file.

taka[Verb]

## 1A. Dummy lexical entry that fills in the feature vector [verb]
##     for anything tagged with the label [Verb]

[Seg]+ ([Verb]:[verb])

## 2. Rules for Block III

[<epsilon>] -> ku / [<bos>] __ [<sigma>]* [verb pol=neg tns=pst]
[<epsilon>] -> li / [<bos>] __ [<sigma>]* [verb pol=pos tns=pst]
[<epsilon>] -> ta / [<bos>] __ [<sigma>]* [verb tns=fut]

## 3. Rules for Block IV

[<epsilon>] -> ni / [<bos>] __ [<sigma>]* [verb per=1 num=sg]
[<epsilon>] -> u /  [<bos>] __ [<sigma>]* [verb per=2 num=sg]
[<epsilon>] -> a /  [<bos>] __ [<sigma>]* [verb per=3 num=sg gen=1,2]
[<epsilon>] -> tu / [<bos>] __ [<sigma>]* [verb per=1 num=pl]
[<epsilon>] -> m /  [<bos>] __ [<sigma>]* [verb per=2 num=pl]
[<epsilon>] -> wa / [<bos>] __ [<sigma>]* [verb per=3 num=pl gen=1,2]

## 4. Rules for Block V

[<epsilon>] -> ha / [<bos>] __ [<sigma>]* [verb pol=neg]

## 5. Regular expression for "si" prefix

([<epsilon>]:si)[<sigma>]* [verb per=1 num=sg pol=neg]

## 6. Phonological "cleanup" rules

aa -> a
au -> u
```

FIGURE 3.7 An implementation of a mini-fragment of Swahili, following Stump (2001)

verbs matching this feature specification can only pass through the *si*-transducer. Verbs with any other feature specification will pass through Blocks IV and V. This is a straightforward way to implement Stump's notion of a portmanteau rule block that competes with a set of cascaded blocks.

Thus, the combination of all the blocks is as follows:

$$Prefixes = Dummy \circ BlockIII \circ [si \cup_P [BlockIV \circ BlockV]] \quad (3.7)$$

For a particular verb such as *taka[Verb]* we have the following composition:

$$taka[Verb] \circ Prefixes \quad (3.8)$$

But note again that one can easily factor the problem differently. For example one can trivially redefine Block III as follows:

```
([<epsilon>]:ku) ([<sigma>]*) [verb pol=neg tns=pst]
([<epsilon>]:li) ([<sigma>]*) [verb pol=pos tns=pst]
([<epsilon>]:ta) ([<sigma>]*) [verb tns=fut]          (3.9)
```

This defines a transducer that on the one hand introduces a prefix – *ku-*, *li-*, or *ta-* – and on the other adds a specification for polarity and tense features to the verb. In this definition, the verbal feature matrix added by the affix serves as a filter to restrict the verbal affixes of the input, which are unconstrained with respect to the feature specifications for polarity and tense. The input to Block III for the verb *taka*, for example, would be:

```
taka[verb num={sg,pl} per={1,2,3} pol={pos,neg}
                          tns={pst,fut}]          (3.10)
```

In other words, in this specification *taka* is a verb with no restrictions on the specifications of the features NUM, PER, POL, or TNS. The new definition of Block III above will restrict these features and simultaneously introduce the related affixes. But this is completely equivalent to a unification mechanism that combines the affixal features with the (underspecified) features of the stem. The result is then:

```
kutaka[verb num={sg,pl} per={1,2,3} pol=neg tns=pst]
litaka[verb num={sg,pl} per={1,2,3} pol=pos tns=pst]
tataka[verb num={sg,pl} per={1,2,3} pol={pos,neg}
                                    tns=fut]      (3.11)
```

Once again, the realizational–inferential theory turns out to be formally equivalent under refactorization to a lexical–incremental theory.

3.3.3 Double Plurals in Breton

Another issue that Stump discusses is the issue of multiple exponence. We have already seen an example of that in Swahili, where negative polarity is spelled out in a couple of places, specifically in the Block III tense morpheme and as the Block V prefix *ha-*. Another example is Breton diminutive plurals such as *bag+où+ig+où* "little boats", where the base is *bag* "boat", the diminutive suffix is *-ig* and the two *-où*'s

represent the plural; note that the singular diminutive is *bagig* "little boat", so that it seems that both the stem and the diminutive affix have a plural marker.

Stump analyzes this as a case of word-to-stem derivation, whereby *-ig* suffixation derives a stem from the base *bag*. Stump's derivation is complex and depends upon a number of assumptions. The first is that Breton pluralization involves two rule blocks, although for normal (non-diminutive) nouns, there is typically just one plural suffix:

$$\mathrm{PF}(< \mathrm{X}, \{\mathrm{Num} : \alpha\} >) =_{\mathrm{def}} \mathrm{Nar}_1(\mathrm{Nar}_0(< \mathrm{X}, \{\mathrm{Num} : \alpha\} >)) \qquad (3.12)$$

Here, "Nar" is the *narrowest applicable rule* where "narrowest" is defined in the Paninian sense: the rule whose featural domain specifications are the narrowest in a block of rules that still subsumes the form in question. For normal plurals, it seems, only the inner "zero" block has an associated plural rule. "PF" denotes *Paradigm Function*, that is, a function that mediates the spell out of a cell in a paradigm. More specifically a Paradigm Function is "a function which, when applied to the root of a lexeme L, paired with a set of morphosyntactic properties appropriate to L, determines the word form occupying the corresponding cell in L's paradigm" (Stump, 2001, page 32). For *bag* "boat", the narrowest applicable rule applying on the inner block simply attaches the default plural affix *-où*.

In the case of *bagig* 'little boat', things are more interesting. Stump assumes a diminutive rule for Breton defined as a "word-to-stem" rule (page 204):

$$\mathrm{DR}_{\mathrm{dimin}}(\mathrm{X}) =_{\mathrm{def}} \mathrm{X}ig \qquad (3.13)$$

He assumes a "universal metarule for word to stem derivatives" (= Stump's 28, page 204):

> If X, Y are roots and M is a word-to-stem rule such that M(X)=Y, then for each set σ of morphosyntactic properties such that PF($< X, \sigma >$) is defined, if PF($< X, \sigma >$) = $< Z, \sigma' >$, then $\mathrm{RR}_{0,\sigma,\{\mathrm{L-index(Y)}\}}(< Y, \sigma'' >) =_{\mathrm{def}} < M(Z), \sigma'' >$.

Here, *RR* is Stump's designation for a *realization rule* – a rule that phonologically realizes a particular set of morphosyntactic features, and $\mathrm{RR}_{0,\sigma,\{\mathrm{L-index(Y)}\}}$ designates a realization rule that applies in a particular block (here, Block 0), for a particular set of morphosyntactic features (σ), to a particular lexical item (designated as "L–index(Y)"). In plain language, Stump's universal metarule states that the inflection

of a word-to-stem derivative is the same as applying that word-to-stem derivation to the inflected form of the base. Thus we have, for *bagig*, *bagoùig*.

Here the assumption of two blocks for plural formation comes into play so that we have as an intermediate step in the derivation:

$$\mathrm{RR}_{1,\{\mathrm{NUM:pl}\},\mathrm{N}}(< \mathrm{bagoùig}, \sigma >) \qquad (3.14)$$

The default rule for pluralization in Block 1 is to suffix *-où*, so we end up with the observed form *bagoùigoù*.

The universal word-to-stem metarule is applied in this case in Breton, and also in another similar example in Kikuyu. Some of Stump's other assumptions, however, seem rather more specific to Breton, and it is hard to tell how universal these could reasonably be expected to be.

We turn now to a computational treatment of the same phenomenon. A mini-grammar in lextools format is given in Figure 3.8. This grammar produces the following forms for the singular and plural diminutives and non-diminutives, with the diminutive forms being derived via the optional rule that introduces diminutive features:

```
bag[ou] [noun num=sg]     bag
bag[ou] [noun num=sg]     bagig
bag[ou] [noun num=pl]     bagou
bag[ou] [noun num=pl]     bagouigou
```

While this account might be criticized as ad hoc or non-explanatory, it is far from obvious that it is less ad hoc than Stump's account, or any less explanatory. And it does have the advantage of being easier to understand.

As before, it is possible to refactor the analysis so that the inferential–realizational rules that we have presented above are recast as lexical–incremental. For example, the plural rule can be rewritten as a pair of rules:

```
[ou]  -> ou
[sg]  -> [pl]
```

This pair of rules introduces the suffix *-où*, and changes the number feature of a singular base to plural. These two rules can be represented as a single transducer P, which can then be composed with a singular base to produce the plural form. Thus, again, the feature

```
## Lextools symbol file for Breton
Letter  a b c d e f g h i j k l m n o p q r s t u v w x y z
## Note: no nasalized or accented vowels in this short fragment
num sg pl
Feat    dim num
Category:  noun num
## ed marks nouns that take plurals in -ed
InflClass  ou ed

#####################################################################
## Transducer definitions:
##
## 1. Lexicon
##
## Here we define the entry for "bag" 'boat', which has a diacritic
## [ou] indicating that the plural is -ou. If one objects that -ou
## should be the default, then one can always replace this with a
## generic diacritic -- e.g. '[xx]' -- which will get spelled out as
## [ou] if no other plural mark is there.
##
## [noun] has all possible feature specifications for nouns.

bag[ou][noun]

## 2. Diminutive rule
##
## This optionally introduces the feature [dim]. Then it
## introduces an affix "-ig", if the feature [dim] is present. The
## affix is marked to go after the inflection class marker (the
## indicator of which plural affix the noun takes). The diminutive
## suffix itself is marked to take -ou, but again if one prefers it
## could be marked with a generic diacritic.

optional
[<epsilon>] -> [dim] / __ [<eos>]

obligatory
[<epsilon>] -> ig[ou] / [InflClass] __ [<sigma>]* [dim]

## 3. Plural spell out.
##
## Note that this spells out every [ou] as "-ou" in the context of a
## plural marking. The result: double plural marking for "bagig"

[ou] -> ou / __ [<sigma>]*[noun num=pl]

## 4. Clean up rule. This just deletes feature specification and
##    category information to derive the surface form.

[InflClass]|[Feat]|[<category>] -> [<epsilon>]
```

FIGURE 3.8 An implementation of a mini-fragment of Breton

"plural" is now introduced by an affix rule under the generic affixation operation of composition. And once again, the difference between an inferential–realizational approach and a lexical–incremental approach turns out to be a matter of factorization.

3.4 Equivalence of Inferential–Realizational and Lexical–Incremental Approaches: A Formal Analysis

It is also possible to argue for the equivalence of inferential–realizational and lexical–incremental approaches on the basis of a formal analysis of the semantics of morphological operations. The argument is the same as what we have already presented in computational terms, but stated a bit differently.

As an example consider Blevins' (2003) analysis of West Germanic. Blevins provides an analysis of the stem morphology of weak verbs in English, German, Frisian, and Dutch within the realizational framework of Stump (2001). He observes that in all West Germanic languages, the past tense, perfect participle, and passive participle all share the same stem, which is formed with a dental. In English, this is exemplified by examples such as the following:

1. PAST: John whacked the toadstool
2. PERF: John has whacked the toadstool
3. PASS: The toadstool was whacked

The past form in Sentence 1, the perfect participle in Sentence 2, and the passive participle in Sentence 3 all share the same phonological property. In other West Germanic languages, some of these forms may have additional material. Thus in German:

4. PAST: Er mähte das Heu
   "He mowed the hay"
5. PERF: Er hat das Heu *ge*mäht
   "He has mowed the hay"
6. PASS: Das Heu wurde *ge*mäht
   "The hay was mowed"

The common feature of the forms is the dental **-t**, but there is additional material in each case: in the past there is a stem vowel *-e*, and in the perfect and passive participles, there is the prefix *ge-*.

Blevins argues that one cannot view the dental suffix as being a single morpheme with a common semantics, and indeed this is also the conclusion reached by analyses based on morphemes, such as that of Pinker (1999), who argues for a set of distinct homophonous dental morphemes. This duplication is an embarrassment for lexical accounts to be sure, but it is important to bear in mind that it is simply a fact of the data and has to be incorporated somewhere in the model.

In realizational accounts, such as the one Blevins provides, such duplications are handled by allowing many-to-one mappings between semantic features and the morphological *exponents* of those features. Thus a realization function $\mathfrak{R}$ is defined as follows, for English, where $F_d(X) = Xd$ is a function that suffixes *-d* to the stem:

$$\begin{aligned} \mathfrak{R}([\text{PAST}]) &= F_d(X) \\ \mathfrak{R}([\text{PERF}]) &= F_d(X) \\ \mathfrak{R}([\text{PASS}]) &= F_d(X) \end{aligned} \tag{3.15}$$

Thus we have a many-to-one mapping between three semantic features and a single exponent.

Let us try to define these notions more formally as follows. First of all, we will use the abstract catenation operator "·" to represent the catenation of *-d* with the stem, and so we can redefine $F_d(X)$ as a single place function that concatenates *-d* to a stem, as follows:

$$F_d(X) = \lambda(\mathbf{X})[\mathbf{X} \cdot \mathbf{d}] \tag{3.16}$$

Second, the realization expressions presumably do not just realize, say, [past], but realize it with respect to a certain base, the same base to which *-d* is ultimately attached. Let us assume an operator $\oplus$ to represent the addition of the relevant feature. Thus we would write:

$$\mathfrak{R}(\lambda(\mathbf{X})[\mathbf{X} \oplus \text{PAST}]) = \lambda(\mathbf{X})[\mathbf{X} \cdot \mathbf{d}] \tag{3.17}$$

Now, one assumes that what it means to realize a particular feature or set of features on a stem by means of a particular morphological exponent is that one adds the feature and realizes the exponent of that feature.[6] So we should be able to collapse the above into:

$$\lambda(\mathbf{X})[\mathbf{X} \oplus \text{PAST} \wedge \mathbf{X} \cdot \mathbf{d}] \tag{3.18}$$

Here, $\wedge$ simply denotes the fact that both the feature combination and the catenation operations take place. But, we can condense this expression further by collapsing the two combinatoric expressions into one:

$$\lambda(\mathbf{X})[\mathbf{X} < \oplus, \cdot >< \text{PAST}, \mathbf{d} >] \tag{3.19}$$

[6] Note that this statement is neutral as to incremental versus realizational approaches. In any case, one has as input a form that lacks a set of morphosyntactic features and a particular morphological exponent; and one ends up with a new form that has the morphosyntactic features and the particular exponent.

Here <PAST, **d** > is simply a pairing of the morphosyntactic/semantic feature with the phonological exponence. We use $< \oplus, \cdot >$ to represent a catenation pair which combines elements on the morphosyntactic/semantic side using $\oplus$ and elements on the phonological side using "·"; see Sproat (2000) for a similar binary catenation pair applied to the simultaneous combination of linguistic elements with linguistic elements and graphemic elements with graphemic elements. In this formulation, we also need to consider X to be a morphosyntactic–phonological pairing, but we will leave this implicit in our notation. The above expression thus just describes a function that takes an element **X** and combines it (via $< \oplus, \cdot >$) with another expression <PAST, **d** >. This is clearly just a formulation of a standard lexical–incremental model.

3.5 Conclusions

For the past half century there has been a debate in morphological theory between, on the one hand, theories that maintain that morphology involves the assembly of small atomic meaning-bearing pieces (morphemes); and on the other, theories that maintain that the basic operations of morphology are rules that introduce or modify meanings and simultaneously effect phonological changes on the bases to which they apply. From a purely computational or formal point of view, as we have argued, the differences between such approaches are less significant than they at first appear to be.

In many cases there may be bona fide reasons for preferring one approach over another. For example, to explain the complex paradigms of Sanskrit nominal morphology, one would like to have recourse to a model where the morphological function – the cells in the paradigm and their associated features – are dissociated from the actual spellout of those features. On the other hand, there seems to be little benefit in such a model if the task is to account for, say, noun plural formation in Kannada, where this simply involves attaching the plural morpheme *-guḷu* to the end of the noun. As much as anything else, what is at issue here is not whether inferential–realizational or lexical–incremental theories are sometimes motivated, but whether they are always motivated, and whether there is any justification in the all-or-nothing view that has prevailed in morphological thinking. Such a monolithic view is often justified on the basis of Occam's razor: if a theory can account for certain phenomena elegantly, and is at least capable of accounting for other

phenomena, if not so elegantly, then that theory is to be preferred over another theory that fails to account for certain kinds of data. Unfortunately this is not the situation that obtains with morphological theories.

Both lexical–incremental theories and inferential–realizational theories can account for the data; we have seen that they are formally and computationally equivalent. Of course, as a reader of an earlier version of this book pointed out to us, if someone is interested in how humans process morphology, rather than the formal computational description of such processing, they might be inclined to wonder: "so what?" But this clearly appeals to psycholinguistic evidence, and there the data does not seem to be that friendly to either side: there is increasing evidence that human morphological processing involves a powerful set of analogical reasoning tools that are sensitive to various effects including frequency, semantic and phonetic similarity, register, and so on (Hay and Baayen, 2005; Baayen and Moscoso del Prado Martín, 2005). It is not clear that the mechanisms at work particularly favor lexical–incremental over inferential-realizational models. What these issues ultimately come down to are matters of taste. There is of course nothing wrong with taste, but it is something that is less obviously in the purview of Occam's razor.

Appendix 3A: Lextools

The following unix manual page describes the file formats and regular expression syntax for the *lextools* grammar development toolkit. It is a copy of the manual page available at the AT&T website (http://www.research.att.com/~alb/lextools/lextools.5.htm), and also available at the web page for this book http://compling.ai.uiuc.edu/catms.

```
NAMES
Lextools symbol files, regular expression syntax, general
file formats, grammar formats - lextools file formats

DESCRIPTION
Symbol Files
The lextools symbol file specifies the label set of an
application. Labels come in three basic flavors:

Basic labels

Superclass labels

Category labels
```



```
Basic labels are automatically assigned a unique integer
representation (excluding 0, which is reserved for
"<epsilon>"), and this information is compiled by lexmakelab
into the basic label file, which is in the format specified
in fsm(5).

Superclass labels are collections of basic labels: a superclass
inherits all of the integral values of all of the
basic labels (or superclass labels) that it contains.

Category labels are described below.

The lines in a symbol file look as follows:

superclass1          basic1 basic2 basic3

You may repeat the same superclass label on multiple (possibly
non-adjacent) lines: whatever is specified in the new
line just gets added to that superclass. The "basic" labels
can also be superclass labels: in that case, the superclass
in the first column recursively inherits all of the labels
from these superclass labels.

The one exception is a category expression which is specified
as follows:

Category:          catname feat1 feat2 feat3

The literal "Category:" must be the first entry: it is case
insensitive.  "catname" should be a new name. "feat1" labels
are their values.

The following sample symbol file serves to illustrate:
dletters   a b c d e f g h i j k l m n o p
dletters   q r s t u v w x y z
uletters   A B C D E F G H I J K L M N O P
uletters   Q R S T U V W X Y Z
letters    dletters uletters
gender     masc fem
case       nom acc gen dat
number     sg pl
person     1st 2nd 3rd
Category:  noun gender case number
Category:  verb number person

For this symbol set, the superclass "dletters" will contain
the labels for all the lower case labels, "uletters" the
upper case letters, and "letters" both. Defined categories
are "noun" and "verb". "noun", for instance, has the features
"gender", "case" and "number". The feature "gender"
can have the values "masc" and "fem".

(NB: this way of representing features is inherited in concept
from Lextools 2.0.)
```

Some caveats:

You should not use the reserved terms "<epsilon>" or "<sigma>" in a symbol file. If you use the -L flag to lexmakelab you should also not use any of the special symbols that it introduces with that flag (see lextools(1)).

Symbol files cannot contain comments. Backslash continuation syntax does not work.

Regular Expressions
Regular expressions consist of strings, possibly with specified costs, conjoined with one or more operators.

strings are constructed out of basic and superclass labels. Labels themselves may be constructed out of one or more characters. Characters are defined as follows:

If 2-byte characters are specified (Chinese, Japanese, Korean...), a character can be a pair of bytes if the first byte of the pair has the high bit set.

In all other conditions a character is a single byte.

Multicharacter tokens MUST BE delimited in strings by a left bracket (default: "[") and right bracket (default: "]"). This includes special tokens "<epsilon>", "<sigma>", "<bos>" and "<eos>". This may seem inconvenient, but the regular expression compiler has to have some way to figure out what a token is. Whitespace is inconvenient since if you have a long string made up of single-character tokens, you don't want to be putting spaces between every character: trust me. You may also use brackets to delimit single-character tokens if you wish.

Some well-formed strings are given below:

```
abc[foo]
[<epsilon>]ab[<sigma>]
```

Note that the latter uses the superclass "<sigma>" (constructed by lexmakelab to include all symbols of the alphabet except "<epsilon>"): in order to compile this expression, the superclass file must have been specified using the -S flag.

If features are specified in the label set, then one can specify the features in strings in a linguistically appealing way as follows:

```
food[noun gender=fem number=sg case=nom]
```

Order of the feature specifications does not matter: the order is determined by the order of the symbols in the symbol file. Thus the following is equivalent to the above:


```
food[noun case=nom number=sg gender=fem]
```

The internal representation of such feature specifications looks as follows: "food[_noun][nom][sg][fem]".

Unspecified features will have all legal values filled in. Thus

```
food[noun case=nom number=sg]
```

will produce a lattice with both [fem] and [masc] as alternatives. Inappropriate feature values will cause a warning during the compilation process. Since features use superclasses, again, in order to compile such expressions, the superclass file must have been specified using the -S flag.

Costs can be specified anywhere in strings. They are specified by a positive or negative floating point number within angle brackets. The current version of lextools assumes the tropical semiring, so costs are accumulated across strings by summing. Thus the following two strings have the same cost:

```
abc[foo]<3.0>
a<-1.0>b<2.0>c<1.0>[foo]<0.5><0.5>
```

Note that a cost on its own -- i.e. without an accompanying string -- specifies a machine with a single state, no arcs, and an exit cost equal to the cost specified.

Regular expressions can be constructed as follows. First of all a string is a regular expression. Next, a regular expression can be constructed out of one or two other regular expressions using the following operators:

```
regexp1*              Kleene star
regexp1+              Kleene plus
regexp1^n             power
regexp1?              optional
!regexp1              negation
regexp1 | regexp2     union
regexp1 & regexp2     intersection
regexp1 : regexp2     cross product
regexp1 @ regexp2     composition
regexp1 - regexp2     difference
```

In general the same restrictions on these operations apply as specified in fsm(1). For example, the second argument to "-" (difference) must be an unweighted acceptor. Note also that the two argument to ":" (cross product) must be acceptors. The argument n to "^" must be a positive integer. The arguments to "@" (composition) are assumed to be transducers.

The algorithm for parsing regular expressions finds the longest stretch that is a string, and takes that to be the (first) argument of the unary or binary operator immediately to the left, or the second argument of the binary operator immediately to the right. Thus "abcd | efgh" represents the union of "abcd" and "efgh" (which is reminiscent of Unix regular expression syntax) and "abcd*" represents the transitive closure of "abcd" (i.e., not "abc" followed by the transitive closure of d, which is what one would expect on the basis of Unix regular expression syntax).

The precedence of the operators is as follows (from lowest to highest), the last parenthesized group being of equal precedence:

```
| & - : (* + ? ^)
```

But this is hard to remember, and in the case of multiple operators, it may be complex to figure out which elements get grouped first. The use of parentheses is highly recommended:
use parentheses to disambiguate "!(abc | def)" from "(!abc) | def".

Spaces are never significant in regular expressions.

Escapes, Comment Syntax and Miscellaneous other Comments can appear in input files, with the exception of symbol files. Comments are preceded by "#" and continue to the end of the line.

You can split lines or regular expressions within lines onto multiple lines if you include "\ " at the end of the line, right before the newline character.

Special characters, including the comment character, can be escaped with "\ ". To get a "\ ", escape it with "\ ": "\ ".

Lexicons
The input to lexcomplex is simply a list of regular expressions. The default interpretation is that these expressions are to be unioned together, but other interpretations are possible: see lextools(1) for details.

If any of the regular expressions denotes a relation (i.e., a transducer) the resulting union also denotes a relation, otherwise it denotes a language (i.e., an acceptor).

Arclists
An arclist (borrowing a term from Tzoukermann and Liberman's 1990 work on Spanish morphology) is a simple way to specify a finite-state morphological grammar. Lines can be of one of

the following three formats:

```
instate    outstate    regexp
finalstate
finalstate cost
```

Note that cost here should be a floating point number not enclosed in angle brackets. State names need not be enclosed in square brackets: they are not regular expressions.

The following example, for instance, specifies a toy grammar for English morphology that handles the words, "grammatical", "able", "grammaticality", "ability" (mapping it to "able + ity"), and their derivatives in "un-":

```
START   ROOT     un [++] | [<epsilon>]
ROOT    FINAL    grammatical | able
ROOT    SUFFIX   grammatical
ROOT    SUFFIX   abil : able
SUFFIX  FINAL    [++] ity
FINAL   1.0
```

Paradigms
The paradigm file specifies a set of morphological paradigms, specified as follows.

Each morphological paradigm is introduced by the word "Paradigm" (case insensitive), followed by a bracketed name for the paradigm:

```
Paradigm        [m1a]
```

Following this are specifications of one of the following forms:

```
Suffix     suffix    features
Prefix     prefix    features
Circumfix  circumfix features
```

The literals "Suffix", "Prefix" and "Circumfix" are matched case-insensitively. The remaining two fields are regular expressions describing the phonological (or orthographic) material in the affix, and the features. The "Circumfix" specification has a special form, namely "regexp...regexp". The three adjacent dots, which must be present, indicate the location of the stem inside the circumfix. In all cases, features are placed at the end of the morphologically complex form. There is no provided mechanism for infixes, though that would not be difficult to add.

One may specify in a third field in the "Paradigm" line another previously defined paradigm from which the current paradigm inherits forms:

```
Paradigm        [mo1a] [m1a]
```

In such a case, a new paradigm will be set up, and all the forms will be inherited from the prior paradigm except those forms whose features match entries specified for the new paradigm: in other words, you can override, say, the form for "[noun num=sg case=dat]" by specifying a new form with those features. (See the example below.) One may also add additional entries (with new features) in inherited paradigms.

A sample set of paradigms (for Russian) is given below:

```
Paradigm         [m1a]
Suffix  [++]     [noun num=sg case=nom]
Suffix  [++]a    [noun num=sg case=gen]
Suffix  [++]e    [noun num=sg case=prep]
Suffix  [++]u    [noun num=sg case=dat]
Suffix  [++]om   [noun num=sg case=instr]
Suffix  [++]y    [noun num=pl case=nom]
Suffix  [++]ov   [noun num=pl case=gen]
Suffix  [++]ax   [noun num=pl case=prep]
Suffix  [++]am   [noun num=pl case=dat]
Suffix  [++]ami  [noun num=pl case=instr]
Paradigm         [mo1a]  [m1a]
Paradigm         [m1e]   [m1a]
Suffix  [++]"ov  [noun num=pl case=gen]
Suffix  [++]"ax  [noun num=pl case=prep]
Suffix  [++]"am  [noun num=pl case=dat]
Suffix  [++]"ami[noun num=pl case=instr]
Suffix  [++]"ami[noun num=pl case=instr]
```

Note that "[mo1a]" inherits all of "[m1a]", whereas "[m1e]" inherits all except the genitive, prepositional, dative and instrumental plurals.

See lextools(1) for some advice on how to link the paradigm labels to individual lexical entries in the lexicon file argument to lexparadigm.

Context-Free Rewrite Rules
The input to lexcfcompile is a set of expressions of the following form:

```
NONTERMINAL -> regexp
```

The "->" must be literally present. "NONTERMINAL" can actually be a regular expression over nonterminal symbols, though the only useful regular expressions in this case are unions of single symbols. The "regexp" can in principle be any regular expression specifying a language (i.e., not a relation) containing a mixture of terminals and non-terminals. However, while lexcfcompile imposes no restrictions on what you put in the rule, the algorithm implemented in GRMCfCompile, which lexcfcompile uses, can only handle certain kinds of context-free grammars. The user is strongly advised to read and understand the description in grm(1) to understand the restrictions on the kinds of

context-free grammars that can be handled.

By default the start symbol is assumed to be the first non-terminal mentioned in the grammar; see lextools(1) for further details.

The following grammar implements the toy English morphology example we saw above under Arclists, this time putting brackets around the constituents (and this time without the mapping from "abil" to "able"):

```
[NOUN] -> \[ ( \[ [ADJ] \] | \[ [NEGADJ] \] ) ity \]
[NOUN] -> \[ [ADJ] \] | \[ [NEGADJ] \]
[NEGADJ] -> un \[ [ADJ] \]
[ADJ] -> grammatical | able
```

Context-Dependent Rewrite Rules
A context-dependent rewrite rule file consists of specifications of one of the following two forms:

```
phi -> psi / lambda __ rho
phi => psi / lambda __ rho
```

In each case "phi", "psi", "lambda" and "rho" are regular expressions specifying languages (acceptors). All but "psi" must be unweighted (a requirement of the underlying GRMCd-Compile; see grm(1), grm(3)). The connectors "->", "=>", and "/" must literally occur as such. The underbar separating "lambda" and "rho" can be any number of consecutive underbars. The interpretation of all such rules is that "phi" is changed to "psi" in the context "lambda" on the left and "rho" on the right.

The difference between the two productions, "->" and "=>" is the following. "->" denotes a mapping where any element of "phi" can be mapped to any element of "psi". With "=>", the inputs and outputs are matched according to their order in the symbol file: this is most useful with single (superclass) symbol to single (superclass) symbol replacements.
For example, suppose you have the following entries in your symbol file:

```
V           a e i o u
+voiced     b d g
-voiced     p t k
```

The rule:

```
[-voiced] -> [+voiced] / V __ V
```

will replace any symbol in {p,t,k} with any symbol in {b,d,g} between two vowels. Probably what you want in this case is the following:

```
[-voiced] => [+voiced] / V __ V
```

```
This will replace "p" with "b", "t" with "d" and "k" with
"g". The matching is done according to the order of the symbols
in the symbol file. If you had specified instead:

+voiced     b g d
-voiced     p t k

then "t" would be replaced with "g" and "k" with "d". Similarly
with

+voiced     b d g
-voiced     p t k x

"p", "t" and "k" would be replaced as in the first case, but
"x" would be ignored since there is nothing to match it to:
nothing will happen to "x" intervocalically. Use of the
matching rule specification "=>" thus requires some care in
labelset management.

Beginning of string and end of string can be specified as
"[<bos>]" and "[<eos>]", respectively: these are added to
the label set by default if you use the -L flag to lexmakelab.

A line may also consist of one of the following specifications
(which are case insensitive):

left-to-right
right-to-left
simultaneous
optional
obligatory

The first three set the direction of the rule application;
the last two set whether the rule application is obligatory
or optional; see grm(1). All specifications are in effect
until the next specification or until the end of the file.
The default setting is obligatory, left-to-right. In prac-
tice the user will rarely need to fiddle with these default
settings.

Replacements
A replacement specification (for lexreplace) is a file
consisting of lines like the following:

foo.fst      a|b|c|d

The first column specifies a single fsm that must exist in
the named file. The remainder of the line specifies a union
of labels to be replaced in the topology fsm argument to
lexreplace with said fsm. Specified substitutions for a
given label will override any previous substitutions. In the
following case:

foo.fst      a|b|c|d
bar.fst      a
```



```
you will foo.fst for "b", "c" and "d", and bar.fst for "a".

See also grmreplace in grm(1).


Currency Expressions
Currency specification files contain lines of the following
form:

sym major-expr point minor-expr large-number

Each entry is a regular expression.  See lextools(1) for a
description of the function of the entries.

Note that the whitespace separator MUST BE A TAB: the regular
expressions themselves may contain non-tab whitespace.
There must therefore be four tabs. You must still have tabs
separating entries even if you split an entry across multiple
lines (with "\ ").
```

Appendix 3B: XFST Implementation of Sanskrit

Dale Gerdemann has very kindly provided us with his XFST (Beesley and Karttunen, 2003) reimplementation of the *lextools* model of Sanskrit presented in Section 3.3.1. We reproduce his implementation verbatim below.

```
define VShort a | e | i | o | u | au | ai;
define VLong a= | e= | i= | o= | u= | a=u | a=i;
define Vowel VShort | VLong;
define UCons k | c | t%. | t | p | kh | ch | t%.h | th | ph |
sh | s%. | s | f;
define VCons g | j | d%. | d | b | gh | jh | d%.h | dh | bh;
define SCons n%~ | n%. | n | m | v | y;
define Cons UCons | VCons | SCons;
define Seg Cons | Vowel;
define gend masc | fem | neut;
define case nom | acc | instr | dat | abl | gen | loc;
define num sg | du | pl;
define noun Noun case= case gend= gend num= num;
define Grade Guna | Vrddhi | Zero | ivat | us;
define GradeClass GZN | VIUs;
define Diacritics Ending | MascNoun | Strong | Middle | Weakest
| Grade | GradeClass;

define Class MascNoun;

define DummyLexicon Seg* GradeClass Class;

define Features MascNoun -> [noun & $masc] Ending;
```

```
define Endings [
Ending -> 0 || case= nom gend= masc num= sg \_
.o.
Ending -> {am} || case= acc gend= masc num= sg \_
.o.
Ending -> a= || case= instr gend= masc num= sg \_
.o.
Ending -> e || case= dat gend= masc num= sg \_
.o.
Ending -> {as} || case= abl gend= masc num= sg \_
.o.
Ending -> i || case= loc gend= masc num= sg \_
.o.

Ending -> a={u} || case= nom gend= masc num= du \_
.o.
Ending -> bh{y}a={m} || case= instr gend= masc num= du \_
.o.
Ending -> {os} || case= gen gend= masc num= du \_
.o.

Ending -> {as} || case= nom gend= masc num= pl \_
.o.
Ending -> bh{is} || case= instr gend= masc num= pl \_
.o.
Ending -> bh{yas} || case= dat gend= masc num= pl \_
.o.
Ending -> a= m || case= gen gend= masc num= pl \_
.o.
Ending -> {su} || case= loc gend= masc num= pl \_
];


define Referral [
case= abl gend= masc num= sg (->) case= gen gend= masc num= sg
.o.

case= nom gend= masc num= du (->) case= acc gend= masc num= du
.o.
case= instr gend= masc num= du (->) case= dat gend= masc
num= du
.o.
case= instr gend= masc num= du (->) case= abl gend= masc
num= du
.o.
case= gen gend= masc num= du (->) case= loc gend= masc num= du
.o.

case= nom gend= masc num= pl (->) case= acc gend= masc num= pl
.o.
case= dat gend= masc num= pl (->) case= abl gend= masc num= pl
];


define Filter~ [?* Ending];
```



```
define Stem [
[..] -> Weakest // \_ GradeClass
.o.
Weakest -> Strong // \_ GradeClass [noun & $masc & $nom]
.o.
Weakest -> Strong // \_ GradeClass [noun & $masc & $acc & $sg]
.o.
Weakest -> Strong // \_ GradeClass [noun & $masc & $acc & $du]
.o.
Weakest -> Middle // \_ GZN
.o.
Weakest -> Middle // \_ GradeClass [noun & $masc & $instr & $du]
.o.
Weakest -> Middle // \_ GradeClass [noun & $masc & $dat & $du]
.o.
Weakest -> Middle // \_ GradeClass [noun & $masc & $abl & $du]
.o.
Weakest -> Middle // \_ GradeClass [noun & $masc & $instr & $pl]
.o.
Weakest -> Middle // \_ GradeClass [noun & $masc & $dat & $pl]
.o.
Weakest -> Middle // \_ GradeClass [noun & $masc & $abl & $pl]
.o.
Weakest -> Middle // \_ GradeClass [noun & $masc & $loc & $pl]
];


define Grademap [
[Middle GZN] -> Zero
.o.
[Strong GZN] -> Guna
.o.
[Strong VIUs] -> Vrddhi
.o.
[Middle VIUs] -> ivat
.o.
[Weakest VIUs] -> us
];

define Grade [
{avant} -> {avat} // \_ Zero
.o.
{iv}a={ms} -> {u}s%. // \_ us
.o.
{iv}a={ms} -> {ivat} // \_  ivat
.o.
Grade -> 0
];

define Phonology [
[{vant}|{v}a={ms}] -> {v}a={n} // \_ noun .#.
.o.
{vat} -> {vad} // \_ noun VCons
];
```

```
define Suffixes [
DummyLexicon
.o.
Features
.o.
Endings
.o.
Referral
.o.
Filter
.o.
Stem
.o.
Grademap
.o.
Grade
.o.
Phonology
];

define Lexicon [bh{agavant} GZN MascNoun] |
[{tas}th{iv}a={ms} VIUs MascNoun];

# Add spaces for nice printout

define Spacing [
[..] -> % ||  \_ $\backslash$[Seg | case | gend | num]
.o.
[..] -> % || num \_
];

define Morphology [
Lexicon .o. Suffixes
];

## Morphology (as above) defines a transducer with input:
##
## {tasthiv[a=]ms[VIUs][MascNoun], bhagavant[GZN][MascNoun]}*
## *Multicharacter symbols marked up as in FSM
##
## These inputs are mapped to
##
## {tasthivat[Noun case=loc gend=masc num=pl]su
##  tasthivad[Noun case=instr gend=masc num=du]bhy[a=]m
##    ...
##  bhagavat[Noun case=acc gend=masc num=pl]as
##  bhagavat[Noun case=instr gend=masc num=sg][a=]
##    ...
##  }
##

define Lemmatizer [
[Morphology .o. noun -> 0].i .o. [Class | GradeClass] -> 0
];
```



```
define MorphologicalTagger [
0 <- noun .o. [Lexicon .o. Suffixes ].l .o. Seg -> 0
];


read regex Morphology .o. Spacing;
read regex Lemmatizer;
read regex MorphologicalTagger .o. Spacing;
```

4

A Brief History of Computational Morphology

4.1 Introduction

Automatic morphological analysis dates back to the earliest work in computational linguistics on Machine Translation during the 1950s (Andron, 1962; Woyna, 1962; Bernard-Georges et al., 1962; Boussard and Berthaud, 1965; Vauquois, 1965; Schveiger and Mathe, 1965; Matthews, 1966; Brand et al., 1969; Hutchins, 2001). There have been many applications over the years including the Porter stemmer (Porter, 1980) heavily used in information retrieval applications (Dolby et al., 1965; Attar et al., 1978; Choueka, 1983; Büttel et al., 1986; Meya-Lloport, 1987; Choueka, 1990; Koskenniemi, 1984), spelling correction (McIlroy, 1982; Hankamer, 1986), text input systems (Becker, 1984; Abe et al., 1986), and morphological analysis for text-to-speech synthesis (Allen et al., 1987; Church, 1986; Coker et al., 1990). Many of these earlier applications used quite ad hoc approaches including hard-coding much of the linguistic information into the system. For example, in the system reported in Coker et al. (1990), a lot of the morphological analysis is mediated by tables coded as C header files and spelling-change rules written as C functions.

In many of the early applications, there was little interest in getting the morphological analysis correct as long as the resulting behavior served the purpose of the system. For example McIlroy's *spell* program (McIlroy, 1982) was concerned only with those derivations that would affect the performance of the program. Thus, as McIlroy notes (page 94):

> ...we are interested in soundness of results, not of method. Silly derivations like *forest* = *fore+est* are perfectly acceptable and even welcome. Anything goes, provided it makes the list [of elements needed in the dictionary] shorter without impairing its discriminatory power.

But the most interesting work in computational morphology both from a theoretical and a computational point of view, has been more principled than this and has depended heavily upon finite-state methods. Indeed the history of computational morphology over the last thirty years has been completely dominated by finite-state approaches. As we saw in Chapter 2, with the possible exception of reduplication, which (at least for reasons of elegance) may require further mechanisms, there are no morphological phenomena that fall outside the regular domain. Even non-local dependencies (of the German *ge-...-en* variety), can be handled by purely finite-state devices, at the cost of having duplication of structure. So it is natural that computational implementations should seek this lower bound.[1]

Dominating finite-state morphology since the early 1980s has been the approach based on finite-state transducers, originally investigated in the 1970s by Ron Kaplan and Martin Kay at Xerox PARC, whose first practical implementation was due to Koskenniemi (Koskenniemi, 1983). Because of its centrality in the history of computational morphology, we will focus in this chapter on a review of Koskenniemi's approach and attempt to elucidate some of its theoretical underpinnings.

Alternative approaches to computational morphology have largely been of two varieties. The first is explicitly finite-state approaches that are based on an explicitly finite-state model of morphotactics but with more or less ad-hoc computational devices (e.g., C functions) to implement spelling change or phonological rules. Such systems include the DECOMP module of the MITalk text-to-speech synthesis system (Allen et al., 1987) and Hankamer's *keçi* Turkish morphological analyzer (Hankamer, 1986).

More ad hoc approaches include the work of Byrd and colleagues (Byrd et al., 1986) and Church (Church, 1986). In these two pieces of work, for example, affixation is modeled as stripping rules that relate a derived word to another word. For example, a rule that strips off *-able* and replaces it with *-ate* would relate *evaluable* to *evaluate*. These kinds of approaches are reminiscent of the early work on "suffix stripping" found in McIlroy's SPELL program (McIlroy, 1982) and the Porter stemmer (Porter, 1980).

[1] Dale Gerdemann (p.c.) points us to an interesting example of local dependency in Kanuri (<http://www.cogsci.uni-osnabrueck.de/%7Ejt/kanuri/>), where person and number are marked with either prefixes or suffixes, but not both: as he points out, "one way or another, a finite-state network must record the fact that you get a prefix just in case you don't get a suffix, and this involves doubling the size of the network."

Then there are extensions to basic finite-state models. For example, both Bear (1986) and Ritchie et al. (1992) present systems that integrate finite-state morphology and phonology, with unification of morphosyntactic features. Rather than relying entirely upon finite-state morphotactics to restrict combinations of affixes, these systems control the combinatorics by insisting that the morphosyntactic feature matrices associated with the morphemes correctly unify. So, for example, one need not have separate continuation lexica (see below) expressing the different possible suffixes that might follow a noun, a verb, or an adjective stem. Rather one could have a single suffix sublexicon, with the stem-suffix combinations filtered by feature unification. And as we have already discussed, there has been a lot of work in the DATR framework that has allowed for sophisticated models of morphological inheritance that are nonetheless finite-state equivalent.

One thing to bear in mind in relation to the discussion in Chapter 3 is that virtually all practical approaches to computational morphology have assumed a lexical–incremental approach. The formal equivalence of this and inferential–realizational approaches, at least at the computational level, has already been discussed and will not be repeated here. But even those proponents of inferential–realizational approaches who were not convinced by the preceding arguments should still find the ensuing discussion to be of interest. Koskenniemi's Two-Level approach, in particular, has been used to develop wide-coverage morphological analyzers for a wide range of languages. If the ability to describe and implement a large number of linguistic phenomena is a valid metric of success, then one would have to agree that these approaches to morphology have been quite successful, even if one does not agree with their fundamental theoretical assumptions.

4.2 The KIMMO Two-Level Morphological Analyzer

The KIMMO two-level morphological analyzer has been described in a number of places including Antworth (1990); Sproat (1992); Karttunen and Beesley (2005). In the early descriptions, as in Koskenniemi's original presentation, the system is presented at the level of the machine rather than in terms of the algebraic operations that are involved in the system. But it is the algebraic operations that are really more important in understanding how the system works, and we will thus focus mostly on this aspect here.

Let us start with an illustration of the difference between a "machine-level" view and the "algebraic" view, a distinction that we have alluded to previously. Suppose we want to describe how we might check if a string is in a regular language, and therefore if it is accepted by the deterministic finite automaton (DFA) corresponding to the language. In that case we might describe how we would start the machine in its initial state and start reading from the beginning of the string. Each time we read a character from the string, we check the state we are in, see which arc is labeled with the character we are reading, and move into the state pointed to by that arc. We keep doing this until we get to the end of the string, or we fail because we get to a state where we are unable to read the current input character. (Since the machine is deterministic, we are guaranteed that in that case we cannot backtrack and try a different path.) If, once we have read all the characters in the string, we find ourselves in a final state of the automaton, then the string is in the regular language of the automaton; otherwise the string is not in the regular language of the automaton. This is a machine-level description since it describes the process in terms of the low-level operations of the machine.

An alternative description eschews the machine-level details and concentrates instead on what regular algebraic operations are involved in the process. In this case, the operation is simple. Represent the string as a trivial finite-state automaton with a single path. Then *intersect* the string machine with the language DFA. If the intersection is non-null (and in fact in this case it will be identical to the string-machine) then the string is in the language of the DFA, otherwise it is not. The machine-level operations described in the previous paragraph still go on here: they are part of the implementation of the intersection operation. But the algebraic view allows one to ignore the algorithmic details and focus instead on the algebraic operation that is being performed. With these issues in mind, let us turn to the core ideas in the KIMMO system.

4.2.1 KIMMO Basics

In Koskenniemi's (1983) presentation, there are three aspects of the system that are central. The first is the representation of dictionaries as *tries*, following Knuth (1973). The second is the representation of morphological concatenation via *continuation lexica*. If one is reading a word, and reaches a leaf node of a trie before one has completed

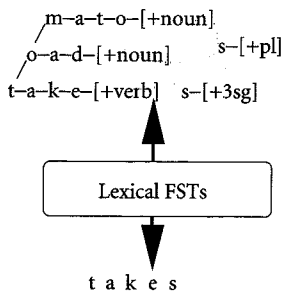

FIGURE 4.1 Schematic representation of Koskenniemi's two-level morphology system. The input "takes" is matched via a set of parallel FSTs to the lexicon, which is represented as a set of *tries*. A fragment of these *tries* is shown here, with the main lexicon containing the words "take", "toad", and "tomato", and *continuation lexica* containing the suffixes for noun plural and verbal third singular

reading the word, one can look at the leaf node of the trie and see if there are pointers to other tries where one can continue the search; these other tries are the continuation lexica. Note that this is completely equivalent to splicing the continuation lexicon tries onto the leaf nodes in the first tree. Finally, there are the finite-state transducers that implement the surface-lexical phonological correspondences, processes that in Koskenniemi's original Finnish version included vowel harmony and consonant lenition. Koskenniemi somewhat quaintly describes the rules as serving the function of "slightly distorting lenses" through which one views the lexicon. As a practical matter – and this is the distinctive property that gives Koskenniemi's system the name of "Two-Level Morphology" (though more properly it is "Two-Level Phonology") – each transducer reads the lexical and surface "tapes" simultaneously. This requires that the lexical and surface strings be passed by each of the transducers, or in other words that the strings must be members of the regular relations implemented by each of the individual machines. Or in other words again, the transducers are logically *intersected*. As Koskenniemi notes, it is possible to replace all the individual machines with one "big machine," which is simply the intersection of all the individual rule machines. A schematic figure of the Koskenniemi system is given in Figure 4.1.

Tries and continuation lexica can straightforwardly be replaced by finite-state automata, with which they are completely formally equivalent. One can think of a trie as simply a finite-state automaton where each path leads to a distinct final state.

Continuation lexica are easily modeled by grafting a continuation lexicon onto the terminal node(s) that allow the continuation in

question. One could even model this using concatenation, but this is not in general correct since concatenation would allow every "leaf" node of a stem transducer to proceed to the continuation lexicon. However, this problem can easily be solved by "tagging" both the end of the stem and the beginning of a matching continuation lexicon with a distinctive tag, different for each continuation lexicon. One could then concatenate the lexica and filter illicit combinations of tags using a regular filter. Let B' and A' be, respectively, the stem lexicon and the affix set augmented as just described. Let T be a set of tags and τ an individual tag. Then the appropriate construction is:

$$\forall \tau \in T, (B' \cdot A') \cap \neg(\Sigma^* \tau [T - \tau] \Sigma^*) \tag{4.1}$$

This just states that one can obtain the desired result by concatenating the continuation lexicon A with B and then filtering illegal combinations, namely, each τ followed by something that is in T but is not τ.

The situation with the transducers is more tricky, since Koskenniemi's system depends upon intersection, for which regular relations are not in general closed. However, all cases involving a non-regular result of intersection involve unbounded deletions or insertions, which Koskenniemi (obviously) never needs. It can be shown, in fact, that as long as one bounds the number of deletions or insertions by some finite k, then regular relations are closed under intersection. We turn to this point in the next subsection.

4.2.2 FST Intersection

Of course, regular relations are not in general closed under intersection. Kaplan and Kay (1994) demonstrate this using a simple counterexample. Consider two regular relations:

- $R_1 = \{< a^n, b^n c^* > | n \geq 0\}$, which can be expressed as the regular expression $(a : b)^*(\epsilon : c)^*$
- $R_2 = \{< a^n, b^* c^n > | n \geq 0\}$, which can be expressed as the regular expression $(\epsilon : b)^*(a : c)^*$

The intersection of these two relations is:

$$R_1 \cap R_2 = \{< a^n, b^n c^n > | n \geq 0\}$$

This is clearly not regular, since the right-hand language $b^n c^n$ is well known not to be regular.

The construction above depends upon there being an unbounded difference in length between the input and output strings. Indeed, one can show that as long as one allows only a finite bound k on the difference between any pair of input and output strings in a relation, then regular relations are closed under intersection. More formally we define *k-length-difference-bounded* regular relations as follows:

Definition 6 *A regular relation R is* k-length-difference-bounded *iff* $\exists k$, *such that* $\forall(x, y) \in R, -k \leq |x| - |y| \leq k$.

We then state the following theorem:

Theorem 4.1 k-length-difference-bounded *regular relations S are closed under intersection, for all k.*

The proof depends upon the following lemma:

Lemma 4.1 Same-length *regular relations are closed under intersection.*

Lemma 4.1 is discussed in Kaplan and Kay (1994). We prove Theorem 4.1 by reduction of *k-length-difference-bounded* relations to *same-length* relations.

Proof of Theorem 4.1. Consider first the case where:

$$R_1 =< X_1, Y_1 >$$
and
$$R_2 =< X_2, Y_2 >,$$

two regular relations, are exact *k-length-difference* relations such that for any string x_1 in the domain of R_1, and every corresponding $y_1 = R_1(x_1)$ (a string in the image of x_1 under R_1), $abs(|x_1| - |y_1|) = k$; and similarly for R_2. Without loss of generality, we will assume that the output of R_1 is the shorter so that $|x_1| - |y_1| = k$; and similarly for R_2.

Now, consider for a moment the case of two regular languages L_1 and L_2 and their intersection $L_3 = L_1 \cap L_2$. Let Σ be the alphabet of L_1 and L_2, let ξ be a symbol not in Σ, and ξ^k a k-length string of ξ. Then clearly the following equality holds:

$$L_3 \cdot \xi^k = L_1 \cdot \xi^k \cap L_2 \cdot \xi^k$$

That is, intersecting the result of concatenating L_1 with ξ^k and L_2 with ξ^k yields $L_3 = L_1 \cap L_2$ concatenated with ξ^k.

Returning to R_1 and R_2, observe that Y_1 and Y_2 – the right-hand sides of these two relations – are regular languages and that by assumption any string in Y_1 or Y_2 is exactly k symbols shorter than the corresponding string in X_1 or X_2. Consider then the relation Ξ_k, which simply adds a string ξ^k to the end of any input string. It is easy to show that Ξ_k is a regular relation: it can be implemented by an FST that has self loops to the start state over all elements in Σ and then terminates by a sequence of arcs and states k long, mapping from ϵ to ξ. Thus, $R_1' = R_1 \circ \Xi_k$ and $R_2' = R_2 \circ \Xi_k$ are regular relations since regular relations are closed under composition. Indeed, by construction R_1' and R_2' are *same-length* relations, and thus by Lemma 4.1 their intersection is also a regular relation. Furthermore, the right-hand side of $R_1' \cap R_2'$ is just

$$Y_1 \cdot \xi^k \cap Y_2 \cdot \xi^k = (Y_1 \cap Y_2) \cdot \xi^k,$$

so $R_1' \cap R_2' =< X_1 \cap X_2, (Y_1 \cap Y_2) \cdot \xi^k >$. Finally, note that Ξ^{-1}, which removes a final string of ξ^k, is a regular relation (since regular relations are closed under inversion) so we have:

$$< X_1 \cap X_2, (Y_1 \cap Y_2) \cdot \xi^k > \circ \Xi^{-1} = < X_1 \cap X_2, Y_1 \cap Y_2 > = R_1 \cap R_2$$

Thus $R_1 \cap R_2$ must be a regular relation, again because regular relations are closed under composition. Thus exact *k-length-difference* regular relations are closed under intersection.

The final step is to extend this result to *k-length-difference-bounded* regular relations. For any such relation R we can partition the relation into a union of relations $\bigcup_{i=0 \rightarrow k} R_i$, where each R_i is an exact *i-length-difference* relation (where R_0 is a same-length relation). To see this, observe that since R is *k-length-difference-bounded*, it must be implemented by a transducer T in which any cycles must be length preserving. Now consider any path p from the initial state q_0 to a $q_f \in F$ (F the set of final states), where no state on the path is visited more than once – the path does not pass through a cycle. By assumption this path must implement an *i-length-difference* relation for $0 \leq i \leq k$. Since all cycles must be length preserving, one can add to p any cycles that start and end at any $q_i \in p$, and still have an *i-length-difference*

relation; call this resulting sub-transducer $\hat{p}$. Clearly, for any i such that $0 \leq i \leq k$, one can find the (possibly empty) set P of all $\hat{p_i}$ such that $\hat{p_i}$ is an *i-length-difference* relation. P, a sub-transducer of T, is itself a finite-state transducer and thus implements a regular relation. Thus each R_i is a regular relation, and more particularly an *i-length-difference* regular relation.

Now consider two *k-length-difference-bounded* relations R_1 and R_2. We divide each into R_{1_i}'s and R_{2_i}'s, respectively, as above. We have already shown that the intersection of each pair R_{1_i} and R_{2_i} is also a regular relation. Furthermore $R_1 \cap R_2$ is just $\bigcup_{i=0 \rightarrow k}(R_{1_i} \cap R_{2_i})$: this is because for $R_{1_i} \cap R_{2_j}$ to be non-null, it must always be the case that $i = j$, since otherwise a string pair $< x_1, y_1 >$ in R_{1_i} could never match any $< x_2, y_2 >$ in R_{2_i}, since on at least one side of the pairs, the string lengths would not match. Since regular relations are closed under union, we have completed the proof.

Note that the above construction does not work for the non-*k-length-difference-bounded* relations $R_1 = \{< a^n, b^n c^* >\}$ and $R_2 = \{< a^n, b^* c^n >\}$ discussed in Kaplan and Kay (1994). Consider R_1. For us to be able to use the construction, one would have to know how many ξs to add to the a side in order to match the extra cs. This could be done by factoring R_1 into a series of *zero-length-difference-bounded* (same length), *1-length-difference-bounded*, *2-length-difference-bounded* relations and so forth as above. The problem is that the factorization is infinite and there is therefore no way to enumerate the cases. The same holds for R_2.

Koskenniemi did not explicitly describe his system in terms of intersection, much less in terms of *k-length-difference-bounded* regular relations. But his algorithms were in effect computing intersections of multiple transducers that allowed both insertions and deletions, and in fact he had to take special care to ensure that the transducers did not get arbitrarily out of alignment. Indeed, rewrite rules involving deletions or insertions are not in general *k-length-difference-bounded* relations. But if this is the case, how can Koskenniemi's system actually be modeled as the intersection of a set of rule transducers since this would appear to involve the intersection of a set of non-*k-length-difference-bounded* relations? One way around this is to observe that the intersection of a non-*k-length-difference-bounded* relation with a *k-length-difference-bounded* relation is a *k-length-difference-bounded* relation. Thus, so long as one can impose a hard upper bound on the length difference between

the input and output, there is no problem implementing the system in terms of relation intersection.

Note, finally, that Koskenniemi's system is not the only system that depends upon intersections of transducers. The decision-tree compilation algorithm reported in Sproat and Riley (1996) models decision trees (and more generally decision forests) as the intersection of a set of weighted rule transducers, each one representing a leaf node of the tree.

4.2.3 Koskenniemi's Rule Types

The other innovation of Koskenniemi's approach was his formalization of two-level rewrite rules; while he did not provide a compiler for these rules, the rules served to specify the semantics underlying the transducers that he built by hand. All rules in his system followed a template in that they were all of the following form:

CorrespondencePair **operator** LeftContext _ RightContext

That is, the rules specified conditions for the occurrence of a correspondence pair – a pairing of a lexical and a surface symbol (one of which might be empty, modeling deletion or insertion) – in a given left or right context. The contexts could be regular expressions, but the correspondence pair was a single pair of symbols, and thus was not as general as the $\phi \rightarrow \psi$ formulation from Kaplan and Kay (1994); See Section 4.2.4.

Koskenniemi's rules came in four flavors, determined by the particular **operator** used. These were:

Exclusion rule	a:b /⇐ LC _ RC
Context restriction rule	a:b ⇒ LC _ RC
Surface coercion rule	a:b ⇐ LC _ RC
Composite rule	a:b ⇔ LC _ RC

The interpretation of these was as follows:

- **Exclusion rule**: *a* cannot be realized as *b* in the stated context.
- **Context restriction rule**: *a* can only be realized as *b* in the stated context (i.e., nowhere else).
- **Surface coercion rule**: *a* must be realized as *b* in the stated context.
- **Composite rule**: *a* is realized as *b* obligatorily and only in the stated context.

Note that in many ways Koskenniemi's formalism for rules was better defined than the ones that had previously been used in generative

phonology. For one thing, each rule type specified a direct relation between the underlying and surface forms, something that was not possible within generative phonology due to the arbitrary number of ordered rewrite rules: in general, in generative phonology there was no way to know how a given lexical form would surface, short of applying all rules in the specified order and seeing what the outcome was. Koskenniemi's rules, in contrast, specified the relation directly.

Ignoring for the moment that traditional generative phonological rules were not two-level, one can ask which of Koskenniemi's rules correspond to the rule types (basically just obligatory or optional rewrite rules) of generative phonology. In fact only the *surface coercion rule* has a direct counterpart: it corresponds pretty directly to an obligatory rewrite rule. All the other two-level rule types depend upon global knowledge of the system. Thus the *context restriction rule* is equivalent to a situation in a traditional generative account where there is but one optional rule that changes *a* into *b*; but note that this is a property of the system, not of a specific rule. The *composite rule*, which is just a combination of *context restriction* and *surface coercion* is similar, but in this case the unique rule changing *a* into *b* is obligatory. Note that since one could write, say, a *context restriction* rule that relates *a* to *b* in one environment, and then also write another *context restriction* rule that allows *a* to become *b* in another environment, it is perfectly possible in Koskenniemi's system to write an inconsistent grammar. A lot of the work in designing later two-level systems involved writing de-buggers that would catch these kinds of conflicts. Finally, the *exclusion rule* is again global in nature: it is equivalent to the situation in a traditional generative grammar where there is no rule that relates *a* to *b* in the specified environment.

But really, Koskenniemi's rules can best be thought of as involving constraints on correspondence pairs. Constraints were virtually non-existent as a device in early generative phonology, but have since become quite popular in various theories of phonology including Declarative Phonology (Coleman, 1992), One-Level Phonology (Bird and Ellison, 1994) and Optimality Theory (Prince and Smolensky, 1993).

4.2.4 Koskenniemi's System as a Historical Accident

Koskenniemi's development of Two-Level Morphology can be thought of as a fortuitous accident of history. It had been known since

C. Douglas Johnson's PhD thesis (C. D. Johnson, 1972) that "context-sensitive" rewrite rules of the kind that had become familiar in generative phonology described regular relations and could thus be implemented using finite-state transducers (FSTs). By the late 1970s Ron Kaplan and Martin Kay at Xerox PARC were developing algorithms for the automatic compilation of FSTs from rewrite rules in a format that would be familiar to linguists, namely:

$$\phi \rightarrow \psi / \lambda \_ \rho \tag{4.2}$$

Here, ϕ, ψ, λ and ρ could be arbitrary regular expressions. Furthermore, since regular relations are closed under composition, this meant that one could write a series of ordered rules of the kind found in SPE (Chomsky and Halle, 1968), compile each of the rules into a transducer and then compose the entire series of rules together to form a single transducer representing the entire rule system. Kaplan and Kay finally published their algorithm many years later (Kaplan and Kay, 1994), and there has been subsequent work on a simpler and more efficient algorithm (Mohri and Sproat, 1996).

But in the late 1970s and early 1980s there was just one problem: computers were simply not fast enough, nor did they have enough memory to compile rule systems of any serious complexity. Indeed complex rule systems of several tens of rules over a reasonable-sized alphabet (say, 100 symbols) can easily produce FSTs with several hundred thousand states with a similar number of arcs, with a total memory footprint of several megabytes. While any PC today could easily handle this, this was simply not viable around 1980.[2] So Koskenniemi proposed a compromise: avoid the space complexities of rule compilation and composition by instead hand-coding transducers and intersecting them rather than composing them. By hand-coding the transducers, and using various space-saving techniques (e.g., the use of "wildcard" symbols to represent arcs that would be traversed if no other arc matched the current symbol), Koskenniemi was able to build a very compact system. By carefully specifying an algorithm for traversing the arcs in the multiple machines – thus implementing the intersection algorithm "on the fly" – he was able to avoid doing full transducer intersection.

Koskenniemi's two-level morphology was remarkable in another way: in the early 1980s most computational linguistic systems were toys. This included parsers, which were usually fairly restricted in the kinds

[2] Recall Bill Gates' 1981 statement that "640k ought to be enough for anybody."

of sentences they could handle; dialog systems, which only worked in very limited domains; and models of language acquisition, which were only designed to learn simple grammatical constraints. In contrast, Koskenniemi's implementation of Finnish morphology was quite real in that it handled a large portion of inflected words that one found in real Finnish text. To some extent this reflects the fact that it is easier to get a quite complete coverage of morphology in any language than it is to have a similar coverage of syntax, let alone dialog. But it also reflects Koskenniemi's own decision to develop a full-fledged system, rather than present a mere "proof of concept" of his ideas.

While two-level morphology was originally motivated by the difficulties, at the time, with Kaplan and Kay's approach to cascaded rewrite rules, the model quickly took on a life of its own. Koskenniemi took it to be a substantive theoretical claim that only two levels of analysis were necessary, a claim that was fairly radical in its day (at least in contrast to generative phonology), but which has since been superseded by claims that only one level is needed (Bird and Ellison, 1994) or even that synchronic accounts in phonology are not needed since phonological systems are merely a residue of the process of historical change (Blevins, 2003).

Nevertheless, practical considerations of developing morphological analyzers have led people to not rely wholly on the two-level assumption. Since transducers can be combined both by composition (under which they are always closed) and by intersection (under which they are closed under certain conditions), combinations of these two operations may be used in any given system; see, for example, Karttunen et al. (1992). Indeed, one of the beauties of finite-state techniques is that the calculus of the combination of regular languages and relations is expressive enough that one can develop modules of systems without regard to following any particular overall design: thus, for handling certain phenomena it may be more convenient to think in terms of a two-level system; for others, it may be easier to write cascaded rules. No matter: the two components can be combined as if one had built them both in one way or the other.

While Koskenniemi certainly did not invent finite-state approaches to morphology and phonology, he was the first to develop a system that worked fully using finite-state techniques, and he is thus to be given much credit for bringing the field of finite-state morphology to maturity and building the way for the renaissance of finite-state approaches to language and speech that has developed over the past two decades.

4.3 Summary

This chapter has given a very brief selective overview of the history of computational morphology. We have focused here on the dominant paradigm, namely finite-state approaches; a more complete and "eclectic" survey can be found in Sproat (1992). The dominant paradigm within the dominant paradigm has been two-level morphology, pioneered by Koskenniemi. While this approach was clearly an accident of history it has had an enormous influence on the field. While two-level rules are no longer strictly speaking necessary – compilation of large sets of cascaded rewrite rules are well within the capability of current computers – it is sometimes convenient to be able to describe phenomena in terms of reference to both surface and underlying levels. And in any case, since one ends up with a finite-state transducer that computes the desired relation, it is of little consequence how that transducer was constructed.

The latter point has been brought out especially nicely in Karttunen (1998). In that paper, Karttunen argues that Optimality Theory (Prince and Smolensky, 1993) can be implemented via a cascade of transducers that are combined using *lenient composition*. Lenient composition is defined in terms of priority union, which itself is defined as follows:

Definition 7 *The* ***priority union*** $\cup_P$ *of two regular relations* R_1 *and* R_2 *is defined as follows, where* π_1 *denotes* projection *onto the first dimension (domain) of the relation, and the overbar represents the complement of the language:*

$$R_1 \cup_P R_2 \equiv R_1 \cup [\overline{\pi_1(R_1)} \circ R_2]$$

In other words, the priority union of R_1 and R_2 maps any string in the domain of R_1 to its image under R_1 and any strings *not* in the domain of R_1 to their image under R_2.

Lenient composition is then defined as follows:

Definition 8 *The* ***lenient composition*** $\circ_L$ *of two regular relations* R_1 *and* R_2 *is defined as follows:*

$$R_1 \circ_L R_2 \equiv [R_1 \circ R_2] \cup_P R_1$$

or in other words

$$R_1 \circ_L R_2 \equiv [R_1 \circ R_2] \cup [\overline{\pi_1(R_1 \circ R_2)} \circ R_1]$$

Thus, under the lenient composition of R_1 and R_2, strings that are in the domain of $R_1 \circ R_2$ will undergo the composition of the two relations, but strings not in that domain will undergo only the relation R_1.

For Optimality Theory, lenient composition can represent the combination of the "generator function" GEN and some violable constraint C. Recall that the job of GEN is to generate a candidate list of analyses given some input string. For example, the input might be a segment sequence *ak*, and GEN will generate for that sequence a possibly infinite set of candidate syllabifications. The role of a constraint C is to rule out a particular configuration; for example, the sequence *ak* with a generated syllabification $[]_{ons}[a]_{nucl}[k]_{coda}$ might violate a constraint FILLONS which requires onsets to be non-empty.[3] Let us say that C is the constraint FILLONS, and for the sake of simplicity assume that this constraint is checked immediately on the output of GEN. In that case, and assuming that all the outputs of GEN for *ak* have an empty onset, then GEN $\circ$ FILLONS will produce no output at all for *ak*. In that case, lenient composition "rescues" *ak* by allowing all the outputs of GEN. In another case, such as *ka*, for which GEN might produce $[k]_{ons}[a]_{nucl}[]_{coda}$, the output of GEN would pass FILLONS and so would pass through the normal composition GEN $\circ$ FILLONS, filtering any of GEN's outputs that happen to violate FILLONS; but other constraints might be violated later on.

As Karttunen shows, a rank-ordered list of constraints can be modeled using lenient composition. In particular, if one leniently composes the constraints together in a cascade in the order in which the constraints are ranked – the constraint at the left-hand side of an OT tableau first, and so on – then it is guaranteed that the selected output will be the one that violates the lowest ranked constraint among the violated constraints (or no constraint at all). This is because lenient composition will rescue an input to GEN from a constraint C if and only if all analyses produced by GEN for that input, and available as the input to C, violate C. So, for a sequence of ranked constraints $C_1, C_2, \ldots C_n$, lenient composition will rescue analyses only for the first C_i for which no analyses whatever pass. All constraints prior to C_i must involve regular composition (the first half of the priority union), and therefore some analyses will potentially be filtered for any given input.[4] So the

[3] As Karttunen shows, constraints such as FILLONS can readily be implemented in terms of regular languages or relations.

[4] The model just sketched does not deal with cases where one must distinguish differing numbers of violations of the same constraint. As Karttunen shows, this can be accomplished within the framework by implementing multiple violations of a single constraint as a single violation of a version of the constraint that "counts" an exact number of violations; and then cascading the set of such count-specific constraint versions via lenient composition. Of course, this approach cannot handle more than a bounded number of violations of any given constraint, though as Karttunen notes, this is unlikely to be a practical problem. However, as Dale Gerdemann points out, even with reasonable bounds, Karttunen's automata can grow large. More recent work by Gerdemann and van Noord (2000) addresses this problem by replacing Karttunen's counting method with a method based on exact matching. Their method, like Karttunen's, uses lenient composition.

OT finger of salvation will point at the forms that are rescued from C_i and survive the lenient composition of all subsequent constraints.

But as Karttunen points out, this raises an interesting question: sets of traditional generative rewrite rules can be modeled by a single FST constructed by composition of the individual rules. Depending upon whether the rules involved are obligatory or optional, and depending upon how they are written, one could achieve either a unique output or a lattice of possible outputs. In a similar vein, an Optimality Theory system can be implemented by a single FST constructed by lenient composition of GEN and the individual constraints. Again the resulting FST may have a single optimal output or a set of optimal outputs. At a computational level, then, there is no difference between these two approaches despite the fact that in the linguistics literature, the two approaches are considered to be as different as night and day. This is not to say that it may not be easier or more natural to describe certain phenomena in Optimality Theoretic terms than in traditional terms. But this is not always the case. A good example of where it is not is San Duanmu's (1997) treatment of cyclic compounds in Shanghai. In that article Duanmu presents a traditional cyclic analysis of stress in Shanghai noun compounds, an analysis that takes up about a page of text. He then spends the remaining ten or so pages of the article laying out a fairly complex Optimality Theory analysis. While for some this may count as an advance in our understanding of the linguistic phenomenon in question, it is far from obvious to us that it does. Given the reductionist argument that Karttunen presents, it is not clear, given the ultimate computational equivalence of the two approaches, why one has to shoehorn data into one theory or the other.

In the previous chapter, we already applied the same reductionist reasoning to argue that views that are considered radically different in the literature on theoretical morphology are really notational variants of one another when looked at from a computational point of view.

5

Machine Learning of Morphology

5.1 Introduction

There is a disconnect between computational work on syntax and computational work on morphology. Traditionally, all work on computational syntax involved work on parsing based on hand-constructed rule sets. Then, in the early 1990s, the "paradigm" shifted to statistical parsing methods. While the rule formalisms – context-free rules, Tree-Adjoining grammars, unification-based formalisms, and dependency grammars – remained much the same, statistical information was added in the form of probabilities associated with rules or weights associated with features. In much of the work on statistical parsing, the rules and their probabilities were learned from treebanked corpora; more recently there has been work on inducing probabilistic grammars from unannotated text. We discuss this work elsewhere.

In contrast, equivalent statistical work on morphological analysis was, through much of this time period, almost entirely lacking (one exception being Heemskerk, 1993). That is, nobody started with a corpus of morphologically annotated words and attempted to induce a morphological analyzer of the complexity of a system such as Koskenniemi's (1983); indeed such corpora of fully morphologically decomposed words did not exist, at least not on the same scale as the Penn Treebank. Work on morphological induction that did exist was mostly limited to uncovering simple relations between words, such as the singular versus plural forms of nouns, or present and past tense forms of verbs. Part of the reason for this neglect is that hand-constructed morphological analyzers actually work fairly well. Unlike the domain of syntax, where broad coverage with a hand-built set of rules is very hard, it is possible to cover a significant portion of the morphology of even a morphologically complex language such

as Finnish, within the scope of a doctoral-dissertation-sized research project. Furthermore, syntax abounds in structural ambiguity, which can often only be resolved by appealing to probabilistic information – for example, the likelihood that a particular prepositional phrase is associated with a head verb versus the head of the nearest NP. There is ambiguity in morphology too, as we have noted elsewhere; for example, it is common for complex inflectional systems to display massive syncretism so that a given form can have many functions. But often this ambiguity is only resolvable by looking at the wider context in which the word form finds itself, and in such cases importing probabilities into the morphology to resolve the ambiguity would be pointless.

Recently, however, there has been an increased interest in statistical modeling both of morphology and of morphological induction and in particular the unsupervised or lightly supervised induction of morphology from raw text corpora. One recent piece of work on statistical modeling of morphology is Hakkani-Tür et al. (2002), which presents an n-gram statistical morphological disambiguator for Turkish. Hakkani-Tür and colleagues break up morphologically complex words and treat each component as a separate tagged item, on a par with a word in a language like English. The tag sequences are then modeled with standard statistical language-modeling techniques; see Section 6.1.

A related approach to tagging Korean morpheme sequences is presented in Lee et al. (2002). This paper presents a statistical language-modeling approach using syllable trigrams to calculate the probable tags for unknown morphemes within a Korean *eojeol*, a space-delimited orthographic word. For *eojeol*-internal tag sequences involving known morphemes, the model again uses a standard statistical language-modeling approach. With unknown morphemes, the system backs off to a syllable-based model, where the objective is to pick the tag that maximizes the tag-specific syllable n-gram model. This model presumes that syllable sequences are indicative of part-of-speech tags, which is statistically true in Korean: for example, the syllable conventionally transcribed as *park* is highly associated with personal names, since *Park* is one of the the most common Korean family names.

Agglutinative languages such as Korean and Turkish are natural candidates for the kind of approach just described. In these kinds of languages, words can consist of often quite long morpheme sequences, where the sequences obey "word-syntactic" constraints, and each morpheme corresponds fairly robustly to a particular morphosyntactic

feature bundle, or tag. Such approaches are harder to use in more "inflectional" languages where multiple features tend to be bundled into single morphs. As a result, statistical n-gram language-modeling approaches to morphology have been mostly restricted to agglutinative languages.

We turn now from supervised statistical language modeling to the problem of morphological induction, an issue that has received a lot of attention over the past few years. We should say at the outset that this work, while impressive, is nonetheless not at the stage where one can induce a morphological analyzer such as Koskenniemi's system for Finnish. For the most part, the approaches that have been taken address the issue of finding simple relations between morphologically related words, involving one or two affixes. This is not by any means to trivialize the contributions, merely to put them in perspective relative to what people have traditionally done with hand-constructed systems.

Automatic methods for the discovery of morphological alternations have received a great deal of attention over the last couple of decades, with particular attention being paid most recently to *unsupervised* methods.

It is best to start out with a definition of what we mean by morphological learning since there are a couple of senses in which one might understand that term. The first sense is the discovery, from a corpus of data, that the word *eat* has alternative forms *eats*, *ate*, *eaten* and *eating*. Thus the goal is to find a set of morphologically related forms as evidenced in a particular corpus. In the second sense, one wants to learn, say, that the past tense of regular verbs in English involves the suffixation of *-ed*, and from that infer that a new verb, such as *google*, would (with appropriate spelling changes) be *googled* in the past tense. In this second sense, then, the goal is to infer a set of rules from which one could derive new morphological forms for words for which we have not previously seen those forms.

Clearly the second sense is the stronger sense and more closely relates to what human language learners do. That is, while it is surely true that part of the process of learning the morphology of a language involves cataloging the different related forms of words, ultimately the learner has to discover how to generalize. For the present discussion we may remain agnostic as to whether the generalization involves learning rules or is done via some kind of analogical reasoning (cf. the classic debates between Rumelhart and McClelland (1986) and Pinker and Prince (1988)). Suffice it to say that such generalization must take place.

This stronger sense was the problem to which earlier supervised approaches to morphology addressed themselves. Thus the well-known system by Rumelhart and McClelland (1986) proposed a connectionist framework which, when presented with a set of paired present- and past-tense English verb forms, would generalize from those verb forms to verb forms that it had not seen before. Note that "generalize" here does not mean the same as "generalize correctly," and indeed there was much criticism of the Rumelhart and McClelland work, most notably by Pinker and Prince (1988) (and see also Sproat, 1992: Chapter 4). Other approaches to supervised learning of morphological generalizations include van den Bosch and Daelemans (1999) and Gaussier (1999).

Supervised approaches of course have the property that they assume that the learner is presented with a set of alternations that are known to be related to one another by some predefined set of morphological alternations. This begs the question of how the teacher comes by that set in the first place. It is to the latter question that the work on unsupervised learning of morphology addresses itself in that, as noted above, it aims to find the set of alternate forms as evidenced in a particular corpus. Indeed, while it is obviously an end goal of unsupervised approaches to not only learn the set of alternations that are found in a corpus, but also to generalize from such forms, the former is an interesting and hard problem in and of itself and so some of the work in this area has tended to focus on this first piece. Note, that it is reasonable to assume that once one has a list of alternation exemplars, one could apply a supervised technique to learn the appropriate generalizations; this is the approach taken, for example, in Yarowsky and Wicentowski (2001).

Since most of the work in the past ten years has been on unsupervised approaches, we will focus in this discussion on these, and in particular three recent influential examples, namely Goldsmith (2001), Yarowsky and Wicentowski (2001) and Schone and Jurafsky (2001). Other recent work in this area includes Sharma et al. (2002); Snover et al. (2002); Baroni et al. (2002); Creutz and Lagus (2002); Wicentowski (2002); H. Johnson and Martin (2003).

5.2 Goldsmith, 2001

We start with a discussion of Goldsmith's *minimum description length* (MDL) approach – called *Linguistica* – to the learning of affixation alternations. While this is not the first piece of work in unsupervised

morphological acquisition, it is certainly the most cited, and since Goldsmith has made his system available on the Internet, it has come to represent a kind of standard against which other systems are compared.

Goldsmith's system starts with an unannotated corpus of text of a language – the original paper demonstrated the application to English, French, Spanish, Italian, and Latin – and derives a set of *signatures* along with words that belong to those signatures. Signatures are simply sets of affixes that are used with a given set of stems. Thus, one signature in English is (using Goldsmith's notation) *NULL.er.ing.s*, which includes the stems *blow*, *bomb*, *broadcast*, *drink*, *dwell*, *farm*, *feel*, all of which take the suffixes Ø, *-er*, *-ing* and *-s* in the corpus that Goldsmith examines. One is tempted to think of signatures as being equivalent to paradigms, but this is not quite correct, for two reasons.

First of all, notice that *NULL.er.ing.s* contains not only the clearly inflectional affixes *-ing* and *-s*, but the (apparently) derivational affix *-er*. Whether or not one believes in a strict separation of derivational from inflectional morphology – cf. Beard (1995) – is beside the point here: most morphologists would consider endings such as *-s* and *-ing* as constituting part of the paradigm of regular (and most irregular) verbs in English, whereas *-er* would typically not be so considered.

Second, notice also that the set is not complete: missing is the past tense affix, which does show up in other signatures, such as *NULL.ed.er.ing.s* – including such verbs as *attack*, *back*, *demand*, and *flow*. An examination of the verbs belonging to the *NULL.er.ing.s* signature will reveal why: many of them – *blow*, *broadcast*, *drink*, *dwell*, *feel* – are irregular in their past tense form and thus cannot take *-ed*. However neither *bomb* nor *farm* are irregular, and the reason they show up in this signature class is presumably because they simply were never found in the *-ed* form in the corpus. A system would have to do more work to figure out that *bomb* and *farm* should perhaps be put in the *NULL.ed.er.ing.s* class, but that the other verbs in *NULL.er.ing.s* should be put in various different paradigms. Goldsmith briefly discusses the more general problem of going from signatures to paradigms, but note in any case that his system is not capable of handling some of the required alternations, such as *blow/blew*, since his methods only handle affixation – and are tuned in particular to suffixation.

The system for deriving the signatures from the unannotated corpus involves two steps. The first step derives candidate signatures and signature-class membership, and the second evaluates the candidates. We turn next to a description of each of these.

5.2.1 Candidate Generation

The generation of candidates requires first and foremost a reasonable method for splitting words into potential morphemes. Goldsmith presents a couple of approaches to this, one which he terms the *take-all-splits* heuristic and the second which is based on what he terms *weighted mutual information*. Since the second converges more rapidly on a reasonable set, we will limit our discussion to this case.

The method first starts by generating a list of potential affixes. Starting at the right edge of each word in the corpus, which has been padded with an end-of-word marker "#", collect the set of possible suffixes up to length six (the maximum length of any suffixes in the set of languages that Goldsmith was considering), and then for each of these suffixes, compute the following metric, where N_k here is the total number of k-grams:

$$\frac{freq(n_1, n_2 \ldots n_k)}{N_k} \log \frac{freq(n_1, n_2 \ldots n_k)}{\prod_1^k freq(n_i)} \tag{5.1}$$

The first 100 top ranking candidates are then chosen, and words in the corpus are segmented according to these candidates, where that is possible, choosing the best parse for each word according to the following metric, also used in the *take-all-splits* approach, which assigns a probability to the analysis of a word w into a stem $w_{1,i}$ and a suffix $w_{i+1,l}$, where l is the length of the word:

$$P(w = w_{1,i} + w_{i+1,l}) = \frac{1}{\sum_{j=1}^{l-1} H(w_{1,j}, w_{j+1,l})} e^{-H(w_{1,i}, w_{i+1,l})} \tag{5.2}$$

where

$$H(w_{1,i}, w_{i+1,l}) = -(i \log(freq(stem = w_{l,i})) + (l - i) \log(freq(suffix = w_{i+1,j}))) \tag{5.3}$$

Finally, suffixes that are not optimal for at least one word are discarded.

The result of this processing yields a set of stems and associated suffixes including the null suffix. The alphabetized list of suffixes associated with each stem constitutes the signature for that stem. Simple heuristic weeding is possible at this point: remove all signatures associated with but one stem and all signatures involving one suffix. Goldsmith terms the remaining signatures *regular signatures*, and these constitute the set of suffixes associated with at least two stems.

5.2.2 Candidate Evaluation

The set of signatures and associated stems constitutes a proposal for the morphology of the language in that it provides suggestions on how to decompose words into a stem plus suffix(es). But the proposal needs to be evaluated: in particular, not all the suggested morphological decompositions are useful, and a metric is needed to evaluate the utility of each proposed analysis. Goldsmith proposes an evaluation metric based on minimum description length. The best proposal will be the one that allows for the most compact description of the corpus (in terms of the morphological decomposition) and the morphology itself. This is, of course, a standard measure in text compression: a good compression algorithm is one that minimizes the size of the compressed text plus the size of the model that is used to encode and decode that text.

The compressed length of the model – the morphology – is given by:

$$\lambda\langle T\rangle + \lambda\langle F\rangle + \lambda\langle \Sigma\rangle \tag{5.4}$$

Here, $\lambda\langle T\rangle$ represents the length (in bits) of a list of pointers to $\langle T\rangle$ stems, where T is the set of stems, and the notation $\langle\rangle$ represents the cardinality of that set. $\lambda\langle F\rangle$ and $\lambda\langle\Sigma\rangle$ represent the equivalent pointer-list lengths for suffixes and signatures, respectively.

The first of these, $\lambda\langle T\rangle$, is given by:

$$\sum_{t\in T}(\log(26) * \mathit{length}(t) + \log\frac{[W]}{[t]}) \tag{5.5}$$

T is the set of stems, $[W]$ is the number of word tokens in the corpus and $[t]$ is the number of tokens of the particular stem t: here and elsewhere $[X]$ denotes the number of tokens of X. (The log(26) term assumes an alphabet of twenty-six letters.)

The term $\lambda\langle F\rangle$ (where F is Goldsmith's notation for *suffixes*) is:

$$\sum_{f\in \mathit{suffixes}}(\log(26) * \mathit{length}(f) + \log\frac{[W_A]}{[f]}) \tag{5.6}$$

Here, $[W_A]$ is the number of tokens of morphologically analyzed words, and $[f]$ is the number of tokens of the suffix f.

The signature component's contribution can be defined, for the whole signature component, as:

$$\sum_{\sigma\in\Sigma}\log\frac{[W]}{[\sigma]} \tag{5.7}$$

where Σ is the set of signatures.

TABLE 5.1 Top ten signatures for English from Goldsmith (2001), with a sample of relevant stems

1.	NULL.ed.ing.s	*accent, afford, attempt*
2.	's.NULL.s	*adolescent, amendment, association*
3.	NULL.ed.er.ing.s	*attack, charm, flow*
4.	NULL.s	*aberration, abstractionist, accommodation*
5.	e.ed.es.ing	*achiev, compris, describ*
6.	e.ed.er.es.ing	*advertis, enforc, pac*
7.	NULL.ed.ing	*applaud, bloom, cater*
8.	NULL.er.ing.s	*blow, drink, feel*
9.	NULL.d.s	*abbreviate, balance, costume*
10.	NULL.ed.s	*acclaim, bogey, burden*

Finally the compressed length of the corpus in terms of the morphological model is given by:

$$\sum_{w\in W}[w]\left[\log\frac{[W]}{[\sigma(w)]} + \log\frac{[\sigma(w)]}{[\mathit{stem}(w)]} + \log\frac{[\sigma(w)]}{[\mathit{suffix}(w)\in\sigma(w)]}\right] \tag{5.8}$$

or in other words, assuming the maximum likelihood estimate for probabilities:

$$\sum_{w\in W}[w][-\log(P(\sigma(w))) - \log(P(\mathit{stem}(w)|\sigma(w))) - \log(P(\mathit{suffix}(w)|\sigma(w)))] \tag{5.9}$$

As noted above, Goldsmith tested his method on corpora from English, French, Italian, Spanish, and Latin. For each of these languages, he lists the top ten signatures. For English these are reproduced here in Table 5.1, along with associated stems. Goldsmith also evaluated the results for English and French. Having no gold standard against which to compare, he evaluated the results subjectively, classifying the analyses into the categories *good*, *wrong analysis*, *failed to analyze*, *spurious analysis*. Results for English, for 1,000 words, were 82.9% in the *good* category, with 5.2% *wrong*, 3.6% *failure*, and 8.3% *spurious*. Results for French were roughly comparable.

As we noted in the introduction to this section, Goldsmith's work stands out in that it has become the de facto gold standard for subsequent work on unsupervised acquisition of morphology, partly because he has made his system freely available, but partly also because he has provided a simple yet powerful metric for comparing morphological analyses. Goldsmith's method, of course, makes no explicit use

of semantic or syntactic information. An obvious objection to his approach as a model of what human learners do is that children clearly have access to other information besides the set of words in a corpus of (spoken) language: they know something about the syntactic environment in which these words occur and they know something about the intended semantics of the words. But Goldsmith argues that this is a red herring. He states (page 190):

> Knowledge of semantics and even grammar is unlikely to make the problem of morphology discovery significantly easier. In surveying the various approaches to the problem that I have explored (only the best of which have been described here), I do not know of any problem (of those which the present algorithm deals with successfully) that would have been solved by having direct access to either syntax or semantics.

But while no one can yet claim to have a model that well models the kind of syntactic or semantic information that a child is likely to have access to, it has been demonstrated in subsequent work that having a system that models syntactic and semantic information does indeed help with the acquisition of morphology. We turn now to a discussion of some of this subsequent work.

5.3 Schone and Jurafsky, 2001

Schone and Jurafsky's approach, a development of their earlier work reported in Schone and Jurafsky (2000), uses semantic, orthographic, and syntactic information derived from unannotated corpora to arrive at an analysis of inflectional morphology. The system is evaluated for English, Dutch, and German using the CELEX corpus (Baayen et al., 1996).

Schone and Jurafsky note problems with approaches such as Goldsmith's that rely solely on orthographic (or phonological) features. For example, without semantic information, it would be hard to tell that *ally* should not be analyzed as *all+y*, and since Goldsmith's approach does not attempt to induce spelling changes, it would be hard to tell that *hated* is not *hat+ed*. On the other hand, semantics by itself is not enough. Morphological derivatives may be semantically distant from their bases – consider *reusability* versus *use* – so that it can be hard to use contextual information as evidence for a morphological relationship. Furthermore, contextual information that would allow one to derive semantics can be weak for common function words, so there is effectively no information that would lead one to prevent *as* being derived from *a+s*.

Previous approaches also tend to limit the kinds of morphological alternations they handle. As we have seen Goldsmith's method was developed with suffixational morphology in mind (though it could certainly be extended to cover other things). Schone and Jurafsky's system at least extends coverage to prefixes and circumfixes. Their system, then, according to their own summary (page 2):

- considers circumfixes
- automatically identifies capitalizations by treating them similarly to prefixes
- incorporates frequency information
- uses distributional information to help identify syntactic properties, and
- uses transitive closure to help find variants that may not have been found to be semantically related but which are related to mutual variants.

Schone and Jurafsky use the term *circumfix* somewhat loosely to denote apparently true circumfixes such as the German past participle circumfix *ge—t*, as well as combinations of prefixes and suffixes more generally. Their method for finding prefixes, suffixes and circumfixes is as follows:

1. Strip off prefixes that are more common than some predetermined threshold.[1]
2. Take the original lexicon, plus the potential stems generated in the previous step, and build a trie out of them.
3. As in Schone and Jurafsky (2000), posit potential suffixes wherever there is a *branch* in the trie, where a branch is a subtrie of a node where splitting occurs. See Figure 5.1.
4. Armed with a set of potential suffixes, one can obtain potential prefixes by starting with the original lexicon, stripping the potential suffixes, reversing the words, building a trie out of the reversed words, and finding potential suffixes of these reversed strings, which will be a set of potential prefixes in reverse.
5. Identify candidate circumfixes, defined as prefix-suffix combinations that are attached to some minimum number of stems that are also shared by other potential circumfixes. The stems here are actually called *pseudostems* since, of course, they may not actually correspond to morphological stems. Note that since NULL will

[1] As Schone and Jurafsky note, if you do not do this initial step some potential circumfixes may be missed.

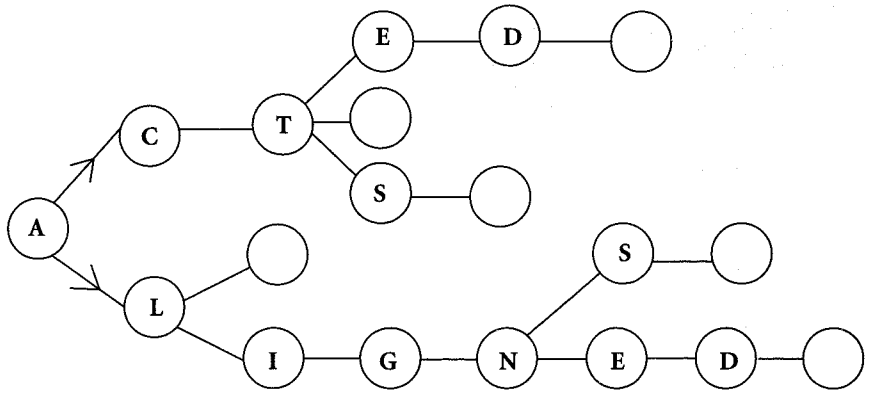

FIGURE 5.1 A sample trie showing *branches* for potential suffixes NULL (empty circle), *-s* and *-ed*: from Schone and Jurafsky (2001, Figure 2). Used with permission of the authors and the Association for Computational Linguistics

in general be among the potential prefixes and suffixes found in previous stages, pure suffixes or prefixes will simply be circumfixes where one of the components is NULL. Schone and Jurafsky also define a *rule* to be a pair of candidate circumfixes sharing some minimum number of pseudostems.

The output of these steps for English, German, and Dutch, with particular settings for the minima mentioned above, produces a large set of rules (about 30,000 for English) of which some are reasonable (e.g. *-s*⇒ ∅, *-ed*⇒*-ing*) but many of which are not (e.g. *s-*⇒ ∅, as induced from such seeming alternations as *stick/tick* or *spark/park*.)

To compute semantic information, Schone and Jurafsky use a version of Latent Semantic Analysis introduced by Schütze (1998), that uses an $N \times 2N$ term–term matrix. N represents the $N-1$ most frequent words, plus a "glob" in the Nth position corresponding to all other words. For each row, the first N columns represent words that occur in a window to the left of the row's word, and the last N columns words that occur within a window to its right. Singular value decomposition is then performed on a normalized version of the $N \times 2N$ matrix M, to produce the product of two orthogonal matrices U and V^T and the diagonal matrix of squared eigenvalues (singular values) D. The diagonal squared eigenvalue entries in D are ordered so that the first k of them account for the first k most significant dimensions of the $|M|$-dimensional space.

$$M = UDV^T \tag{5.10}$$

Each word w is assigned the semantic vector $\Omega_w = U_w D_k$, where U_w is the row of U corresponding to w and D_k are the first k singular values from Equation 5.10.

It is then necessary to compute a similarity between pairs of words, and this is accomplished using *normalized cosine scores* (NCS). For each word w_k, $k \in (1, 2)$, vectors for 200 other words are randomly selected, and the means (μ_k) and variances (σ_k) of the cosine values between w_k and each of the 200 other words are computed. The cosine of w_1 and w_2 are then normalized, and the NCS is computed as follows:

$$NCS(w_1, w_2) = min_{k\in(1,2)} \frac{cos(\Omega_{w1}, \Omega_{w2}) - \mu_k}{\sigma_k} \tag{5.11}$$

Assuming random NCS are normally distributed as $N(0, 1)$ and similarly the distribution of means and variances of true correlations as $N(\mu_T, \sigma_T^2)$, we can define a function that gives the area under the curve of the distribution from NCS to infinity:

$$\Phi_{NCS} = \int_{NCS}^{\infty} e^{-(\frac{-(x-\mu)}{\sigma})^2} dx \tag{5.12}$$

The probability that an NCS is non-random is then given by:

$$P(NCS) = \frac{n_T \Phi_{NCS}(\mu_T, \sigma_T)}{(n_R - n_T)\Phi_{NCS}(0, 1) + n_T \Phi_{NCS}(\mu_T, \sigma_T)} \tag{5.13}$$

where n_T is the number of terms in the distribution of true correlations. Finally the probability that w_1 and w_2 are related is given by:

$$P_{sem}(w_1 \Rightarrow w_2) = P(NCS(w_1, w_2)) \tag{5.14}$$

Schone and Jurafsky assume a threshold for P_{sem} (0.85 in their implementation) above which a potential relationship is considered valid.

The semantic probability is supplemented with an "orthographic" probability defined as follows:

$$P_{orth} = \frac{2\alpha f(C_1 \Rightarrow C_2)}{max_{\forall Z} f(C_1 \Rightarrow Z) + max_{\forall W} f(W \Rightarrow C_2)} \tag{5.15}$$

Here $f(A \Rightarrow B)$ is the frequency of the alternation involving circumfix A and circumfix B, and α is a weight between 0 and 1. Thus the probability of an alternation is the weighted ratio of the count of the alternation over the sum of the count of the alternation between the left circumfix and all circumfixes, and all circumfixes and the right circumfix. The orthographic probability is combined with the semantic probability as follows (P_{s-o} denotes the combined semantic-orthographic

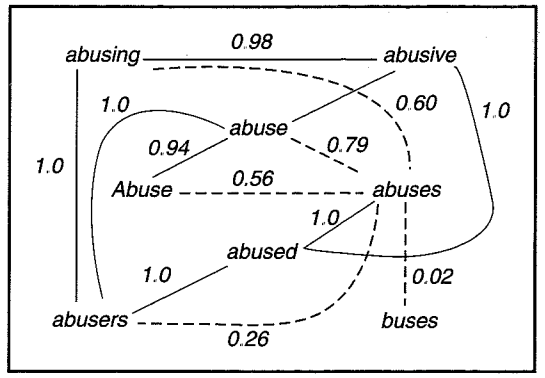

FIGURE 5.2 Semantic transitive closure of PPMVs, from Schone and Jurafsky (2001, Figure 4). Used with permission of the authors and the Association for Computational Linguistics

probability):

$$P_{s-o}(valid) = P_{sem} + P_{orth} - (P_{sem}P_{orth}) \tag{5.16}$$

The similarities of local syntactic context between pairs of an alternation are also computed. For words that are members of each side of an alternation – the *pair of potential morphological variants* or *PPMV* – one first computes contextual words that are either far more frequent or far less frequent than expected by chance; these sets are the *signatures* for the words. Signatures are also collected for randomly chosen words for the corpus, and for unvalidated PPMVs. Lastly, one computes the NCS and the probabilities between the signatures of the ruleset and the unvalidated PPMVs. (The probability of the syntactic correspondence is computed as in Equation 5.13.) The probability of a correspondence being valid is given as a combination of the orthographic/semantic probability P_{s-o} and the syntactic environment-based probability P_{syntax}:

$$P(valid) = P_{s-o} + P_{syntax} - (P_{s-o}P_{syntax}) \tag{5.17}$$

The above method, while it captures a lot of valid PPMVs, still fails to capture some that are valid because, for example, they are not sufficiently represented in the corpus. However many of these can be reconstructed by following the transitive closure of PPMVs. This is exemplified in Figure 5.2. The probability of an alternation given a single path is modeled as the product of the probabilities of the individual links, along with a decay factor. For multiple paths between two nodes the probabilities of the paths are summed.

TABLE 5.2 Comparison of the F-scores for suffixing for full Schone and Jurafsky method with Goldsmith's algorithm for English, Dutch, and German

	English	German	Dutch
Linguistica	81.8	84.0	75.8
Schone & Jurafsky	88.1	92.3	85.8

The output of Schone and Jurafsky's method is a set of what they term *conflation sets* – a directed graph linking words to their relatives. The conflation sets can then be compared to the evaluation corpus – CELEX – by computing the *correct*, *deleted*, and *inserted* members of the found set against the true set. These scores can then be converted to F-scores. F-scores for Goldsmith's method, in comparison with Schone and Jurafsky's full method, are shown in Table 5.2, evaluating for suffixes only since Goldsmith's system only handles suffixes. (Schone and Jurafsky also evaluate their own system for circumfixes.)

5.4 Yarowsky and Wicentowski, 2001

Yarowsky and Wicentowski (2001) present a lightly supervised method for inducing analyzers of inflectional morphology. The method consists of two basic steps. First a table of alignments between root and inflected forms is estimated from data. For example, the table might contain the pair *take/took*, indicating that *took* is a particular inflected form of *take*. Second, a supervised morphological analysis learner is trained on a weighted subset of the table. In considering possible alignment pairs, Yarowsky and Wicentowski concentrate on morphology that is expressed by suffixes or by changes in the root (as in *take/took*).

The method depends upon the following resources:

- A table of inflectional categories for the language; such a table for English would indicate that verbs have past tense forms. Along with these inflectional categories, a list of canonical suffixal exponents of these categories is needed. For past tense verbs in English, for example, canonical suffixes would be *-ed*, *-t* and *-Ø*, the latter being found in cases like *took*.
- A large corpus of text.
- A list of candidate noun, verb, and adjective roots, which can be obtained from a dictionary, plus a rough method for guessing parts of speech of words in the corpus.

- A list of consonants and vowels for the language.
- A list of common function words (optional).

The suffixes are useful for providing plausible alignments for many pairs of root and inflected form. For example, the existence of *announce* and *announced* in a corpus, along with the knowledge that *-ed* is a possible past-tense suffix, is sufficient to propose that the two words in question should be paired with *announce* being the root and *announced* the inflected past form.

But in many cases the suffixes are not sufficient. For example, we would like to pair the root *sing* with *sang*, not with *singed*. A key to deciding such cases is the observation that there is a substantial difference in frequency between, on the one hand, *sang/sing* (ratio: 1.19/1 in Yarowsky and Wicentowski's corpus), and on the other *singed/sing* (0.007/1). However, this is not quite enough, since a priori we have no way of deciding which of the two ratios is a more reasonable ratio for a past tense/present tense pair: some inflectional categories, for example, may be systematically far less frequent than their roots. Yarowsky and Wicentowski address this issue by computing the distribution of ratios over the corpus for word pairs from a given inflected/root class. Figure 5.3 shows the distribution of $\log \frac{VBD}{VB}$, where *VBD* and *VB* are the Penn Treebank tags for, respectively, past tense verbs and verb roots. One problem is that the computation of the $\log \frac{VBD}{VB}$ distribution depends upon knowing the correct inflected/non-inflected pairs, which by assumption we do not know. Yarowsky and Wicentowski get around this problem by observing that the distribution is similar for regular and irregular verbs in English. They then estimate the distribution from regular verb pairs, which can be detected to a first approximation by simply looking for pairs that involve stripping the regular affix (in this case *-ed*). Initially the set of such pairs will be noisy; it will contain bogus pairings like *singed/sing* above. However, the estimate can be iteratively improved as improved alignments between inflected forms and their roots are computed. One is also not limited to considering the distribution of just one type of pair, such as *VBD/VB*. So, for example, *VBG/VBD* (where *VBG* is the Penn treebank tag for the gerund/participle ending in *-ing*) is also useful evidence for a putative *VBD* form.

A second form of evidence for the relatedness of forms is contextual evidence of the kind also used by Schone and Jurafsky (2001). Cosine similarity measures between weighted contextual vectors give a good

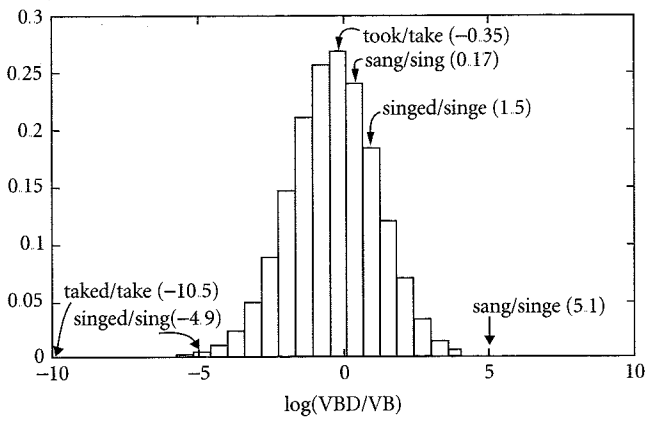

FIGURE 5.3 The $\log \frac{VBD}{VB}$ estimator from Yarowsky and Wicentowski (2001, Figure 1), smoothed and normalized to yield an approximation to the probability density function for the *VBD/VB* ratio. Correct pairings *took/take*, *sang/sing* and *singed/singe* are close to the mean of the distribution. Incorrect pairs *singed/singe*, *sang/singe* and *taked/take* are far out in the tails of the distribution; note that *taked* occurred once in the corpus, presumably as a typo. Used with permission of the authors and the Association for Computational Linguistics

indication that two forms are related, since even though semantically related words such as *sip* and *drink* will also show similar contexts, such words are still more dissimilar than are morphologically related forms.

Also as with Schone and Jurafsky (2001), Yarowsky and Wicentowski use orthographic similarity between the putative root and inflected form, using a weighted Levenshtein distance. Costs for character or character-sequence substitutions are initialized to a predefined set of values. Thus, for example consonant–consonant substitutions are assigned an initial cost of 1.0, whereas single vowel–vowel substitutions are assigned a cost of 0.5. However, these costs are re-estimated as the system converges to a more accurate set of root-inflected form correspondences.

Yarowsky and Wicentowski are not only interested in producing a table of accurate correspondences for particular roots and their inflected forms but are also interested in producing a morphological analyzer that can extend to new cases. The task here is to find the probability of a particular stem change, given a root, a suffix, and a part-of-speech tag:

$$P(stemchange|root, suffix, POS) \tag{5.18}$$

This is estimated by an interpolated backoff model (see Section 6.1) that is the weighted sum of the probability of the stem change given the last three characters of the root, the suffix, and the POS; the last two characters of the root, the suffix, and the POS; and so on down to just the suffix, and the POS; and finally the context-independent stem change $\alpha \rightarrow \beta$:

$$\begin{aligned} P(change|root, suf, POS) &= P(\alpha \rightarrow \beta|root, suf, POS) \\ &\approx \lambda_1 P(\alpha \rightarrow \beta|last_3(root), suf, POS) + \\ &(1-\lambda_1)(\lambda_2 P(\alpha \rightarrow \beta|last_2(root), suf, POS) + \\ &(1-\lambda_2)(\lambda_3 P(\alpha \rightarrow \beta|last_1(root), suf, POS) + \\ &(1-\lambda_3)(\lambda_4 P(\alpha \rightarrow \beta|suf, POS) + \\ &(1-\lambda_4) P(\alpha \rightarrow \beta)))) \end{aligned} \tag{5.19}$$

The λs are re-estimated on each iteration of the algorithm.

In addition to the statistical methods outlined above, Yarowsky and Wicentowski impose a constraint, which they call the *pigeonhole principle*, that requires that there be only one form for a given inflection for a particular word. Note that this is what would more normally be called *morphological blocking* (Aronoff, 1976). Exceptions to this principle such as *dreamed/dreamt* are considered to be sufficiently rare that their existence does not detract significantly from the performance of the model. The pigeonhole principle is used to greedily select the first candidate inflected form for a given position in the paradigm from a rank-ordered list of putative forms.

Table 5.3 shows the performance of the algorithm on four classes of verbs for the first iteration of each of the measures: frequency similarity (FS); Levenshtein (LS); context similarity (CS); various combinations of these metrics; and the fully converged system. As Yarowsky and Wicentowski observe, none of the metrics perform well by themselves, but by combining the metrics one can achieve very high performance.

5.5 Discussion

In this section we have reviewed three of the more prominent pieces of work on morphological induction that have appeared in the past few years.

Other recent work in this area includes:

TABLE 5.3 The performance of the Yarowsky–Wicentowski algorithm on four classes of English verbs

Combination of Similarity Models	# of Iterations	All Words (3888)	Highly Irregular (128)	Simple Concat. (1877)	Non-Concat. (1883)
FS (Frequency Sim)	(Iter 1)	9.8	18.6	8.8	10.1
LS (Levenshtein Sim)	(Iter 1)	31.3	19.6	20.0	34.4
CS (Context Sim)	(Iter 1)	28.0	32.8	30.0	25.8
CS + FS	(Iter 1)	32.5	64.8	32.0	30.7
CS+FS+LS	(Iter 1)	71.6	76.5	71.1	71.9
CS+FS+LS+MS	(Iter 1)	96.5	74.0	97.3	97.4
CS+FS+LS+MS	(Convg)	99.2	80.4	99.9	99.7

From Yarowsky and Wicentowski (2001, Table 9)

- H. Johnson and Martin (2003) propose a method for detecting morpheme boundaries based on what they term *hubs*. Hubs are nodes in a minimized automaton representing the words of the language that have in-degree and out-degree greater than one. As Johnson and Martin note, their idea is an extension on previous work such as Z. Harris (1951) and Schone and Jurafsky (2000).
- Snover et al. (2002) propose a method that estimates probabilities for: the numbers of stems and suffixes; the lengths of the stems and suffixes; the joint probability of hypothesized stem and suffix sets; the number of paradigms; the number of suffixes in each paradigm, and which suffixes are associated with each paradigm; and the paradigm affiliation of stems. These estimates are then used to assign a probability to a particular analysis of the data into a set of hypothesized stems, suffixes and paradigms. A novel search algorithm is proposed for searching the set of possible analyses of the data. The method is shown to outperform Linguistica in terms of F-score on corpora of English and Polish.
- Baroni et al. (2002) propose another method similar to Schone and Jurafsky (2000) that combines semantic and orthographic similarity for the discovery of morphologically-related words. They use an edit-distance measure for orthographic similarity and mutual information for semantic similarity. The reported performance of this method is hard to compare to the results of other work but seems to be degraded from what one would expect from other methods.

TABLE 5.4 Comparison of methods of morphological induction

Method	Correct	Incomplete	Incorrect
Recursive MDL	49.6%	29.7%	20.6%
Sequential ML	47.3%	15.3%	37.4%
Linguistica	43.1%	24.1%	32.8%

From Creutz and Lagus (2002, Table 3).

- Creutz and Lagus (2002) propose a recursive MDL-based approach. Words are segmented all possible ways into two parts, including no split, and the segmentation that results in the minimum total cost is chosen. If a split is chosen, then the two chosen components are recursively submitted to the same procedure. As Creutz and Lagus note, this approach has the danger of falling into local optima. To avoid this, words that have already been segmented are resegmented when they occur again (since this may lead to a reanalysis of the word).
  A second method involved a maximum likelihood estimate of the $P(data|model)$, estimated using Expectation Maximization; in this case a linear rather than recursive procedure is used, starting with a random segmentation of the corpus.
  A comparison of the two methods proposed by Creutz and Lagus, along with the performance of Linguistica on morpheme boundary detection in a sample of 2,500 Finnish words is shown in Table 5.4 (Creutz and Lagus' Table 3). Note that *correct* means that all critical morpheme boundaries were detected, *incomplete* means that only some of the boundaries were detected, and *incorrect* means that at least one boundary was incorrect.

One clear principle that can be gleaned from the more successful work in morphological induction is that multiple sources of evidence are crucial. Thus morphological induction is similar to other problems in natural language processing, such as Named Entity Recognition – e.g. Collins and Singer (1999) – in that one often does better by combining evidence from multiple sources than from relying on just one kind of evidence.

Relevant evidence for morphological induction includes:

- **Orthographic or phonological similarity between morphologically related forms.** While it is not always the case that members of a paradigm share phonological properties (*go/went* is an obvious counterexample), it is overwhelmingly the most common situation. Note that the notion of sharing phonological properties can be quite varied and needs to be parameterized to language-particular cases. Methods such as Yarowsky and Wicentowski (2001) and Schone and Jurafsky (2001) go far towards dealing with a variety of cases involving circumfixation or stem changes. But no published work to date handles infixation, the kind of interdigitation found in Semitic morphology, or productive reduplication.
  The key here is clearly to allow for a variety of possible ways in which two forms may be phonologically related, without opening the flood gates and allowing all kinds of silly correspondences. The current approaches, in addition to being too limiting in the correspondences they allow, are also too limiting in another sense: they target the kinds of alternations that are known to be active in the languages they are being applied to. Obviously this is not realistic as a model for morphological induction in the general case.
  A traditional linguistic approach to this issue would be to constrain the search for possible correspondences to those kinds of alternations that are known to occur across languages. So one might start with a taxonomy of morphological alternations (one such taxonomy is given in Sproat, 1992) and allow correspondences only within those classes of alternations. This would be a reasonable approach, but it is also worth considering whether one could induce the range of observed alternation types completely automatically.
- **Syntactic context.** Various methods for morphological induction depend upon related forms occurring in similar syntactic contexts. This is reasonable for some instances of inflectional morphology. For example in English, plural nouns have roughly the same syntactic distribution as singular nouns, modulo the fact that as subjects they co-occur with different verb forms. Similarly, there is no syntactic difference between present and past tense verb forms.
  But such similarity in syntactic context is not a feature of inflectional variants in general. For example, in many languages nouns are case marked precisely to mark differing syntactic contexts for the noun forms in question. So one would not in general expect, for example, a nominative and ablative form of a noun to occur in similar syntactic environments.

Even tense marking on verbs can result in different syntactic contexts in some languages. For example in Georgian (A. Harris, 1981; Aronson, 1990), different verb tenses imply different case markings on subjects and direct objects. Aorist verbs, for example, take ergative–absolutive marking, whereas present verbs take nominative–accusative marking. Assuming one tags different nominal case forms with different syntactic tags, these tense differences would therefore correspond to different syntactic environments.

And once one moves away from inflectional morphology into more derivational kinds of alternations, especially those that result in a change of grammatical category, one expects quite different syntactic environments.

- **Semantic context.** Words that are morphologically related tend to occur in semantically similar environments. Thus *eat* will tend to occur in environments that are similar no matter what the tense is. Similarly, one would expect a nominative singular form such as *equus* "horse" in Latin to occur in similar semantic environments to other inflected forms such as *equī* (nominative plural) or *equōrum* (genitive plural).

  Semantic context is expected to be more stable than syntactic context across derivational relatives. Thus both *donate* and *donation* might be expected to co-occur with such context words as *money*, *fund*, *dollars*, and *charity*. Semantic drift will, of course, contribute to a weakening of this expectation: *transmit* and *transmission* (in the automotive sense) will clearly tend to occur in different semantic contexts. It is a fair question, however, whether native speakers even recognize such forms as related.

Thus despite the substantial progress in recent years on automatic induction of morphology, there are also still substantial limitations on what current systems are able to handle. Nonetheless, the problems described in the preceding bullet items are just that: problems, and problems which we have every reason to believe will be seriously addressed in subsequent work.

Part II

Computational Approaches to Syntax

approaches, and the use of re-ranking as a post-process for context-free (or finite-state) approaches. Going beyond context-free models comes at an additional efficiency cost, but many rich linguistic dependencies are difficult to model with context-free grammars. Some of the most interesting approaches use models of varying complexity at various stages, to try to get rich and accurate annotations at bargain basement efficiencies. It is clear that more research must be done to understand the most effective trade-offs between these levels of the Chomsky hierarchy.

While computational approaches to syntax have turned increasingly to statistical methods based on large annotated corpora, this does not mean that there is less need for the kind of linguistic knowledge that has driven grammar engineering over the years. More than ever, the formal mechanisms exist to apply rich constraints for syntactic processing and disambiguation; the search for such constraints (or features) will continue to be largely driven by linguistic knowledge. There is much potential in approaches that combine the flexibility of some of the stochastic approaches we have discussed with linguistically informed feature exploration.

References

Abe, M., Ooshima, Y., Yuura, K., and Takeichi, N., 1986. "A Kana-Kanji translation system for non-segmented input sentences based on syntactic and semantic analysis." In: *Proceedings of the 11th International Conference on Computational Linguistics (COLING)*, pp. 280–285.

Abney, S., 1996. "Partial parsing via finite-state cascades." *Natural Language Engineering* 2 (4), 337–344.

—— 1997. "Stochastic attribute-value grammars." *Computational Linguistics* 23 (4), 597–617.

Aho, A. V., Sethi, R., and Ullman, J. D., 1986. *Compilers, principles, techniques, and tools*. Addison-Wesley, Reading, MA.

Ajdukiewicz, K., 1935. "Die syntaktische konnexität." In: McCall, S. (Ed.), *Polish Logic 1920–1939*. Oxford University Press, Oxford, pp. 207–231.

Allauzen, C., Mohri, M., and Roark, B., 2003. "Generalized algorithms for constructing language models." In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 40–47.

Allen, J., Hunnicutt, M. S., and Klatt, D., 1987. *From Text to Speech: the MITalk System*. Cambridge University Press, Cambridge, UK.

Alshawi, H., Srinivas, B., and Douglas, S., 2000. "Learning dependency translation models as collections of finite-state head transducers." *Computational Linguistics* 26 (1), 45–60.

Anderson, S., 1992. *A-Morphous Morphology*. Cambridge University Press, Cambridge, UK.

Andron, D., 1962. "Analyse morphologique du substantif russe." Tech. rep., Centre d'Etudes pour la Traduction Automatique, Université de Grenoble 1.

Andry, F., Fraser, N., Thornton, S., and Youd, N., 1993. "Making DATR work for speech: lexicon compilation in SUNDIAL." *Computational Linguistics* 18 (3), 245–267.

Antworth, E., 1990. *PC-KIMMO: A Two-Level Processor for Morphological Analysis*. Occasional Publications in Academic Computing, 16. Summer Institute of Linguistics, Dallas, TX.

Archangeli, D., 1984. "Underspecification in Yawelmani phonology and morphology." Ph.D. thesis, MIT.

Aronoff, M., 1976. *Word Formation in Generative Grammar*. MIT Press, Cambridge, MA.

—— 1994. *Morphology by Itself: Stems and Inflectional Classes*. No. 22 in Linguistic Inquiry Monographs. MIT Press, Cambridge, MA.

Aronson, H., 1990. *Georgian: A Reading Grammar.* Slavica, Columbus, OH.

Attar, R., Choueka, Y., Dershowitz, N., and Fraenkel, A., 1978. "KEDMA—linguistic tools for retrieval systems." *Journal of the ACM* 25, 55–66.

Baayen, R. H., and Moscoso del Prado Martín, F., 2005. "Semantic density and past-tense formation in three Germanic languages." *Language* 81 (3), 666–698.

—— Piepenbrock, R., and van Rijn, H., 1996. *The CELEX Lexical Database* (CD-rom). Linguistic Data Consortium.

Baker, J., 1979. "Trainable grammars for speech recognition." In: *Speech Communication papers for the 97th Meeting of the Acoustical Society of America*, pp. 547–550.

Baker, M., 1985. "The mirror principle and morphosyntactic explanation." *Linguistic Inquiry* 16 (3), 373–416.

Bar-Hillel, Y., 1953. "A quasi-arithmetical notation for syntactic description." *Language* 29, 47–58.

Barg, P., 1994. "Automatic acquisition of DATR theories from observations." Tech. rep., Heinrich-Heine Universität, Düsseldorf, theories des Lexicons: Arbeiten des Sonderforschungsbereichs 282.

Baroni, M., Matiasek, J., and Trost, H., 2002. "Unsupervised discovery of morphologically related words based on orthographic and semantic similarity." In: *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*, pp. 48–57.

Bat-El, O., 2001. "In search for the roots of the C-root: The essence of Semitic morphology." Paper presented at the Workshop on Roots and Template Morphology, Los Angeles, USC, handout available at: <http://www.tau.ac.il/humanities/lingui/downloads/batel/c_root.rtf>.

Bear, J., 1986. "A morphological recognizer with syntactic and phonological rules." In: *Proceedings of the 11th International Conference on Computational Linguistics (COLING)*, pp. 272–276.

Beard, R., 1995. *Lexeme-Morpheme Base Morphology*. SUNY, Albany, NY.

Becker, J., 1984. Multilingual word processing. *Scientific American* (July 1984), 96–107.

Beesley, K., 1989. "Computer analysis of Arabic morphology: a two-level approach with detours." In: *Proceedings of the 3rd Annual Symposium on Arabic Linguistics*. University of Utah, pp. 155–172.

—— and Karttunen, L., 2000. "Finite-state non-concatenative morphotactics." In: *Proceedings of the 5th Workshop of the ACL Special Interest Group on Computational Phonology (SIGPHON-2000)*, pp. 1–12.

—— —— 2003. *Finite State Morphology*. CSLI Publications. University of Chicago Press.

Bernard-Georges, A., Laurent, G., and Levenbach, D., 1962. "Analyse morphologique du verbe allemand." Tech. rep., Centre d'Etudes pour la Traduction Automatique, Université de Grenoble 1.

Besag, J., 1974. "Spatial interaction and the statistical analysis of lattice systems (with discussion)." *Journal of the Royal Statistical Society, Series D* 36, 192–236.

Bikel, D. M., 2004. "Intricacies of Collins' parsing model." *Computational Linguistics* 30 (4), 479–511.

Billot, S., and Lang, B., 1989. "The structure of shared parse forests in ambiguous parsing." In: *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 143–151.

Bilmes, J., 1997. "A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models." Tech. rep., ICSI-TR-97-021, International Computer Science Institute, Berkeley, CA.

—— and Kirchhoff, K., 2003. "Factored language models and generalized parallel backoff." In: *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL), Companion Volume*, pp. 4–6.

Bird, S., and Ellison, T., 1994. "One-level phonology: Autosegmental representations and rules as finite automata." *Computational Linguistics* 20 (1), 55–90.

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M. P., Roukos, S., Santorini, B., and Strzalkowski, T., 1991. "A procedure for quantitatively comparing the syntactic coverage of English grammars." In: *DARPA Speech and Natural Language Workshop*, pp. 306–311.

Blaheta, D., and Charniak, E., 2000. "Assigning function tags to parsed text." In: *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 234–240.

Blevins, J., 2003. "Stems and paradigms." *Language* 79 (4), 737–767.

Bod, R., 1993a. "Data-oriented parsing as a general framework for stochastic language processing." In: Sikkel, K., and Nijholt, A. (Eds.), *Proceedings of Twente Workshop on Language Technology (TWLT6)*. University of Twente, The Netherlands, pp. 37–46.

—— 1993b. "Using an annotated corpus as a stochastic grammar." In: *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 37–44.

—— 1998. *Beyond Grammar: An Experience-Based Theory of Language*. CSLI Publications, Stanford, CA.

—— 2001. "What is the minimal set of fragments that achieves maximal parse accuracy?" In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 66–73.

Böhmová, A., Hajič, J., Hajičová, E., and Hladká, B. V., 2002. "The Prague dependency treebank: Three-level annotation scenario." In: Abeille, A.

(Ed.), *Treebanks: Building and Using Syntactically Annotated Corpora*. Kluwer Academic Publishers, Dordrecht, pp. 103–127.

Boussard, A., and Berthaud, M., 1965. "Présentation de la synthèse morphologique du français." Tech. rep., Centre d'Etudes pour la Traduction Automatique, Université de Grenoble 1.

Brand, I., Klimonow, G., and Nündel, S., 1969. "Lexiko-morphologische Analyse." In: Nündel, S., Klimonow, G., Starke, I., and Brand, I. (Eds.), *Automatische Sprachübersetzung: Russisch-deutsch*. Akademie-Verlag, Berlin, pp. 22–64.

Brew, C., 1995. "Stochastic HPSG." In: *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 83–89.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S., 1990. "A statistical approach to machine translation." *Computational Linguistics* 16 (2), 79–85.

—— Della Pietra, V. J., deSouza, P. V., Lai, J. C., Mercer, R. L., 1992. "Class-based n-gram models of natural language." *Computational Linguistics* 18 (4), 467–479.

Buchholz, S., and Marsi, E., 2006. "CoNLL-X shared task on multilingual dependency parsing." In: *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pp. 149–164.

Buckwalter, T., 2002. *Buckwalter Arabic morphological analyzer*, version 1.0. Linguistic Data Consortium, Catalog # LDC2002L49, ISBN 1-58563-257-0.

Butt, M., Dyvik, H., King, T., Masuichi, H., and Rohrer, C., 2002. "The parallel grammar project." In: *Proceedings of the COLING-2002 Workshop on Grammar Engineering and Evaluation*, pp. 1–7.

Büttel, I., Niedermair, G., Thurmair, G., and Wessel, A., 1986. "MARS: Morphologische Analyse für Retrievalsysteme: Projektbericht." In: Schwarz, C., and Thurmair, G. (Eds.), *Informationslinguistische Texterschliessung*. Georg Olms Verlag, Hildesheim, pp. 157–216.

Byrd, R., Klavans, J., Aronoff, M., and Anshen, F., 1986. "Computer methods for morphological analysis." In: *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 120–127.

Carroll, G., and Charniak, E., 1992. "Two experiments on learning probabilistic dependency grammars from corpora." In: Weir, C., Abney, S., Grishman, R., and Weischedel, R. (Eds.), *Working Notes of the Workshop on Statistically-Based NLP Techniques*. AAAI Press, Menlo Park, CA, pp. 1–13.

Carstairs, A., 1984. "Constraints on allomorphy in inflexion." Ph.D. thesis, University of London, Indiana University Linguistics Club.

Charniak, E., 1997. "Statistical parsing with a context-free grammar and word statistics." In: *Proceedings of the 14th National Conference on Artificial Intelligence*, pp. 598–603.

—— 2000. "A maximum-entropy-inspired parser." In: *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 132–139.

—— 2001. "Immediate-head parsing for language models." In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 116–123.

—— and Johnson, M., 2005. "Coarse-to-fine n-best parsing and MaxEnt discriminative reranking." In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 173–180.

Chelba, C., 2000. "Exploiting syntactic structure for natural language modeling." Ph.D. thesis, The Johns Hopkins University, Baltimore, MD.

—— and Jelinek, F., 1998. "Exploiting syntactic structure for language modeling." In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics (ACL) and 17th International Conference on Computational Linguistics (COLING)*, pp. 225–231.

—— —— 2000. "Structured language modeling." *Computer Speech and Language* 14 (4), 283–332.

Chen, S., and Goodman, J., 1998. "An empirical study of smoothing techniques for language modeling." Tech. rep., TR-10-98, Harvard University.

Chomsky, N., 1957. *Syntactic Structures*. Mouton, The Hague.

—— 1995. *The Minimalist Program*. MIT Press, Cambridge, MA.

—— and Halle, M., 1968. *The Sound Pattern of English*. Harper and Row, New York.

Choueka, Y., 1983. "Linguistic and word-manipulation in textual information systems." In: Keren, C., and Perlmutter, L. (Eds.), *Information, Documentation and Libraries*. Elsevier, New York, pp. 405–417.

—— 1990. "Responsa: An operational full-text retrieval system with linguistic components for large corpora." In: Zampolli, A. (Ed.), *Computational Lexicology and Lexicography*. Giardini, Pisa, p. 150.

Church, K., 1986. "Morphological decomposition and stress assignment for speech synthesis." In: *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 156–164.

Clark, S., 2002. "Supertagging for combinatory categorial grammar." In: *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, pp. 19–24.

—— and Curran, J. R., 2004. "Parsing the WSJ using CCG and log-linear models." In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 104–111.

—— Hockenmaier, J., and Steedman, M., 2002. "Building deep dependency structures with a wide-coverage CCG parser." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 327–334.

Cocke, J., and Schwartz, J. T., 1970. "Programming languages and their compilers: Preliminary notes." Tech. rep., Courant Institute of Mathematical Sciences, New York University.

CODIUL, 1989. "Diccionario elemental del Ulwa (sumu meridional)." Tech. rep., CODIUL/UYUTMUBAL, Karawala Región Autónoma Atlántico Sur, Nicaragua; Centro de Investigaciones y Documentación de la Costa Atlantica, Managua and Bluefields, Nicaragua; Center for Cognitive Science, MIT, Cambridge, MA.

Cohen-Sygal, Y., and Wintner, S., 2006. "Finite-state registered automata for non-concatenative morphology." *Computational Linguistics* 32 (1), 49–82.

Coker, C., Church, K., and Liberman, M., 1990. "Morphology and rhyming: Two powerful alternatives to letter-to-sound rules for speech synthesis." In: Bailly, G., and Benoit, C. (Eds.), *Proceedings of the ESCA Workshop on Speech Synthesis*, pp. 83–86.

Coleman, J., 1992. "Phonological representations – their names, forms and powers." Ph.D. thesis, University of York.

Collins, M. J., 1997. "Three generative, lexicalised models for statistical parsing." In: *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 16–23.

—— 1999. "Head-driven statistical models for natural language parsing." Ph.D. thesis, University of Pennsylvania.

—— 2000. "Discriminative reranking for natural language parsing." In: *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pp. 175–182.

—— 2002. "Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms." In: *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1–8.

—— and Duffy, N., 2002. "New ranking algorithms for parsing and tagging: Kernels over discrete structures and the voted perceptron." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 263–270.

—— and Koo, T., 2005. "Discriminative reranking for natural language parsing." *Computational Linguistics* 31 (1), 25–69.

—— and Roark, B., 2004. "Incremental parsing with the perceptron algorithm." In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 111–118.

—— and Singer, Y., 1999. "Unsupervised models for named entity classification." In: *Proceedings of the 3rd Conference on Empirical Methods in Natural Language Processing (EMNLP) and Very Large Corpora*, pp. 100–110.

Corbett, G., and Fraser, N., 1993. "Network morphology: a DATR account of Russian nominal inflection." *Journal of Linguistics* 29, 113–142.

Creutz, M., and Lagus, K., 2002. "Unsupervised discovery of morphemes." In: *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*, pp. 21–30.

Culy, C., 1985. "The complexity of the vocabulary of Bambara." *Linguistics and Philosophy* 8, 345–351.

Dolby, J., Earl, L., and Resnikoff, H., 1965. "The application of English-word morphology to automatic indexing and extracting." Tech. Rep. M-21-65-1, Lockheed Missiles and Space Company, Palo Alto, CA.

Duanmu, S., 1997. "Recursive constraint evaluation in Optimality Theory: evidence from cyclic compounds in Shanghai." *Natural Language and Linguistic Theory* 15, 465–507.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G., 1998. *Biological Sequence Analysis*. Cambridge University Press, Cambridge, UK.

Earley, J., 1970. "An efficient context-free parsing algorithm." *Communications of the ACM* 6 (8), 451–455.

Eisner, J., 1996a. "Efficient normal-form parsing for combinatory categorial grammar." In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 79–86.

—— 1996b. "Three new probabilistic models for dependency parsing: An exploration." In: *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pp. 340–345.

—— 1997. "Bilexical grammars and a cubic-time probabilistic parser." In: *Proceedings of the 5th International Workshop on Parsing Technologies (IWPT)*, pp. 54–65.

—— and Satta, G., 1999. "Efficient parsing for bilexical context-free grammars and head automaton grammars." In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 457–464.

Evans, R., and Gazdar, G., 1989a. "Inference in DATR." In: *Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 66–71.

—— —— 1989b. "The semantics of DATR." In: Cohn, A. (Ed.), *Proceedings of the Seventh Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (AISB)*. Pitman/Morgan Kaufmann, London, pp. 79–87.

—— —— and Weir, D., 1995. "Encoding lexicalized tree adjoining grammars with a nonmonotonic inheritance hierarchy." In: *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 77–84.

Finkel, R., and Stump, G., 2002. "Generating Hebrew verb morphology by default inheritance hierarchies." In: *Proceedings of the ACL-02 Workshop on Computational Approaches to Semitic Languages*, pp. 9–18.

Francis, N. W., and Kucera, H., 1982. *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton Mifflin, Boston.

Frazier, L., and Fodor, J. D., 1978. "The sausage machine: a new two-stage parsing model." *Cognition* 6, 291–325.

Freund, Y., Iyer, R., Schapire, R., and Singer, Y., 1998. "An efficient boosting algorithm for combining preferences." In: *Proceedings of the 15th International Conference on Machine Learning (ICML)*, pp. 170–178.

Gabbard, R., Marcus, M. P., and Kulick, S., 2006. "Fully parsing the Penn treebank." In: *Proceedings of the 2006 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2006)*, pp. 184–191.

Gaussier, E., 1999. "Unsupervised learning of derivational morphology from inflectional lexicons." In: *Proceedings of the Workshop on Unsupervised Learning in Natural Language Processing*, pp. 24–30.

Geman, S., and Johnson, M., 2002. "Dynamic programming for parsing and estimation of stochastic unification-based grammars." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 279–286.

Gerdemann, D., and van Noord, G., 2000. "Approximation and exactness in finite state optimality theory." In: *Proceedings of the 5th Workshop of the ACL Special Interest Group on Computational Phonology (SIGPHON-2000)*.

Gildea, D., 2001. "Corpus variation and parser performance." In: *Proceedings of the 6th Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 167–202.

Goldsmith, J., 2001. "Unsupervised acquisition of the morphology of a natural language." *Computational Linguistics* 27 (2), 153–198.

Good, I. J., 1953. "The population frequencies of species and the estimation of population parameters." *Biometrica* V 40 (3,4), 237–264.

Goodman, J., 1996a. "Efficient algorithms for parsing the DOP model." In: *Proceedings of the 1st Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 143–152.

—— 1996b. "Parsing algorithms and metrics." In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 177–183.

—— 1998. "Parsing inside-out." Ph.D. thesis, Harvard University.

Hakkani-Tür, D., Oflazer, K., and Tür, G., 2002. "Statistical morphological disambiguation for agglutinative languages." *Computers and the Humanities* 36 (4), 381–410.

Hall, K., 2004. "Best-first word-lattice parsing: Techniques for integrated syntactic language modeling." Ph.D. thesis, Brown University, Providence, RI.

—— and Johnson, M., 2004. "Attention shifting for parsing speech." In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 40–46.

—— and Novak, V., 2005. "Corrective modeling for non-projective dependency parsing." In: *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pp. 42–52.

Halle, M., and Marantz, A., 1993. "Distributed morphology and the pieces of inflection." In: Hale, K., and Keyser, S. J. (Eds.), *The View from Building 20*. MIT Press, Cambridge, MA, pp. 111–176.

Hankamer, J., 1986. "Finite state morphology and left to right phonology." In: *Proceedings of the West Coast Conference on Formal Linguistics, Volume 5*. Stanford Linguistic Association, Stanford, pp. 41–52.

Harlow, S., 1981. "Government and relativization in Celtic." In: Heny, F. (Ed.), *Binding and Filtering*. Croom Helm, London, pp. 213–254.

—— 1989. "The syntax of Welsh soft mutation." *Natural Language and Linguistic Theory* 7 (3), 289–316.

Harris, A., 1981. *Georgian Syntax: A Study in Relational Grammar*. Cambridge University Press, Cambridge, UK.

Harris, Z., 1951. *Structural Linguistics*. University of Chicago Press, Chicago.

Harrison, M., 1978. *Introduction to Formal Language Theory*. Addison Wesley, Reading, MA.

Hay, J., and Baayen, R., 2005. "Shifting paradigms: gradient structure in morphology," *Trends in Cognitive Sciences* 9, 342–348.

Heemskerk, J., 1993. "A probabilistic context-free grammar for disambiguation in morphological parsing." In: *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 183–192.

Henderson, J., 2003. "Inducing history representations for broad coverage statistical parsing." In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2003)*, pp. 103–110.

—— 2004. "Discriminative training of a neural network statistical parser." In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 95–102.

Hindle, D., and Rooth, M., 1993. "Structural ambiguity and lexical relations." *Computational Linguistics* 19 (1), 103–120.

Hockenmaier, J., and Steedman, M., 2002. "Generative models for statistical parsing with combinatory categorial grammar." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 335–342.

—— —— 2005. "CCGbank manual." Tech. rep., MS-CIS-05-09, Department of Computer and Information Science, University of Pennsylvania.

Hockett, C., 1954. "Two models of grammatical description." *Word* 10, 210–231.

Hollingshead, K., Fisher, S., and Roark, B., 2005. "Comparing and combining finite-state and context-free parsers." In: *Proceedings of the 2005 Human Language Technology Conference and Conference on Empirical*

Methods in Natural Language Processing (HLT-EMNLP) 2005, pp. 787–794.

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

Huang, L., and Chiang, D., 2005. "Better k-best parsing." In: *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pp. 53–64.

Hutchins, W. J., 2001. "Machine translation over 50 years." *Histoire, Epistemologie, Langage* 22 (1), 7–31, available at: <http://ourworld.compuserve.com/homepages/wjhutchins/HEL.pdf>.

Inkelas, S., and Zoll, C., 1999. "Reduplication as morphological doubling." Tech. Rep. 412-0800, Rutgers Optimality Archive.

—— —— 2005. *Reduplication: Doubling in Morphology.* Cambridge University Press, Cambridge, UK.

Jansche, M., 2005. "Algorithms for minimum risk chunking." In: *Proceedings of the 5th International Workshop on Finite-State Methods in Natural Language Processing (FSMNLP).*

Jelinek, F., 1998. *Statistical Methods for Speech Recognition.* The MIT Press, Cambridge, MA.

—— and Lafferty, J. D., 1991. "Computation of the probability of initial substring generation by stochastic context-free grammars." *Computational Linguistics* 17 (3), 315–323.

—— and Mercer, R. L., 1980. "Interpolated estimation of Markov source parameters from sparse data." In: *Proceedings of the First International Workshop on Pattern Recognition in Practice*, pp. 381–397.

Jijkoun, V., and de Rijke, M., 2004. "Enriching the output of a parser using memory-based learning." In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 311–318.

Johnson, C. D., 1972. *Formal Aspects of Phonological Description.* Mouton, The Hague.

Johnson, H., and Martin, J., 2003. "Unsupervised learning of morphology for English and Inuktitut." In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2003), Companion Volume*, pp. 43–45.

Johnson, M., 1998a. "Finite-state approximation of constraint-based grammars using left-corner grammar transforms." In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics (ACL) and 17th International Conference on Computational Linguistics (COLING)*, pp. 619–623.

—— 1998b. "PCFG models of linguistic tree representations." *Computational Linguistics* 24 (4), 617–636.

—— 2002a. "The DOP estimation method is biased and inconsistent." *Computational Linguistics* 28 (1), 71–76.

—— 2002b. "A simple pattern-matching algorithm for recovering empty nodes and their antecedents." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 136–143.

—— and Roark, B., 2000. "Compact non-left-recursive grammars using the selective left-corner transform and factoring." In: *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pp. 355–361.

—— Geman, S., Canon, S., Chi, Z., and Riezler, S., 1999. "Estimators for stochastic 'unification-based' grammars." In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 535–541.

Joshi, A. K., 1985. "How much context-sensitivity is necessary for assigning structural descriptions." In: Dowty, D., Karttunen, L., and Zwicky, A. (Eds.), *Natural Language Parsing.* Cambridge University Press, Cambridge, UK, pp. 206–250.

—— and Schabes, Y., 1997. "Tree-adjoining grammars." In: Rozenberg, G., and Salomaa, A. (Eds.), *Handbook of Formal Languages. Vol 3: Beyond Words.* Springer-Verlag, Berlin/Heidelberg/New York, pp. 69–123.

—— and Srinivas, B., 1994. "Disambiguation of super parts of speech (or supertags): Almost parsing." In: *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, pp. 154–160.

—— Levy, L. S., and Takahashi, M., 1975. "Tree adjunct grammars." *Journal of Computer and System Sciences* 10 (1), 136–163.

—— Vijay-Shanker, K., and Weir, D., 1991. "The convergence of mildly context-sensitive formalisms." In: Sells, P., Shieber, S., and Wasow, T. (Eds.), *Processing of Linguistic Structure.* MIT Press, Cambridge, MA, pp. 31–81.

Kaplan, R. M., and Bresnan, J., 1982. "Lexical-functional grammar: A formal system for grammatical representation." In: Bresnan, J. (Ed.), *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, MA, pp. 173–281.

—— and Kay, M., 1994. "Regular models of phonological rule systems." *Computational Linguistics* 20, 331–378.

—— Riezler, S., King, T., Maxwell III, J. T., Vasserman, A., and Crouch, R., 2004. "Speed and accuracy in shallow and deep stochastic parsing." In: *Proceedings of the 2004 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2004)*, pp. 97–104.

Karttunen, L., 1998. "The proper treatment of optimality in computational phonology." In: *Proceedings of the 2nd International Workshop on Finite-State Methods in Natural Language Processing (FSMNLP)*, pp. 1–12.

—— 2003. "Computing with realizational morphology." In: Gelbukh, A. (Ed.), *Computational Linguistics and Intelligent Text Processing.* Vol. 2588 of *Lecture Notes in Computer Science.* Springer Verlag, Heidelberg, pp. 205–216.

Karttunen, L., and Beesley, K., 2005. "Twenty-five years of finite-state morphology." In: Arppe, A., Carlson, L., Lindén, K., Piitulainen, J., Suominen, M., Vainio, M., Westerlund, H., and Yli-Jyrä, A. (Eds.), *Inquiries into Words, Constraints and Contexts (Festschrift in the Honour of Kimmo Koskenniemi and his 60th Birthday)*. Gummerus Printing, Saarijärvi, Finland, pp. 71–83, available on-line at: <http://cslipublications.stanford.edu/site/SCLO.html>.

—— Kaplan, R. M., and Zaenen, A., 1992. "Two-level morphology with composition." In: *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, pp. 141–148.

Kasami, T., 1965. "An efficient recognition and syntax analysis algorithm for context-free languages." Tech. rep., AFCRL-65-758, Air Force Cambridge Research Lab., Bedford, MA.

Katz, S. M., 1987. "Estimation of probabilities from sparse data for the language model component of a speech recogniser." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35 (3), 400–401.

Kay, M., 1986. "Algorithm schemata and data structures in syntactic processing." In: Grosz, B. J., Sparck-Jones, K., and Webber, B. L. (Eds.), *Readings in Natural Language Processing*. Morgan Kaufmann, Los Altos, pp. 35–70.

Kiraz, G., 2000. *Computational Approach to Non-Linear Morphology*. Cambridge University Press, Cambridge, UK.

Klein, D., 2005. "The unsupervised learning of natural language structure." Ph.D. thesis, Stanford, Palo Alto, CA.

—— and Manning, C. D., 2002. "A generative constituent-context model for improved grammar induction." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 128–135.

—— —— 2003a. "A* parsing: Fast exact Viterbi parse selection." In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2003)*, pp. 119–126.

—— —— 2003b. "Accurate unlexicalized parsing." In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 423–430.

—— —— 2004. "Corpus-based induction of syntactic structure: Models of dependency and constituency." In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 478–485.

Kneser, R., and Ney, H., 1995. "Improved backing-off for m-gram language modeling." In: *Proceedings of the 1995 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 1995)*, pp. 181–184.

Knuth, D., 1973. *The Art of Computer Programming*. Vol. 3. Addison-Wesley, Reading, MA.

Kornai, A., 1991. "Formal phonology." Ph.D. thesis, Stanford University, distributed by Garland Publishers.

Koskenniemi, K., 1983. "Two-level morphology: a general computational model for word-form recognition and production." Ph.D. thesis, University of Helsinki, Helsinki.

—— 1984. "FINSTEMS: a module for information retrieval." In: *Computational Morphosyntax: Report on Research 1981–1984*. University of Helsinki, Helsinki, pp. 81–92.

Kumar, S., and Byrne, W., 2004. "Minimum Bayes-risk decoding for machine translation." In: *Proceedings of the 2004 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2004)*, pp. 169–176.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N., 2001. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data." In: *Proceedings of the 18th International Conference on Machine Learning*, pp. 282–289.

Lambek, J., 1958. "The mathematics of sentence structure." *American Mathematical Monthly* 65, 154–169.

Lari, K., and Young, S., 1990. "The estimation of stochastic context-free grammars using the inside-outside algorithm." *Computer Speech and Language* 4 (1), 35–56.

Lee, G. G., Lee, J.-H., and Cha, J., 2002. "Syllable-pattern-based unknown morpheme segmentation and estimation for hybrid part-of-speech tagging of Korean." *Computational Linguistics* 28 (1), 53–70.

Levy, R., and Manning, C. D., 2004. "Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation." In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 327–334.

Lewis, H., and Papadimitriou, C., 1981. *Elements of the Theory of Computation*. Prentice-Hall, Englewood Cliffs, NJ.

Lewis, R. L., 1998. "Reanalysis and limited repair parsing: Leaping off the garden path." In: Fodor, J. D., and Ferreira, F. (Eds.), *Reanalysis in Sentence Processing*. Kluwer Academic Publishers, Dordrecht, pp. 247–284.

Lewis II, P., and Stearns, R., 1968. "Syntax-directed transduction." *Journal of the Association for Computing Machinery* 15 (3), 465–488.

Lieber, R., 1980. "On the organization of the lexicon." Ph.D. thesis, MIT, Cambridge, MA.

—— 1987. *An Integrated Theory of Autosegmental Processes*. SUNY Series in Linguistics. SUNY Press, Albany, NY.

—— 1992. *Deconstructing Morphology: Word Formation in a Government-Binding Syntax*. University of Chicago Press, Chicago.

Lin, D., 1998a. "Dependency-based evaluation of minipar." In: *Workshop on the Evaluation of Parsing Systems*, pp. 48–56.

—— 1998b. "A dependency-based method for evaluating broad-coverage parsers." *Natural Language Engineering* 4 (2), 97–114.

Lombardi, L., and McCarthy, J., 1991. "Prosodic circumscription in Choctaw morphology." *Phonology* 8, 37–72.

Magerman, D. M., 1995. "Statistical decision-tree models for parsing." In: *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 276–283.

—— and Marcus, M. P., 1990. "Parsing a natural language using mutual information statistics." In: *Proceedings of the 8th National Conference on Artificial Intelligence*, pp. 984–989.

Mangu, L., Brill, E., and Stolcke, A., 1999. "Finding consensus among words: Lattice-based word error minimization." In: *Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech)*, pp. 495–498.

Manning, C. D., and Carpenter, B., 1997. "Probabilistic parsing using left corner language models." In: *Proceedings of the 5th International Workshop on Parsing Technologies (IWPT)*, pp. 147–158.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A., 1993. "Building a large annotated corpus of English: The Penn Treebank." *Computational Linguistics* 19 (2), 313–330.

Matthews, P., 1966. "A procedure for morphological encoding." *Mechanical Translation* 9, 15–21.

—— 1972. *Inflectional Morphology: a Theoretical Study Based on Aspects of Latin Verb Conjugations*. Cambridge University Press, Cambridge, UK.

Maxwell III, J. T., and Kaplan, R. M., 1998. "Unification-based parsers that automatically take advantage of context freeness." MS., Xerox PARC, available at: <http://www2.parc.com/istl/groups/nltt/xle/doc/xle-performance.ps>.

McCallum, A., and Li, W., 2003. "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons." In: *Proceedings of the 7th Conference on Computational Natural Language Learning (CoNLL)*, pp. 188–191.

—— Freitag, D., and Pereira, F. C. N., 2000. "Maximum entropy Markov models for information extraction and segmentation." In: *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pp. 591–598.

McCarthy, J., 1979. "Formal problems in Semitic morphology and phonology." Ph.D. thesis, MIT, Cambridge, MA, distributed by Indiana University Linguistics Club (1982).

—— and Prince, A., 1986. "Prosodic morphology." MS. University of Massachusetts, Amherst, and Brandeis University.

—— —— 1990. "Foot and word in prosodic morphology: The Arabic broken plural." *Natural Language and Linguistic Theory* 8, 209–284.

McDonald, R., Pereira, F. C. N., Ribarov, K., and Hajič, J., 2005. "Non-projective dependency parsing using spanning tree algorithms." In: *Proceedings of the Conference on Human Language Technology Conference and Empirical Methods in Natural Language Processing (HLT-EMNLP)*, pp. 523–530.

McIlroy, M. D., 1982. "Development of a spelling list." *IEEE Transactions on Communications* 30 (1), 91–99.

Melamed, I. D., 2003. "Multitext grammars and synchronous parsers." In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2003)*, pp. 158–165.

—— Satta, G., and Wellington, B., 2004. "Generalized multitext grammars." In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 661–668.

Mel'čuk, I., 1988. *Dependency Syntax: Theory and Practice*. SUNY Press, Albany, NY.

Meya-Lloport, M., 1987. "Morphological analysis of Spanish for retrieval." *Literary and Linguistic Computing* 2, 166–170.

Michaelis, J., 2001a. "Derivational minimalism is mildy context-sensitive." In: Moortgat, M. (Ed.), *Logical Aspects of Computational Linguistics, LNCS/LNAI Vol. 2014*. Springer-Verlag, Berlin/Heidelberg/New York, pp. 179–198.

—— 2001b. "Transforming linear context-free rewriting systems into minimalist grammars." In: Retoré, G. M. C. (Ed.), *Logical Aspects of Computational Linguistics, LNCS/LNAI Vol. 2099*. Springer-Verlag, Berlin/Heidelberg/New York, pp. 228–244.

Mohri, M., 1994. "Syntactic analysis by local grammars automata: an efficient algorithm." In: *Papers in Computational Lexicography: COMPLEX '94*. Research Institute for Linguistics, Hungarian Academy of Sciences, Budapest, pp. 179–191.

—— 1997. "Finite-state transducers in language and speech processing." *Computational Linguistics* 23, 269–311.

—— 2002. "Generic epsilon-removal and input epsilon-normalization algorithms for weighted transducers." *International Journal of Foundations of Computer Science* 13 (1), 129–143.

—— and Nederhof, M.-J., 2001. "Regular approximation of context-free grammars through transformation." In: Junqua, J.-C., and van Noord, G. (Eds.), *Robustness in Language and Speech Technology*. Kluwer Academic Publishers, Dordrecht, pp. 153–163.

—— and Riley, M., 1999. "Network optimizations for large vocabulary speech recognition." *Speech Communication* 28 (1), 1–12.

—— —— 2002. "An efficient algorithm for the n-best-strings problem." In: *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, pp. 1313–1316.

Mohri, M., and Roark, B., 2006. "Probabilistic context-free grammar induction based on structural zeros." In: *Proceedings of the 2006 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2006)*, pp. 312–319.

—— and Sproat, R., 1996. "An efficient compiler for weighted rewrite rules." In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 231–238.

—— Pereira, F. C. N., and Riley, M., 2002. "Weighted finite-state transducers in speech recognition." *Computer Speech and Language* 16 (1), 69–88.

Montague, R., 1974. *Formal Philosophy: Papers of Richard Montague*. Yale University Press, New Haven, CT.

Moore, R. C., 2000. "Removing left recursion from context-free grammars." In: *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 249–255.

Morgan, T., 1952. *Y Treigladau a'u Cystrawen*. University of Wales Press, Cardiff.

Nederhof, M.-J., 2000. "Practical experiments with regular approximation of context-free languages." *Computational Linguistics* 26 (1), 17–44.

Newman, S., 1944. *Yokuts Language of California*. Viking Fund Publications in Anthropology, New York.

Ney, H., Essen, U., and Kneser, R., 1994. "On structuring probabilistic dependencies in stochastic language modeling." *Computer Speech and Language* 8, 1–38.

Nijholt, A., 1980. *Context-free Grammars: Covers, Normal Forms, and Parsing*. Springer Verlag, Berlin/Heidelberg/New York.

Nivre, J., and Nilsson, J., 2005. "Pseudo-projective dependency parsing." In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 99–106.

Pereira, F. C. N., and Riley, M., 1997. "Speech recognition by composition of weighted finite automata." In: Roche, E., and Schabes, Y. (Eds.), *Finite-State Language Processing*, MIT Press, Cambridge, MA, pp. 431–453.

—— and Schabes, Y., 1992. "Inside–outside reestimation from partially bracketed corpora." In: *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 128–135.

—— and Wright, R. N., 1997. "Finite-state approximation of phrase-structure grammars." In: Roche, E., and Schabes, Y. (Eds.), *Finite-State Language Processing*. MIT Press, Cambridge, MA, pp. 149–173.

Pesetsky, D., 1985. "Morphology and logical form." *Linguistic Inquiry* 16, 193–246.

Pinker, S., 1999. *Words and Rules*. Weidenfeld and Nicholson, London.

—— and Prince, A., 1988. "On language and connectionism: Analysis of a parallel distributed processing model of language acquisition." In: Pinker, S., and Mehler, J. (Eds.), *Connections and Symbols. Cognition* special issue, MIT Press, pp. 73–193.

Pollard, C., 1984. "Generalized phrase structure grammars." Ph.D. thesis, Stanford University.

—— and Sag, I., 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Porter, M., 1980. "An algorithm for suffix stripping." *Program* 14 (3), 130–137.

Prince, A., and Smolensky, P., 1993. "Optimality theory." Tech. Rep. 2, Rutgers University, Piscataway, NJ.

Pullum, G., and Zwicky, A., 1984. "The syntax–phonology boundary and current syntactic theories." In: *Ohio State Working Papers in Linguistics*. No. 29. Department of Linguistics, The Ohio State University, Columbus, OH, pp. 105–116.

Ratnaparkhi, A., 1996. "A maximum entropy model for part-of-speech tagging." In: *Proceedings of the 1st Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 133–142.

—— 1997. "A linear observed time statistical parser based on maximum entropy models." In: *Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1–10.

—— 1999. "Learning to parse natural language with maximum entropy models." *Machine Learning* 34, 151–175.

Resnik, P., 1992. "Left-corner parsing and psychological plausibility." In: *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, pp. 191–197.

Riezler, S., Prescher, D., Kuhn, J., and Johnson, M., 2000. "Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training." In: *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 480–487.

—— King, T., Kaplan, R. M., Crouch, R., Maxwell III, J. T., and Johnson, M., 2002. "Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 271–278.

Ritchie, G., Russell, G., Black, A., and Pulman, S., 1992. *Computational Morphology: Practical Mechanisms for the English Lexicon*. MIT Press, Cambridge, MA.

Roark, B., 2001. "Probabilistic top-down parsing and language modeling." *Computational Linguistics* 27 (2), 249–276.

—— 2002. "Markov parsing: lattice rescoring with a statistical parser." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 287–294.

Roark, B., 2004. "Robust garden path parsing." *Natural Language Engineering* 10 (1), 1–24.

—— and Johnson, M., 1999. "Efficient probabilistic top-down and left-corner parsing." In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 421–428.

—— Saraclar, M., Collins, M. J., and Johnson, M., 2004. "Discriminative language modeling with conditional random fields and the perceptron algorithm." In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 47–54.

Rosenfeld, R., 1996. "A maximum entropy approach to adaptive statistical language modeling." *Computer Speech and Language* 10, 187–228.

Rosenkrantz, S. J., and Lewis II, P., 1970. "Deterministic left corner parsing." In: *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata*, pp. 139–152.

Rumelhart, D., and McClelland, J., 1986. "On learning the past tense of English verbs." In: McClelland, J., and Rumelhart, D. (Eds.), *Parallel Distributed Processing*, Volume 2. MIT Press, Cambridge, MA, pp. 216–271.

Saul, L., and Pereira, F. C. N., 1997. "Aggregate and mixed-order Markov models for statistical language processing." In: *Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 81–89.

Schabes, Y., Abeille, A., and Joshi, A. K., 1988. "Parsing strategies with 'lexicalized' grammars: application to tree adjoining grammars." In: *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, pp. 578–583.

Schone, P., and Jurafsky, D., 2000. "Knowledge-free induction of morphology using latent semantic analysis." In: *Proceedings of the 4th Conference on Computational Natural Language Learning (CoNLL)*, pp. 67–72.

—— —— 2001. "Knowledge-free induction of inflectional morphologies." In: *Proceedings of the 2nd Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 1–9.

Schütze, H., 1998. "Automatic word sense discrimination." *Computational Linguistics* 24 (1), 97–124.

Schveiger, P., and Mathe, J., 1965. "Analyse d'information de la déclinaison du substantif en hongrois (du point de vue de la traduction automatique)." *Cahiers de Linguistique Théorique et Appliquée* 2, 263–265.

Seidenadel, C. W., 1907. *The Language Spoken by the Bontoc Igorot*. Open Court Publishing Company, Chicago.

Sha, F., and Pereira, F. C. N., 2003. "Shallow parsing with conditional random fields." In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2003)*, pp. 213–220.

Sharma, U., Jugal, K., and Das, R., 2002. "Unsupervised learning of morphology for building lexicon for a highly inflectional language." In: *Proceedings*

of the ACL-02 Workshop on Morphological and Phonological Learning, pp. 1–10.

Shieber, S. M., and Schabes, Y., 1990. "Synchronous tree-adjoining grammars." In: *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, pp. 253–258.

Snover, M., Jarosz, G., and Brent, M., 2002. "Unsupervised learning of morphology using a novel directed search algorithm: Taking the first step." In: *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*, pp. 11–20.

Sproat, R., 1985. "On deriving the lexicon." Ph.D. thesis, MIT, Cambridge, MA, distributed by MIT Working Papers in Linguistics.

—— 1992. *Morphology and Computation*. MIT Press, Cambridge, MA.

—— 1997a. "Multilingual text analysis for text-to-speech synthesis." *Natural Language Engineering* 2 (4), 369–380.

—— (Ed.), 1997b. *Multilingual Text-to-Speech Synthesis: The Bell Labs Approach*. Kluwer Academic Publishers, Dordrecht.

—— 2000. *A Computational Theory of Writing Systems*. Cambridge University Press, Cambridge, UK.

—— and Riley, M., 1996. "Compilation of weighted finite-state transducers from decision trees." In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 215–222.

—— Hu, J., and Chen, H., 1998. "EMU: An e-mail preprocessor for text-to-speech." In: *Proceedings of the IEEE Signal Processing Society Workshop on Multimedia Signal Processing*, pp. 239–244.

Srinivas, B., 1996. "Almost parsing technique for language modeling." In: *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, pp. 1169–1172.

—— and Joshi, A. K., 1999. "Supertagging: an approach to almost parsing." *Computational Linguistics* 25 (2), 237–265.

Stabler, E., 1997. "Derivational minimalism." In: Retoré, C. (Ed.), *Logical Aspects of Computational Linguistics, LNCS/LNAI Vol. 1328*. Springer-Verlag, Berlin/Heidelberg/New York, pp. 68–95.

Steedman, M., 1985. "Dependency and coordination in the grammar of Dutch and English." *Language* 61, 523–568.

—— 1986. "Combinators and grammars." In: Oehrle, R., Bach, E., and Wheeler, D. (Eds.), *Categorial Grammars and Natural Language Structures*. Foris, Dordrecht, pp. 417–442.

—— 1996. *Surface Structure and Interpretation*. MIT Press, Cambridge, MA.

Steele, S., 1995. "Towards a theory of morphological information." *Language* 71, 260–309.

Stolcke, A., 1995. "An efficient probabilistic context-free parsing algorithm that computes prefix probabilities." *Computational Linguistics* 21 (2), 165–202.

Stolcke, A., and Segal, J., 1994. "Precise n-gram probabilities from stochastic context-free grammars." In: *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 74–79.

—— König, Y., and Weintraub, M., 1997. "Explicit word error minimization in n-best list rescoring." In: *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech)*, pp. 163–166.

Stump, G., 2001. *Inflectional Morphology: A Theory of Paradigm Structure.* Cambridge University Press, Cambridge, UK.

Thorne, D., 1993. *A Comprehensive Welsh Grammar*. Blackwell, Oxford.

Tjong Kim Sang, E. F., and Buchholz, S., 2000. "Introduction to the CoNLL-2000 shared task: Chunking." In: *Proceedings of the 4th Conference on Computational Natural Language Learning (CoNLL)*, pp. 127–132.

Tzoukermann, E., and Liberman, M., 1990. "A finite-state morphological processor for Spanish." In: *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, pp. 277–286.

van den Bosch, A., and Daelemans, W., 1999. "Memory-based morphological analysis." In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 285–292.

Vauquois, B., 1965. "Présentation d'un programme d'analyse morphologique russe." Tech. rep., Centre d'Etudes pour la Traduction Automatique, Université de Grenoble 1.

Vijay-Shanker, K., and Joshi, A. K., 1985. "Some computational properties of tree adjoining grammars." In: *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 82–93.

—— and Weir, D., 1990. "Polynomial time parsing of combinatory categorial grammars." In: *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 82–93.

—— —— 1993. "Parsing some constrained grammar formalisms." *Computational Linguistics* 19 (4), 591–636.

—— —— and Joshi, A. K., 1987. "Characterizing structural descriptions produced by various grammatical formalisms." In: *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 104–111.

Voutilainen, A., 1994. "Designing a parsing grammar." Tech. rep. 22, University of Helsinki.

Walther, M., 2000a. "Finite-state reduplication in one-level prosodic morphology." In: *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 296–302.

Walther, M., 2000b. "Temiar reduplication in one-level prosodic morphology." In: *Proceedings of the 5th Workshop of the ACL Special Interest Group on Computational Phonology (SIGPHON-2000)*, pp. 13–21.

Wang, W., and Harper, M. P., 2002. "The superARV language model: Investigating the effectiveness of tightly integrating multiple knowledge sources." In: *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 238–247.

—— —— and Stolcke, A., 2003. "The robustness of an almost-parsing language model given errorful training data." In: *Proceedings of the 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2003)*, pp. 240–243.

Weir, D., 1988. "Characterizing mildly context-sensitive grammar formalisms." Ph.D. thesis, University of Pennsylvania.

Wicentowski, R., 2002. "Modeling and learning multilingual inflectional morphology in a minimally supervised framework." Ph.D. thesis, Johns Hopkins University, Baltimore, MD.

Witten, I. H., and Bell, T. C., 1991. "The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression." *IEEE Transactions on Information Theory* 37 (4), 1085–1094.

Woyna, A., 1962. "Morphological analysis of Polish verbs in terms of machine translation." Tech. rep., Machine Translation Research Project, Georgetown University.

Wright, J., 1910. *Grammar of the Gothic Language.* Oxford University Press, Oxford.

Wu, D., 1997. "Stochastic inversion transduction grammars and bilingual parsing of parallel corpora." *Computational Linguistics* 23 (3), 377–404.

XTAG Research Group, 2001. "A lexicalized tree adjoining grammar for English." Tech. rep. IRCS-01-03, IRCS, University of Pennsylvania.

Xu, P., Chelba, C., and Jelinek, F., 2002. "A study on richer syntactic dependencies for structured language modeling." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 191–198.

—— Emami, A., and Jelinek, F., 2003. "Training connectionist models for the structured language model." In: *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 160–167.

Yarowsky, D., and Wicentowski, R., 2001. "Minimally supervised morphological analysis by multimodal alignment." In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 207–216.

Younger, D. H., 1967. "Recognition and parsing of context-free languages in time n^3." *Information and Control* 10 (2), 189–208.