Computational Optimality Theory

Citation to Publisher Version: No citation available.

Citation to *this* Version: Tesar, Bruce. *Computational Optimality Theory*, 1995. Retrieved from doi:10.7282/T3D798RT.

COMPUTATIONAL OPTIMALITY THEORY

by

BRUCE BENSON TESAR

B.S., Western Michigan University, 1990

M.S., University of Colorado, 1992

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirement for the degree of

Doctor of Philosophy

Department of Computer Science

1995

This thesis for the Doctor of Philosophy degree by

Bruce Benson Tesar

has been approved for the

Department of Computer Science

by

Paul Smolensky

James Martin

Michael Mozer

Clayton Lewis

Geraldine Legendre

Date __________

Tesar, Bruce Benson (Ph.D., Computer Science)

Computational Optimality Theory

Thesis directed by Professor Paul Smolensky

Abstract

In Optimality Theory, a linguistic input is assigned a grammatical structural description by selecting, from an infinite set of candidate structural descriptions, the description which best satisfies a ranked set of universal constraints. Cross-linguistic variation is explained as different rankings of the same universal constraints. Two questions are of primary interest concerning the computational tractibility of Optimality Theory. The first concerns the ability to compute optimal structural descriptions. The second concerns the learnability of the constraint rankings.

Parsing algorithms are presented for the computation of optimal forms, using dynamic programming. These algorithms work for grammars in Optimality Theory employing universal constraints which may be evaluated on the basis of information local within the structural description. This approach exploits optimal substructure to construct the optimal description, rather than searching for the solution by moving from one entire description to another.

A class of learning algorithms, the Constraint Demotion algorithms, are presented, which solve the problem of learning constraint rankings based upon hypothesized structural descriptions (an important subproblem of the general problem of language learning). Constraint Demotion exploits the implicit negative evidence available in the form of the competing (suboptimal) structural descriptions of the input. The data complexity of this algorithm is quadratic in the number of constraints.

Acknowledgments

First, I would like to thank the National Science Foundation, for the NSF Graduate Fellowship that supported me for three of my five years of graduate study, and for the research assistanceship support I received through NSF grant IRI-9213894 to Paul Smolensky and Geraldine Legendre. Thanks also to the Computer Science Department of the University of Colorado, for a teaching assistanceship my first year of graduate study.

I would like to thank the members of my committee, James Martin, Michael Mozer, Clayton Lewis, and Geraldine Legendre, for their support and for reading my dissertation.

Many thanks are due to the members of the Boulder Connectionist Research Group, for encouragement, intellectual and moral support, and for generally being a good bunch of people. Thanks go to Mike Mozer for his help and support, especially during my first two years. Thanks also to Kevin Markey, for convincing me that I did not want to do a dissertation based upon lots of simulations of a complex system (while simultaneously demonstrating that it could in fact be done, and well). Gratitude is also due Brian Bonnlander and Sreerupa Das, my fellow connectionist classmates who started the program in Boulder at the same time I did: to Rupa for her enthusiasm in exchanging cultural activities, and to Brian for being just about my only friend out west who appreciates good jazz music.

I wish to thank the Cognitive Science Department of The Johns Hopkins University, for graciously hosting me during my final year of graduate study. I also wish to thank the Linguistics Department and the Center for Cognitive Science at Rutgers University, for hosting me during a very stimulating and valuable two week visit. Special thanks go to my primary hosts at Rutgers, Jane Grimshaw and Alan Prince, for valuable discussions, support, and other important things.

I unfortunately cannot list all of the other people with whom I have had valuable interaction over the past several years. I would like to thank Bruce Hayes, David Haussler, and Mark Liberman for particularly useful discussions. I would also like to offer collective thanks to the many people I met and talked with at the 1993 Connectionist Models Summer School and the 1994 First International Summer Institute for Cognitive Science.

The writing of this dissertation was made considerably more difficult by tendonitis and arm nerve irritation, leaving me unable to type or manually write for any significant extent. I wish to thank everyone who assisted me over the past year and a half with typing and writing, as I struggled to transmit my ideas from my head onto paper. The list is long, but I want to be sure and recognize Geraldine Legendre, Phyllis Bellver, Bill Raymond, Syed Waqar Hasib, and my typist at Johns Hopkins, Lauren Kim, who actually typed the majority of this dissertation. Thanks also to IBM for developing the VoiceType Dictation system. It really works!

I have received a great deal of support from my friends and relatives back at "the home front", in Kalamazoo, MI. Thanks to all my friends, especially Fred James and Mario Pena, for tolerating my occasional disjointed descriptions of my research. Thanks to Tom Lawson, the first true cognitive scientist I ever met, for remaining a good friend and colleague. Special thanks to my mother and my father, for everything that they've done for me.

The person who had the largest impact upon my work in this dissertation is my adviser, Paul Smolensky. I owe him tremendous thanks for many things, despite the fact that he deviously lured me into linguistics after I had strategically decided to avoid language as a research area. Thanks for everything from being a truly wonderful adviser to doing most of the typing on the papers we co-authored. Thanks also for being a good friend.

My final and deepest thanks go to Esther Tesar, for all of her love and support, and for agreeing to marry me.

Contents

Chapter 1. Introduction

In Optimality Theory (Prince and Smolensky 1993), Universal Grammar provides a set of highly general universal constraints which apply in parallel to assess the well-formedness of possible structural descriptions of linguistic inputs. Grammar is described in terms of an optimization function: the possible structural description of an input that optimally satisfies the constraints is the grammatical one. Evidence in favor of this characterization of Universal Grammar is provided elsewhere. Much of the work to date addresses phonology: see Prince and Smolensky (1993) and the several dozen works cited therein, notably McCarthy and Prince (1993a). Work addressing syntax includes Grimshaw (1993) and Legendre, Raymond and Smolensky (1993), as well as the papers presented at the 1995 MIT Workshop on Optimality in Syntax.

The grammars characterized by Optimality Theory are functions mapping inputs to outputs; they are not directly concerned with computation. However, one can ask interesting computational questions about Optimality Theoretic grammars. There are two major problems of computational linguistics. The first concerns the processing of linguistic forms. The second concerns the learning of grammars for specific languages. This dissertation examines how these two problem areas are characterized within Optimality Theory, and will provide answers to some specific computational questions.

1.1 Optimality Theory

This section presents a self-contained presentation of the relevant principles of Optimality Theory. These principles are consistently exemplified using the Basic CV Syllable Theory of Prince and Smolensky (1993, §6), which is developed concurrently.

1.1.1 Constraints and Their Violation

(1) Grammars are functions.

A grammar is a specification of a function which assigns to each input a unique structural description or *output*. (A grammar does not provide an algorithm for computing this function, e.g., by sequential derivation.)

In the Basic CV Syllable Theory, an input is a string of Cs and Vs, that is, a member of $\{C,V\}^+$, e.g., /VCVC/. An output is a parse of the string into syllables, which is notated as follows:

(2)	a.	.V.CVC.	= $[_\sigma$ V] $[_\sigma$ CVC]
	b.	⟨V⟩.CV.⟨C⟩	= V $[_\sigma$ CV] C
	c.	⟨V⟩.CV.C□́.	= V $[_\sigma$ CV] $[_\sigma$ C□́]
	d.	.□V.CV.⟨C⟩	= $[_\sigma$ □V] $[_\sigma$ CV] C

A syllable consists of a minimum of one and a maximum of three positions. The nucleus position is mandatory, and may be filled by a vowel (V). The onset position, which appears before the nucleus, and the coda position, which appears after the nucleus, are optional, and may be filled by consonants (C).

What (1) asserts is that the grammar assigns to the input /VCVC/ one structural description, such as one of the possibilities listed under (2). Which one gets assigned is determined by subsequent principles. The possibilities shown depict phenomena referred to as underparsing and overparsing. Underparsing is represented in (2b) by the first V and the final C, both of which are not parsed into syllable positions. They are present in the overall structural description, but will not be pronounced in the surface form. Overparsing is represented in (2d) by the unfilled onset position in the first syllable (denoted by □; an unfilled nucleus position is denoted □́). This position does not have an input segment parsed into it, but will be filled by phonetic interpretation, and so will be realized in the surface form (this phenomenon is often referred to as epenthesis).

(3) A linguistic input is mapped to a set of possible outputs by *GEN*.

Universal Grammar provides the set of possible inputs, the set of possible outputs, and a function *GEN* which, given any input *I*, generates the set of candidate outputs *GEN*(*I*), for *I*. The input *I* is a substructure contained within each of its candidate outputs in *GEN*(*I*).

In the CV case, for any input *I*, the candidate outputs in *GEN*(*I*) consist in all possible parses of the string into syllables, including the possible over- and underparsing structures exemplified above in (2). The structural descriptions generated by *GEN* are strings of syllables with the following restrictions: nuclei are mandatory, onsets and codas are optional, and positions are assumed to contain at most one input segment. The order of the input segments must be preserved, and each input segment must either be placed in a syllabic position or marked as unparsed in the structural description. Further, a C may only be parsed as an onset or a coda, while V may only be parsed as a nucleus. Notice that this is a statement of the universal set of structural descriptions to be considered, and not the inventory of optimal forms for any particular language.

(4) Well-formedness is assessed on the basis of universal constraints.

Universal Grammar provides a set of universal constraints which assess the well-formedness of candidate outputs for a given input in parallel (i.e., simultaneously). Given a candidate output, each constraint assesses a set of MARKS each of which corresponds to one violation of the constraint; the collection of all marks assessed a candidate c is denoted MARKS(c).

The Basic CV Syllable Structure constraints dealt with here are as follows:

(5)	ONS	Syllables must have onsets.
	NOCODA	Syllables must not have codas.
	PARSE	Input segments must be parsed (into syllabic positions).
	FILL^{Nuc}	A nucleus position must be filled (with a V).
	FILL^{Ons}	An onset position must be filled (with a C).

1.1.2 Optimality and Harmonic Ordering

The central notion of optimality now makes its appearance. The idea is that by examining the marks assigned by the universal constraints to all the candidate outputs for a given input, there is one which is least marked, or optimal: this is the one and only well-formed description that may be assigned to the input by the grammar. The relevant notion of 'least marked' is not the simplistic one of just counting numbers of violations. Rather, in a given language, different constraints have different strengths or priorities: they are not all equal in force. When a choice must be made between satisfying one constraint or another, the stronger must take priority. The result is that the weaker will be violated in a well-formed description.

(6) The relative importance of constraints is determined by their ranking in a strict dominance hierarchy.

The grammar of each language *ranks* the universal constraints in a *dominance hierarchy*; when constraint $\mathbb{C}_1$ dominates another $\mathbb{C}_2$ in the hierarchy, it is denoted $\mathbb{C}_1 \gg \mathbb{C}_2$ The ranking is total: the hierarchy determines the relative dominance of every pair of constraints:

$$\mathbb{C}_1 \gg \mathbb{C}_2 \gg \cdots \gg \mathbb{C}_n$$

A grammar's constraint hierarchy induces on all the candidate outputs a *Harmonic ordering* as follows. Let *a* and *b* be two candidate parses with sets of marks MARKS(*a*) and MARKS(*b*). To compare *a* and *b*, first cancel all the marks they have in common, getting lists of 'uncancelled' marks: MARKS′(*a*) and MARKS′(*b*); now, no mark occurring in one list occurs in the other. Then determine which list of uncancelled marks contains the worst mark: a mark assessed by the highest-ranking constraint violated in the two lists. This candidate is *less Harmonic* or *has lower Harmony* than the other; if it is *a*, then this relationship is denoted: $a \prec b$ (MARKS(*a*) $\prec$ MARKS(*b*) is also sometimes written). For a given input, the most Harmonic of the candidate outputs provided by *GEN* is the *optimal* candidate: it is the one assigned to the input by the grammar. Only this optimal candidate is well-formed; all less Harmonic candidates are ill-formed.

Harmonic ordering in the CV case can be illustrated with a specific grammar, L_1.

(7) Constraint Hierarchy for L_1: ONS $\gg$ NOCODA $\gg$ FILL^{Nuc} $\gg$ PARSE $\gg$ FILL^{Ons}

In the following *constraint tableau*, the candidates of (2) are shown, along with their constraint violations. The most dominant constraints in L_1 appear to the left:

Table 1

Constraint Tableau for $L_1 = \Sigma^{\mathrm{CV}}_{\mathrm{ep,del}}$

Candidates	**Ons**	**NoCoda**	**Fill$^{\mathrm{Nuc}}$**	**Parse**	**Fill$^{\mathrm{Ons}}$**
/VCVC/ →					
☞ *d.* .□V.CV.⟨C⟩				*	*
b. ⟨V⟩.CV.⟨C⟩				* *	
c. ⟨V⟩.CV.C□́.			*	*	
a. .V.CVC.	*	*			

The candidates in this tableau have been listed in Harmonic order, from highest to lowest Harmony; the optimal candidate is marked manually (determining that this candidate is optimal requires demonstrating that it is more Harmonic than any of the infinitely many competing candidates). Starting at the bottom of the tableau, verify that $a \prec c$. The first step is to cancel common marks: here, there are none. The next step is to determine which candidate has the worst mark, i.e., violates the most highly ranked constraint: it is *a*, which violates ONS. Therefore *a* is the less Harmonic. In determining that $c \prec b$, first cancel the common mark *PARSE; *c* then earns the worst remaining mark of the two, *FILL$^{\mathrm{Nuc}}$. The importance of retaining multiple marks can be seen in the comparison of *b* to *d*: one *PARSE mark cancels, leaving MARKS′(*b*) = {*PARSE} and MARKS′(*d*) = {*FILL$^{\mathrm{Ons}}$}. The worst mark is the uncancelled *PARSE incurred by *b*: so $b \prec d$.

The final central principle of Optimality Theory is:

(8) Language typology is explained by re-ranking the universal constraints.

Cross-linguistic variation is principally due to variation in language-specific rankings of the universal constraints. Analysis of the optimal forms arising from all possible rankings of the constraints provided by Universal Grammar gives the typology of possible human languages. Universal Grammar may impose restrictions on the possible rankings of its constraints.

If in L_1's constraint hierarchy we exchange the two FILL constraints, we get L_2:

(9) Constraint Hierarchy for L_2: ONS ≫ NOCODA ≫ FILL$^{\text{Ons}}$ ≫ PARSE ≫ FILL$^{\text{Nuc}}$

Now the tableau corresponding to Table 1 becomes Table 2; the columns have been re-ordered to reflect the constraint re-ranking, and the candidates have been re-ordered to reflect the new Harmonic ordering:

Table 2

Constraint Tableau for $L_2 = \Sigma^{\text{CV}}{}_{\text{del,ep}}$

Candidates	ONS	NOCODA	FILL$^{\text{Ons}}$	PARSE	FILL$^{\text{Nuc}}$
/VCVC/ →					
☞ *c.* 〈V〉.CV.C□́.				*	*
b. 〈V〉.CV.〈C〉				* *	
d. .□V.CV.〈C〉			*	*	
a. .V.CVC.	*	*			

Like L_1, all syllables in L_2 are CV; /VCVC/ gets syllabified differently, however. In L_2, underparsing is used to avoid onsetless syllables, and overparsing to avoid codas (L_2 is Prince and Smolensky's (1993) language $\Sigma^{\text{CV}}{}_{\text{del,ep}}$).

In the CV theory, Universal Grammar imposes no restrictions on the ranking of the constraints I have discussed here (5). Analysis of all possible rankings of these constraints reveals that the resulting typology of basic CV syllable structures is an instantiation of Jakobson's typological generalizations (Jakobson 1962, Clements & Keyser 1983): see Prince and Smolensky (1993, §6). In this typology, syllable structures may have required or optional onsets, and, independently, forbidden or optional codas.

1.2 Computational Questions in Optimality Theory

Central to language processing is the assigning of structural descriptions to linguistic material via algorithms. Within the Optimality Theory framework, the most obvious such mapping is the assigning of a structural description to a linguistic input. The corresponding computational problem is the problem of computing the optimal description, or parse, of an input. I will call this the parsing problem in Optimality Theory.

(10) The parsing problem: computing the optimal parse of an input.

The exact correspondence of the competence input/output mapping to the performance processes of comprehension and production remains an open issue, although the input/output mapping is more naturally understood as a mapping from an underlying form to a description including the surface phonetic form, and thus as corresponding to production.

The computability of optimal descriptions has been of real concern to people working in Optimality Theory, because a grammar is described in terms of the simultaneous generation, evaluation, and comparison of an infinite number of candidates. This description can make the problem appear intractable: how could an algorithm efficiently compute the optimal description if it must generate and evaluate an infinite number of candidate descriptions? This dissertation will show that it is not necessary to generate and evaluate an infinite number of candidates in order to compute the optimal description. Provably correct algorithms will be given which efficiently compute the optimal description for grammars satisfying a few basic conditions.

Cross-linguistic variation is explained in Optimality Theory through the rankings of the universal constraints. Therefore, an important part of language learning in Optimality Theory is learning the correct ranking of the universal constraints for a given language, from positive data. I will call this the learnability problem in Optimality Theory.

(11) The learnability problem: learning the correct constraint ranking from positive data.

One challenging aspect of this problem in Optimality Theory is imposed by the use of violable constraints. Given a grammatical description, a learner might observe that it violates some of the universal constraints. But if grammatical descriptions are allowed to violate constraints, how can anything be learned from those observations? There is also a combinatorial concern. The number of distinct total rankings is a factorial function of the number of constraints: 10 constraints have 10! = 3,628,800 distinct rankings. If the amount of data required to learn the correct ranking scales as the number of possible rankings, then a grammar with many constraints could require a prohibitively large amount of data to be learned successfully. This dissertation will show that a significant subproblem of the learnability problem, that of inferring rankings from hypothesized structural descriptions, has an efficient solution. Provably correct algorithms will be given which compute the correct ranking from positive data in the form of structural descriptions, and require an amount of data on the order of only the square of the number of constraints.

To be precise about the goals of this work: it is not claimed here that any of these algorithms are psychologically "plausible", in the sense of being detailed models of human processing. Developing plausible psychological models of language processing and acquisition based upon Optimality Theory proposals is a more ambitious project. This work is more preliminary. This work addresses the very serious doubts that these problems can be solved by any algorithms whatsoever. Nevertheless, I sincerely hope that some of the principles and insights underlying these algorithms will eventually contribute to psychological models.

1.3 Overview

Chapter 2 and Chapter 3 address the parsing problem in Optimality Theory. Chapter 2 discusses parsing when the linguistic representational structures may be described by a regular formal grammar; the Basic CV Syllable Theory is such as case, and is used as an example. Chapter 3 discusses parsing when the linguistic representational structures are described by context-free formal grammars. The parsing algorithm for the context-free case is a direct extension of the algorithm for the regular case. Chapter 4 addresses the learnability problem in Optimality Theory. Algorithms are there presented which apply to any linguistic grammar within the framework of Optimality Theory. Chapter 5 briefly summarizes the main results, and discusses some possible directions for future research.

Chapter 2. Parsing: Regular Structures

In Optimality Theory, grammaticality is defined in terms of optimization over a large (often infinite) space of candidates. This raises the question of how grammatical forms might be computed. This chapter and the next examine this question in detail. It will be shown that parsing in Optimality Theory is tractable, given reasonable restrictions on the universal constraints and on *GEN*. An algorithm is described which computes the optimal parse of the input in time that is linear in the length of the input. As an illustration, the Basic CV Syllable Theory is discussed and a complete algorithm is detailed for grammars defined within that theory. The main ideas of the parsing algorithms will be presented in the course of presenting that example. A later section will explain how to create a complete parser for any grammar with regular representational structures.

2.1 The Parsing Problem in Optimality Theory

Recall from chapter 1 the parsing problem, repeated here for convenience:

(10) The parsing problem: computing the optimal parse of an input.

Although Optimality Theory is easily understood mathematically in terms of the generation and evalutation of all candidates in parallel, it is unnecessary, and in fact counterproductive, to consider computing optimal forms in those terms. Intuitively, the algorithm presented in this chapter works by gradually constructing a few candidate parses as it works its way through the input. When the end of the input is reached, only a few complete parses have been constructed, one of which is guaranteed to be the optimal structural description (relative to the entire candidate set). This will be accomplished by making some assumptions about the grammar being parsed. This chapter discusses the case in which the core set of “skeletal structures” employed by the grammar may be described by a formal regular grammar (the notion of a core set of “skeletal structures” is formalized in section 2.2 by the concept of *position grammar*). The universal constraints of the grammar are assumed to be "local" in a sense to be made precise below. Under these assumptions, a grammar may be parsed in linear time. The next chapter will discuss the case in which the position grammar is a general context-free grammar, and will provide an algorithm for that case with a complexity that, while not linear, is still reasonable.[1]

2.1.1 The Technique: Dynamic Programming

The challenge is to efficiently choose the optimal structural description from an infinite set of candidates. The solution is to avoid dealing with whole structural descriptions, and instead build up the optimal one piece by piece. The basic technique used to do this is dynamic programming (see, e.g., Corman, Leiserson, & Rivest 1990). The algorithm presented here is related to chart parsing (see, e.g., Kay 1980), an algorithm used in natural language parsing that employs dynamic programming. Dynamic programming has also been used for optimization in sequence comparison (Sankoff & Kruskal 1983) and Hidden Markov models (see, e.g., Rabiner 1989). The algorithm presented here combines the use of dynamic programming for language structure processing with dynamic programming for optimization, resulting in optimization-based language processing.

2.1.2 The Main Idea: An Intuitive Explanation of Dynamic Programming

Due to the nature of the problem under consideration, the analysis presented in this paper will at times involve considerable formal complexity. With that in mind, the fundamental idea underlying the analysis, dynamic programming, is here introduced via an intuitive analogy. Suppose that there are two towns, X and Y. In between these towns is a river, which must be crossed in order to travel from X to Y. There are three bridges across the river: A, B, and C. Suppose that we wish to find the shortest - the optimal - route from X to Y.

We know that any path between X and Y must cross one of the three bridges. There are many different ways to get from Town X to each of the three bridges, and many different ways to get from each of the bridges to Town Y. However, we can simplify our problem by first only considering the best way

[1]Ellison (1994) has independently developed some work on computing optimal forms in Optimality Theory that is similar in principle to part of the work in this chapter, although expressed in the formalism of finite state automata. Among the additional ideas provided in this chapter are independent characterizations of the formal description of the grammar, the input and the parser, as well as a method for creating a parser from a description of the grammar.

to get from X to A, the best way from X to B, and the best way from X to C. Having found each of these "sub-routes", we could make a small chart for future reference: it would have three entries, each giving the route and the distance of the route to one of the bridges. Next, we could consider the best way to get to Y from each of the three bridges. Once we determine the shortest route from bridge A to town Y, we can easily calculate the shortest route from X to Y which crosses bridge A, by adding the distance of the shortest route from A to Y with the chart entry giving the distance from X to A. In the same fashion, we can calculate the shortest route from X to Y crossing B, by combining the shortest route from B to Y and using the already calculated shortest route from X to B. The same can be done for bridge C. At this point, we need only choose the shortest of three routes: the shortest route of those for each of the three bridges.

Notice that there are many possible routes between X and Y: just considering bridge A, every possible route from X to A may be combined with every possible route from A to Y. In fact, the problem is best understood in that fashion, as the problem of searching the space of all possible routes between X and Y to find the shortest one. But while the problem is most easily stated and understood in those terms, it is not most easily solved in those terms. The above illustration gives the essence of dynamic programming: break a large problem, like traveling from X to Y, into smaller sub-problems, like traveling from X to A, and traveling from A to Y.

The value of this way of thinking is perhaps even more apparent if we change the problem so that there are two rivers between X and Y: the second river having three bridges, D, E, and F. In this case, we would first put into our chart the shortest route from X to the bridges A, B, and C. Next, for bridge D, we would consider the shortest route from each of the bridges A, B, and C. We would then make another chart entry giving the shortest route from town X to bridge D: this will be the shortest of three routes, the shortest route from X to D via bridge A, via bridge B and via bridge C. Next, similar short entries would be written down for bridges E and F. Finally, we could calculate the shortest route from town X to town Y by considering the shortest route via bridge D, via E and via F. Again, at the end, we need only compare three complete routes between X and Y.

The algorithm presented in this paper will use dynamic programming to compute optimal forms. Each segment of the input is something like a river in the above illustration. There are a limited number of ways to deal with one input segment, and the best way to do each can be recorded in a table. Once all of the input segments have been considered in order, only a very few entire parses of the input need be compared in order to determine the optimal one.

2.2 A Formal Characterization of The Basic CV Syllable Theory

Recall the Basic CV Syllable Theory, as laid out in Chapter 1. An input to the grammar is a sequence of segments categorized as consonants and vowels, that is, a member of $\{C,V\}^+$. The structural descriptions generated by *GEN* are strings of syllables with the following restrictions: nuclei are mandatory, onsets and codas are optional, and positions are assumed to contain at most one input segment. The order of the input segments must be preserved, and each input segment must either be placed in a syllabic position or marked as unparsed in the structure. Further, a C may only be parsed as an onset or a coda, while V may only be parsed as a nucleus. Notice that this is a statement of the universal set of structural descriptions to be considered, and not the inventory for any particular language. For a given input, *GEN* generates all possible syllable structures that contain the input, and meet the restrictions just given[1]. The constraints of the theory are given in (5) (repeated here for convenience):

(5)	ONS	Syllables must have onsets.
	NOCODA	Syllables must not have codas.
	PARSE	Input segments must be parsed (into syllabic positions).
	FILL^{Nuc}	A nucleus position must be filled (with a V).
	FILL^{Ons}	An onset position must be filled (with a C).

These constraints are violable and may be ranked differently by different languages.

[1]Strictly speaking, Prince and Smolensky (1993) describe these restrictions by universally fixing the constraints NUC, *COMPLEX, *M/V, and *P/C at the top of the hierarchy. This insures that they are unviolated in optimal forms, so I here treat them as part of *GEN*.

Prince and Smolensky (1993) implicitly describe the space of possible structural descriptions. Immediately below, I give a formal description of this space. This description is used when explaining the parser for the CV Theory.

For computational purposes, I will regard a structural description of an input as a string of *syllabic positions*, referred to as a *position structure*, which are matched with the input segments. The positions are represented by the symbols {o,n,d}, for onset, nucleus, and coda, respectively ('C' is reserved for consonant). In a given structural description, each position may be filled with at most one input segment, and each input segment may be parsed into at most one position. Any input segment not parsed into a syllabic position is so marked in the structural description. For a given position structure, each allowable way of matching the input with the structure counts as a candidate structural description. An allowable matching is one in which the order of the input segments is preserved, and in which V segments are only parsed into n positions, while C segments are only parsed into o and d positions.

Figure 1 shows some examples of candidate parses for the input /VC/ as expressed in these terms.

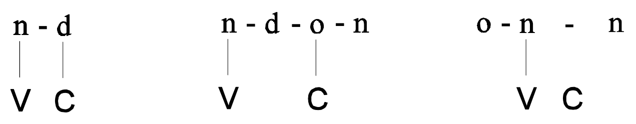

Figure 1 Parses for /VC/ expressed with position strings.

The lower case letters are syllable positions. Syllable positions with vertical bars under them are filled by the input segments immediately under the vertical bars. Any syllable position without a vertical bar underneath is unfilled in that parse. An input segment (V or C) which is not underneath a vertical bar is unparsed.

I will use the following *position grammar* to describe the set of allowable position structures:

(12) $S \Rightarrow e \mid oO \mid nN$

$O \Rightarrow nN$

$N \Rightarrow e \mid dD \mid oO \mid nN$

$D \Rightarrow e \mid oO \mid nN$

The terminals in the position grammar are the syllabic positions and the empty string (e). The non-terminals {S, O, N, D} may be thought of as corresponding to states in the derivation of a position structure. S is the starting state. O signifies that the last position generated was an onset (o), N that a nucleus (n) was just generated, and D a coda (d). Those non-terminals which may evaluate to e correspond to possible finishing states. O is not a finishing state, because a syllable with an onset must also have a nucleus. This position grammar guarantees that each syllable has a nucleus, that onsets precede nuclei, that codas follow nuclei, and that there is at most one of each type of position per syllable.

It should here be emphasized that the position grammar just discussed is a descriptive formalism useful in understanding *GEN*; it is NOT a computational mechanism. The actual computational mechanism understandable in terms of the position grammar is the set of operations contained in the Operations Set, described below.

2.3 Parsing the CV Syllable Theory

The algorithm proceeds by creating a table, called the Dynamic Programming Table, and filling in the cells of the table. Once all of the cells have been filled, the optimal form is quite easily determined. Section 2.3.1 describes the table and explains how it contributes to computing the optimal form. Section 2.3.2 describes the operations used to fill the cells of the table, discussing both how they relate to the table and how they relate to CV Syllable Theory.

2.3.1 The Dynamic Programming Table

Table 3 shows the Dynamic Programming Table for the input /VC/, with the constraint ranking ONS $\gg$ NOCODA $\gg$ FILL^{Nuc} $\gg$ PARSE $\gg$ FILL^{Ons}.

Table 3

Dynamic Programming Table for /VC/

	BOI	i_1 = V	i_2 = C
S		〈V〉	〈VC〉
O	□	.□V.□	.□V.C
N	□□́	□V	.□V.〈C〉
D	.□□́□.	.□V□.	.□VC.

Optimal Parse: .□V.〈C〉 This parse is represented in cell [N,i_2].

Each cell in this table contains a structure. Each column of this table stands for a segment of the input except the first column, BOI, which corresponds to the "beginning of the input". Notice that each cell in the column headed i_1 contains a V; further, every structure in the column headed i_2 contains both a V and a C, in the correct order. The label on each row is a non-terminal of the position grammar, and corresponds to a type of syllable position. Notice that for each structure in the N row, the last-generated position in the structure is a nucleus. The O row contains structures ending in an onset, while the D row contains structures ending in a coda. The S row only contains structures in which no positions at all have been generated (i.e., all of the input segments seen are unparsed).

Each cell contains a structure representing the best way of parsing the input up through the segment for that column. The last column (the column for the last input segment) includes the complete parses to be considered. The optimal parse is easily chosen from among this set of possibilities.

In general, a given input string I = $i_1 i_2 ... i_J$ is parsed by constructing a dynamic programming table. The table has one column for each segment of the input, plus a first column, BOI. The BOI column is present because positions may be generated at the beginning, before any of the input has been examined (this would correspond to epenthesis at the beginning of the utterance). Each cell corresponds to a *partial description*, which is a structural description of part of the input. Ultimately, each cell contains the optimal way of parsing the input up through the segment heading the column, with the last syllable position in the structure matching the row label. The table cell $[N,i_2]$ corresponds to the optimal way of parsing up through the second segment of the input, with a nucleus being the last structural position in the partial description. Each cell also contains the constraint violation marks assessed the partial description, and representing the Harmony of that description (these marks are not depicted in Table 3).

The parsing algorithm proceeds by filling in the columns of the table one at a time, left to right. After the best way of parsing the input through segment i_{j-1} ending in each non-terminal has been calculated (the entries of column i_{j-1}), those values are then used to determine the best way (for each possible final position) of parsing the input through segment i_j. Once all the values for the last column are determined, the Harmony values in the table cells of the last column in rows corresponding to possible finishing states are compared (this is explained in greater detail below). The cell containing the highest Harmony value thus also contains the optimal parse of the input.

2.3.2 The Operations Set

Operations are used to fill cells in the Dynamic Programming (DP) Table. An operation works by taking the partial description in a previously filled cell, adding an element of structure to it, and putting the new description in the new cell. A cell entry is determined by considering all of the operations that might fill the cell, and selecting the one with the highest resulting Harmony to actually fill the cell. This is the essence of dynamic programming: because the partial descriptions in later cells contain the partial descriptions listed in earlier cells, the earlier cell entries may be used directly, rather than explicitly recalculating all of the possibilities for later cells.

Each operation is based upon one of three primitive actions. The three primitive actions are:

(13) (a) parsing a segment of input into a syllabic position;

(b) underparsing an input segment;

(c) overparsing a syllabic position.

Primitive actions (a) and (c) involve generating positions, so they must be coordinated with productions in the position grammar of *GEN*; (b) does not involve position generation. On the other hand, actions (a) and (b) consume input, while (c) does not. Operations are versions of the primitive actions coordinated with the specifics of the model (*GEN* and the universal constraints). An operation may be specified by four things: the new cell (being filled), the previous cell containing the description being added to, the structure added to the partial description, and the constraint violation marks incurred by the operation. A candidate structural description of an input may thus be viewed as resulting from a sequence of operations. It should be emphasized that an operation does not transform one entire structural description into another, but merely adds to a partial description.

As an example, consider the actions that might fill cell [O,i_2] of the DP Table. Recall that the structure in this cell must contain input segments i_1 and i_2, and the last syllabic position in the structure must be an onset. One possibility is the underparsing action: take the structure from the cell immediately to the left in the same row, [O,i_1], and add to it the input segment i_2 marked as underparsed. We don't need to consider any other ways of filling this cell with i_2 underparsed, because we have already guaranteed that [O,i_1] contains the best way of parsing through i_1 ending in an onset. The resulting Harmony of the operation will be the Harmony listed in [O,i_1], with the mark {*PARSE} added to it (indicating the constraint violated by underparsing). If i_2 is a consonant, then another possibility is to parse i_2 into a newly generated onset position. This requires having a structure from the previous column to which an onset position may be legally appended. The position grammar shows that an onset position may be generated directly from the non-terminals S, N, and D; this corresponds to the intuitive notions that an onset must be at the beginning of a syllable, and may be the first position of a description (generated from S), may immediately follow a nucleus position (generated from N), or may immediately follow a coda position (generated from D). An onset position may not immediately follow another onset position, because then the first onset belongs to a syllable with no nucleus. Fortunately, we have already determined that the cells [S,i_1], [N,i_1], and [D,i_1] do contain the optimal partial descriptions for these three cases. Finally, the cell [O,i_2] may be filled by an overparsing operation that would take a structure which already contains i_2 and append an unfilled onset position.

The set of possible operations is called the Operations Set, and is organized to indicate what operations may fill each type of cell (the cells are here typed by row). Table 4 shows the Operations Set for the CV syllable theory. Each row in the table corresponds to an operation. The new cell column shows the type of cell to be filled by the operation. The condition column contains any additional conditions that must be met in order for the operation to apply (in this case, the restriction of V to nuclei, etc.). The previous cell column indicates the relative position of the cell containing the partial description being added to by the operation. The structure column indicates the additional structure added by the operation. The violations column shows the constraint violation marks incurred by the added structure. The final two columns are informational: the production column lists the position grammar production used by the operation if one is used, and the operation type column indicates the type of operation.

Table 4

The Operations Set

					Information	
New Cell	Condition	Previous Cell	Struc	Violations	Production	Operation Type
[S,i_j]		[S,i_{j-1}]	⟨i_j⟩	{*PARSE}		Underparsing
[O,i_j]		[O,i_{j-1}]	⟨i_j⟩	{*PARSE}		Underparsing
[O,i_j]	IF i_j=C	[S,i_{j-1}]	o/i_j	{}	S ⇒ oO	Parsing
[O,i_j]	IF i_j=C	[N,i_{j-1}]	o/i_j	{}	N ⇒ oO	Parsing
[O,i_j]	IF i_j=C	[D,i_{j-1}]	o/i_j	{}	D ⇒ oO	Parsing
[O,i_j]		[S,i_j]	o/□	{*FILL^{Ons}}	S ⇒ oO	Overparsing
[O,i_j]		[N,i_j]	o/□	{*FILL^{Ons}}	N ⇒ oO	Overparsing
[O,i_j]		[D,i_j]	o/□	{*FILL^{Ons}}	D ⇒ oO	Overparsing
[N,i_j]		[N,i_{j-1}]	⟨i_j⟩	{*PARSE}		Underparsing
[N,i_j]	IF i_j=V	[S,i_{j-1}]	n/i_j	{*ONS}	S ⇒ nN	Parsing
[N,i_j]	IF i_j=V	[O,i_{j-1}]	n/i_j	{}	O ⇒ nN	Parsing
[N,i_j]	IF i_j=V	[N,i_{j-1}]	n/i_j	{*ONS}	N ⇒ nN	Parsing
[N,i_j]	IF i_j=V	[D,i_{j-1}]	n/i_j	{*ONS}	D ⇒ nN	Parsing
[N,i_j]		[S,i_j]	n/□́	{*ONS *FILL^{Nuc}}	S ⇒ nN	Overparsing
[N,i_j]		[O,i_j]	n/□́	{*FILL^{Nuc}}	O ⇒ nN	Overparsing
[N,i_j]		[N,i_j]	n/□́	{*ONS *FILL^{Nuc}}	N ⇒ nN	Overparsing
[N,i_j]		[D,i_j]	n/□́	{*ONS *FILL^{Nuc}}	D ⇒ nN	Overparsing
[D,i_j]		[D,i_{j-1}]	⟨i_j⟩	{*PARSE}		Underparsing
[D,i_j]	IF i_j=C	[N,i_{j-1}]	d/i_j	{*NOCODA}	N ⇒ dD	Parsing
[D,i_j]		[N,i_j]	d/□	{*NOCODA}	N ⇒ dD	Overparsing

The Operations Set relates to the DP Table as follows. The Operations Set gives all of the possible operations that may fill a given cell in the Dynamic Programming Table. Each of the possible operations "competes" to fill in the cell. The product of each operation is a partial structure consisting of two things: the partial structure contained in the operation's previous cell with the operation's additional structure appended to it, and the Harmony of the new partial structure, which consists of the list of marks in the operation's previous cell with the marks incurred by the operation added to it. The operation producing the most harmonic partial description (that is, the one whose resulting list of marks is least offensive with respect to the constraint ranking of the grammar) actually gets to fill the cell. Told from the point of view of the algorithm, examine each of the operations which can fill the current cell, select the one which produces the most harmonic partial structure, and place that operation's partial structure and list of marks into the current cell.

The cell [S,BOI] is the starting cell: no input has yet been examined, and no positions have been generated. So, [S,BOI] has a Harmony value of no constraint violations in it. The other cells in the BOI column may be filled from there by overparsing operations. The cells in the BOI column may only be filled by overparsing operations, as there are no input segments for other operations to work with.

One crucial aspect has not yet been explained about the application of this formula. The Parsing and Underparsing operations have a previous state cell from the previous column in the DP Table, i_{j-1}. However, the Overparsing operations refer to other cells in the same column of the DP Table as the cell being filled. How, in general, can these cells be filled, if the value for each cell in the column depends upon the values in the other cells of the column? The answer involves some intricate details of the algorithm, and is given in the next section.

Notice that, in the Operations Table, the Parsing operations contain IF conditions. These are used to enforce constraints of the CV theory that consonants (C) may only fill onsets and codas, and vowels (V) only nuclei. These restrictions are assumed to be part of *GEN*, and so are included here as IFs.

2.3.3 Limiting Structure: Position Grammar Cycles

The overparsing operations consume no input, and so they map between cells within a single column. In principle, an unbounded number of such operations could apply, and in fact structures with arbitrary numbers of unfilled positions are contained in the output space of *GEN* (as formally defined). However, the algorithm need only explicitly consider a finite number of such operations within a column. The position grammar has four non-terminals. Therefore, at most three overparsing operations can take place consecutively without the repeating of a non-terminal. A set of consecutive overparsings that both begins and ends with the same non-terminal can be considered a *cycle*. An example of a cycle of overparsings is an entire epenthesized syllable. The FILL constraints serve to penalize overparsings by penalizing any structural positions unfilled by input segments.

This fact is not specific to the Basic Syllable Theory. For any theory within Optimality Theory, the constraints must ban cycles of overparsings in order for the optimal value to be well-defined. If the constraints make a description containing such a cycle more Harmonic than a description differing only by the removal of that cycle, then there is no optimal value, because one could always increase the Harmony by adding more such cycles of overparsings. If such cycles have no Harmony consequences, then there will be an infinite number of optimal descriptions, as any optimal description can have more cycles of overparsings added. Thus, for the optimization with respect to the constraints to be well-defined and reasonable, the constraints must strictly penalize overparsing cycles. The number of non-terminals in the position grammar bounds the number of consecutive overparsings that may occur without having a cycle.

Operations are properly applied to the Dynamic Programming Table by first filling in all cells of a column considering only underparsing and parsing operations (which only use values from the previous column). Three passes are then made through the column cells, considering the overparsing operations: if the resulting Harmony of an overparsing operation into a cell from another cell in the same column is higher than the Harmony already listed in the cell, replace the Harmony in the cell with that resulting from the considered overparsing operation.

The ban on overparsing cycles is the crucial observation that allows the algorithm to complete the search in a finite amount of time; although the space of structural descriptions to be searched is infinite, there is a provably correct (input-dependent) bound on the space of descriptions that actually need to be considered.

2.3.4 Selecting the Optimal Parse

Once the entire table has been completed, the optimal parse may be selected. In the position grammar, certain non-terminals may evaluate to the empty string. This means that they can be the last non-terminal in a derivation, and therefore that the syllable position to which each corresponds is a valid end of syllable position. Therefore, the cells in the final column, in rows corresponding to these non-terminals, contain valid complete parses of the input. For the Basic Syllable Theory, the non-terminals are N and D, signifying that a syllable may end in a nucleus or a coda, and S, for the null parse. These three entries are compared, and the entry with the highest Harmony is selected as the optimal parse of the input.

2.3.5 Overview of the Parser

NOTE: OP(i_j) stands for the result (structure and marks) of applying operation OP for column i_j.

```
Set [S,BOI] to no structure and no violation marks
Fill each other cell in column BOI with the best overparsing operation that currently applies
Repeat until no cell entries change
    For each row X in BOI
        For each overparsing operation OP for X
            If Harmony(OP(BOI)) > Harmony([X,BOI]), set [X,BOI] to OP(BOI)
```

```
For each column $i_j$, proceeding from left to right
    For each row X
        Fill $[X,i_j]$ with the result of the underparsing operation for X
        For each parsing operation OP for X
            If Harmony $(OP(i_j)) >$ Harmony$([X,i_j])$, set $[X,i_j]$ to $OP(i_j)$
    Repeat until no cell entries change
        For each row X
            For each overparsing operation OP for X
                If Harmony $(OP(i_j)) >$ Harmony$([X,i_j])$, set $[X,i_j]$ to $OP(i_j)$
Select from the final column the most Harmonic of the entries in rows S, N, and D
```

2.3.6 A Sample Parse

Table 5 is an example of the filled Dynamic Programming Table for the input /VC/, with the constraint ranking ONS $\gg$ NOCODA $\gg$ FILL^{Nuc} $\gg$ PARSE $\gg$ FILL^{Ons}.

Table 5

The Completed Dynamic Programming Table for /VC/.

	BOI	i_1 = "V"	i_2 = "C"	
S	START	under from:[S,BOI] *PARSE ⟨V⟩	under from:[S,i_1] *PARSE *PARSE ⟨VC⟩	
O	over from:[S,BOI] *FILL^{Ons} .□	over from:[N,i_1] *FILL^{Ons} *FILL^{Ons} .□V.□	parse from:[N,i_1] *FILL^{Ons} .□V.C	
N	over from:[O,BOI] *FILL^{Ons} *FILL^{Nuc} .□□́	parse from:[O,BOI] *FILL^{Ons} .□V	under from:[N,i_1] *FILL^{Ons} *PARSE .□V.⟨C⟩	☞
D	over from:[N,BOI] *FILL^{Ons} *FILL^{Nuc} *NOCODA .□□́□.	over from:[N,i_1] *FILL^{Ons} *NOCODA .□V□.	parse from:[N,i_1] *FILL^{Ons} *NOCODA .□VC.	

Optimal Parse: .□V.⟨C⟩ This parse is represented in cell [N,i_2].

The top line of each cell contains on the left an indication of the type of operation that filled the cell, and on the right (after the 'from:' label) the row and column designation of the previous cell (the already-filled cell whose structure was added onto by the operation to fill the current cell). The abbreviations indicate the kind of operation that filled the cell: 'over' for overparsing, 'under' for underparsing, and 'parse' for parsing. The constraint violation marks assessed the partial description are given on the middle line of each cell, and the bottom of each cell shows the partial description represented by that cell. The cell containing the optimal parse is indicated manually, and the cells which constitute the steps in the construction of the optimal parse are double-lined.

Parsing begins by filling the cells of the first column. The first cell, [S,BOI], is automatically filled with no structure, which incurs no constraint violations. Next, the cell [O,BOI] is filled. For this, the Operations Set is consulted. The Operations Set lists seven operations that can fill a cell in the O row. However, the underparsing and parsing operations do not apply here because they make reference to entries in an earlier column, which does not exist here. Of the three overparsing operations, two require entries in cells not yet filled: [N,BOI] and [D,BOI]. The remaining operation uses the entry in [S,BOI] as the previous cell and adds an unfilled onset position. This structure is placed in the cell, and the incurred mark listed in the operation. Next, the cell [N,BOI] is filled. Of the nine operations listed for a cell in the nucleus row, two may be considered here. The first is for previous cell [S,BOI], and results in violations of ONS and FILL^{Nuc}. The second is for previous cell [O,BOI], and results in violations of FILL^{Ons} and FILL^{Nuc}. Because ONS $\gg$ FILL^{Ons}, the result of the first operation has lower Harmony than the result of the second; thus, the second operation gets to fill the cell. The cell [D,BOI] is filled similarly. That completes the first pass through the column for the overparsing operations. Next, a second pass is performed; now, for each cell, all of the overparsing operations may be considered, because each cell in the column contains an entry. However, no further overparsing operations change any of the cell entries, because none improve the Harmony of the entry.

Now, column i_1 must be filled. The cells are first filled via the underparsing and parsing operations. We will focus in detail on how cell [O,i_1] gets filled. First, the one underparsing operation fills the cell; this results in a structure which has an unfilled onset position, and in which the first input segment, i_1 = V, is left unparsed. Next, the three parsing operations are considered. But none apply, because the input segment is a V, and an onset position may only have a C parsed into it. The underparsing and parsing operations for the rest of the column are now performed. The results of the steps up to this point are shown in Table 6.

Table 6

The DP Table with the Underparsing and Parsing Operations Completed for Column i_1

	BOI	i_1 = V	i_2 = C
S	START	under [S,BOI] *PARSE ⟨V⟩	
O	over [S,BOI] *FILL^{Ons} .□	under [O,BOI] *FILL^{Ons} *PARSE .□⟨V⟩	
N	over [O,BOI] *FILL^{Ons} *FILL^{Nuc} .□□́	parse [O,BOI] *FILL^{Ons} .□V	
D	over [N,BOI] *FILL^{Ons} *FILL^{Nuc} *NOCODA .□□́□.	under [N,BOI] *FILL^{Ons} *FILL^{Nuc} *NOCODA *PARSE .□□́□.⟨V⟩	

Finally, we consider the overparsing operations that might fill the cell. It is important to remember that each of the cells in the column have already been filled on the basis of underparsing and parsing operations before any overparsing operations are considered. For [O,i_1], there are three overparsing operations to be considered. The first adds an unfilled onset to the structure in [S,i_1], yielding ⟨V⟩.□, which incurs marks *PARSE and *FILL^{Ons}. The second adds an unfilled onset to the structure in cell [N,i_1], giving .□V.□, which incurs marks *FILL^{Ons} and *FILL^{Ons}. The third one has previous cell [D,i_1], giving .□□́□.⟨V⟩.□, which incurs marks *FILL^{Ons}, *FILL^{Nuc}, *NOCODA, *NOCODA, and *FILL^{Ons}. Of the three, the second overparsing operation has the highest resulting Harmony. Importantly, it also has higher Harmony than the entry already in cell [O,i_1], because PARSE $\gg$ FILL^{Ons}. Therefore, the result of this overparsing operation replaces the earlier entry in the cell. Overparsing also replaces the entry in [D,i_1].

On the next pass through the column, no cell entries are replaced by further overparsing operations, so the column is complete.

Once all of the columns have been filled, the optimal parse may be selected. The final candidates are the structures in the cells in the final column, and in rows S, N, and D. These rows are considered because these are the non-terminals that may evaluate to the empty string e in the position grammar. In the complete table above, the optimal parse is in cell [N,i_2].

2.3.7 Ties

It is possible for two different operations to tie for optimality when attempting to fill a cell. To illustrate, there are two ways to derive an essentially identical partial description: first insert and then delete, or first delete and then insert. In this case, the tie might be seen as a kind of anomaly, having no significance to the ultimate phonetic realization. However, if more than one truly different partial description for the same cell incurred identical marks, including all of them in the cell permits all of the optimal descriptions to be recovered from the table, if that cell should happen to figure in the set of descriptions ultimately found to be optimal.

2.4 Creating Parsers for Grammars

The previous section illustrated the parsing method with the Basic CV Syllable Theory. However, the parsing method is not limited to this theory or to phonology. It is a general method for parsing in Optimality Theory when the position grammar is regular. This section describes how to create a parser for any Optimality-Theoretic grammar satisfying the appropriate conditions. The next section provides a general definition and analysis of the parsing method.

For any given grammar with a regular position structure grammar, the Operations Set may be constructed as follows. First, for any cell [X,i_j] where X is a non-terminal (x is the corresponding structural position unless X is S), one allowable operation is to underparse the input segment. So, include the underparsing operation that takes the structure in [X,i_{j-1}] and adds an underparsed i_j to it. For each position grammar production with the non-terminal X on the right-hand side, two operations are possible: the generated position x has the next input segment parsed into it, or it is left unfilled. So, for each production Y⇒xX generating X, create two operations: a parsing operation which takes the structure in [Y,i_{j-1}] and appends a position x with i_j parsed into it, and an overparsing operation which takes the structure in [Y,i_j] and appends an unfilled position x. Add to each operation any conditions which restrict its application (such as the restriction of vowels to nucleus positions in the Basic Syllable Theory). Finally, each operation must be supplied with marks indicating the constraint violations incurred by its application.

The Dynamic Programming Table will have one row for each non-terminal in the position grammar. For any given input, it will have a first column, BOI, and one subsequent column for each input segment.

2.5 Formal Analysis

2.5.1 Formal Definition of Optimality Theory Grammars

(14) **Def.** The *position structure grammar* Γ is a formal regular grammar which generates *position structures.* Each terminal symbol of Γ is a structural position. Each production should be of one of two forms: X ⇒ yY, X ⇒ e. The start symbol is S.

(15) **Def.** A *unit* of structure is any of the following: (a) a single input segment marked as underparsed; (b) a single position filled with a input segment; (c) a single unfilled position.

(16) **Def.** Suppose there are an input I = i_1 ... i_J, and a position structure P = p_1 ... p_K. A *matching* μ is a string of units, where the set of possible matchings between P and I is *MATCH*(P,I), defined by the following recurrence:

$$MATCH(P,I) = M[p_K,i_J]$$

$$M[p_0,i_0] = \{the\ \ null\ \ structure\}$$

$$M[p_0,i_j] = \{s+\langle i_j\rangle \,|\, s\in M[p_0,i_{j-1}]\}$$

$$M[p_k,i_0] = \{s+p_k/\square \,|\, s\in M[p_{k-1},i_0]\}$$

$$M[p_k,i_j] = \{s+\langle i_j\rangle \,|\, s\in M[p_k,i_{j-1}]\} \cup \{s+p_k/i_j \,|\, s\in M[p_{k-1},i_{j-1}]\} \cup \{s+p_k/\square \,|\, s\in M[p_{k-1},i_j]\}$$

The '+' operator here means concatenation. The '/' operator here means to fill the position to the left of the operator with the material to the right of the operator. The set {the null structure} contains precisely one object, which consists of exactly no structure. Asserting the existence of this object at the base of the recurrence allows the rest of the recurrence to begin building descriptions by concatenating structure to the null structure.

This definition may be augmented by further restrictions for different specific instances of *GEN* (for example, forbidding consonants from being parsed into nucleus positions).

(17) **Def.** The set of candidate structural descriptions for input I is

$$GEN(I) = \bigcup_{P \in L(\Gamma)} MATCH(P,I)$$

L(Γ) is the set of all position structures generated by Γ.

(18) **Def.** The structural description(s) assigned to I by the grammar is $maxH\{GEN(I)\}$. The function maxH{} returns the structural description(s) with maximum Harmony. Implicit in the definition of maxH is a set of universal constraints and the ranking of those constraints.

What has just been given is a *definition* of the optimal form, in terms of position structures and matchings. Nothing has yet been said about *algorithms*.

2.5.2 The Computing Recurrence

(19) **Def.** The function *OPT*(I) is defined by the computing recurrence

$$\begin{aligned}
OPT(I) &= maxH\{\Omega[X^f, i_J] \mid X^f \Rightarrow e \in \Gamma\} \\
\Omega[S, i_0] &= \{the\ \ null\ \ structure\} \\
\Omega[S, i_j] &= \{\Omega[S, i_{j-1}] + \langle i_j \rangle\} \\
\Omega[X^k, i_0] &= maxH\{\Omega[X^m, i_0] + x^k/\square \mid X^m \Rightarrow x^k X^k \in \Gamma\} \\
\Omega[X^k, i_j] &= maxH\left\{\begin{array}{c}
\{\Omega[X^k, i_{j-1}] + \langle i_j \rangle\} \cup \\
\{\Omega[X^m, i_{j-1}] + x^k/i_j \mid X^m \Rightarrow x^k X^k \in \Gamma\} \cup \\
\{\Omega[X^m, i_j] + x^k/\square \mid X^m \Rightarrow x^k X^k \in \Gamma\}
\end{array}\right\}
\end{aligned}$$

where i_0 is the equivalent of BOI in the earlier description of the DP table. For notational convenience, from this point forward, '+' will be treated as a set operator, appending its second argument to each member of the set which is its first argument.

The computing recurrence bears a resemblance to the matching recurrence, but the two are quite different in important ways. The first index of the computing recurrence is not a specific piece of a specific position structure, but a non-terminal of Γ.

The computing recurrence also has a complexity not present in the matching recurrence. There is a circularity in the dependencies of each set of terms with an identical value for the second index (the input segment). Notice that each such set of terms corresponds precisely to a column of cells in the dynamic programming table. The value of each term is dependent upon the values of the other terms in the set. Thus, each set of terms is a set of interdependent equations. We want to determine conditions under which a solution to each set of equations exists and is efficiently computable.

(20) **Def.** A *partial description* π is a contiguous substring (of units) of a structural description μ (generated by *GEN*) which includes the first unit of μ. A partial description is said to have a last non-terminal, which corresponds to the last non-terminal in the derivation of the part of the position structure contained in the partial description.

The set of partial descriptions for an input includes all complete structural descriptions, as they are those partial descriptions where the contiguous substring is in fact the entire string of units making up the description. This observation is useful, as it is now possible to give one definition of Harmony for all partial descriptions, which includes the definition of Harmony for all complete structural descriptions.

(21) **Def.** A universal constraint is *local* if it may be evaluated solely on the basis of the internal configuration of a unit of structure, and the type of position which immediately precedes it.

(22) **Def.** The *Harmony* of a partial description is the collection of the constraint violation marks assessed each unit of the partial description.

(23) **Lemma** The value of each term $\Omega[X^k,i_j]$ of the computing recurrence, if one exists, is a set of partial descriptions containing input segments $i_1..i_j$ and having last non-terminal X^k.

Proof

This result is established through correspondence between the computing recurrence and the matching recurrence. Observe that the matching recurrence defines matchings by the recursive concatenation of units. Therefore, every string of units which is the value of some term of the matching recurrence for some position structure P is a partial description.

First, observe that both recurrences have the null structure as their base case. Thus, $\Omega[S,i_0]$ is a partial description containing no input segments, and with final non-terminal S.

Now, consider some term $\Omega[X^k,i_j]$ of the computing recurrence. The possible structures considered as values for this term are described by the three parts of the main computing recurrence formula (the last equation in (19)). Assume that all of $\{\Omega[X^m,i_{j-1}]\}_{m=1..K}$ contain as values only partial descriptions which contain input segments $i_1..i_{j-1}$.

Consider the first part of the main computing recurrence formula. This considers the structure ω formed by appending a unit with i_j underparsed to the partial description in $\Omega[X^k,i_{j-1}]$. This directly corresponds to the first part of the main matching recurrence formula. By hypothesis, $\Omega[X^k,i_{j-1}]$ is a valid partial description containing $i_1..i_{j-1}$ with last non-terminal X^k. Then ω is also a valid partial description containing $i_1..i_j$ with last non-terminal X^k.

Consider the second part of the main computing recurrence formula. This considers structures $\{\omega^m\}$ formed by appending a unit, with i_j parsed into position x^k, to the partial descriptions in selected $\{\Omega[X^m,i_{j-1}]\}$. This directly corresponds to the second part of the main matching recurrence formula, provided that the consecutive positions x^m followed by x^k are part of some valid position structure. This is guaranteed by the condition on the terms in this part of the computing recurrence formula; this condition only permits consideration of appending a position x^k if there is a derivation rule in Γ permitting the derivation of position x^k from non-terminal X^m. By hypothesis, each $\Omega[X^m,i_{j-1}]$ is a valid partial description containing $i_1..i_{j-1}$ and with last non-terminal X^m. Thus, $\{\omega^m\}$ are valid partial descriptions containing $i_1..i_j$ with last non-terminal X^k.

Consider the third part of the main computing recurrence formula. This part considers structures $\{\omega^m\}$ formed by appending an unfilled x^k to the structures in selected $\{\Omega[X^m,i_j]\}$. This part directly corresponds to the third part of the main matching recurrence formula, provided that the consecutive positions x^m followed by x^k are part of some valid position structure. The provision is guaranteed by the condition on the terms in this part of the computing recurrence formula; this condition only permits consideration of appending a position x^k if there is a derivation rule in Γ permitting the derivation of position x^k from non-terminal X^m. Therefore, if $\{\Omega[X^m,i_j]\}$ contain only valid partial descriptions containing $i_1..i_j$ with last non-terminal X^k, then the structures obtained by appending the unfilled x^k are also valid partial descriptions containing $i_1..i_j$ with last non-terminal X^k.

Call the number of unfilled positions in a structure following the last unit containing an input segment the *final overparsing length* of that structure. The partial descriptions considered by the first two parts of the main computing recurrence formula, discussed above, have final overparsing length 0. Because the third part of the computing recurrence formula only considers structures of final overparsing length one or greater, the earlier results for the first two parts of the computing recurrence formula establish that all structures of final overparsing length zero that are considered are valid partial descriptions. Now, assume that all considered structures of final overparsing length N-1 are valid partial descriptions. Any considered structure ω^m of final overparsing length N is considered only through the third part of the computing recurrence formula. The structure being appended to, $\Omega[X^m,i_j]$, must be of final overparsing length N-1. By hypothesis, that structure is a valid partial description. Therefore, ω^m is a valid partial description. By mathematical induction, all structures considered by the third part of the main computing recurrence formula are valid partial descriptions containing $i_1..i_j$ with last non-terminal X^k.

This exhausts the main computing recurrence formula, as all three parts have now been shown to consider only valid partial descriptions. Thus, only partial descriptions containing $i_1..i_j$ with last non-terminal X^k may be values of the terms of the computing recurrence.

(24) **Def.** A *cycle of overparsing units* is a contiguous string of unfilled positions in which the first and last units are the same type of position.

(25) **Def.** A Harmony function is said to *ban overparsing cycles* if, for any partial description which contains a cycle of overparsing units, removing all but the first unit of the cycle results in a new partial description which is strictly more harmonic.

(26) **Lemma** Suppose that the Harmony function bans overparsing cycles. Then for any X^k and i_j, of all partial descriptions containing i_1 through i_j and with last non-terminal X^k, there exist a subset that have maximum Harmony, and that subset is finite.

Proof

Let K be the number of types of non-terminals. In any candidate partial description, there are only a finite number of units containing input segments (j of them, to be precise). By the pigeonhole principle, between any two units containing consecutive input segments, there cannot be more unfilled positions than there are types of positions without repeating a position, thus creating an overparsing cycle. Identical reasoning gives the same bound on the number of units that may occur before the first input segment, or after the last input segment. Thus, a partial description cannot have more than (j+1)*K unfilled units without overparsing cycles. Thus, there are only a finite number of candidate partial descriptions which do not contain any overparsing cycles.

By assumption, any candidate containing an overparsing cycle is strictly less harmonic than some candidate without that overparsing cycle. Thus, there are only a finite number of candidates, those not containing overparsing cycles, which could possibly have maximum Harmony. Because the set of optimal partial descriptions is contained in this finite set, and because there is guaranteed to be at least one candidate that does not contain overparsing cycles, the set of candidate partial descriptions with maximum Harmony is finite, and contains at least one partial description.

(27) **Lemma** Suppose that the Harmony function bans overparsing cycles, and that the universal constraints are local. Then for any set of computing recurrence terms $\{\Omega[X^k,i_j]\}_{k=1..K}$ the solution values for these terms exist. For each term $\Omega[X^k,i_j]$, the solution value is precisely the set of partial descriptions of maximum Harmony containing $i_1..i_j$ and with final non-terminal X^k.

Proof

By lemma (23), the computing recurrence only considers, as possible values for $\Omega[X^k,i_j]$, partial descriptions containing $i_1..i_j$ and with final non-terminal X^k. By lemma (26), we know that for each recurrence term $\Omega[X^k,i_j]$, there exist a positive finite set of partial descriptions containing $i_1..i_j$ and with final non-terminal X^k, that have maximum Harmony. Assign to each term its corresponding set of optimal partial descriptions. Consider some partial description π assigned to $\Omega[X^k,i_j]$.

Suppose that the last unit of π contains i_j underparsed. Then the partial description containing all but this last unit is an optimal partial description for $\Omega[X^k,i_{j-1}]$. Suppose, to the contrary, that it weren't. Then a better solution for $\Omega[X^k,i_j]$ could be constructed by appending the last unit of π to an optimal value for $\Omega[X^k,i_{j-1}]$. The locality of the universal constraints guarantees that the marks assessed the final unit will be the same when that unit is appended to any partial description with last position X^k. The Harmony of any partial description including the final unit is thus simply the collection of violation marks assessed the partial description without the final unit, with the marks assessed the final unit added to it. This contradicts the assumption that π is an optimal solution. Thus, π satisfies the recurrence, as it is considered by the first part of the recurrence, and is of maximum Harmony out of all possible partial descriptions considered for $\Omega[X^k,i_j]$.

Suppose that the last unit of π contains i_j parsed into x^k. Then π with this last unit removed is an optimal partial description for the term $\Omega[X^m,i_{j-1}]$, where X^m is the last non-terminal of this partial description. Suppose, to the contrary, that it weren't. Then a better partial description for $\Omega[X^k,i_j]$ could be constructed by appending the last unit of π to an optimal value for $\Omega[X^m,i_{j-1}]$. This contradicts the assumption that π is an optimal solution. Thus, π satisfies the recurrence, as it is considered by the second part of the recurrence, and is of maximum Harmony out of all partial descriptions considered for $\Omega[X^k,i_j]$.

Suppose now that the last unit of π does not contain input segment i_j. By containment, π must include a unit containing i_j, so the last unit of π must be an unfilled x^k. Then π with this last unit removed is an optimal partial description for $\Omega[X^m,i_j]$, where X^m is the last non-terminal of this partial description. Suppose, to the contrary, that it weren't. Then a better partial description for $\Omega[X^k,i_j]$ could be constructed by appending the last unit of π to an optimal value for $\Omega[X^m,i_j]$. This contradicts the assumption that π is an optimal solution. Thus, π satisfies the recurrence, as it is considered by the third part of the recurrence, and is of maximum Harmony out of all partial descriptions considered for $\Omega[X^k,i_j]$.

This demonstrates that assigning the optimal partial descriptions to each of the computing recurrence terms solves the recurrence. Finally, no other solutions exist, because the recurrence only assigns to a term partial descriptions of maximum Harmony among those considered. The base case, $\Omega[S,i_0]$, is fixed, so for each term considering descriptions of length one or greater, the optimal values are considered. So any other partial descriptions are disallowed by the recurrence in virtue of their sub-optimality.

(28) **Theorem** Suppose that the Harmony function bans overparsing cycles, and that the universal constraints are local. Then *OPT*(I) = maxH{*GEN*(I)}.

Proof

First, consider the set of descriptions selected by *OPT*(I). By definition, *OPT*(I) only considers recurrence terms $\Omega[X^k,i_J]$ where $X^k \Rightarrow e$ is in Γ. This ensures that each term has as values only descriptions with valid final positions, and thus that each term has as values only descriptions generated by *GEN*(I). Thus, the set of candidates optimized over by *OPT*(I) is a subset of the candidates optimized over by *GEN*(I). Then *OPT*(I) $\leq$ maxH{*GEN*(I)}.

Now, consider the set of descriptions selected by maxH{*GEN*(I)}. Each such description π must have a valid final non-terminal X^k. Thus, such a description is one of the descriptions optimized over by $\Omega[X^k, i_J]$. Thus, for each π, $\pi \leq \Omega[X^k, i_J] \leq$ *OPT*(I). Then maxH{*GEN*(I)} $\leq$ *OPT*(I).

It follows that *OPT*(I) = maxH{*GEN*(I)}.

2.5.3 The Dynamic Programming Algorithm

(29) **Def.** The *iterative solution procedure* for a set of computing recurrence equations $IterSolve(\{\Omega[X^k, i_j]\}_k)$ is:

```
For each $X^k$, set $\Omega[X^k,i_j]$ to $\{\Omega[X^k,i_{j-1}] + \langle i_j \rangle\}$
For each $X^k$, set $\Omega[X^k,i_j]$ to $maxH\{\Omega[X^k,i_j],\ \{\Omega[X^m,i_{j-1}] + x^k/i_j \mid X^m \Rightarrow x^k X^k \in \Gamma\}\}$
Repeat
    For each $X^k$, set $\Omega[X^k,i_j]$ to $maxH\{\Omega[X^k,i_j],\ \{\Omega[X^m,i_j] + x^k/\square \mid X^m \Rightarrow x^k X^k \in \Gamma\}\ \}$
Until no $\Omega[X^k,i_j]$ has changed during a pass
Return $(\{\Omega[X^k,i_j]\}_k)$
```

(30) **Def.** The *dynamic programming algorithm* DP(I) is:

```
Set $\Omega[S,i_0]$ to {the null structure}
For each $X^k \neq S$, set $\Omega[X^k,i_0]$ to $\infty$
Repeat
    For each $X^k$, set $\Omega[X^k,i_0]$ to $maxH\{\Omega[X^k,i_0],\ \{\Omega[X^m,i_0] + x^k/\square \mid X^m \Rightarrow x^k X^k \in \Gamma\}\ \}$
Until no $\Omega[X^k,i_0]$ has changed during a pass
For each $i_j$, set $\{\Omega[X^k,i_j]\}_k$ to $IterSolve(\{\Omega[X^k,i_j]\}_k)$
Set $Answer$ to $maxH\{\Omega[X^f,i_J] \mid X^f \Rightarrow e \in \Gamma\}$
Return ($Answer$)
```

The symbol ∞ represents the “worst possible Harmony” value. This is used as a way of marking terms that have not yet been assigned a partial description as a candidate value.

This algorithm directly corresponds to the dynamic programming algorithm described in the earlier sections. Each recurrence term $\Omega[X^k,i_j]$ is the same as cell $[X^k,i_j]$ of the Dynamic Programming Table. Index i_0 is the same as column heading BOI. The Operations Set simply lists the pre-calculated consequences of each operation considered by *IterSolve()*.

(31) **Theorem** Suppose that *GEN* is defined as above, the universal constraints are local, and the resulting Harmony function bans overparsing cycles. Then DP(I) computes the optimal structural description of I.

Proof

By theorem (28), *OPT*(I) is the optimal structural description of I. It remains to be shown that DP(I) correctly computes it.

The nature of the algorithm is to consider certain candidate solutions, and keep any solution considered unless another solution is explicitly considered which is more Harmonic. Therefore, it can be proven that all optimal solutions for a computing recurrence term are computed by showing that all optimal solutions are explicitly considered by the algorithm.

Observe that $\Omega[S,i_0]$ is assigned the optimal solution by the recurrence, because there is only one possible candidate. The repeat loop in the main DP() is separate only because this first set of terms may only consider partial descriptions with unfilled structure (no input segments have yet been considered). The logic proving this loop correct is the same as that for the other sets of inter-related terms as given below. However, in this first set the only solutions of final overparsing length zero are the pre-assigned values of the null structure and ∞.

Consider the term $\Omega[X^k,i_j]$, assuming that $\{\Omega[X^m,i_{j-1}]\}_{m=1..K}$ contain all of their optimal solutions. By the lemma (27), we know that the correct solutions for this term are those partial descriptions containing $i_1..i_j$ with final non-terminal X^k that have maximum Harmony, and that a positive, finite number of such descriptions exist. Consider some arbitrary optimal solution π.

Suppose that the last unit of π contains i_j underparsed. Then it is explicitly considered by the first step of *IterSolve()*. Suppose that the last unit of π contains i_j parsed into a position. Then it is explicitly considered by the second step of *IterSolve()*.

Suppose now that the last unit of π does not contain i_j. By containment, π must include a unit containing i_j, so there must be some finite substring of unfilled positions following the unit containing i_j. Call the number of such unfilled positions the final overparsing length of π. The solutions considered in the first two steps, where the final unit did contain i_j, are thus solutions with final overparsing length zero. It will next be shown that if all optimal solutions of final overparsing length N-1 have been considered, then all optimal solutions of length N will be considered.

Suppose that π has final overparsing length N. Then the partial description containing all but this last unit is an optimal solution of length N-1. By hypothesis, this solution of length N-1 has already been considered. Because it is an optimal solution, when it was considered, it was added as an optimal value for $\Omega[X^m,i_j]$, thus changing the value of $\Omega[X^m,i_j]$. This change ensures that another pass of the third step of *IterSolve()* will take place after. On this next pass, when $\Omega[X^k,i_j]$ is evaluated, π will be considered, because the third step explicitly considers solutions which append an unfilled x^k to all optimal solutions already determined for $\Omega[X^m,i_j]$.

Therefore, *IterSolve()* finds the correct values for every term $\Omega[X^k,i_j]$.

The only remaining step of the algorithm computes $maxH\{\Omega[X^f,i_J] \mid X^f \Rightarrow e \in \Gamma\}$ just as specified for *OPT*(I). Therefore, DP(I) correctly computes *OPT*(I).

(32) **Corollary** The Basic CV Syllable Theory grammars may be correctly computed by DP(I).

Proof

First, observe that the set of position structures is defined by an appropriate formal grammar. Next, observe that each of the universal constraints is local in the appropriate sense. Next, observe that the two FILL constraints, along with the locality of the other constraints, guarantee that descriptions with overparsing cycles are sub-optimal. Removing all but the first member of an overparsing cycle eliminates the FILL violations incurred by the removed units. It cannot incur any new violations, because the constraints are local to units, and no new units are added when cycle units are removed. Thus, the Harmony function bans overparsing cycles.

Finally, consider the hard conditions banning the parsing of V into an onset or coda position and banning the parsing of C into a nucleus position. Both of these conditions are entirely local (they may be evaluated for individual units) and simply rule out certain operations considered in step two of *IterSolve()*. In fact, they may be treated as superordinate (undominated) constraints for the purposes of the algorithm.

Thus, all the conditions of the main theorem are satisfied, so the theorem applies.

2.5.4 Computational Complexity

Each column in the DP Table is processed in constant time for any fixed grammar: the number of cells in each column is the number of non-terminals in the position grammar, and the number of passes through the column is bounded from above by the number of non-terminals, due to the fact that overparsing cycles are sub-optimal. There is one column for each input segment (plus the BOI column). Therefore, the algorithm is linear in the size of the input.

2.6 Conditions on and Extensions of the Algorithm

2.6.1 Parsing More Than One Segment into a Position

The algorithm as defined in section 2.5 inherits from the Basic Syllable Theory the simplifying assumptions that each syllable position may have at most one input segment parsed into it. As stated in a footnote above, this constraint actually comes from ranking the constraint *COMPLEX as superordinate. More sophisticated syllable theories will need to have a space of candidate parses which permits violations of this constraint. This may be accommodated by the current algorithmic scheme. What is required are more complex versions of the parse operation. One parse operation could parse the new input segments into the last syllable position in a partial structure (as opposed to adding a new position with the new segment parsed into it). This operation would then incur a mark indicating the violation of *COMPLEX, along with marks for any other constraints violated.

What the algorithm requires is that a new input segment may be parsed by a parse operation into either the current last syllable position, or into a newly generated syllable position. What is not permissible under the current form of the algorithm is the parsing of the current segment into a position preceding the final position in the partial structure; this would violate the sense of locality required by the algorithm. Notice that this does not require that two input segments sharing a syllable position be consecutive in the input. What it requires is that, if two input segments are not consecutive but are sharing a single syllable position, that all intervening input segments be either parsed into the same position or unparsed.

2.6.2 Locality

A property of the Basic CV Syllable Theory important to the success of the algorithm is the "locality" of the constraints. Each constraint may be evaluated on the basis of at most one input segment and two consecutive syllable positions. What really matters here is that the constraint violations incurred by an operation can be determined solely on the basis of the operation itself. The information used by the constraints in the Basic Syllable Theory include the piece of structure added and the very end of the partial description being added on to (the last syllabic position generated). These restrictions on constraints are sufficient conditions for the algorithm given.

An example of a constraint that would not be local in the context of the Basic Syllable Theory is a constraint which requires that the number of syllables be at least two, as when a word must contain a foot, and a foot must be binary at the level of syllables. That constraints referring to feet are not easily computed using the formal description given in this chaper should not be surprising, as there is no explicit representation of feet in the structures. To properly handle such theories, a more complex set of position structures will probably be required, perhaps a context-free space of structures in which foot nodes may dominate one or more syllable nodes, and so forth. In that case, the binary foot constraint would be local in the sense relevant to context-free position structures in Optimality Theory: the constraint could be evaluated solely on the basis of a foot node and the syllable nodes immediately dominated by it. This notion of locality will be formalized in the next chapter.

Other examples of non-locality are alignment constraints (McCarthy & Prince 1993b). Alignment constraints typically refer to the alignment of boundaries of constituents, where in principle an arbitrary amount of structure may intervene between the two boundaries. Alignment constraints can be non-local in two ways. One involves the relationship between constituent boundaries and context-free structures, and will be discussed in the next chapter. The other way occurs with a type of alignment constraint which incurs multiple violations, one for each instance of a type of structure which intervenes between the relevant boundaries. This non-locality is of a restricted form, in the sense that a constraint violation is incurred independently by each relevant unit of intervening structure. Thus, the violation is "local" to the unit of structure given the knowledge that it is between the two relevant boundaries, and independent of how "far" the unit is from either of the boundaries. This may well make many classes of alignment constraints amenable to parsing with dynamic programming. Efficient parsing in systems which use alignment constraints is the subject of current research.

It is important to distinguish between locality of *constraints* and locality of *effects*. Independent of computational concerns, Optimality Theory provides striking illustrations of how local constraints can have very non-local effects. This is reflected in the dynamic programming algorithm by the fact that more than one partial description is maintained at each stage. Each partial description in a column represents a different set of decisions on what to do up to that point in the input. The several partial descriptions are maintained because those early decisions may interact with decisions about later parts of the input. The efficiency of the dynamic programming algorithm for regular structures stems from the fact that at each stage, only a constant number of partial descriptions need to be maintained (in the case of Basic Syllable Theory, four).

Chapter 3. Parsing: Context-Free Position Structures

Using a context-free position structure grammar increases the complexity of the parsing task. Instead of a linear structure of non-terminals running parallel to the terminal positions, as in the regular position grammar case, a string of positions may be dominated by a hierarchy of non-terminals. The space of possible structural descriptions will be in general richer, because in addition to a large space of possible strings of positions, each string of positions may support several different descriptive structures. However, while this may increase the complexity of parsing within Optimality Theory, it does not render the problem intractable.

This chapter will present algorithms for parsing with context-free position structure grammars. The special case of a context-free grammar in a restricted canonical form will be discussed in detail.

3.1 Context-Free Position Structure Grammars

Using a context-free position structure grammar does not greatly affect the conceptual functioning of *GEN*. The position structures still have positions as terminals, and *GEN* still generates all allowable ways of matching the input string to the position string for each generated position structure. Notice that we are here still assuming the input to be a linear string. The increase in complexity is thus confined entirely to the space of considered position structures.

3.2 An Algorithm for "Revised" Chomsky Normal Form Position Structure Grammars

Chomsky Normal Form (Chomsky 1959) is an often-studied canonical form for context-free grammars. Intuitively, it enforces binary branching in the derivation tree.

(33) A formal grammar is in Chomsky Normal Form (CNF) if every production has for its right side either two non-terminals or one terminal.

CNF is of interest to computer scientists because, for any arbitrary context-free grammar, there is a formally equivalent grammar in Chomsky Normal Form. Branching which is at most binary is of interest to linguists, because it is often considered a desirable constraint on linguistic structures, particularly in syntax. It is useful here because the productions are restricted to a couple of simple forms which are more easily analyzed. This chapter will discuss in detail a canonical form which is slightly less restrictive than

Chomsky Normal Form. It enforces at most binary branching, and is here called "Revised" Chomsky Normal Form (RCNF).

(34) A formal grammar is in "Revised" Chomsky Normal Form (RCNF) if every production has for its right side one of the following forms: (a) two non-terminals; (b) one non-terminal; (c) one terminal. The production $S \Rightarrow e$ is also permitted, where S is the start symbol, and e is the empty string.

One nice property of the pure CNF is that structures cannot be generated with cycles of non-terminals singly dominating other non-terminals (e.g., A dominates only B dominates only C dominates only A ...). The RCNF by itself permits such structures. This is relevant for analyses in Optimality Theory, because excess structure is often penalized by constraints which are violated by extra material which is phonetically realized (that is, unfilled terminal positions). A well-defined grammar will have to ensure that the faithfulness constraints are sufficient to restrict the amount of structure actually present in optimal descriptions, either by ensuring that the specific position structure grammar does not permit cycles of singly dominating non-terminals (as is the case in the example discussed below), or by having constraints which penalize unwarranted higher structure such as non-terminals in a description, or by some other means.

3.2.1 The Dynamic Programming Table

The Dynamic Programming Table is more complex for context-free position structures. It is no longer a simple two-dimensional grid. The table now has three indices: one indicating the identity of a non-terminal (just like the row index in the regular grammar case), and two other indices indicating the range of the substring of the input string covered by the non-terminal. The cell [X,2,5] should contain the structure of and the Harmony of the best way of parsing input segments i_2 through i_5 with a structure that has the non-terminal X at the root (that is, X dominates the rest of the structure). The table has a cell for every non-terminal, for every legitimate substring range of the input. Thus, a cell for a substring of one input segment is permitted, such as [X,3,3], but no cells exist with the starting index greater than the ending index, like [X,3,2].

The DP Table is perhaps best envisioned as a set layers, one for each non-terminal. A layer looks like the following (the cells are labeled):

Table 7

Example Layer of the Dynamic Programming Table

[X,1,3]		
[X,1,2]	[X,2,3]	
[X,1,1]	[X,2,2]	[X,3,3]
i_1	i_2	i_3

It will help to visualize the table as follows. For each non-terminal X, there is a row containing one cell for every segment of the input: [X,a,a] for each input segment i_a. The next row will contain one fewer cell, and each cell corresponds to a pair of consecutive input segments: [X,a,a+1]. The next row has one less cell yet, with each cell covering a substring of three consecutive segments. This pattern continues up to the top row, which contains only one cell: [X,1,J], where J is the length of the string. For each substring length, there is a collection of rows, one for each non-terminal, which will collectively be referred to as a *level*. The first level contains the cells which only cover one input segment; the number of cells in this level will be the number of input segments multiplied by the number of non-terminals. Level two contains cells which cover input substrings of length two, and so on. The top level contains one cell for each non-terminal.

In the rest of this chapter, table cells will be denoted with square brackets, [], as above. Tree structures will be denoted with parentheses: a parent node X with child nodes Y and Z is denoted X(Y,Z).

3.2.2 The Operations Set

As in the regular grammar case, the operations set contains operations corresponding to underparsing, parsing, and overparsing actions. Also, the parsing and overparsing operations are matched with position structure grammar productions. However, there is additional complexity introduced by the context-free position grammar. The regular position grammar had only one type of production. The CNF position grammar has two main types of productions, one with a single terminal on the right hand side, and one with two non-terminals on the right hand side. We need parsing and overparsing operations to correspond to each of these.

The first type of parsing operation involves productions which generate a single terminal. Because we are assuming that an input segment may only be parsed into at most one position, and that a position may have at most one input segment parsed into it, this parsing operation may only fill a cell which covers exactly one input segment. The second kind of parsing operation is matched to a position grammar production in which a parent non-terminal generates two child non-terminals. This kind of operation fills the cell for the parent non-terminal by combining cell entries for the two child so that the substrings covered by each of them combine (concatenatively, with no overlap) to form the input substring covered by the cell being filled. The resulting Harmony in the cell being filled will be the combination of the marks assessed each of the substructures for the two child non-terminals, plus any additional marks incurred as a result of the production itself. This operation has to consider all possible ways in which the child non-terminals, taken in order, may combine to cover the substring, and select the way with the best resulting Harmony. Because the child non-terminals must be contiguous and in order, this amounts to considering each of the ways in which the substring can be divided into two pieces. This can be expressed as the formula shown in the Previous Cell column for this parsing operation, with b ranging in value from a to (c-1).

Underparsing operations are not matched with position grammar productions, but they are more complex for other reasons. A DP Table cell which covers only one input segment may be filled by an underparsing operation which marks the input segment as unparsed; this is just as in the regular grammar case. In general, any partial description covering any substring of the input may be extended to cover an adjacent input segment by marking that additional segment as unparsed. Thus, a cell covering a given substring of length greater than one may be filled in two mirror-image ways via underparsing: by taking a substructure which covers all but the leftmost input segment and adding that segment as unparsed, and by taking a substructure which covers all but the rightmost input segment and adding that segment as unparsed.

Overparsing operations are more complex in the context-free case. The reason is that more than just one unfilled position needs to be considered. A single non-terminal may dominate an entire subtree, in which none of the syllable positions at the leaves of the tree are filled. Thus, the optimal "unfilled structure" for each non-terminal must be determined. We will denote the optimal overparsing structure for non-terminal X with [X, 0], and we will refer to such an entity as a base overparsing structure. A set of such structures must be computed, one for each non-terminal, before parsing may begin. Because these values are not dependent upon the input, they may be "compiled" into the Operations Set. In fact, this is exactly what was done in the regular grammar case, it was just less apparent because the possible overparsing structures were less complex. The actual computation of these values will be discussed in further detail below.

The DP Table is filled as follows. First, all of the cells covering input substrings of length 1 are filled. Then, all of the cells covering substrings of length 2 are filled, utilizing the entries in the cells of the length 1 level. The cells of the other levels are filled by order of increasing substring length. The final level to be filled has one cell for each non-terminal, including the start symbol. When this level has been completely filled, the cell [S,1,J] will contain the optimal parse of the input, and its Harmony.

Table 8 shows templates for the kinds of operations.

Table 8

The Different Operation Forms

New Cell	Previous Cell(s)	Structure	Production	Operation Type
[X,a,a]	[X,0]	$\langle i_a \rangle$		Underparsing / Overparsing
[X,a,a]	none	$X(x/i_a)$	$X \Rightarrow x$	Parsing
[X,a,a]	$[X^k,a,a]$	$X(X^k)$	$X \Rightarrow X^k$	Overparsing
[X,a,a]	$[X^k,a,a]$ $[X^m,0]$	$X(X^k,X^m)$	$X \Rightarrow X^k X^m$	Overparsing
[X,a,a]	$[X^k,0]$ $[X^m,a,a]$	$X(X^k,X^m)$	$X \Rightarrow X^k X^m$	Overparsing
[X,a,c]	[X,a+1,c]	$\langle i_a \rangle$		Underparsing
[X,a,c]	[X,a,c-1]	$\langle i_c \rangle$		Underparsing
[X,a,c]	$\max_b\{[X^k,a,b]$ $[X^m,b+1,c]\}$	$X(X^k,X^m)$	$X \Rightarrow X^k X^m$	Parsing
[X,a,c]	$[X^k,a,c]$	$X(X^k)$	$X \Rightarrow X^k$	Overparsing
[X,a,c]	$[X^k,a,c]$ $[X^m,0]$	$X(X^k,X^m)$	$X \Rightarrow X^k X^m$	Overparsing
[X,a,c]	$[X^k,0]$ $[X^m,a,c]$	$X(X^k,X^m)$	$X \Rightarrow X^k X^m$	Overparsing

3.2.3 The Overparsing Operations

The overparsing operations must be considered after the underparsing and parsing operations. This is for the same reasons as in the regular grammar case: the overparsing operations may make use of cell entries in the same level. Further, the application of overparsing operations must go through several cycles, again just as in the regular grammar case. In the regular grammar case, a successive sequence of overparsing operations resulted in a sequence of unfilled positions. In the context-free grammar case, overparsing operations add entirely unfilled subtrees. The unfilled subtree may contain several unfilled positions, and the terminals of several consecutive overparsing operations will not necessarily be in succession in the terminal position string. For example, if $[X^k,0]$ and $[X^m,0]$ are base overparsing structures, and we start with a structure in which the non-terminal Z dominates a substring of the input, the first overparsing operation might put $[X^k,0]$ to the left of Z, all dominated by X^p, while the second overparsing operation might put $[X^k,0]$ to the right, resulting in the structure $X^q(X^p([X^k,0],Z), [X^m,0])$. Notice that in this structure dominated by non-terminal X^q, the unfilled positions of $[X^k,0]$ precede the positions dominated by Z, while the unfilled positions of $[X^m,0]$ follow.

The same principle that guaranteed a limit on the number of successive overparsing operations in an optimal form in the regular grammar case must be in force here. The faithfulness constraints must penalize overparsing. In particular, the constraints should prevent overparsing from adding an entire overparsed non-terminal more than once. That is, cycles of overparsing must be suboptimal. Given that the universal constraints meet this criterion, the overparsing operations may be repeatedly considered for a given level until none of them increase the Harmony of the entries in any of the cells.

3.2.4 Calculating the Base Overparsing Structures

The base overparsing structures, denoted above as [X,0], represent the optimal structures dominated by each non-terminal that contain none of the input segments. These are best calculated using a dynamic programming algorithm quite similar to the main parsing algorithm. First, for each non-terminal X which has a production rule permitting it to dominate a terminal x, set [X,0] to contain the structure with the corresponding terminal x left unfilled. Next, for each value [X,0], for each production $X \Rightarrow X^k X^m$, consider the Harmony of the structure with root X dominating $[X^k,0]$ and $[X^m,0]$, and choose the more Harmonic of that structure and the prior value of [X,0].

This procedure amounts to iterating overparsing operations over cells for each non-terminal. The same conditions which guarantee that a cycle of overparsing operations will not occur in the general case also apply to this case. A "pure" implementation of the general algorithm might not pre-compute these base overparsing structures; it might instead compute from scratch the optimal unfilled structure for a non-terminal for each location in which an overparsing operation for that non-terminal is considered. But, the locality assumptions being employed here require that there not be any interaction between the structure dominated by a non-terminal and structure above that non-terminal. So, we are guaranteed that the optimal unfilled structure dominated by a given non-terminal will be the same in any location where the overparsing is considered. Thus, it is efficient to pre-compute these base overparsing structures, and simply recall them every time an overparsing operation is considered. This strategy was employed in the regular grammar case as well, but it is not apparent because there is only one way to overparse for a given non-terminal, given the assumption that a non-terminal may only dominate one specific terminal.

Notice that if a grammar is developed within Optimality Theory which does not meet the locality assumptions in their entirety such that the optimal unfilled structure for a non-terminal is different in different locations, but sufficiently restricts the interactions so that parsing may be conducted efficiently, the above "pure" version will quite straightforwardly handle the case.

3.2.5 Computational Complexity

The algorithm has a computational complexity of N^3 in the size of the input.

3.3 An Example: Context-Free Pseudo-Syllable Structure

This section will illustrate the context-free case with a simple artificial grammar (artificial in that it does not correspond to any proposed linguistic analysis). The set of structures considered here are based upon the "pseudo-syllable": a peak (p) surrounded by one or more pairs of margin positions (m). These structures exhibit context-free behavior, in that margin positions to the left of a peak should be balanced with margin positions to the right.

The set of inputs is $\{C,V\}^+$. The position grammar is:

(35) $S \Rightarrow F \mid e$

$F \Rightarrow Y \mid YF$

$Y \Rightarrow MR$

$R \Rightarrow PM \mid YM$

$M \Rightarrow m$

$P \Rightarrow p$

The constraints are:

(36)	*m/V	Do not parse a V into a margin position.
	*p/C	Do not parse a C into a peak position.
	PARSE	Input segments must be parsed (into positions).
	FILL^{m}	A margin position must be filled.
	FILL^{p}	A peak position must be filled.

In this grammar, *GEN* generates candidates with a V parsed into a margin position, and with a C parsed into a peak position, as well as vice-versa. The conditions on parsing segment types into positions are now contained in the violable constraints, rather than in the definition of *GEN*, as was the case with the Basic CV Syllable Theory.

The non-terminal Y dominates a pseudo-syllable, which consists of a left margin position and a rhyme (R) which in turn consists of a center constituent and a right margin. The center constituent may either be a single peak position or an entire pseudo-syllable. The non-terminal F is a pseudo-foot, and permits hierarchical grouping of a sequence of pseudo-syllables.

Consider the case of the following ranking: {*m/V, *p/C, PARSE} $\gg$ {FILL^{p}} $\gg$ {FILL^{m}}. This ranking ensures that in optimal descriptions the balancing of margin positions around a peak is perfect, and that a V will only be parsed as a peak, while a C will only be parsed as a margin. Further, all input segments will be parsed, and unfilled positions will be included only as necessary to produce a sequence of balanced pseudo-syllables.

For example, the input /VC/ receives the description S(F(Y(M(□),R(P(V),M(C))))). The surface string for this description is □VC: the first □ was "epenthesized" to balance with the one following the peak V. Balanced inputs like /CCVCC/ and /CVCCCVCC/ will receive completely faithful descriptions which incur no constraint violations.

Table 9 shows the operations of the Operations Set for cells headed by the non-terminals M, Y, and S. Other dynamic programming table cell entries are denoted with square brackets [].

Table 9

The Operations Set for M, Y, and S

New Cell	Structure	Violations	Production	Operation Type
[M,a,a]	[M,0] $\langle i_a \rangle$	{*FILL^m *PARSE}		Underparsing / Overparsing
[M,a,a]	M(m/i_a)	if i_a=V {*m/V}	M ⇒ m	Parsing
[M,a,c]	$\langle i_a \rangle$ [M,a+1,c]	{*PARSE}		Underparsing
[M,a,c]	[M,a,c-1] $\langle i_c \rangle$	{*PARSE}		Underparsing
[Y,a,a]	Y([M,0], [R,0]) $\langle i_a \rangle$	{*PARSE *FILL^p *FILL^m *FILL^m}		Underparsing / Overparsing
[Y,a,c]	$\langle i_a \rangle$ [Y,a+1,c]	{*PARSE}		Underparsing
[Y,a,c]	[Y,a,c-1] $\langle i_c \rangle$	{*PARSE}		Underparsing
[Y,a,c]	$\max_b${Y([M,a,b], [R,b+1,c])}	{}	Y ⇒ MR	Parsing
[Y,a,c]	Y([M,0], [R,a,c])	{*FILL^m}	Y ⇒ MR	Overparsing
[Y,a,c]	Y([M,a,c], [R,0])	{*FILL^m *FILL^p}	Y ⇒ MR	Overparsing
[S,a,a]	[S,0] $\langle i_a \rangle$	{*PARSE}		Underparsing / Overparsing
[S,a,c]	[S,a,c-1] $\langle i_c \rangle$	{*PARSE}		Underparsing
[S,a,c]	S([F,a,c])	{}	S ⇒ F	Overparsing

The base overparsing structures are shown in Table 10.

Table 10

The Base Overparsing Structures

Root	Structure	Violations
[M,0]	M(m/□)	{*FILL^m}
[P,0]	P(p/□)	{*FILL^p}
[R,0]	R([P,0], [M,0])	{*FILL^m *FILL^p}
[Y,0]	Y([M,0], [R,0])	{*FILL^m *FILL^p *FILL^m}
[F,0]	F([Y,0])	{*FILL^m *FILL^p *FILL^m}
[S,0]	the null structure	{}

Now, the dynamic programming table for the input /VC/ will be shown. Because the table is three-dimensional, it will be shown as several "slices", one for each non-terminal.

Table 11

Cells for Non-Terminal M

*PARSE ⟨V⟩ M(m/C)	
*PARSE *FILL^m M(m/□) ⟨V⟩	M(m/C)
V	C

Table 12

Cells for Non-Terminal P

*PARSE ⟨C⟩ P(p/V)	
P(p/V)	*PARSE *FILL^{p} P(p/□) ⟨C⟩
V	C

Table 13

Cells for Non-Terminal R

R(P(p/V), M(m/C))	
*FILL^{m} R(P(p/V), M(m/□))	*FILL^{p} R(P(p/□), M(m/C))
V	C

Table 14

Cells for Non-Terminal Y

*FILL^{m} Y(M(m/□), R(P(p/V), M(m/C)))	
*FILL^{m} *FILL^{m} Y(M(m/□), R(P(p/V), M(m/□)))	*FILL^{p} *FILL^{m} Y(M(m/□), R(P(p/□), M(m/C)))
V	C

Table 15

Cells for Non-Terminal F

*FILL^{m} F(Y(M(m/□), R(P(p/V), M(m/C))))	
*FILL^{m} *FILL^{m} F(Y(M(m/□), R(P(p/V), M(m/□))))	*FILL^{p} *FILL^{m} F(Y(M(m/□), R(P(p/□), M(m/C))))
V	C

Table 16

Cells for Non-Terminal S

*FILL^{m} S(F(Y(M(m/□), R(P(p/V), M(m/C)))))	
*FILL^{m} *FILL^{m} S(F(Y(M(m/□), R(P(p/V), M(m/□)))))	*FILL^{p} *FILL^{m} S(F(Y(M(m/□), R(P(p/□), M(m/C)))))
V	C

The optimal description is the one contained in cell [S,1,2].

3.4 Formal Analysis of the "Revised" Chomsky Normal Form Case

3.4.1 Formal Definition of Optimality Theory Grammars

(37) **Def.** The *position structure grammar* Γ is a formal context-free grammar in "Revised" Chomsky Normal Form which generates *position structures*. Each terminal symbol of Γ is a structural position. The start symbol is S.

(38) **Def.** A *local region* is any of the following: (a) a non-terminal and the child non-terminals that it immediately dominates; (b) a non-terminal which dominates a terminal symbol, along with the terminal and the segment filling the terminal position.

(39) **Def.** Suppose there are an input I = $i_1 \ldots i_J$ composed of segments from the segment alphabet G, and a position structure P, which is a tree consisting of terminals $p_1 \ldots p_K$ and non-terminals $N_1 \ldots N_T$. A matching μ is a string of units, where the set of possible matchings between P and I is *MATCH*(P,I), defined by the following recurrence:

$$MATCH(P,I) \;=\; M[S,1,J]$$

$$\begin{aligned}
M[N_t,0] \;=\; & \{N_t(p_k/g) \mid N_t(p_k)\in P,\ g\in G\} \cup \\
& \{N_t(s_k) \mid s_k\in M[N_k,0],\ N_t(N_k)\in P\} \cup \\
& \{N_t(s_k,s_m) \mid s_k\in M[N_k,0],\ s_m\in M[N_m,0],\ N_t(N_k,N_m)\in P\} \cup \\
& \{\ \textit{the null structure} \mid P = S()\}
\end{aligned}$$

$$\begin{aligned}
M[N_t,a,a] \;=\; & \{s_t+\langle i_a\rangle \mid s_t\in M[N_t,0]\} \cup \\
& \{N_t(p_k/i_a) \mid N_t(p_k)\in P\} \cup \\
& \{N_t(s_k) \mid s_k\in M[N_k,a,a],\ N_t(N_k)\in P\} \cup \\
& \{N_t(s_k,s_m) \mid s_k\in M[N_k,a,a],\ s_m\in M[N_m,0],\ N_t(N_k,N_m)\in P\} \cup \\
& \{N_t(s_k,s_m) \mid s_k\in M[N_k,0],\ s_m\in M[N_m,a,a],\ N_t(N_k,N_m)\in P\}
\end{aligned}$$

$$\begin{aligned}
M[N_t,a,c] \;=\; & \{s_t+\langle i_a\rangle \mid s_t\in M[N_t,a+1,c]\} \cup \\
& \{s_t+\langle i_c\rangle \mid s_t\in M[N_t,a,c-1]\} \cup \\
& \{N_t(s_k,s_m) \mid a\le b\le c,\ s_k\in M[N_k,a,b],\ s_m\in M[N_m,b+1,c],\ N_t(N_k,N_m)\in P\} \cup \\
& \{N_t(s_k) \mid s_k\in M[N_k,a,c],\ N_t(N_k)\in P\} \cup \\
& \{N_t(s_k,s_m) \mid s_k\in M[N_k,a,c],\ s_m\in M[N_m,0],\ N_t(N_k,N_m)\in P\} \cup \\
& \{N_t(s_k,s_m) \mid s_k\in M[N_k,0],\ s_m\in M[N_m,a,c],\ N_t(N_k,N_m)\in P\}
\end{aligned}$$

The ‘/’ operator here means to fill the position to the left of the operator with the material to the right of the operator. The set {the null structure} contains precisely one object, which consists of exactly no structure.

This definition may be augmented by further restrictions for different specific instances of *GEN* (for example, forbidding consonants from being parsed into nucleus positions).

(40) **Def.** The set of candidate structural descriptions for input I is

$$GEN(I) \;=\; \bigcup_{P\in L(\Gamma)} MATCH(P,I)$$

L(Γ) is the set of all position structures generated by Γ.

(41) **Def.** The structural description assigned to I by the grammar is $maxH\{GEN(I)\}$. The function maxH{} returns the structural description with maximum Harmony. Implicit in the definition of maxH is a set of universal constraints and the ranking of those constraints.

3.4.2 The Computing Recurrence

(42) **Def.** The function *OPT*(I) is defined by the following computing recurrence (where G is the segment alphabet):

$$OPT(I) = \Omega[S,1,J]$$

$$\Omega[X^t,0] = maxH\left\{\begin{array}{c} \{X^t(x^k/g) \mid X^t \Rightarrow x^k \in \Gamma,\ g \in G\} \cup \\ \{X^t(\Omega[X^k,0]) \mid X^t \Rightarrow X^k \in \Gamma\} \cup \\ \{X^t(\Omega[X^k,0],\Omega[X^m,0]) \mid X^t \Rightarrow X^k X^m \in \Gamma\} \cup \\ \{\ \textit{the null structure} \mid X^t = S,\ S \Rightarrow e \in \Gamma\} \end{array}\right\}$$

$$\Omega[X^t,a,a] = maxH\left\{\begin{array}{c} \{\Omega[X^t,0]+\langle i_a\rangle\} \cup \\ \{X^t(x^k/i_a) \mid X^t \Rightarrow x^k \in \Gamma\} \cup \\ \{X^t(\Omega[X^k,a,a]) \mid X^t \Rightarrow X^k \in \Gamma\} \cup \\ \{X^t(\Omega[X^k,0],\Omega[X^m,a,a]) \mid X^t \Rightarrow X^k X^m \in \Gamma\} \cup \\ \{X^t(\Omega[X^k,a,a],\Omega[X^m,0]) \mid X^t \Rightarrow X^k X^m \in \Gamma\} \end{array}\right\}$$

$$\Omega[X^t,a,c] = maxH\left\{\begin{array}{c} \{\Omega[X^t,a+1,c]+\langle i_a\rangle\} \cup \\ \{\Omega[X^t,a,c-1]+\langle i_c\rangle\} \cup \\ \{X^t(\Omega[X^k,a,b],\Omega[X^m,b+1,c]) \mid a \le b \le c,\ X^t \Rightarrow X^k X^m \in \Gamma\} \cup \\ \{X^t(\Omega[X^k,a,c]) \mid X^t \Rightarrow X^k \in \Gamma\} \cup \\ \{X^t(\Omega[X^k,0],\Omega[X^m,a,c]) \mid X^t \Rightarrow X^k X^m \in \Gamma\} \cup \\ \{X^t(\Omega[X^k,a,c],\Omega[X^m,0]) \mid X^t \Rightarrow X^k X^m \in \Gamma\} \end{array}\right\}$$

(43) **Def.** A *partial description* π is a tree which is a subtree of a structural description μ (generated by *GEN*). A partial description is classified by its root non-terminal. Each leaf of a partial description consists of a position (a terminal of the position grammar) which either contains an input segment or an overparsed segment.

(44) **Def.** A universal constraint is *local* if it may be evaluated solely on the basis of the internal configuration of a local region.

(45) **Def.** The *Harmony* of a partial description is the collection of the constraint violation marks assessed each local region of the partial description.

(46) **Def.** An overparsing region is a local region where one child non-terminal dominates all of the input segments dominated by the root non-terminal. A local region with no child non-terminals is not an overparsing region.

(47) **Lemma** The value of each term $\Omega[X^t,0]$ of the computing recurrence, if one exists, is a set of partial descriptions containing no input segments and having root non-terminal X^t.

Proof

This result is established through correspondence between the computing recurrence for $\Omega[X^t,0]$ and the matching recurrence for $M[N_t,0]$.

Consider some term $\Omega[X^t,0]$ of the computing recurrence. The possible structures considered as values for this term are described by the four parts of the recurrence.

Consider the first part of the computing recurrence. This considers structures $\{\omega^k\}$ formed by creating a non-terminal dominating a terminal with a non-input segment parsed into the terminal position. This directly corresponds to the first part of the matching recurrence, provided that the parent-child relation $X^t(x^k)$ is part of some position structure, which is guaranteed by the first condition on this part of the computing recurrence. The root non-terminal is X^t, and it covers no input segments. Thus, $\{\omega^k\}$ are valid partial descriptions containing no input segments and with root non-terminal X^t.

Consider the last part of the computing recurrence. This considers the null parse, where the start symbol (S) dominates no other structure. This directly corresponds to the last part of the matching recurrence. The root non-terminal is S, and it covers no input segments. Thus, this is a valid partial description containing no input segments and with root non-terminal S.

The second and third parts of the computing recurrence are analogous to the fourth and fifth parts of the computing recurrence for terms $\Omega[X^t,a,c]$ discussed in lemma (49), and the proof is similarly analogous.

(48) **Lemma** The value of each term $\Omega[X^t,a,a]$ of the computing recurrence, if one exists, is a set of partial descriptions containing the input segment i_a and having root non-terminal X^t.

Proof

This result is established through correspondence between the computing recurrence for $\Omega[X^t,a,a]$ and the matching recurrence for $M[N_t,a,a]$.

Consider some term $\Omega[X^t,a,a]$ of the computing recurrence. The possible structures considered as values for this term are described by the five parts of the recurrence. By lemma (47) all terms $\Omega[X^t,0]$ contain as values only valid partial descriptions covering no input segments and with root non-terminal X^t.

Consider the second part of the computing recurrence. This considers structures $\{\omega^k\}$ formed by setting X^t to dominate the terminal position x^k with i_a parsed into it. This directly corresponds to the second part of the matching recurrence, provided that the parent-child relation $X^t(x^k)$ is part of some valid position structure, which is guaranteed by the condition on this part of the computing recurrence. This structure has root non-terminal X^t, and covers only input segment i_a. Thus, $\{\omega^k\}$ are valid partial descriptions containing i_a with root non-terminal X^t.

The first part of the computing recurrence is analogous to the first part of the computing recurrence for terms $\Omega[X^t,a,c]$, discussed in lemma (49). The proof for this part is similarly analogous.

The third, fourth, and fifth parts of the computing recurrence are analogous to the fourth, fifth, and sixth parts of the computing recurrence for terms $\Omega[X^t,a,c]$, discussed in lemma (49). The proof for these parts is similarly analogous.

(49) **Lemma** The value of each term $\Omega[X^t,a,c]$ of the computing recurrence, if one exists, is a set of partial descriptions containing input segments i_a through i_c and having root non-terminal X^t.

Proof

This result is established through correspondence between the computing recurrence and the matching recurrence, for terms $[X^t,a,c]$.

Consider some term $\Omega[X^t,a,c]$ of the computing recurrence. The possible structures considered as values for this term are described by the six parts of the recurrence. Assume that all of the computing recurrence terms covering strict subsequences of i_a through i_c contain as values only valid partial descriptions containing input segments i_a through i_c with root non-terminal X^t.

Consider the first part of the computing recurrence. This considers the structure ω formed by adding the input segment i_a underparsed to the partial description $\Omega[X^t,a+1,c]$. This directly corresponds to the first part of the matching recurrence formula. By hypothesis, $\Omega[X^t,a+1,c]$ is a valid partial description containing i_{a+1} through i_c with root non-terminal X^t. Then ω is also a valid partial description, and it contains i_a through i_c with root non-terminal X^t. Directly analogous reasoning applies to the second part of the computing recurrence.

Consider the third part of the computing recurrence. This considers structures $\{\omega^b\}$ formed by setting the non-terminal X^t to immediately dominate the partial descriptions in terms $\Omega[X^k,a,b]$ and $\Omega[X^m,b+1,c]$. This directly corresponds to the third part of the matching recurrence, provided that the parent-child relation $X^t(X^k,X^m)$ is part of some valid position structure, which is guaranteed by the second condition on this part of the computing recurrence. By hypothesis, the terms $\Omega[X^k,a,b]$ and $\Omega[X^m,b+1,c]$ are valid partial descriptions meeting the terms of this lemma. The first condition on this part of the computing recurrence ensures that each of the segments of the subsequence is covered by precisely one of the child non-terminals. Thus, $\{\omega^b\}$ are valid partial descriptions containing input segments i_a through i_c with root non-terminal X^t.

Consider the last part of the computing recurrence. This considers structures $\{\omega^k\}$ formed by setting non-terminal X^t to immediately dominate the structures in $\Omega[X^k,a,c]$ and $\Omega[X^m,0]$. This part directly corresponds to the last part of the matching recurrence, provided that the parent-child relation $X^t(X^k,X^m)$ is part of some valid position structure, which is guaranteed by the condition on this part of the computing recurrence. By lemma (47), $\Omega[X^m,0]$ may only contain a valid partial description covering no input segments with root X^m. Thus, if $\Omega[X^k,a,c]$ contains a valid partial description containing input segments i_a through i_c with root non-terminal X^t, then the resulting structure created by this computing recurrence term for $\Omega[X^t,a,c]$ is also a valid partial description containing input segments i_a through i_c with root non-terminal X^t. Analogous reasoning applies to the fourth and fifth parts of the computing recurrence.

Call the number of overparsing regions in a structure which are not dominated by any non-overparsing local region the final overparsing length of that structure. The partial descriptions considered by the last three parts of the computing recurrence formula add an overparsing region to the root, and thus only consider structures of final overparsing length one or greater. The first three parts of the computing recurrence, as discussed above, only consider valid partial descriptions. Because these first three parts are the only parts that can consider structures of final overparsing length zero, it follows that all structures of final overparsing length zero that are considered are valid partial descriptions. Now, assume that all considered structures of final overparsing length N-1 are valid partial descriptions. Any considered structure ω^k of final overparsing length N is considered only through the last three parts of the computing recurrence. The structure being added to, $\Omega[X^k,a,c]$, must be of final overparsing length N-1. By hypothesis, that structure is a valid partial description. Therefore, ω^k is a valid partial description. By mathematical induction, all structures considered by the last three parts of the main computing recurrence formula are valid partial descriptions.

This exhausts the computing recurrence, as all parts have been shown to consider only valid partial descriptions. Thus, only partial descriptions containing input segments i_a through i_c with root non-terminal X^t may be values of the terms of the computing recurrence.

(50) **Def.** A cycle of overparsing regions is a sequence of overparsing regions where each successive region includes as a child non-terminal the root non-terminal of the preceding region in the sequence, where each region dominates all of the input segments dominated by the region following it in the sequence, and where the root non-terminals of the first and last regions of the sequence are the same non-terminal.

(51) **Def.** A Harmony function is said to ban overparsing cycles if, for any partial description which contains a cycle of overparsing regions, removing all but the first region of the cycle results in a new partial description which is strictly more Harmonic.

(52) **Lemma** Suppose that the Harmony function bans overparsing cycles. Then for any X^t, a, and c, of all the partial descriptions containing i_a through i_c and with root non-terminal X^t, there exist a subset that have maximum Harmony, and that subset is finite.

Proof

Let T be the number of types of non-terminals, and L be the length of the input subsequence covered by the candidate partial descriptions (that is, L=c-a+1). First, observe that any non-terminal in a candidate partial description must cover a contiguous subsequence (of length zero or greater) of the input. There are only a finite number of such contiguous subsequences (1 + L*(L+1)/2 of them).

A subtree which covers no input segments cannot have depth greater than the number of types of non-terminals without some non-terminal dominating another non-terminal of the same type, thus creating an overparsing cycle. Thus, the subtree cannot contain more than 2^T non-terminals without overparsing cycles.

By the pigeonhole principle, for any subsequence of length one or greater, there cannot be more non-terminals covering precisely that subsequence than there are types of non-terminals without repeating a non-terminal type, thus creating an overparsing cycle. Thus, a partial description which contains no overparsing cycles will contain at most T non-terminals covering any given input subsequence, for L*(L+1)/2 such subsequences, and therefore will contain not more than T*L*(L+1)/2 non-terminals which cover at least one input segment.

Because the root of any candidate partial description must cover all of the input segments, any non-terminal covering no input segments must be dominated by some non-terminal which does cover an input segment. Therefore, the number of maximal subtrees containing no input segments cannot be greater than the number of non-terminals which do cover input segments.

A maximal subtree is one which covers no input segments, with a root whose parent does cover an input segment. As argued above, those maximal subtrees contain not more than 2^T non-terminals. Therefore, there cannot be more than $2^T*(TL(L+1)/2)$ non-terminals covering no input segments in any partial description not containing overparsing cycles.

Therefore, for a partial description to not contain overparsing cycles: (a) the number of non-terminals which cover at least one input segment cannot exceed $TL(L+1)/2$; (b) the number of non-terminals which cover no input segments cannot exceed $2^T*(TL(L+1)/2)$. Thus, there are only a finite number of candidate partial descriptions which do not contain any overparsing cycles.

By assumption, any candidate containing an overparsing cycle is strictly less Harmonic than some candidate without that overparsing cycle. Thus, there are only a finite number of candidates, those not containing overparsing cycles, which could possibly have maximum Harmony. Because the set of optimal partial descriptions is contained in this finite set, and because there is guaranteed to be at least one candidate that does not contain overparsing cycles, the set of candidate partial descriptions with maximum Harmony is finite, and contains at least one partial description.

(53) **Lemma** Suppose that the Harmony function bans overparsing cycles, and that the universal constraints are local. Then for any set of computing recurrence terms $\{\Omega[X^t,a,c]\}_{t=1..T}$ the solution values for these terms exist. For each term $\Omega[X^t,a,c]$, the solution value is precisely the set of partial descriptions of maximum Harmony containing i_a through i_c and with root non-terminal X^t.

Proof

By lemmas (47), (48), and (49), the computing recurrence only considers, as possible values for $\Omega[X^t,a,c]$, partial descriptions covering i_a through i_c with root non-terminal X^t. By lemma (52), for each term $\Omega[X^t,a,c]$, there exists a finite set of such partial descriptions that have maximum

Harmony. Assign to each term its corresponding set of optimal partial descriptions. Consider some partial description π assigned to $\Omega[X^t,a,c]$.

Suppose that π contains i_a underparsed. Then the partial description containing all but the underparsed i_a is an optimal partial description for $\Omega[X^t,a+1,c]$. Suppose, to the contrary, that it weren't. Then a better solution for $\Omega[X^t,a,c]$ could be constructed by adding underparsed i_a to an optimal value for $\Omega[X^t,a+1,c]$. The locality of the universal constraints guarantees that the marks assessed the underparsed segment will be the same when added to any partial description with root X^t. This contradicts the assumption that π is an optimal solution. Thus, π satisfies the recurrence, as it is considered by the first term of the computing recurrence, and is of maximum Harmony out of all possible partial description considered for $\Omega[X^t,a,c]$. Analogous reasoning applies if π contains i_c underparsed.

Suppose that π has root non-terminal X^t dominating $\Omega[X^k,a,b]$ and $\Omega[X^m,b+1,c]$. Then removing the root non-terminal leaves two partial descriptions which are optimal partial descriptions for terms $\Omega[X^k,a,b]$ and $\Omega[X^m,b+1,c]$. Suppose, to the contrary, that one of them weren't, say term $\Omega[X^k,a,b]$ (without loss of generality). Then a better solution for $\Omega[X^t,a,c]$ could be constructed by using an optimal value of $\Omega[X^k,a,b]$ as a child. The locality of the universal constraints guarantees that the marks assessed the top local region will be the same when any partial description with root non-terminal X^k is used as a child. This contradicts the assumption that π is an optimal solution. Thus, π satisfies the recurrence, as it is considered by the third part of the computing recurrence, and is of maximum Harmony out of all possible partial descriptions considered for $\Omega[X^t,a,c]$.

The other cases involving partial descriptions with the root non-terminal dominating other non-terminals may be proved in analogous fashion to the case just proved.

Suppose that π has root non-terminal X^t directly dominating a terminal. This directly satisfies parts of the computing recurrences for $\Omega[X^t,a,a]$ and $\Omega[X^t,0]$ without reference to other term values.

This demonstrates that assigning the optimal partial descriptions to each of the computing recurrence terms solves the recurrence. Finally, no other solutions exist, because the recurrence only assigns to a term partial descriptions of maximum Harmony among those considered. So, any other partial descriptions are disallowed by the recurrence in virtue of their sub-optimality.

(54) **Theorem** Suppose that the Harmony function bans overparsing cycles and that the universal constraints are local. Then *OPT*(I) = maxH{*GEN*(I)}.

Proof

First, consider the set of descriptions selected by *OPT*(I). By definition, it only considers the recurrence term Ω[S,1,J]. This ensures that only descriptions generated by *GEN*(I) are considered. Thus, the set of candidates optimized over by *OPT*(I) is a subset of the candidates optimized over by *GEN*(I). Then *OPT*(I) ≤ maxH{*GEN*(I)}.

Now, consider the set of descriptions selected by maxH{*GEN*(I)}. Each such description π must have root non-terminal S. Thus, such a description is one of the descriptions optimized over by Ω[S,1,J]. Thus, for each π, π ≤ Ω[S,1,J] = *OPT*(I). Then maxH{*GEN*(I)} ≤ *OPT*(I).

It follows that *OPT*(I) = maxH{*GEN*(I)}.

The Dynamic Programming Algorithm

(55) **Def.** The base overparsing calculation procedure, $BaseCalc(\{\Omega[X^t,0]\}_t)$, is:

```
For each $X^t$, set $\Omega[X^t,0]$ to $\infty$
If $S \Rightarrow e$ is in $\Gamma$, set $\Omega[S,0]$ to {the null structure}
For each $X^t$, set $\Omega[X^t,0]$ to $maxH\{\Omega[X^t,0],\ \{X^t(x^t/g) \mid X^t \Rightarrow x^t \in \Gamma,\ g \in G\}\ \}$
Repeat
    For each $X^t$
        Set $\Omega[X^t,0]$ to $maxH\{\Omega[X^t,0],\ \{X^t([X^k,0]) \mid X^t \Rightarrow X^k \in \Gamma\}\ \}$
        Set $\Omega[X^t,0]$ to $maxH\{\Omega[X^t,0],\ \{X^t([X^k,0],[X^m,0]) \mid X^t \Rightarrow X^k X^m \in \Gamma\}\ \}$
Until no $\Omega[X^t,0]$ has changed during a pass
Return $(\{\Omega[X^t,0]\}_t)$
```

(56) **Def.** The iterative solution procedure for input sequences of length one, $IterSolve1(\{\Omega[X^t,a,a]\}_t)$, is:

```
For each $X^t$, set $\Omega[X^t,a,a]$ to $\{\Omega[X^t,0]+\langle i_a\rangle\}$
For each $X^t$, set $\Omega[X^t,a,a]$ to $maxH\{\Omega[X^t,a,a],\ \{X^t(x^t/i_a) \mid X^t \Rightarrow x^t \in \Gamma\}\ \}$
Repeat
    For each $X^t$, set $\Omega[X^t,a,a]$ to
        $maxH\{\Omega[X^t,a,a],\ \{X^t(\Omega[X^k,a,a]) \mid X^t \Rightarrow X^k \in \Gamma\}\ \}$
    For each $X^t$, set $\Omega[X^t,a,a]$ to
        $maxH\{\Omega[X^t,a,a],\ \{X^t(\Omega[X^k,0],\Omega[X^m,a,a]) \mid X^t \Rightarrow X^k X^m \in \Gamma\}\ \}$
    For each $X^t$, set $\Omega[X^t,a,a]$ to
        $maxH\{\Omega[X^t,a,a],\ \{X^t(\Omega[X^k,a,a],\Omega[X^m,0]) \mid X^t \Rightarrow X^k X^m \in \Gamma\}\ \}$
Until no $\Omega[X^t,a,a]$ has changed during a pass
Return $(\{\Omega[X^t,a,a]\}_t)$
```

(57) **Def.** The iterative solution procedure for input sequences of length greater than one, $\mathit{IterSolve}(\{\Omega[X^t,a,c]\}_t)$, is:

```
For each $X^t$, set $\Omega[X^t,a,c]$ to $\{\Omega[X^t,a+1,c]+\langle i_a\rangle\}$
For each $X^t$, set $\Omega[X^t,a,c]$ to $\{\Omega[X^t,a,c-1]+\langle i_c\rangle\}$
For each $X^t$
    For b=a to c
        set $\Omega[X^t,a,c]$ to
            $maxH\{\Omega[X^t,a,c],\ \{X^t([X^k,a,b],[X^m,b+1,c]) \mid X^t \Rightarrow X^k X^m \in \Gamma\}\ \}$
Repeat
    For each $X^t$, set $\Omega[X^t,a,c]$ to
        $maxH\{\Omega[X^t,a,c],\ \{X^t(\Omega[X^k,a,c]) \mid X^t \Rightarrow X^k \in \Gamma\}\ \}$
    For each $X^t$, set $\Omega[X^t,a,c]$ to
        $maxH\{\Omega[X^t,a,c],\ \{X^t(\Omega[X^k,0],\Omega[X^m,a,c]) \mid X^t \Rightarrow X^k X^m \in \Gamma\}\ \}$
    For each $X^t$, set $\Omega[X^t,a,c]$ to
        $maxH\{\Omega[X^t,a,c],\ \{X^t(\Omega[X^k,a,c],\Omega[X^m,0]) \mid X^t \Rightarrow X^k X^m \in \Gamma\}\ \}$
Until no $\Omega[X^t,a,c]$ has changed during a pass
Return $(\{\Omega[X^t,a,c]\}_t)$
```

(58) **Def.** The dynamic programming algorithm DP(I) (where J is the input length) is:

```
Set $(\{\Omega[X^t,0]\}_t)$ to $BaseCalc(\{\Omega[X^t,0]\}_t)$
For a=1 to J, set $(\{\Omega[X^t,a,a]\}_t)$ to $IterSolve1(\{\Omega[X^t,a,a]\}_t)$
For d=1 to (J-1)
    For a=1 to (J-d)
        Set $(\{\Omega[X^t,a,a+d]\}_t)$ to $IterSolve(\{\Omega[X^t,a,a+d]\}_t)$
Return (Ω[S,1,J])
```

(59) **Theorem** Suppose that *GEN* is defined as above, the universal constraints are local, and the resulting Harmony function bans overparsing cycles. Then DP(I) computes the optimal structural description of I.

Proof

By theorem (54), *OPT*(I) is the optimal structural description of I. It remains to be shown that DP(I) correctly computes it.

The nature of the algorithm is to consider certain candidate solutions, and keep any solution considered unless another solution is explicitly considered which is more Harmonic. Therefore, it can be proven that all optimal solutions for a computing recurrence term are computed by showing that all optimal solutions are explicitly considered by the algorithm.

Consider the term $\Omega[X^k,a,c]$. Assume that all terms of Ω covering strict subsequences of ia through ic contain all of their optimal solutions. By lemma (53), we know that the correct solutions for this term are those partial descriptions containing input i_a through i_c with root non-terminal X^t that have maximum Harmony, and that a finite number of such descriptions exist. Consider some arbitrary optimal solution π.

Suppose π contains i_a underparsed. Then it is explicitly considered by the first step of *IterSolve()*. Suppose that the root non-terminal of π immediately dominates a terminal. Then it is explicitly considered by step three of *BaseCalc()*, and step two of *IterSolve1()*.

Suppose that the root non-terminal of π immediately dominates two non-terminals, each of which covers a strict subsequence of i_a through i_c. Then it is explicitly considered by step six of *IterSolve()*.

Suppose that the root non-terminal of π immediately dominates another non-terminal which covers all of i_a through i_c. Then it is explicitly considered by the steps in the repeat loop of *IterSolve()*. That all such solutions are considered within this loop can be proven by mathematical induction over the final overparsing length of the solutions, in a manner analogous to the final part of the proof for lemma (49).

3.4.3 Computational Complexity

Each block of cells for a subsequence of the input (the block has one cell for each non-terminal) is processed in time linear in the length of the input subsequence. This is a consequence of the fact that in general operations filling a cell must consider all ways of dividing the subsequence into two pieces (see step four of *IterSolve*()). For any fixed grammar, the number of passes through the block is bounded from above by the number of non-terminals, due to the fact that overparsing cycles are sub-optimal. The number of such blocks is the number of input subsequences (equivalently, the number of cells in a layer), which is on the order of the square of the length of the input. If N is the length of the input, the algorithm has computational complexity $O(N^3)$.

3.5 The General Case: Position Structure Grammar Productions of Arbitrary Form

Handling grammars with position structure grammars that are not in "Revised" Chomsky Normal Form requires making the relationship to chart parsing more explicit. This is done by expanding the dynamic programming table in the following way. Consider the block of cells for some subsequence of the input. The dynamic programming table described previously contained blocks with one cell for each non-terminal of the position structure grammar. In the general case, there will also be one cell for each left-aligned substring of the righthand side of each production.

Here is an illustration. Consider the production $N \Rightarrow WXYZ$. The left-aligned substrings of the righthand side of this production are W, WX, and WXY. The dynamic programming table will contain a cell for each of these. These left-aligned substrings directly correspond to uncompleted edges in traditional chart parsing. There is no separate cell for the entire righthand side, WXYZ, because that is a completed edge, and the result is stored in the cell for the non-terminal on the lefthand side of the production, N.

A cell corresponding to a left-aligned substring is filled by considering all ways of combining entries for the last non-terminal of the substring with complementary entries for the substring with that non-terminal removed. The parsing operations for filling cell WXY will consider combinations of entries in cells for WX with entries in cells for Y. This corresponds to "extending an edge" in traditional chart parsing. Notice that the algorithm in the RCNF case can now be understood in terms of extending edges. With at most binary branching, any left-aligned substring only consists of one non-terminal, so that the uncompleted edge may be identified with a subtree dominated by the single non-terminal.

The complexity of the parsing algorithm in the general case is exactly the same as the RCNF case: cubic in the length of the input. The expansion of the dynamic programming table is sensitive only to the position structure grammar; the table still has, for a fixed position structure grammar, a constant number of cells for each subsequence of the input.

3.6 Discussion

3.6.1 Locality

As discussed in the previous chapter, alignment constraints are examples of proposed non-local constraints. There is a kind of non-locality which does not arise when regular position structures are employed, but can arise when context-free structures are employed. This kind of non-locality violates the "vertical" locality implicit in the definition of a local region. Consider an alignment constraint that requires the alignment of the left boundary of each constituent X with the boundary of some constituent Y. If X must immediately dominate Y, or vice-versa, then the constraint is local in the relevant sense, because the constraint can be evaluated based on whether Y is the leftmost constituent immediately dominated by X, or vice-versa. However, if some other non-terminal (call it Z) may dominate Y and in turn be dominated by X, then the location of Y with respect to the left boundary is "hidden" by Z. The restricted form of this non-locality may permit the following resolution. The dynamic programming table could be expanded so that every cell for a partial description with root non-terminal Z is expanded into two cells: one in which there is a Y as the leftmost constituent, and one in which there isn't. This would make the information relevant to the alignment constraint available in the local region with parent non-terminal X. Both partial descriptions with root Z could be considered, with the resulting Harmony (including the alignment violation) of each correctly determined.

Chapter 4. The Learnability of Optimality Theory

4.1 The Learning Problem in Optimality Theory

4.1.1 Optimality Theory and Language Acquisition in General

Substantive knowledge of language in Optimality Theory consists in knowledge of the universal set of possible inputs, outputs and *GEN* (3); of the universal constraints (4); of the ranking of those constraints particular to the language (6); and of the lexicon providing the particular underlying forms in the language-particular inputs. Under the assumption that what is universal in this knowledge is not learned but innate, what remains then to be learned is the lexicon and the constraint ranking.

The general problem of language acquisition has many challenges. Among the questions to be answered are: what kind of information is accessible to the learner from the environment, what kinds of language-specific principles must be learned, what is the structure of the lexicon, and so forth. There are some famous problems which must be faced ultimately no matter what linguistic framework is employed. For example, given an observed utterance, how can the learner segment it into morphemes, and how can the learner determine which morpheme corresponds to which linguistic element (syntactic element, semantic meaning, etc.). This chapter will not answer all of these questions.

It is commonly proposed in the literature on language acquisition that the learner needs lexical knowledge to learn the grammar, and grammatical knowledge to learn the lexicon (see, for example, Pinker 1987, Gleitman 1990, Grimshaw 1994). This suggests that the learner maintains both a hypothesized lexicon and a hypothesized grammar during learning, and is repeatedly evaluating possible lexical entries with respect to its active hypothesized grammar, and then evaluating possible grammars with respect to its active hypothesized lexicon. The learner iterates in this fashion until a stable configuration, the target language, is reached. Clearly, this high-level account raises many important questions, questions for which no one has all of the answers. It is nevertheless quite worthwhile to investigate the implications of Optimality Theory for this overall view. In particular, one can ask what procedures could be used by the learner to acquire the lexicon given a hypothesized grammar, and what procedures could learn the constraint ranking given a hypothesized lexicon. The first question is discussed very briefly in the next section, and little more will be said about it. The second question, the learning of the constraint rankings, is the subject of the rest of this chapter.

In fact, the work in this chapter addresses a subproblem of the problem of learning constraint rankings. The subproblem is that of learning rankings based upon (hypothesized) descriptions. This is not trivial, due to the highly interactive nature of the constraints in Optimality Theory. The early experience of many linguists working in Optimality Theory has been that deducing a ranking consistent with a set of analyzed data can be very difficult. This raises the question of how a child could learn the correct constraint ranking, given that the child has much more impoverished data sources. In response, the problem of learning constraint rankings is here divided into two parts. The first part is the hypothesizing of a structural description of an utterance, given the actual information (phonetic, semantic, etc.) available to the child. The second part, the part addressed by the present work, is that of inferring a constraint ranking given hypothesized descriptions.

This division is not a frequent one in work on formal language learnability. Much current work in formal learnability is done within the Principles and Parameters Framework (Chomsky 1981). The parameter settings constitute language-specific knowledge of the core grammar. A parameter setting determines an inviolable constraint on grammatical descriptions. Thus, at the level of entire descriptions, the parameters are entirely independent: a single parameter setting can rule a description ungrammatical, regardless of the settings of the other parameters. As a consequence, the problem of inferring parameter settings (and thus, of inferring the grammar) from entire structural descriptions is often perceived to be a trivial one. The only challenge, on this view, lies in contending with the underdetermination of the complete description by the actual information available to the child.

This challenge, however, is rather severe, because the parameter settings, while being independent with respect to entire descriptions, can interact in their relationship to surface phonetic forms. One response to this has been to try and restrict the parameters so as to guarantee identifiable consequences of each parameter setting individually, at the level of surface forms. Unfortunately, this results in a conflict between the goals of learnability and the goals of linguistic explanation: learnability favors parameters which do not interact, while linguistic explanation favors parameters which do interact (this is discussed below in the section on Learnability and Linguistic Theory).

In Optimality Theory, even the problem of inferring rankings from entire descriptions is not trivial, because constraint interaction is pervasive to all levels. Constraint interaction is the primary explanatory mechanism; it is entirely unavoidable. Thus, constraint interaction must be faced and dealt with by any attempt to address learning in Optimality Theory. This makes the problem of learning constraint rankings from descriptions a relevant and important one. The following situation will be assumed by the work in this chapter: the learner has access to positive data in the form of grammatical structural descriptions, including the underlying form. The goal is to find a solution to the problem of learning constraint rankings, given this information. The hope is that the solution to this problem will play an important role in a solution to the more general problem of learning constraint rankings given the kinds of information available to children.

One advantage afforded this part of the learning problem is that any results will be entirely general to the framework of Optimality Theory. Nothing specific to syllable structure or syntactic binding or any other specific phenomenon is assumed. This work is an investigation of what contributions to language learning are made by the formal structure of the framework.

4.1.2 Learning the Lexicon

Prince and Smolensky (1993, §9) give a proposal for the acquisition of the lexicon. They propose a principle of lexicon optimization. This involves having an observed phonetic form, and also several hypothesized underlying forms, each of which is mapped to a description with the observed phonetic form by the grammar. Notice that this is not the problem of choosing from among several candidate outputs for one input, but of choosing from among several inputs, each of which supports the same output. Lexicon optimization states that the underlying form with the most Harmonic description should be selected for the lexicon. The relative Harmony of the several descriptions is determined with respect to the constraint ranking of the grammar, just as when comparing candidate descriptions for a given input. In this way, the same principles of optimization which underlie the functioning of the grammar also explain the structure of the lexicon.

4.1.3 The Learning Problem: Learning Constraint Rankings

Two major concerns are perhaps most apparent. The first is that of trying to learn anything from positive data in a framework that employs violable constraints. Given a grammatical description, you may observe that it violates some of the universal constraints. But if grammatical descriptions are allowed to violate constraints, how can anything be learned from those observations? The second concern is a combinatorial one. The number of distinct total rankings is a factorial function of the number of constraints. For example, a set of 10 constraints has 10! = 3,628,800 distinct rankings. If the amount of data required to determine the correct ranking scales as the number of possible rankings, then a grammar with many constraints could require a prohibitively large amount of data to be learned successfully.

The initial data for the learning problem are pairs consisting of an input and its well-formed (optimal) description. For example, the learner of the CV language L_1 might have as an initial datum the input /VCVC/ together with its well-formed description .□V.CV.⟨C⟩ (2.d). Because *GEN* and the universal constraints are part of Universal Grammar, it is assumed that the learner is able to generate candidate structural descriptions of an input, and is also able to determine what constraint violations are assessed any description.

Together with any single piece of explicit positive evidence comes a large mass of implicit negative evidence. Every alternative description of this input is known to be ungrammatical; for example, if the description .□V.CV.⟨C⟩ (2.d) is observed, then the faithful description of the input /VCVC/, *.V.CVC. (2.a), is ungrammatical. Thus the learner knows that, with respect to the unknown constraint hierarchy,

(60) .V.CVC. ≺ .□V.CV.⟨C⟩

Furthermore, corresponding Harmonic comparisons must hold for every sub-optimal description.

Thus each single piece of positive initial data conveys a large amount of inferred comparative data of the form:

(61) [sub-optimal description *sub-opt* of input *I*] ≺ [optimal description *opt* of input *I*]

Such pairs are what feed the learning algorithms presented below. Each pair carries the information that the constraints violated by the sub-optimal description *sub-opt* must out-rank those violated by the optimal description *opt*, that, in some sense, it must be the case that *marks*(*sub-opt*) ≫ *marks*(*opt*), or more informally, *loser-marks* ≫ *winner-marks*. The algorithm now presented is nothing but a means of making this observation precise, and deducing its consequences.

4.2 The Recursive Constraint Demotion (RCD) Algorithm

Before giving a general description of the algorithm, its use is first illustrated by showing how the algorithm learns the language L_1 from the single positive example which we have discussed, /VCVC/ → .□V.CV.〈C〉, as shown in Table 1. It is then shown how the algorithm learns the different language L_2 from its (different) description of the same input.

4.2.1 An Example: Learning CV Syllable Structure

In the first language we consider, L_1, the correct description of /VCVC/ is .□V.CV.〈C〉. This fact forms our starting point, one initial datum. Table 1 gives the marks incurred by the well-formed description (labeled *d*) and by three ill-formed descriptions. From this table we form the table of *mark-data pairs*, shown as Table 17.

Table 17

Mark-data Pairs (L_1)

	sub-opt ≺ *opt*	*marks*(*sub-opt*)	*marks*(*opt*)
a ≺ *d*	.V.CVC. ≺ .□V.CV.〈C〉	{*ONS *NOCODA}	{*PARSE *FILL^{Ons}}
b ≺ *d*	〈V〉.CV.〈C〉 ≺ .□V.CV.〈C〉	{*PARSE *PARSE}	{*PARSE *FILL^{Ons}}
c ≺ *d*	〈V〉.CV.C□́. ≺ .□V.CV.〈C〉	{*PARSE *FILL^{Nuc}}	{*PARSE *FILL^{Ons}}

In order that each sub-optimal description *sub-opt* be less harmonic than the optimal description *opt*, the marks incurred by the former, *marks*(*sub-opt*) must collectively be worse than *marks*(*opt*). What this means more precisely is that *sub-opt* must incur the worst uncancelled mark, compared to *opt*. So the first step in the learning algorithm is to cancel the common marks in Table 17, as shown in Table 18:

Table 18

Mark-data Pairs after Cancellation (L_1)

	sub-opt ≺ *opt*	*loser-marks* = *marks′*(*sub-opt*)	*winner-marks* = *marks′*(*opt*)
a ≺ *d*	.V.CVC. ≺ .□V.CV.⟨C⟩	{*ONS *NOCODA}	{*PARSE *FILL^{Ons}}
b ≺ *d*	⟨V⟩.CV.⟨C⟩ ≺ .□V.CV.⟨C⟩	{~~*PARSE~~ *PARSE}	{~~*PARSE~~ *FILL^{Ons}}
c ≺ *d*	⟨V⟩.CV.C□́. ≺ .□V.CV.⟨C⟩	{~~*PARSE~~ *FILL^{Nuc}}	{~~*PARSE~~ *FILL^{Ons}}

The canceled marks have been struck out. Note that the cancellation operation which transforms *marks* to *marks′* is only well-defined on *pairs* of sets of marks; e.g., *PARSE is canceled in the pairs *b* ≺ *d* and *c* ≺ *d*, but not in the pair *a* ≺ *d*. Note also that cancellation of marks is done token-by-token: in the row *b* ≺ *d*, one but not the other mark *PARSE in *marks*(*b*) is canceled. As we will see, the algorithm is sensitive not to absolute numbers of marks, but only to whether *sub-opt* or *opt* incurs more marks of a given type (Optimality Theory recognizes a relative distinction between more- or less-violation of a constraint, but not an absolute quantitative measure of degree of constraint violation.). The Mark-data Pairs after cancellation are the data on which the rest of the algorithm operates.

The second part of the algorithm proceeds recursively, finding first the constraints that may be ranked highest while being consistent with the mark-data pairs, then eliminating those constraints from the problem and starting over again to rank the remaining, lower, constraints. Conceived as a sequence of passes, the first pass through the data determines the highest-ranking constraints, the next pass the next-highest ranking constraints, and so forth down the hierarchy. If the data provide enough information to completely determine the total ranking, then only one constraint will be returned by each pass. In general, however, the result of the algorithm will be a *stratified hierarchy* of the form:

(62) Stratified Domination Hierarchy

$$\{\mathbb{C}_1, \mathbb{C}_2, ..., \mathbb{C}_3\} \gg \{\mathbb{C}_4, \mathbb{C}_5, ..., \mathbb{C}_6\} \gg ... \gg \{\mathbb{C}_7, \mathbb{C}_8, ..., \mathbb{C}_9\}$$

The constraints $\mathbb{C}_1$, $\mathbb{C}_2$, ..., $\mathbb{C}_3$ comprise the first stratum in the hierarchy: they are not ranked with respect to one another, but they each dominate all the remaining constraints. A stratified hierarchy may not totally Harmonically order all candidate descriptions for a given input, since it fails to specify the relative ranking of constraints which may conflict. This has the consequence, discussed below, that the output of the algorithm may fail to decide between candidates for which relevant training data is unavailable. Note: Henceforth, 'hierarchy' will mean 'stratified hierarchy'; when appropriate, hierarchies will be explicitly qualified as 'totally ranked.'

The initial state of the learner can be taken to be the completely degenerate stratified hierarchy in which all the constraints are lumped together in a single stratum. Learning proceeds to refine this hierarchy into a sequence of smaller strata. Each pass of the algorithm begins with a set of not-yet-ranked constraints, and ends by selecting some of them to constitute the next-lower stratum in the hierarchy. At each step, mark-data pairs which are accounted for by the already-ranked constraints are eliminated so that only not-yet-ranked constraints are left.

The key observation is this: for any given pair, if an uncancelled mark is incurred by the winner, then it must be dominated by a mark incurred by the loser: otherwise, the winner wouldn't win (this is the Cancellation/Domination Lemma of Prince and Smolensky (1993, 130, 148, 221)). Therefore, any constraint assessing an uncancelled mark to the winner cannot be among the highest ranked constraints in the set, and cannot be output as part of the next stratum. Conversely, any constraint that does *not* assess a mark to any winner *can* enter the next stratum, since no evidence exists that it is dominated by any of the remaining constraints.

We illustrate how this process deduces the hierarchy for L_1 from the data in Table 18.

When the algorithm begins, the not-yet-ranked constraints comprise the entire universal set:

(63) *not-yet-ranked-constraints* = {ONS, NOCODA, PARSE, FILL^{Nuc}, FILL^{Ons}}

Examining the rightmost column of the mark-data table (Table 18) we see that two marks, *PARSE and *FILL^{Ons}, appear in the list of uncancelled winner marks. So the two constraints PARSE and FILL^{Ons} must be dominated by other constraints (those violated by the corresponding losers); they cannot be the highest-ranked of the *not-yet-ranked-constraints*. This leaves:

(64) *highest-ranked-constraints* = {ONS, NOCODA, FILL^{Nuc}}

which constitutes the output of the first pass: these three constraints form the highest stratum in the hierarchy. The data do not support any distinctions in ranking among the three, so none are made. Now

(65) *not-yet-ranked-constraints* = {PARSE, FILL^{Ons}}

Now that the highest ranked constraints have been determined, the list of mark-data pairs can be trimmed down by removing any mark-data pairs that are completely accounted for by the constraints selected as highest. This is the case if at least one of the marks incurred by the loser of a pair is among the highest ranked constraints. Such a mark is guaranteed to dominate all of the corresponding winner's marks, because all of the winner's marks were disqualified from being ranked highest.

So we eliminate from the mark-data table every row in which any of the highest-ranked constraints appear. In the current example, we eliminate the pair $a \prec d$ because *ONS appears (or, alternatively, because *NOCODA appears), and also the pair $c \prec d$, because *FILL^{Nuc} appears. The new mark-data table is thus:

Table 19

Mark-data Pairs (L_1, After First Pass)

	sub-opt ≺ *opt*	*loser-marks*	*winner-marks*
$b \prec d$	⟨V⟩.CV.⟨C⟩ ≺ .□V.CV.⟨C⟩	{~~*PARSE~~ *PARSE}	{~~*PARSE~~ *FILL^{Ons}}

At the end of the first pass, we now have the first (highest) stratum (64), a reduced list of not-yet-ranked constraints (65), and a reduced mark-data table (Table 19). Crucially, the reduced mark-data table involves only the *not-yet-ranked-constraints*, so we can now recursively invoke the same algorithm, using the remaining data to rank the remaining constraints. This initiates the next pass.

Repeating this process with the reduced mark-data table (Table 19), we examine the rightmost column of the table, and observe that of the two *not-yet-ranked-constraints* (65), only one, FILL$^{\text{Ons}}$, appears. The remaining constraint, then, is output as the next stratum of highest-ranked constraints:

(66) *highest-ranked-constraints* = {PARSE}

This leaves

(67) *not-yet-ranked-constraints* = {FILL$^{\text{Ons}}$}

The final step of the second pass is to trim the mark-data table, eliminating rows in which the *highest-ranked-constraints* appear. This eliminates the only row in the table, so that the new mark-data table is empty.

Table 20

Mark-data Pairs (L_1, After Second Pass): none

The result of the first two passes is the highest segment of the stratified hierarchy:

(68) {ONS, NOCODA, FILL$^{\text{Nuc}}$} $\gg$ {PARSE}

The third pass operates on an empty mark-data table. Since there are no marks in the rightmost column of such a table, no remaining constraints must be dominated: all the not-yet-ranked constraints are output as the highest-ranked. In this case, that is the one remaining constraint FILL$^{\text{Ons}}$.

(69) *highest-ranked-constraints* = {FILL$^{\text{Ons}}$}

This leaves

(70) *not-yet-ranked-constraints* = {}

so the algorithm terminates, with the final result:

(71) Learned Stratified Hierarchy for L_1:

{ONS, NOCODA, FILL$^{\text{Nuc}}$} $\gg$ {PARSE} $\gg$ {FILL$^{\text{Ons}}$}

This result represents a class of totally-ranked constraint hierarchies which give rise to the target language L_1: the same optimal outputs arise regardless of the ranking of the three highest constraints. One of these refinements of the learned stratified hierarchy (71) is the particular total ranking given in (7): this is but one of the correct hierarchies for the target language.

It is easy to see how the course of learning L_2 differs from that of L_1, assuming the learner's initial datum is the description of the same input, /VCVC/, which is now ⟨V⟩.CV.C□́. (candidate 2.*c*). The mark-data table used by the algorithm, after cancellation, is now:

Table 21

Mark-data Pairs after Cancellation (L_2)

sub-opt ≺ opt		*loser-marks*	*winner-marks*
a ≺ c	.V.CVC. ≺ ⟨V⟩.CV.C□́.	{*Ons *NoCoda}	{*Parse *Fill$^{\text{Nuc}}$}
b ≺ c	⟨V⟩.CV.⟨C⟩ ≺ ⟨V⟩.CV.C□́.	{~~*Parse~~ *Parse}	{~~*Parse~~ *Fill$^{\text{Nuc}}$}
d ≺ c	.□V.CV.⟨C⟩ ≺ ⟨V⟩.CV.C□́.	{~~*Parse~~ *Fill$^{\text{Ons}}$}	{~~*Parse~~ *Fill$^{\text{Nuc}}$}

This table is identical to its L_1 counterpart (Table 18) except that the marks *Fill$^{\text{Nuc}}$ and *Fill$^{\text{Ons}}$ are interchanged. The result of the algorithm is therefore the same as before, (71), with this exchange made.

(72) Learned Stratified Hierarchy for L_2:

{Ons, NoCoda, Fill$^{\text{Ons}}$} ≫ {Parse} ≫ {Fill$^{\text{Nuc}}$}

Again, this stratified hierarchy is correct: its further refinements into totally-ranked hierarchies, including the one we singled out in (7), all give rise to L_2.

That these CV languages can each be learned completely from a single positive example attests to the power of the implicit negative data which comes with each positive example in Optimality Theory.

4.2.2 General Statement of the Recursive Constraint Demotion Algorithm

The RCD algorithm operates on pairs of constraint violation marks. It is here assumed that the learner has already collected all of the observed (optimal) descriptions, selected one or more suboptimal competitors for each optimal description, and determined the constraint violation marks assessed to each description. How the learner might select appropriate suboptimal competitors for an optimal description will be discussed at length below, in the section on the error-driven learning algorithm.

Given:

universal-constraints = a set of universal constraints

mark-data = a table of pairs of mark lists (*loser-marks*, *winner-marks*)

To Find:

A stratified hierarchy with respect to which each of the *loser-marks* is less Harmonic than its corresponding *winner-marks*.

RCD Algorithm:

Set *not-yet-ranked-constraints* = *universal-constraints*

I. Mark Cancellation

For each pair (*loser-marks*, *winner-marks*) in *mark-data*:

a. For each occurrence of a mark *ℂ in both *loser-marks* and *winner-marks* in the same pair, remove that occurrence of *ℂ from both.

b. If, as a result, no *winner-marks* remain, remove the pair from *mark-data*.

II. Recursive Ranking

a. Output *highest-ranked-constraints* = all the constraints in *not-yet-ranked-constraints* which do not have a mark in the column *winner-marks* of *mark-data*; these form the highest-ranked stratum of the *not-yet-ranked constraints*.

b. Remove the *highest-ranked-constraints* from the *not-yet-ranked-constraints*.

c. Remove all rows from *mark-data* which contain any marks assessed by the *highest-ranked-constraints*.

d. Call Recursive Ranking again, with the remaining *mark-data* and the remaining *not-yet-ranked-constraints*.

Note that in step c of Recursive Ranking, the relevant marks (those assessed by the *highest-ranked-constraints*) can only appear in the column *loser-marks*; for any constraint contributing a mark to the column *winner-marks* is not, by step a, among the relevant constraints (those in *highest-ranked-constraints*).

4.2.3 Informal Analysis of the Algorithm

Observe first that multiple uncancelled tokens of the same type of mark in the *mark-data* table, either for winner or loser, have the same effect as a single token. For in Recursive Ranking step a, we simply determine which constraints assess no marks at all in the *winner-marks* column: whether a single or multiple tokens of a mark appear makes no difference. Then in Recursive Ranking step b, a row is removed from *mark-data* if it contains any marks at all assessed by the *highest-ranked-constraints*; multiple tokens of a mark type have the same effect as a single token. Thus, for efficiency considerations below, we can assume that in Mark Cancellation step a, if a row of the *mark-data* table contains multiple tokens of the same type of mark after cancellation, duplicates are eliminated, leaving at most one token of each type. In other words, in the initial mark-data table prior to cancellation, what really matters is, for each constraint, which of *sub-opt* or *opt* incurs more violations of the constraint $\mathbb{C}$; if it is *sub-opt*, then a token of the mark $*\mathbb{C}$ appears in the column *loser-marks*; if it is *opt*, then the token of $*\mathbb{C}$ appears instead in the column *winner-marks*. What matters in the assessment of optimality is only which of two candidates more seriously violates each constraint, not any absolute magnitude of violation.

The correctness of the algorithm should be clear. The *highest-ranked-constraints* output at each pass of the algorithm are exactly the constraints which need not be dominated in order to explain the available *data*; the remaining *not-yet-ranked-constraints*, by contrast, must be dominated and so cannot be highest-ranked. We now show that the algorithm must terminate.

On each pass of the algorithm, at least one constraint must be output. For suppose not. That would mean that every one of the *not-yet-ranked-constraints* appears in the column *winner-marks*, i.e., as an uncancelled mark of an optimal description. But that would mean that every one of the *not-yet-ranked-constraints* must be dominated by one of the other *not-yet-ranked-constraints*: which means there is no ranking of these constraints which is consistent with the *mark-data*, in contradiction to the basic assumption that the *mark-data* derive from some ranking of the given constraints. Notice that RCD indicates when the data are inconsistent, by failing to output any constraints at the end of a pass while some constraints have not yet been ranked.

So on each pass, at least one constraint is eliminated from the *not-yet-ranked-constraints*. Thus the number of passes required cannot be more than the number of universal constraints: call this N_{constr}. The number of steps required for each pass cannot exceed the number of uncancelled marks in the *mark-data* table: each mark in the column *winner-marks* is examined in Recursive Ranking step a to ensure that its corresponding constraint will not be output as a *highest-ranked-constraint*, and, in the worst case, each mark in the column *loser-marks* must be examined in Recursive Ranking step c to determine which rows must be eliminated from *data*. The number of uncancelled marks per row of the table can't exceed the number of constraints N_{constr}, so the total number of steps per pass can't exceed $N_{constr}N_{pairs}$, where N_{pairs} is the number of *mark-data* pairs used. Therefore, the number of steps required for all the passes can't exceed $(N_{constr})^2N_{pairs}$. In the worst case, the RCD algorithm is quadratic in the number of constraints, and linear in the number of mark-data pairs.

4.3 General Constraint Demotion

In this section, we pull out the core part of the RCD algorithm, which we call the Core CD Algorithm, and show how it can be used in a number of ways to yield a family of CD Algorithms which differ primarily in how much data they process at one time.

4.3.1 The Core CD Algorithm

Given:

$\mathcal{H}$ = a stratified hierarchy of constraints

mark-data = a table of pairs of mark lists (*loser-marks*, *winner-marks*)

To Find:

A stratified hierarchy with respect to which each of the *loser-marks* is less harmonic than its corresponding *winner-marks*.

Core CD Algorithm

```
I. Mark Cancellation
    For each pair (loser-marks, winner-marks) in mark-data:
        a. For each occurrence of a mark $*\mathbb{C}$ in both loser-marks and winner-marks in the same
            pair, remove that occurrence of $*\mathbb{C}$ from both.
        b. If, as a result, no winner-marks remain, remove the pair from mark-data.
II. Constraint Demotion
Repeat the following until no demotions occur:
    For each pair (loser-marks, winner-marks) in mark-data
        a. find the mark $*\mathbb{C}_{loser}$ in loser-marks which is highest-ranked in $\mathcal{H}$
        b. for each $*\mathbb{C}_{winner}$ in winner-marks
            if $\mathbb{C}_{loser}$ does not dominate $\mathbb{C}_{winner}$ in $\mathcal{H}$
                $\mathbb{C}_{winner}$ to the stratum immediately below $\mathbb{C}_{loser}$
                (creating such a stratum if it does not already exist)
    Output $\mathcal{H}$
```

4.3.2 Versions of the CD Algorithm

Now we present several ways of using the Core CD Algorithm to learn a language. All start, like RCD, with the stratified hierarchy in which all the universal constraints are in the top stratum: we call this $\mathcal{H}_0$. The constraints in this stratified hierarchy are then demoted as demanded by the data.

(73) Batch CD

a. Compile all available description pairs (*subopt, opt*).

b. Set *mark-data* to {(*marks(subopt), marks(opt)*)}

c. Set $\mathcal{H} = \mathcal{H}_0$

d. Execute Core CD on *mark-data* and $\mathcal{H}$ (once).

Batch CD is closely related to RCD (see discussion below): all the data are processed simultaneously by Core CD.

(74) On-line CD

```
a. Set $\mathcal{H} = \mathcal{H}_0$
  Execute repeatedly:
      b. Select one pair (*subopt*, *opt*).
      c. Set *mark-data* = {(*marks*(*subopt*), *marks*(*opt*))}.
      d. Execute Core CD on *mark-data* and current $\mathcal{H}$.
```

On-line CD operates by applying Core CD separately to each (*subopt*, *opt*) pair.

(75) I/O CD

```
a. Set $\mathcal{H} = \mathcal{H}_0$
  Execute repeatedly:
      b. Select several pairs (*subopt*, *opt*) for one optimal description *opt*.
      c. Set *mark-data* = {(*marks*(*subopt*), *marks*(*opt*))}
      d. Execute Core CD on *mark-data* and current $\mathcal{H}$.
```

I/O CD is one way of using Core CD in between the extremes of Batch CD (all data processed at once) and On-Line CD (one mark-data pair processed at a time). One description *opt* from the language is processed at a time, but a number of competitors *subopt* to *opt* may be processed at once.

4.3.3 Example

We exemplify On-Line CD using the language L_1 above. Recall that On-Line CD operates on one (*loser-marks*, *winner-marks*) pair at a time.

Step a of On-Line CD has us set the initial stratified hierarchy to

(76) $\mathcal{H} = \mathcal{H}_0 = \{\textsc{Parse}, \textsc{Fill}^{\text{Nuc}}, \textsc{Fill}^{\text{Ons}}, \textsc{Ons}, \textsc{NoCoda}\}$

Step b has us pick a (*subopt*, *opt*) pair; suppose we pick $b \prec d$ of (2). Step c has us generate the corresponding pair of mark lists, as shown in the second row of Table 17. Step d has us execute Core CD.

Step I of Core CD is Mark Cancellation; the result is:

Table 22

Mark-pair for On-Line CD, Step 1 (L_1)

sub-opt ≺ *opt*		*loser-marks*	*winner-marks*
b ≺ *d*	〈V〉.CV.〈C〉 ≺ .□V.CV.〈C〉	{~~*PARSE~~ *PARSE}	{~~*PARSE~~ *FILL^{Ons}}

Step II of Core CD is Constraint Demotion. We first find the highest-ranked (uncancelled) mark $*\mathbb{C}_{loser}$ in $\mathcal{H}$; this is *PARSE. We next check to see whether the *winner-marks* are dominated by *PARSE. The only winner mark is $*\mathbb{C}_{winner}$ = *FILL^{Ons}, which is *not* so dominated. So Core CD requires the demotion of $\mathbb{C}_{winner}$ = FILL^{Ons} to the stratum immediately below $\mathbb{C}_{loser}$ = PARSE. Since no such stratum currently exists, we create it. The hierarchy is now

(77) $\mathcal{H}$ = {PARSE, FILL^{Nuc}, ONS, NOCODA} ≫ {FILL^{Ons}}

This completes the execution of Core CD, so we return to step b of the On-Line CD procedure: picking another (*subopt*, *opt*) pair. Suppose this is *a* ≺ *d* of (2):

Table 23

Mark-pair for On-Line CD, Step 2 (L_1)

sub-opt ≺ *opt*		*loser-marks*	*winner-marks*
a ≺ *d*	.V.CVC. ≺ .□V.CV.〈C〉	{*ONS *NOCODA}	{*PARSE *FILL^{Ons}}

We proceed to the Constraint Demotion step of CD, which calls first for finding the highest-ranked of the *loser-marks*; since ONS and NOCODA are both top ranked, either will do: choose, say, ONS. We next check whether each of *winner-marks* is dominated by ONS. FILL^{Ons} is so dominated, so no demotion results. PARSE is not, however, so it is demoted to the stratum immediately below that of ONS; the hierarchy is now:

(77) $\mathcal{H}$ = {FILL^{Nuc}, ONS, NOCODA} ≫ {PARSE, FILL^{Ons}}

Suppose now that the next *sub-opt* ≺ *opt* pair is:

Table 24

Mark-pair for On-Line CD, Step 3 (L_1)

sub-opt ≺ *opt*		*loser-marks*	*winner-marks*
c ≺ *d*	⟨V⟩.CV.C□́. ≺ .□V.CV.⟨C⟩	{~~*Parse~~ *Fill^{Nuc}}	{~~*Parse~~ *Fill^{Ons}}

Since the uncancelled loser mark, *Fill^{Nuc} already dominates the uncancelled winner mark, *Fill^{Ons}, no demotion results, and $\mathcal{H}$ is unchanged. This is an example of an *uninformative* pair, given its location in the sequence of training pairs, since no demotions result.

Suppose the next *sub-opt* ≺ *opt* pair results from a new input, /VC/:

Table 25

Mark-pair for On-Line CD, Step 4 (L_1)

sub-opt ≺ *opt*	*loser-marks*	*winner-marks*
⟨VC⟩ ≺ .□V.⟨C⟩	{~~*Parse~~ *Parse}	{~~*Parse~~ *Fill^{Ons}}

Since the loser mark *Parse does not dominate the winner mark *Fill^{Ons}, we must demote Fill^{Ons} to the stratum immediately below Parse, resulting in the hierarchy:

(79) $\mathcal{H}$ = {Fill^{Nuc}, Ons, NoCoda} ≫ {Parse} ≫ {Fill^{Ons}}

This stratified hierarchy actually gives rise exactly to L_1, so no further data will be informative. It is the same hierarchy as that learned by RCD, (71).

4.3.4 Restrictions on Rankings

Some analyses within Optimality Theory have proposed further restrictions on the rankings of the universal constraints, as a part of Universal Grammar. An example is the Universal Syllable Theory in Prince and Smolensky (1993, §8). The question of how to incorporate restrictions into CD algorithms, and what kinds of restrictions can be accommodated, is open.

The Universal Syllable Theory in Prince and Smolensky (1993, §8) uses a particular type of ranking restriction: a universal markedness scale. A subset of the universal constraints have their relative rankings fixed by universal grammar. The absolute location of these constraints is not fixed, and the ranking of other constraints not in the markedness scale is entirely free. Thus, as long as the relations of the markedness are preserved in the final learned hierarchy, the restriction is met. One possibility for learning such grammars is to use, as the starting hierarchy, one with all constraints in the top stratum except for those in a markedness scale. The constraints in the markedness scale start with their relative rankings set, with the top constraint in the top stratum, the second constraint of the scale in the second stratum, and so forth. Because the relative rankings of the scale must be respected in any target hierarchy, the position of each constraint in this initial hierarchy is at or above its position in all possible target hierarchies[1]. What remains is to ensure that constraint demotion does not result in a hierarchy with the constraints out of order relative to the markedness scale. This could be accomplished by observing when a constraint being demoted belongs to a markedness scale, and ensuring that after demotion it still dominates all constraints below it in the scale (demoting the other constraints if necessary). Correctness would then follow, by the analysis presented in section 4.4.

4.3.5 Input Errors

The CD algorithms assume that the input data contain no errors. It remains to be explored what kinds of errors the CD algorithms might be resistant to, and what reasonable modifications might increase the ability to overcome input errors. RCD is capable of detecting when inconsistencies exist in the input; it is less clear how On-Line CD will behave when given inconsistent input. The issue for the on-line case is h-domination (see section 4.4): if a constraint is accidentally demoted below the stratum it occupies in the target hierarchy, then the resulting hierarchy does not h-dominate the target hierarchy, and the formal proof no longer applies.

[1] Formally, the crucial concept is that of *h-domination*, defined in section 4.4; this proposed initial hierarchy h-dominates all possible target hierarchies.

4.4 Formal Analysis of Constraint Demotion

4.4.1 Stratified Hierarchies

(80) **Def.** A *stratum* is a set of constraints. A *stratified hierarchy* is a linearly ordered set of strata which partition the universal constraints. A hierarchy distinguishes one stratum as the top stratum. Each stratum other than the top stratum is immediately dominated by exactly one other stratum. The top stratum immediately dominates the second stratum, which immediately dominates the third stratum, and so forth.

(81) **Def.** A *total ranking* is a stratified hierarchy where each stratum contains precisely one constraint.

(82) **Def.** A constraint $\mathbb{C}_1$ is said to dominate constraint $\mathbb{C}_2$, denoted $\mathbb{C}_1 \gg \mathbb{C}_2$, in hierarchy $\mathcal{H}$ if the stratum containing $\mathbb{C}_1$ dominates the stratum containing $\mathbb{C}_2$ in hierarchy $\mathcal{H}$.

(83) **Def.** The *offset* of a constraint $\mathbb{C}$ in a hierarchy $\mathcal{H}$ is the number of strata that dominate the stratum containing $\mathbb{C}$. $\mathbb{C}$ is *in a lower stratum* in $\mathcal{H}_1$ than in $\mathcal{H}_2$ if the offset of $\mathbb{C}$ in $\mathcal{H}_1$ is greater than in $\mathcal{H}_2$. $\mathbb{C}$ is *in the same stratum* in $\mathcal{H}_1$ and $\mathcal{H}_2$ if it has the same offset in both.

(84) **Def.** A constraint hierarchy $\mathcal{H}_1$ *h-dominates* $\mathcal{H}_2$ if every constraint is in the same or a lower stratum in $\mathcal{H}_2$ than in $\mathcal{H}_1$.

(85) **Def.** A constraint hierarchy $\mathcal{H}_2$ is called a *refinement* of $\mathcal{H}_1$ if every domination relation $\mathbb{C} \gg \mathbb{C}'$ of $\mathcal{H}_1$ is preserved in $\mathcal{H}_2$.

(86) **Def.** $\mathcal{H}_0$ denotes the stratified hierarchy with all of the constraints in the top stratum.

(87) **Lemma** $\mathcal{H}_0$ h-dominates all hierarchies.

Proof

$\mathcal{H}_0$ h-dominates itself, because h-domination is reflexive (h-domination is satisfied by constraints that are in the same stratum in both hierarchies). Consider some constraint $\mathbb{C}$ in some hierarchy $\mathcal{H}$. $\mathbb{C}$ is either in the top stratum of $\mathcal{H}$, and thus in the same stratum as in $\mathcal{H}_0$, or it is in some lower stratum of $\mathcal{H}$, and thus in a lower stratum than in $\mathcal{H}_0$. Therefore, $\mathcal{H}_0$ h-dominates all hierarchies.

4.4.2 The Target Stratified Hierarchy

(88) **Def.** The *h-dominant target stratified hierarchy*, or simply the 'target,' for a language *L* generated by a total ranking, is denoted $\mathcal{H}_L$, and is defined as follows. The top stratum of the target contains precisely those constraints which never assess uncancelled marks to any optimal description in *L*. The second stratum consists of precisely those constraints which assess uncancelled marks only to optimal descriptions relative to competitors which are assessed at least one uncancelled mark by a constraint in the top stratum. Each stratum consists of precisely those constraints which (a) cannot occur higher in the target, and (b) only assess uncancelled marks to optimal descriptions relative to competitors assessed an uncancelled mark by at least one constraint ranked higher in the target.

(89) **Lemma** For any *L* generated by a total ranking, $\mathcal{H}_L$ exists and is unique.

Proof

Existence follows from the definition, and the assumption that *L* is generated by at least one total ranking of the constraints. The top stratum of $\mathcal{H}_L$ is guaranteed to contain at least the constraint ranked highest in the total ranking. Among the constraints not placed in the top stratum of $\mathcal{H}_L$, one dominates all the remaining others in the total ranking, and is thus guaranteed to meet the requirements for placement in the second stratum. The same logic, applied to subsequent strata, shows that all of the constraints will be placed in a stratum in $\mathcal{H}_L$.

Uniqueness is guaranteed because a constraint cannot meet the requirements for placement in more than one stratum in the hierarchy, because meeting the requirements for one stratum automatically disqualifies it for any lower strata.

(90) **Lemma** Each constraint $\mathbb{C}$ with offset n>0 in $\mathcal{H}_L$ for a language generated by a total ranking has the following property. There must exist an optimal description *opt* with a competing suboptimal description *subopt* such that $\mathbb{C}$ assesses an uncancelled mark to *opt*, *subopt* is assessed an uncancelled mark by a constraint $\mathbb{C}_{n-1}$ with offset precisely n-1, and *subopt* is not assessed any uncancelled marks by any constraints with offset less than n-1.

Proof

Consider some constraint $\mathbb{C}_n$ with offset n>0 in target $\mathcal{H}_L$. Suppose, to the contrary, that no such pair (*subopt*, *opt*) exists for $\mathbb{C}_n$. Recall that if $\mathbb{C}_n$ assesses an uncancelled mark to an optimal description relative to some suboptimal competitor, it must be dominated by some other constraint which assesses an uncancelled mark to the suboptimal competitor, for otherwise the optimal description would not be more Harmonic, and the correct language would not be generated.

One possibility is that $\mathbb{C}_n$ never assesses an uncancelled mark to any optimal description. But then it would have offset 0 in $\mathcal{H}_L$, contradicting the assumption that it has offset greater than 0. The other possibility is that for any *opt* assessed an uncancelled mark by $\mathbb{C}_n$ relative to some *subopt*, *subopt* is assessed an uncancelled mark by a constraint with offset smaller than n-1. But then $\mathbb{C}_n$ could be placed one stratum higher in $\mathcal{H}_L$, with resulting offset n-1, and the resulting hierarchy would generate the same language, contradicting the fact that by definition every constraint is ranked as high as possible in $\mathcal{H}_L$.

Therefore, the supposition must be false, and an appropriate pair must exist.

(91) **Theorem** For any language *L* generated by a total ranking, $\mathcal{H}_L$ has the property that each constraint is ranked as high as possible; that is, $\mathcal{H}_L$ h-dominates every hierarchy $\mathcal{H}'$ which generates *L*.

Proof

Consider a (stratified) hierarchy $\mathcal{H}'$ which generates *L*.

Consider a constraint $\mathbb{C}$ with offset 0 in $\mathcal{H}'$. $\mathbb{C}$ must not assess an uncancelled mark to any optimal description in the language; otherwise, $\mathcal{H}'$ would not generate the language. Therefore, $\mathbb{C}$ must have offset 0 in $\mathcal{H}_L$. It follows that $\mathbb{C}$ has offset in $\mathcal{H}_L$ <= $\mathbb{C}$'s offset in $\mathcal{H}'$, as both are 0.

Assume that each constraint with offset <= n in $\mathcal{H}'$ is in the same or higher stratum in $\mathcal{H}_L$. Consider a constraint $\mathbb{C}_{n+1}$ with offset n+1 in $\mathcal{H}'$. For any pair of an optimal description *opt* with a suboptimal competitor *subopt*, if $\mathbb{C}_{n+1}$ assesses an uncancelled mark to *opt*, *subopt* must be assessed an uncancelled mark by a constraint $\mathbb{C}$ with offset <= n in $\mathcal{H}'$ (that is, $\mathbb{C} \gg \mathbb{C}_{n+1}$ in $\mathcal{H}'$); otherwise, $\mathcal{H}'$ would not generate the language. By hypothesis, any constraint with offset <= n in $\mathcal{H}'$ has offset <= n in $\mathcal{H}_L$. Therefore, $\mathbb{C}_{n+1}$ has offset <= n+1 in $\mathcal{H}_L$.

By mathematical induction, every constraint in $\mathcal{H}'$ is in the same or higher stratum in $\mathcal{H}_L$. It follows directly that every constraint is in the same or lower stratum in $\mathcal{H}'$ than in $\mathcal{H}_L$. Therefore, target $\mathcal{H}_L$ h-dominates $\mathcal{H}'$.

(92) **Corollary** $\mathcal{H}_L$ h-dominates every total constraint ranking which generates *L*.

4.4.3 The Constraint Demotion Algorithm

(93) **Def.** The mark cancellation procedure, MarkCancel({(*loser-marks*, *winner-marks*)}), is:

```
For each pair (loser-marks, winner-marks)
    For each occurrence of *ℂ in both loser-marks and winner-marks
        Remove that occurrence of *ℂ from both lists
    If winner-marks is empty, delete the pair
Return ({loser-marks, winner-marks})
```

(94) **Def.** The constraint demotion procedure, CD({(*loser-marks*, *winner-marks*)},$\mathcal{H}$), is:

```
Set $\mathcal{H}'$ to $\mathcal{H}$
Set mark-data to MarkCancel({(loser-marks, winner-marks)})
Repeat
    For each pair (loser-marks, winner-marks) in mark-data
        Find the constraint $\mathbb{C}_l$ with a mark in loser-marks ranked highest in $\mathcal{H}'$
        For each $\mathbb{C}_w$ with a mark in winner-marks
            If $\mathbb{C}_l$ does not dominate $\mathbb{C}_w$ in $\mathcal{H}'$
                Demote $\mathbb{C}_w$ to the stratum immediately below $\mathbb{C}_l$
Until no demotions occur during a pass
Return ($\mathcal{H}'$)
```

(95) **Lemma** The hierarchy output by CD is h-dominated by the input hierarchy.

Proof

Because constraint demotion only demotes constraints, each constraint is in either the same or lower stratum in the output hierarchy than it was in the input hierarchy.

(96) **Lemma** If the input hierarchy h-dominates $\mathcal{H}_L$, so does the output hierarchy.

Proof

This holds because CD will never demote a constraint lower than necessary. Let $\mathbb{C}_w$ be some constraint demoted by CD. Then there is a mark-data pair (*subopt*, *opt*) requiring that $\mathbb{C}_w$ be dominated by one of the constraints assessing uncancelled marks to *subopt*. Let $\mathbb{C}_l$ be the one with the smallest offset (highest ranked) in $\mathcal{H}$, the input hierarchy, and let n denote its offset. By assumption, $\mathcal{H}$ h-dominates $\mathcal{H}_L$, so $\mathbb{C}_l$ in $\mathcal{H}_L$ has offset >= n. Thus, every constraint assessing an uncancelled mark to *subopt* must have offset >=n. Therefore, $\mathbb{C}_w$ must have offset at least n+1 in $\mathcal{H}_L$. CD demotes $\mathbb{C}_w$ to the stratum immediately below the one containing $\mathbb{C}_l$, so $\mathbb{C}_w$ has offset n+1 in the resulting hierarchy. Thus, $\mathbb{C}_w$ has offset in the output hierarchy less than or equal to its offset in $\mathcal{H}_L$, guaranteeing that the output hierarchy h-dominates $\mathcal{H}_L$.

(97) **Def.** An *informative pair* for a hierarchy $\mathcal{H}'$ is a mark-data pair that, when given as input to CD along with $\mathcal{H}'$, causes at least one demotion to occur. The property of being informative is jointly determined by the mark-data pair and the hierarchy being evaluated.

(98) **Def.** The *h-distance* between a hierarchy $\mathcal{H}_1$ and a hierarchy $\mathcal{H}_2$ h-dominated by $\mathcal{H}_1$ is the sum, over all constraints $\mathbb{C}$, of the difference between the offset of $\mathbb{C}$ in $\mathcal{H}_1$ and in $\mathcal{H}_2$.

(99) **Lemma** Suppose the input hierarchy h-dominates $\mathcal{H}_L$. The *h-distance* between the output hierarchy and $\mathcal{H}_L$ is decreased by at least one (from the h-distance between the input hierarchy and $\mathcal{H}_L$) for each demotion.

Proof

By lemma (95), the input hierarchy h-dominates the output hierarchy. Let $\mathbb{C}$ be a constraint that is demoted, with offset n in the input hierarchy, offset m in the output hierarchy, and offset t in $\mathcal{H}_L$. $\mathbb{C}$ is demoted, so m>n. By lemma (96), the output hierarchy h-dominates $\mathcal{H}_L$, so t>m>n. Therefore, (t-m)<(t-n), so the contribution of $\mathbb{C}$ to h-distance is smaller for the output hierarchy. Thus, the output hierarchy h-distance is at least one less for each constraint demoted.

(100) **Lemma** The h-distance from $\mathcal{H}_0$ to $\mathcal{H}_L$ cannot exceed

$$½(N_{constr} - 1)N_{constr}.$$

Proof

By lemma (87), $\mathcal{H}_0$ h-dominates every hierarchy, and therefore must h-dominate $\mathcal{H}_L$. The greatest h-distance will be when $\mathcal{H}_L$ is a totally ranked hierarchy. The furthest constraint from the top stratum will be the one in the bottom stratum, which has offset (N_{constr}-1). The next lowest constraint has offset (N_{constr}-2), and so forth. Thus, the h-distance will be:

$(N_{constr}-1)+(N_{constr}-2)+...+1+0$ which is precisely $½(N_{constr} - 1)N_{constr}$.

(101) **Theorem** Starting with $\mathcal{H}_0$ and repeatedly presenting the CD algorithm with sets of mark-data pairs, the procedure converges on the target hierarchy $\mathcal{H}_L$ after at most $½(N_{constr} - 1)N_{constr}$ informative pairs.

Proof

By lemma (99), each informative pair reduces the h-distance by at least one. Therefore the target hierarchy is converged upon after a number of informative pairs that is at most the h-distance between $\mathcal{H}_0$ and the target. Lemma (100) guarantees that this distance is at most ½(N_{constr} − 1)N_{constr}.

4.4.4 The Relation of RCD to Batch CD

The first iteration of Batch CD runs once through all the data, demoting a certain number of constraints. The constraints which remain in the top stratum at the end of this first iteration can never be demoted in subsequent iterations through the data; and other constraints can never be promoted into this stratum. So after the first iteration, the top stratum has its final form. All mark-data pairs containing these constraints can be removed and ignored in subsequent iterations; if not, they will not cause further demotions. This same property then holds recursively for subsequent iterations. In general, the *n*th iteration of Batch CD definitively determines the *n*th stratum from the top. This is the same as the corresponding stratum output by RCD on the *n*th call to RCD (a recursive call for $n > 1$).

4.5 Error-Driven Constraint Demotion

Several versions of the Constraint Demotion algorithm have been presented. All of these versions require that one or more suboptimal competitors be “picked” to form mark-data pairs with the observed optimal description. Given that *GEN* provides an infinite number of suboptimal competitors, it is not immediately clear how to algorithmically select suboptimal descriptions. This is particularly significant in light of the fact that some suboptimal competitors will be much more informative than others; if uninformative competitors are consistently selected, no learning will take place. Can informative competitors be efficiently selected?

4.5.1 Using Parsing to Generate Suboptimal Competitors

It is here that the work on the parsing problem in Optimality Theory becomes relevant to learning. In the learning problem, it is assumed that the learner has access to the input form. If the learner has a current hypothesized ranking of the constraints (in general, a stratified hierarchy), then they can compute the optimal parse of that input with respect to the current hypothesized ranking. It is this capacity that is exploited to determine suboptimal competitors.

The algorithm, Error-Driven Constraint Demotion, makes use of Constraint Demotion, and works as follows. The learner has in hand a hypothesized ranking; at the beginning, the initial hierarchy $\mathcal{H}_0$. The learner receives as input a single complete parse along with its underlying form. The learner then computes the optimal parse of the underlying form according to its current ranking. If this parse matches the observed one, then nothing may be learned (no suboptimal candidate could be informative), so the learner does nothing further with that input. If, on the other hand, the computed optimal parse differs from the observed parse, then the CD algorithm is applied, using the observed parse as the winner and the computed parse as the loser. Because the current ranking held the computed parse as optimal, that parse is guaranteed to be informative. At least one constraint demotion must take place in order to modify the ranking so that the observed parse is more Harmonic than the computed parse. Thus, the application of CD will result in a new hypothesized ranking.

For this scheme to be feasible, the learner must be capable of evaluating the relative Harmony of candidate structural descriptions with respect to hierarchies which are not total rankings. The Harmonic ordering of forms defined by Prince and Smolensky (1993, §5) presumes total rankings. However, there is a quite straightforward extension of Harmonic ordering of forms to stratified hierarchies. This extension treats constraints in the same stratum as having equivalent Harmonic value. When comparing two descriptions, a mark assessed by one constraint may cancel with a mark assessed by a different constraint in the same stratum. Harmonic ordering of forms is here defined stratum by stratum, starting at the top of the hierarchy. Two descriptions are ordered relative to a single stratum by listing for each description the marks assessed by all the constraints in the stratum. The description with fewer marks is the more Harmonic relative to that stratum. If they have the same number of marks, the two descriptions are not Harmonically distinguished relative to that stratum.

Consider the following simple illustration:

Table 26

Evaluation with equivalent constraints: $\mathbb{C}_1 \gg \mathbb{C}_2, \mathbb{C}_3 \gg \mathbb{C}_4$

	$\mathbb{C}_1$	$\mathbb{C}_2$	$\mathbb{C}_3$	$\mathbb{C}_4$
F_1	*!			
F_2			*	*!
☞ F_3		*		
F_4			**!	

In this illustration, descriptions F_2 and F_3 violate different constraints which are in the same stratum of the hierarchy. Therefore, the marks cancel, and it is left to the lower-ranked constraint to decide in favor of F_3. Notice that candidate F_4 is still eliminated by the middle stratum because it incurs more than the minimal number of marks to constraints in the middle stratum.

In this way, Harmonic ordering of forms is well-defined over all stratified hierarchies. Because the extensions are entirely internal to the comparison of different lists of marks, the parsing algorithms may be applied to this extension without difficulty. A consequence of this extension is that descriptions with non-identical lists of marks can now have equal Harmony, including the possibility of having ties for optimality.

Given this extension, for each observed piece of positive data, parsing may be used to determine if an informative competitor exists, and if so, to provide such a candidate. This results in an *error-driven* learning algorithm (Wexler & Culicover 1980). Each observed description is compared with a computed description of the underlying form. If the two descriptions match, no error occurs, and so no learning takes place. If the two descriptions differ, this is treated as an error on the part of the current hypothesized ranking, and so constraint demotion is used to adjust the hypothesized ranking consistent with the observed description.

It is important to note that applying CD to a single mark-data pair does not in general guarantee that the observed description (the winner) is now the optimal description of its underlying form. All that is guaranteed is that the observed description is more Harmonic than the particular competitor used. It may well be that the resulting constraint ranking determines a third description to be the optimal one for the underlying form.

This algorithm demonstrates that using the familiar strategy of error-driven learning does not require hard constraints or independently evaluatable parameters. Because Optimality Theory is defined in terms of optimization, errors are defined with respect to the relative Harmony of several entire descriptions, rather than in terms of applying particular criteria to a single description. Constraint demotion accomplishes learning precisely on the basis of the comparison of entire descriptions.

4.5.2 A Family of Error-Driven Learning Algorithms

As with Constraint Demotion, Error-Driven Constraint Demotion is actually a family of related algorithms, all based upon the same basic principle. There are two facts which result in possible variations in this approach. The first is that for a single input, there may be several candidates which tie for optimality. Such ties are more likely to occur early in learning, when many constraints do not have established relative rankings. The second fact is that, when the CD algorithm is applied to a single mark-data pair, the resulting ranking does not necessarily have the winner as optimal. Applying the CD algorithm to a single pair guarantees that the loser will be less harmonic than the winner, but does not rule out the possibility that some other candidate is optimal by the resulting ranking.

As a result, two decisions need to be made about error-driven learning. First, if parsing reveals that several descriptions tie for optimality, the learner may either use each of them to create separate mark-data pairs to learn on, or the learner can simply choose one of them, learning on only the single resulting pair. Creating several mark-data pairs provides more potential information, at the cost of additional processing. If several mark-data pairs are chosen, it would be reasonable to apply the I/O version of CD; parsing is used to select the suboptimal candidates for creating the mark-data pairs. If only one competing candidate is selected for learning, On-Line CD will be employed by default.

The second decision involves what, if any, steps to take after constraint demotion. Because applying constraint demotion on a single mark-data pair (or the set of pairs corresponding to the set of descriptions which tied for optimality) does not ensure that the observed description (the winner) is optimal, the learner could re-parse the input according to the new constraint ranking. If the resulting optimal description(s) is different from the winner, the new optimal description may be used to create a new mark-data pair, to which constraint demotion would be applied. Again, this allows the learner to extract more information out of a single observed description, at the cost of greater processing dedicated to a single observed description. The decision here is whether or not to repeatedly iterate the parse/CD cycle.

4.5.3 The Time-Course of Learning

Some facts about the time-course of learning will be illustrated with an example, again of the learning of L_1. This example employs the input /VC/, and applies error-driven learning by selecting a single candidate from a set of tied optimal descriptions. After constraint demotion is applied to a single mark-data pair, the input is re-parsed according to the new ranking, and this learning cycle is repeated until the observed description becomes the optimal one for the input.

The algorithm starts with the ranking $\mathcal{H}_0$, that is, all constraints tied at the top of the hierarchy, as in (76). The observed optimal description is .□V.〈C〉; this is the winner. Applying the parsing algorithm to the input reveals that eight candidates tie for optimality:

(102)	⟨V⟩⟨C⟩	{*PARSE *PARSE}
	.V.⟨C⟩	{*ONS *PARSE}
	.□V.⟨C⟩	{*FILL^{Ons} *PARSE}
	.V.C□́.	{*ONS *FILL^{Nuc}}
	.□V.C□́.	{*FILL^{Ons} *FILL^{Nuc}}
	.VC.	{*ONS *NOCODA}
	.□VC.	{*FILL^{Ons} *NOCODA}
	⟨V⟩.C□́.	{*PARSE *FILL^{Nuc}}

Although in this case the observed description is one of the eight that tie for optimality, it will not generally be the case that the observed description will be optimal at the start. At the beginning, all constraint violations have equal importance (because no distinctions in ranking have yet been made). Optimality here reduces to having the fewest total number of marks. The eight candidates listed above are precisely those candidates which incur two violation marks (there are no candidates for this input with fewer than two marks).

The suboptimal description chosen for learning on the first application of CD is .V.⟨C⟩. This gives us the following mark-data pair:

Table 27

Mark-data pair for error-driven learning step 1.

	loser-marks	*winner-marks*
.V.⟨C⟩ ≺ .□V.⟨C⟩	{*ONS ~~*PARSE~~}	{*FILL^{Ons} ~~*PARSE~~}

Applying On-Line CD to this mark-data pair results in the demotion of FILL^{Ons}, so the resulting hierarchy is

(103) $\mathcal{H}$ = {PARSE, FILL^{Nuc}, ONS, NOCODA} ≫ {FILL^{Ons}}

Step 2 repeats the procedure on the same input with the same observed description, using the new hierarchy. Applying the parsing algorithm reveals that with the current hierarchy, three candidates tie for optimality:

(104) .□V.〈C〉 {*FILL^{Ons} *PARSE}

.□V.C□́. {*FILL^{Ons} *FILL^{Nuc}}

.□VC. {*FILL^{Ons} *NOCODA}

As it turns out, the observed description is one of the three which tie for optimality. Thus, one of the other two will be selected as the losing candidate. Here, the chosen losing candidate is .□V.C□́. The corresponding mark-data pair is:

<u>Table 28</u>

Mark-data pair for error-driven learning step 2.

	loser-marks	*winner-marks*
.□V.C□́. ≺ .□V.〈C〉	{*FILL^{Nuc} ~~*FILL^{Ons}~~}	{*PARSE ~~*FILL^{Ons}~~}

Applying On-Line CD to this mark-data pair results in the demotion of PARSE down to the second stratum (crucially, below FILL^{Nuc}):

(105) $\mathcal{H}$ = {FILL^{Nuc}, ONS, NOCODA} ≫ {PARSE, FILL^{Ons}}

Step 3 repeats the procedure on this new hierarchy. Applying the parsing algorithm reveals that there are now two candidates which tie for optimality:

(106) .□V.〈C〉 {*FILL^{Ons} *PARSE}

〈V〉〈C〉 {*PARSE *PARSE}

Applying On-Line CD to this final pair results in the correct stratified hierarchy for the language, the one which gives rise to L_1:

(107) $\mathcal{H}$ = {FILL^{Nuc}, ONS, NOCODA} ≫ {PARSE} ≫ {FILL^{Ons}}

That this is the correct hierarchy is recognized by Error-Driven Constraint Demotion when the input is once more parsed and only one candidate, the observed one, is optimal.

Notice that, even though the On-Line version of CD was employed, its repeated use by the error-driven learning scheme allows the entire hierarchy to be learned from one piece of positive data, just as Recursive Constraint Demotion was able to do.

It is instructive to examine carefully the set of candidates which tie for optimality at each step. At first glance, the algorithm appears to have a straightforward monotonic character: each step eliminates some candidates, until only one remains. But, this is not true. In particular, notice that the candidate $\langle$V$\rangle\langle$C$\rangle$ is one of the optimal candidates in step 1 and step 3, but not in step 2. This candidate was not simply eliminated permanently from contention by some particular operation; it moves into and out of optimal status as the hierarchy changes. Thus, even this simple two segment example hints at the kinds of complex dynamics this algorithm is capable of.

It is certainly not being claimed here that no predictions can be made about the time course of learning if this algorithm is used. It is possible that a particular set of constraints, along with a particular definition of *GEN*, would result in restrictive patterns of language development. The point is that such patterns would not be the result of constraint demotion by itself; such patterns would depend on interactions between the algorithm and the specific grammars being acquired.

4.5.4 The Data Complexity of Error-Driven Constraint Demotion

Because Error-Driven Constraint Demotion repeatedly invokes On-Line CD, the data complexity results for On-Line CD apply directly. Each error is precisely an informative example. Thus, the worst case number of errors required to learn the hierarchy for a grammar is

$$\tfrac{1}{2}(N_{constr}\text{-}1)N_{constr}$$

In a grammar with 10 constraints, there are 3,628,800 distinct total rankings of the constraints, but Error-Driven Constraint Demotion will require at most 45 errors to learn the correct hierarchy.

4.6 Principles of Universal Grammar and Principles of Learning

4.6.1 Ties

Error-Driven Constraint Demotion assumes that all competitors to an observed output are suboptimal. This might fail to be the case in two different ways. First, a single input may give rise to multiple optimal outputs which all earn the same set of marks (i.e., the constraints fail to distinguish them). In this case, while it is not actually correct that all competitors of a given observed output are sub-optimal, no errors in the learning result. Recall that the first step in the CD algorithm is the cancellation of common marks. Thus if the 'sub-optimal' description selected is in fact a second optimal description, then all marks cancel and this pair is discarded.

Two descriptions which do not incur identical marks can only tie for optimality if the underlying constraint hierarchy is not consistent with any totally ranked hierarchy, since it must be the case that marks for different constraints are canceling. However, Constraint Demotion will not correctly learn such hierarchies: observing one of the optimal descriptions and assuming that all others are sub-optimal prevents the tie from being learned. This is easily illustrated with two constraints, and two candidates, each of which violates one of the constraints. If the target hierarchy ranks the two constraints in the same stratum, the two descriptions will tie for optimality (assuming every alternative candidate receives at least two marks). Both descriptions will therefore be observed as positive evidence. But when one description is observed, CD will automatically assume the other to be suboptimal, and will demote the constraint violated by the observed description below the other. But, when the other description is observed, CD will reverse the rankings of the constraints. This will continue endlessly, and learning will fail to converge. Notice that this instability will be observed even if the algorithm starts with a hierarchy with the constraints in the same stratum. Not only will the algorithm not converge on the tied hierarchy, but when given the hierarchy it will in time reject it.

It is currently unknown whether it is possible to extend CD to learn languages in which some inputs have multiple optimal outputs which do not earn identical sets of marks. One might, for example, suppose that the target language derives from an underlying stratified hierarchy inconsistent with any totally ranked hierarchy. One might also assume that the learner's initial data consists in a set of inputs, and for each input, *all* its optimal outputs, should there be more than one. The Core CD Algorithm would need to be extended to handle pairs $opt_1 \sim opt_2$ of tying optima. In this extension, each mark in $marks'(opt_1)$ must be placed in the same stratum as a corresponding mark in $marks'(opt_2)$: a somewhat delicate business. This perhaps is more apparent once the possibility of multiple strata, each containing more than one constraint, is considered, requiring the learner to sort out which constraints cancel with which. Indeed, achieving ties for optimality between descriptions which incur different marks is a delicate matter.

However, the inability of CD to learn such hierarchies is actually a desirable property. Typological prediction in Optimality Theory presumes that all attested languages correspond to total rankings of the constraints. It is commonly assumed within Optimality Theory that universal grammar rules out grammars inconsistent with at least one total ranking. The relation of Constraint Demotion to typological prediction is further discussed below, in the section on Principles and Explanatory Adequacy.

The inability of EDCD to learn hierarchies inconsistent with any total ranking would be easily proven if the only consequence of such hierarchies was the possibility of descriptions with non-identical marks tying for optimality. However, that is not the case. Another possible consequence was illustrated above in Table 26. In that case, marks assessed to two different descriptions by two different constraints in the same stratum cancel, but a constraint in a lower stratum then selects one of the candidates. The cancellation of marks from different constraints plays a role, but only one candidate is optimal. It is unknown whether EDCD could possibly learn a hierarchy which is (a) inconsistent with any total ranking; (b) does not have multiple descriptions with non-identical marks tying for optimality, for *any* input.

4.6.2 Optionality

The structure of Optimality Theory endows considerable power to the implicit negative evidence which the CD algorithm exploits. This has interesting consequences for the Subset Principle for language acquisition (Angluin 1978, Berwick 1986, Pinker 1986, Wexler & Manzini 1987). The Subset Principle states that if one possible language is a subset of another, the subset language must be hypothesized first, to avoid unrecoverable overgeneralization. The error-driven algorithm does not make strong predictions about any superset relations between earlier and later hypothesized hierarchies; that is, it does not predict that the languages generated by later hierarchies are contained in languages generated by earlier hierarchies. It should be readily apparent, however, that it also does not predict subset relations.

In linguistic analyses, subset relations have often been connected with optionality. An example is the analysis of some languages as optionally permitting null subjects (Rizzi 1982). This kind of analysis presumes that sentences which permit null subjects contain pairs of otherwise identical sentences which differ only in that the subject is lexically realized in one and not the other, and further that there is no other information distinguishing the structural description of the two sentences. Languages which do not permit null subjects only contain the member of each pair with the subject realized (hence the subset relation).

Samek-Lodovici and Grimshaw, in recent work (Samek-Lodovici 1995, Grimshaw & Samek-Lodovici in prep.), argue that null subjects are not optional, but are in fact related to the focus and the topic-referring status of the subject. Languages which permit null subjects only permit them when the subject is topic-referring, while languages which do not permit null subjects require a lexically realized subject when the subject is topic-referring. They provide an analysis within Optimality Theory in which the focus and topic-referring status of the subject is included in the input, and therefore by containment in the structural description. As a result, there is no subset relation between the two types of languages. The same input which is described as having a null subject in one language will receive a description with an overt subject in the other type of language. Neither description is present in both languages; neither language is a subset of the other.

This difference on optionality is symptomatic of a more general difference between Optimality Theory and the principles and parameters framework. The latter posits a universal set of possible structures, and specific grammars accept or reject possible structures, the resulting language being the set of accepted structures. Optimality Theory posits a universal set of possible *inputs*, and specific grammars determine what the optimal description is of each input. Grammars in Optimality Theory do not reject inputs, they only assign descriptions. Thus, at the level of complete structural descriptions, no two languages can be in a strict subset relation: each language has the same inputs, and assigns a description to each input. Optimality Theory denies the possibility of optionality, at least with respect to complete structural descriptions.

The possible structures in a language are those that occur in the optimal description of at least one input. Arguing that a grammar does not permit some particular structure is not done by showing that the grammar rejects descriptions containing that structure, but instead by showing that the structure cannot occur in any optimal description. An example is Basic Syllable Theory: in some languages, onsets are required, while in others they are optional. If the grammar is defined with a parameter that determines acceptance or rejection of syllables without onsets, a subset relation results. In Basic Syllable Theory, every input is assigned a description, and grammars which require onsets assign descriptions which only contain syllables with onsets.

4.6.3 Principles and Explanatory Adequacy

The learning algorithm operates in the space of stratified hierarchies of constraints. However, the typological predictions made in Optimality Theory are based on the presumption that well formed grammars must correspond to total rankings of the constraints. The discussion of ties above demonstrates that many possible constraint hierarchies which are not consistent with any total ranking are not learnable. Furthermore, the proof of correctness for Constraint Demotion only applies if all of the data are consistent with some total ranking. Because ties in a constraint hierarchy can have effects other than multiple descriptions with non-identical constraint violations tying for optimality, it is not known for certain if all hierarchies inconsistent with any total ranking are unlearnable by Constraint Demotion. What is known is that for such a hierarchy to possibly be learnable, it must be the case that there is not a single input which has multiple descriptions with non-identical constraint violations which tie for optimality.

This property is a consequence of a principle implicit in the learning algorithm. This principle states that the learner should assume that the observed description is optimal for the corresponding input, and that it is the *only* optimal description. This principle resembles other proposed learning principles, such as Clark's Principle of Contrast (E. Clark 1987) and Wexler's Uniqueness Principle (Wexler 1981). Optimality Theory permits a particularly rigorous formal statement of this kind of principle with respect to the acquisition of core grammar.

This suggests the following account of cross-linguistic variation. Optimality Theory provides the following two principles of universal grammar:

(108) Grammaticality is defined in terms of optimization over a large space of candidates, with respect to violable universal constraints.

(109) When universal constraints conflict, that conflict is resolved via strict domination.

The learning algorithm contributes the following principle of learning:

(110) Presume that any observed description is the only optimal description of its underlying form.

It then is in large part a consequence of these principles that the only well-formed adult grammars will be those that are consistent with at least one total ranking of the universal constraints.

Even though the set of target languages is generated from the set of totally ranked hierarchies of the universal constraints, the hypothesis space employed by the CD learning algorithm is the larger space of stratified hierarchies. It is worth emphasizing that the learning algorithm is not merely tolerant of a hypothesis space of stratified hierarchies; the larger hypothesis space is important to the functioning of the algorithm. That learning can be much less difficult when the hypothesis space is larger than the target space is a theme of work in Computational Learning Theory (Pitt & Valiant 1988, Kearns & Vazirani 1994).

This draws into focus an important but occasionally overlooked fact: the set of learnable grammars is a property of a learning system, not of a hypothesis space. Limiting the hypothesis space to the space of observed grammars does make it a trivial matter to prove that unobserved grammars are not learnable by the system, but it is not the only way to achieve such a result. The fact that hypotheses in a hypothesis space can be unlearnable has been the subject of much work in learnability (subset relations, local maxima). However, most work in the principles and parameters framework assumes a hypothesis space equivalent to the set of observed grammars, and then attempts to construct learning systems which guarantee that all the hypotheses are learnable. By contrast, the account of cross-linguistic variation given above assumes a larger hypothesis space, and then attempts to guarantee that all and only those hypotheses which correspond to observed grammars are learnable.

4.7 Learnability and Linguistic Theory

In the principles and parameters framework, cross-linguistic variation is accounted for by a set of parameters, where a specific grammar is determined by fixing each parameter to one of its possible values. Every possible combination of parameter settings is a possible grammar. Work on learnability in this framework focuses on the relationship between data and the parameter values. This relationship is usually discussed in terms of "triggers". A trigger is a datum (for example, a sentence for syntax) which indicates the appropriate value for a specific parameter (see, for example, the definitions of trigger in Gibson & Wexler 1994). It is significant that a trigger provides information about the value of a single parameter, rather than relationships between the values of several parameters[1]. This property is further reinforced by a proposed constraint on learning, the Single Value Constraint (R. Clark 1990, Gibson & Wexler 1994): successive hypotheses considered by a learner may differ by the value of at most one parameter. The result is that learnability concerns in this framework favor parameters which are independent: they interact with each other as little as possible, so that the effects of each parameter setting can be distinguished from the effects of the other parameters. In fact, this property of independence has been proposed as a principle for grammars (Wexler & Manzini 1987). Unfortunately, this results in a conflict between the goals of learnability, which favor independent parameters with restricted effects, and the goals of linguistic theory, which favor parameters with wide-ranging effects and greater explanatory power (see (Safir 1987) for a discussion of this conflict).

Optimality Theory may provide the opportunity for this conflict to be avoided. In Optimality Theory, interaction between constraints is not only possible but mandatory. Cross-linguistic variation is explained not by variation in the structure of individual constraints, but by variation in the relative ranking of the same constraints. Cross-linguistic variation is only possible to the extent that constraints interact. The Constraint Demotion learning algorithm not only tolerates constraint interaction, but is based upon

[1]Under the normal definitions of a trigger, a single datum can be a trigger for more than one parameter, but is such independently. In such a case, the datum would not be interpreted as expressing any relationship between the values of the two parameters.

it. Informative data provide information not about one constraint in isolation, but about the results of interaction between constraints. Constraints which have wide-ranging effects benefit learnability. If these properties are successfully preserved in a more general account of language learning within Optimality Theory, explanation and learnability will work together; they will both favor interacting constraints with wide-ranging effects and explanatory power.

This possibility comes in virtue of the fact that Optimality Theory defines grammaticality in terms of optimization over violable constraints. This central principle makes constraint interaction the main explanatory mechanism. It provides the implicit negative data used by Constraint Demotion precisely because it defines grammaticality in terms of the comparison of candidate descriptions, rather than in terms of the structure of each candidate description in isolation. Constraint Demotion proceeds by comparing the constraint violations assessed candidate descriptions. This makes constraint interaction the basis for learning.

By making constraint interaction the basis for both linguistic explanation and learning, Optimality Theory creates the opportunity for the full alignment of these two goals. The discovery of sets of constraints which interact strongly in ways that participate in diverse linguistic phenomena represents progress for both explanation and learning. Clearly, this is a desirable property for a theoretical framework.

4.8 Other Work on Learnability in Optimality Theory

An alternative approach to learning constraint rankings has been explored recently by Pulleyblank and Turkel (1995a, 1995b). Their work applies a general approach to global optimization, genetic algorithms, to the problem.

In recent work on syntax and sentence pronunciation, Pesetsky (1993) has used a modification of the standard Optimality Theory framework. In his modification, adult grammars can have tied constraints. Further, ties between constraints are not interpreted in the way that Error-Driven Constraint Demotion does. Instead, when a set of constraints are tied, all possible total rankings of that set are considered, and the most harmonic description for each ranking of the set is treated as equal with respect to that set of constraints. Broihier (1995) has investigated some attempts to extend Constraint Demotion to learn grammars with ties of this sort.

Chapter 5. Conclusions

This dissertation has investigated two major computational issues within the framework of Optimality Theory. The results indicate how the computational challenges presented by the framework can be addressed. This chapter summarizes the results presented in this dissertation, and suggests what issues need to be investigated next.

5.1 Parsing

5.1.1 What Has Been Accomplished

The parsing algorithms presented in this dissertation efficiently compute the optimal structural description of an input, for grammars which select the optimal description from an infinite set of candidate descriptions. These algorithms are provably correct for grammars in which the input is a strictly ordered sequence of elements, and the universal constraints are appropriately local. Chapter 2 presented an algorithm for grammars in which *GEN* employs structures properly described by a formal regular grammar; the parser has a time complexity linear in the length of the input. Chapter 3 presented an algorithm for grammars in which *GEN* employs structures properly described by a formal context-free grammar; the parser has a time complexity cubic in the length of the input.

Perhaps the most important lesson of this work for people working in Optimality Theory is that the prospect of an infinite candidate set generated by *GEN* need not be feared. A grammar is a function, not an algorithm. An algorithm need not mimic every aspect of the normal description of a grammar in order to compute the function defined by that grammar.

Further, given a grammar satisfying the stated conditions, the algorithms do not succeed by imposing limitations on the set of descriptions considered which are extrinsic to the grammar. The algorithms have the entire infinite set of candidate descriptions available to them. The algorithms rely on the same universal constraints which guarantee the existence of optimal descriptions in the first place. For example, if the FILL constraints were removed from the Basic CV Syllable Theory, so that descriptions of arbitrary size tied for optimality, then the parsing algorithms would never halt; they would keep building and considering larger and larger partial descriptions without limit. The properties which guarantee convergence of the algorithms are properties internal to the grammar.

A related point is the modular role of the language-specific constraint ranking in the operation of the parser. The constraint ranking is consulted whenever several partial descriptions are compared. This is the only way in which the ranking is used in the course of parsing. The rest of the operation of the algorithm is independent of the particular ranking being used. Therefore, the structure of the parsing algorithm does not need to be changed in response to cross-linguistic variation. Variation is restricted to the constraint ranking, as it is in the linguistic theory itself.

5.1.2 The Next Steps

The most important formal issues needing further investigation are the restrictions on the input ordering and the locality of the constraints. Both of these restrictions are sufficient conditions for the computability of Optimality Theoretic grammars; necessary conditions are not known. Nevertheless, these restrictions play a significant role in the algorithms described in this dissertation. Just how crucial each restriction is to the general dynamic programming approach needs to be better understood.

A more general issue is the relationship between the competence input/output mappings and the performance processes of production and comprehension. The competence grammars of Optimality Theory are typically described as taking underlying forms as inputs, and thus are more naturally related to production processes. But it is premature to assume any kind of strict identity between competence input/output mappings and production. Further, the computability of mappings from surface forms to structural descriptions (the much more commonly investigated direction in natural language processing) deserves much further investigation. In fact, in syntax the dynamic programming algorithms presented here may extend more naturally when surface forms are used as input, because the input is then strictly ordered.

5.2 Learnability

5.2.1 What Has Been Accomplished

The learnability work presented in this dissertation has solved a significant subproblem of the general problem of language learning. That is the subproblem of inferring the grammar (here, the constraint ranking) from hypothesized descriptions in Optimality Theory, a framework employing violable constraints which interact strongly. The Constraint Demotion algorithms efficiently determine the ranking given structural descriptions corresponding to positive data.

The Constraint Demotion algorithms work solely on the basis of the formal structure of the framework. They apply to any grammar in Optimality Theory. They demonstrate that the challenge of learning grammars which employ violable constraints can be solved in Optimality Theory by using the implicit negative evidence inherent in the framework. The algorithms have a worst case data complexity quadratic in the number of constraints, beating the factorial growth of the number of possible total rankings.

5.2.2 The Next Steps

The obvious next step is to work on the other parts of the learning problem, and try to connect them with Constraint Demotion. Other subproblems include hypothesizing structural descriptions from information realistically available to the learner, and hypothesizing underlying forms from paradigmatic sets of descriptions. This will likely require moving down from the level of the formal structure of the framework, and working with specific Optimality Theoretic proposals for linguistic phenomena.

5.2.3 Theoretical Implications

Optimality Theory may provide the opportunity to avoid the conflict between learnability and linguistic explanation faced by much work in the Principles and Parameters framework. The search for triggers which indicate appropriate values for specific parameters favors parameters that interact with each other as little as possible, so that the effects of each parameter setting can be distinguished from the effects of the other parameters. In Optimality Theory, interaction between constraints is not only possible but mandatory: cross-linguistic variation is only possible to the extent that constraints interact. With Constraint Demotion, informative data provide information not about one constraint in isolation, but about the results of interaction between constraints: constraint interaction is the basis for learning. If these properties are successfully preserved in a complete account of language learning within Optimality Theory, explanation and learnability will work together; they will both favor interacting constraints with wide-ranging effects and explanatory power.

5.3 Other Future Work

Optimality Theory shares several underlying principles with connectionist theory, in particular the use of violable constraints and the defining of grammaticality in terms of optimization. Many interesting questions remain open about the possible relationships between the algorithms presented here for Optimality Theory, and connectionist algorithms. The possibilities for embedding Optimality Theoretic grammars within connectionist architectures are worthy of much future investigation.

Bibliography

Angluin, Dana. 1978. Inductive inference of formal languages from positive data. Information and Control 45:117–135.

Berwick, Robert. 1986. The acquisition of syntactic knowledge. Cambridge, MA: MIT Press.

Broihier, Kevin. 1995. Optimality theoretic rankings with tied constraints: Slavic relatives, resumptive pronouns and learnability. Ms., MIT.

Charniak, Eugene. 1993. Statistical language learning. Cambridge, MA: MIT Press.

Chomsky, Noam. 1959. On certain formal properties of grammars. Information and Control 2: 137-167.

Chomsky, Noam. 1965. Aspects of the theory of syntax. Cambridge, MA: MIT Press.

Chomsky, Noam. 1981. Lectures on government and binding. Dordrecht: Foris Publications.

Clark, Eve. 1987. The principle of contrast: A constraint on language acquisition. In Mechanisms of language acquisition, ed. B. MacWhinney. Hillsdale, NJ: Erlbaum.

Clark, Robin. 1990. Papers on learnability and natural selection. Technical Reports in Formal and Computational Linguistics, No. 1, Universite de Geneve.

Clark, Robin. 1992. The selection of syntactic knowledge. Language Acquisition 2:83-149.

Clark, Robin and Ian Roberts. 1993. A computational model of language learnability and language change. Linguistic Inquiry 24(2):299-345.

Clements, G. N and S.J. Keyser. 1983. CV phonology. Cambridge, MA: MIT Press.

Corman, Thomas, Charles Leiserson, and Ronald Rivest. 1990. Introduction to algorithms. Cambridge, MA: MIT Press.

Dresher, B. Elan, and Jonathan D. Kaye. 1990. A computational learning model for metrical phonology. Cognition 34: 137-195.

Ellison, T. Mark. 1994. Phonological derivation in Optimality Theory. Proceedings of the Fifteenth International Conference on Computational Linguistics, 1007-1013.

Frank, Robert, and Shyam Kapur. 1994. On the use of triggers in parameter setting. Ms., University of Delaware and University of Pennsylvania.

Gibson, Edward and Kenneth Wexler. 1994. Triggers. Linguistic Inquiry 25(4).

Gleitman, Lila R. 1990. The structural sources of verb meaning. Language Acquisition 1: 3-55.

Gold, E.M. 1967. Language identification in the limit. Information and Control 10(4): 447-474.

Grimshaw, Jane. 1993. Minimal projection, heads, and inversion. Ms., Rutgers University.

Grimshaw, Jane. 1994. Lexical reconciliation. Lingua 92: 411-430.

Grimshaw, Jane, and Vieri Samek-Lodovici. (in prep). Optimal subjects. Ms., Rutgers University.

Hillman, Abraham, Gerald Alexanderson, and Richard Grassl. 1987. Discrete and combinatorial mathematics. San Francisco, CA: Dellen.

Jakobson, Roman. 1962. Selected writings 1: Phonological studies. The Hague: Mouton.

Kay, Martin. 1980. Algorithmic schemata and data structures in syntactic processing. CSL-80-12, October 1980.

Kearns, Michael J., and Umesh V. Vazirani. 1994. An introduction to computational learning theory. Cambridge, MA: MIT Press.

Lari, K., and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. Computer Speech and Language 4: 35-36.

Legendre, Géraldine, William Raymond, and Paul Smolensky. 1993. Analytic typology of case marking and grammatical voice. Proceedings of the Berkeley Linguistics Society 19.

Legendre, Géraldine, Colin Wilson, Paul Smolensky, Kristin Homer, and William Raymond. 1995. Optimality in *wh*-chains. In University of Massachusetts occasional papers in linguistics 18: Papers in Optimality Theory, eds. J. Beckman, S. Urbanczyk, & L. Walsh. Amherst, MA: GLSA, University of Massachusetts. In press.

Lewis, Harry R., and Christos H. Papadimitriou. 1981. Elements of the theory of computation. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

McCarthy, John and Alan Prince. 1993a. Prosodic morphology I: Constraint interaction and satisfaction. Ms., University of Massachusetts, Amherst, and Rutgers University, New Brunswick, NJ. To appear in the Linguistic Inquiry Monograph Series, Cambridge, MA: MIT Press.

McCarthy, John and Alan Prince. 1993b. Generalized alignment. Yearbook of Morphology 1993: 79-154.

Pesetsky, David. 1993. Ms., MIT.

Pinker, Steven. 1986. Productivity and conservatism in language acquisition. In Language learning and concept acquisition, ed. W. Demopoulos and A. Marras. Norwood, NJ: Ablex.

Pinker, Steven. 1987. The bootstrapping problem in language acquisition. In Mechanisms of language acquisition, ed. B. MacWhinney. Hillsdale, NJ: Erlbaum.

Pitt, L. and L. Valiant. 1988. Computational limitations on learning from examples. Journal of the ACM 35:965-984.

Prince, Alan and Paul Smolensky. 1991. Notes on connectionism and Harmony Theory in linguistics. Technical Report CU-CS-533-91. Department of Computer Science, University of Colorado, Boulder.

Prince, Alan and Paul Smolensky. 1993. Optimality Theory: Constraint interaction in generative grammar. Technical Report CU-CS-696-93, Department of Computer Science, University of Colorado at Boulder, and Technical Report TR-2, Rutgers Center for Cognitive Science, Rutgers University, New Brunswick, NJ. March. To appear in the Linguistic Inquiry Monograph Series, Cambridge, MA: MIT Press.

Pulleyblank, Douglas, and William J. Turkel. 1995a. Traps in constraint ranking space. Paper presented at Maryland Mayfest 95: Formal Approaches to Learnability.

Pulleyblank, Douglas, and William J. Turkel. 1995b. The logical problem of language acquisition in Optimality Theory. Paper presented at Is the Best Good Enough: Workshop on Optimality in Syntax, MIT.

Rabiner, L.R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE 77(2):257-286.

Rizzi, L. 1982. Issues in italian syntax. Dordrecht: Foris Publications.

Safir, Ken. 1987. Comments on Wexler and Manzini. In Parameter setting, eds. T. Roeper and E. Williams, 77-89. Dordrecht: Reidel.

Samek-Lodovici, Vieri. 1995. Topic and focus effects on clause structure: An Optimality Theoretic analysis. Ph.D. Dissertation in preparation, Rutgers University.

Sankoff, David and Joseph Kruskal. 1983. Time warps, string edits, and macromolecules: The theory and practice of sequence comparison. Reading, MA: Addison-Wesley.

Tesar, Bruce. 1994. Parsing in Optimality Theory: A dynamic programming approach. Technical Report CU-CS-714-94, April 1994. Department of Computer Science, University of Colorado, Boulder.

Tesar, Bruce. 1995. Computing optimal forms in Optimality Theory: Basic syllabification. Technical Report CU-CS-763-95, February 1995. Department of Computer Science, University of Colorado, Boulder.

Tesar, Bruce, and Paul Smolensky. 1995. The learnability of Optimality Theory. Proceedings of the Thirteenth West Coast Conference on Formal Linguistics, 122-137.

Tesar, Bruce, and Paul Smolensky. (to appear). The learnability of Optimality Theory: An algorithm and some basic complexity results. Linguistic Inquiry. (revision of Technical Report CU-CS-678-93, October 1993. Department of Computer Science, University of Colorado, Boulder)

Valiant, L. 1984. A theory of the learnable. Communications of the ACM 27(11):1134-1142.

Wexler, Kenneth. 1981. Some issues in the theory of learnability. In The logical problem of language acquisition, eds. C. Baker and J. McCarthy, 30-52. Cambridge, MA: MIT Press.

Wexler, Kenneth and P. Culicover. 1980. Formal principles of language acquisition. Cambridge, MA: MIT Press.

Wexler, Kenneth and M. Rita Manzini. 1987. Parameters and learnability in binding theory. In Parameter setting, eds. T. Roeper and E. Williams, 41-76. Dordrecht: Reidel.