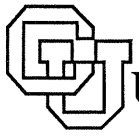
THE LEARNABILITY OF OPTIMALITY THEORY: AN ALGORITHM AND SOME BASIC COMPLEXITY RESULTS

Bruce Tesar, Paul Smolensky

CU-CS-678-93 October 1993



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

THE LEARNABILITY OF OPTIMALITY THEORY: AN ALGORITHM AND SOME BASIC COMPLEXITY RESULTS

CU-CS-678-93 October 1993

Bruce Tesar, Paul Smolensky

Department of Computer Science
University of Colorado at Boulder
Campus Box 430
Boulder, Colorado 80309-0430 USA

The Learnability of Optimality Theory:
An Algorithm and Some Basic Complexity Results

Bruce Tesar & Paul Smolensky

Department of Computer Science &
Institute of Cognitive Science
University of Colorado at Boulder

If Optimality Theory (Prince & Smolensky 1991, 1993) is correct, Universal Grammar provides a set of universal constraints which are highly general, inherently conflicting, and consequently rampantly violated in the surface forms of languages. A language's grammar ranks the universal constraints in a dominance hierarchy, higher-ranked constraints taking absolute priority over lower-ranked constraints, so that violations of a constraint occur in well-formed structures when, and only when, they are necessary to prevent violation of higher-ranked constraints. Languages differ principally in how they rank the universal constraints in their language-specific dominance hierarchies. The surface forms of a given language are structural descriptions of inputs which are optimal in the following sense: they satisfy the universal constraints, or, when these constraints are brought into conflict by an input, they satisfy the highest-ranked constraints possible. This notion of optimality is partly language-specific, since the ranking of constraints is language-particular, and partly universal, since the constraints which evaluate well-formedness are (at least to a considerable extent) universal. In many respects, ranking of universal constraints in Optimality Theory plays a role analogous to parameter-setting in principles-and-parameters theory.

Evidence in favor of this Optimality-Theoretic characterization of Universal Grammar is provided elsewhere; most work to date addresses phonology: see Prince & Smolensky 1993 (henceforth, 'P&S') and the several dozen works cited therein, notably McCarthy & Prince 1993; initial work addressing syntax includes Grimshaw 1993 and Legendre, Raymond & Smolensky 1993. Here, we investigate the learnability of grammars in Optimality Theory.

Under the assumption of innate knowledge of the universal constraints, the primary task of the learner is the determination of the dominance ranking of these constraints which is particular to the target language. We will present a simple and efficient algorithm for solving this problem, assuming a given set of hypothesized underlying forms. (Concerning the problem of acquiring underlying forms, see the discussion of 'optimality in the lexicon' in P & S 1993:§9).

The fact that surface forms are optimal means that every positive example entails a great number of implicit negative examples: for a given input, every candidate output other than the correct form is ill-formed.[1] As a consequence, even a single positive example can greatly constrain the possible grammars for a target language, as we will see explicitly.

In §1 we present the relevant principles of Optimality Theory and discuss the special nature of the learning problem in that theory. Readers familiar with the theory may wish to proceed directly to §1.3. In §2 we present the first version of our learning algorithm, initially, through a concrete example; we also consider its (low) computational complexity. Formal specification of the first version of the algorithm and proof of its correctness are taken up in the Appendix. In §3 we generalize the algorithm, identifying a more general core called *Constraint Demotion* ('CD') and then a family of CD algorithms which differ in how they apply this core to the acquisition data. We sketch a proof of the correctness and convergence of the CD algorithms, and of a bound on the number of examples needed to complete learning. In §4 we briefly consider the issue of ties in the ranking of constraints and the case of inconsistent data. Finally, we observe that the CD algorithm entails a Superset Principle for acquisition:

---

[1] See §4.1 below for a discussion of ties, when more than one form is optimal.

as the learner refines the grammar, the set of well-formed structures *shrinks*.

We offer three related contributions in this paper. First, the existence of the CD algorithms demonstrates formally that Optimality Theoretic grammars are efficiently learnable. Second, these algorithms provide a first suggestion of how to analyze the actual language acquisition process within Optimality Theory, and suggest that acquisition may respect a Superset Principle. Last but not least, the CD algorithms have a very real practical value for linguists working in Optimality Theory. Given language data and a hypothesized set of constraints, the algorithms quickly and easily provide a class of constraint rankings which account for the data, or directly determine that no such rankings exist.

1. The Learning Problem in Optimality Theory

The novel features of the learnability problem, and the character of the learning algorithm, depend on a number of the detailed properties of Optimality Theory. We begin with a self-contained presentation of the relevant principles of the theory, to the level of detail required for presentation of the learning algorithm. We consistently exemplify the principles, and in §2, the learning algorithm, using the Basic CV Syllable Structure Theory of P&S (§6), which we develop concurrently.

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 CV theory, an input is a string of Cs and Vs, e.g., /VCVC/. An output is a parse of the string into syllables, which we notate as follows:

(1′) 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

(These four forms will be referred to throughout the paper, and will be consistently labelled *a*–*d* as in (1′).) Possible output *a* is an onsetless open syllable followed by a closed syllable. Parse *b* contains only one, open, syllable: the initial V and final C of the input are not parsed into syllable structure; following ideas related to 'Stray Erasure' (McCarthy 1979, Steriade 1982, Itô 1986, 1989), we assume that these unparsed segments are not phonetically realized when the phonological output *b* is subjected to phonetic interpretation. Parse *c* consists of a pair of open syllables, in which the nucleus of the second syllable is not filled by underlying material. This is taken to mean that phonetic realization of the phonological structure *c* provides an epenthetic segment in the final nucleus, following a well-developed conception of epenthesis (established, e.g., by Selkirk 1981, LaPointe & Feinstein 1982, Broselow 1982, Archangeli 1984, Kaye & Lowenstamm 1984, Piggott & Singh 1985, Itô 1986, 1989). As in parse *b*, the initial underlying V is unparsed in parse *c*. Parse *d* is also a pair of open syllables, but this time it is the onset of the first syllable which is unfilled by underlying material, and the final underlying C which is unparsed.

In derivational terms, the first and second syllables of 1′.c respectively constitute the result of deletion and epenthesis rules. Optimality Theory lacks derivations and rules; only the output phonological

structure 1′.c, and the input /VCVC/ (which is in fact contained in the output), are recognized. In place of deletion we have *underparsing*, in which underlying material is not parsed into syllable structure; in place of epenthesis we have *overparsing*, in which syllable positions in the output are not filled by underlying material.

What (1) asserts is that the grammar assigns to the input /VCVC/ one structural description, such as one of the possibilities listed under (1′.a–d). Which one gets assigned is determined by subsequent principles.

(2) ***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 parsings of the string into syllables, including the possible over- and underparsing structures exemplified above in (1′.b-d). All syllables are assumed to contain a Nucleus position, with optional preceding Onset and following Coda positions. Here we adopt what P&S (§6) dub the *Basic* CV Syllable Structure Theory, in which the simplifying assumption is made that the syllable positions Onset and Coda may each contain at most one C, and the position Nucleus may contain at most one V.[2]

(3) **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 CV constraints we will deal with here are as follows:

(3′) **Universal CV Syllable Structure Constraints.**

ONS	Syllables have onsets.
–COD	Syllables do *not* have codas.
PARSE	Underlying material is parsed into syllable structure.
FILL^{Nuc}	Nucleus positions are filled with underlying material.
FILL^{Ons}	Onset positions (when present) are filled with underlying material.

We can illustrate how these constraints assess marks by considering the candidate outputs in (1′.a–d). Parse (1′.a) = .V.CVC. violates ONS in its first syllable and –COD in its second; the remaining constraints are satisfied. We denote the single mark which ONS assesses .V.CVC. as *ONS; so the complete set of marks incurred by the candidate .V.CVC. is

$$\textit{marks}(\text{.V.CVC.}) = \{\text{*ONS}, \text{*–COD}\}$$

---

[2] The possibility of overparsing means that the candidate set for any input is in fact infinite. However, it can be shown by examining the constraints of the Basic CV theory that the distribution of unfilled syllable positions is highly limited, entailing that optimal candidates always reside in a particular finite subset of the infinite candidate set (P&S §6.2.3).

The first candidate is a *faithful* parse: it involves neither under- nor over-parsing, and therefore satisfies the *faithfulness constraints* PARSE and FILL. By contrast the underparsed candidate (1′.b) = ⟨V⟩.CV.⟨C⟩ violates PARSE, and more than once:

$$marks(\langle V\rangle.CV.\langle C\rangle) = \{*\textsc{Parse}, *\textsc{Parse}\}$$

As we will see, it is crucial that multiple marks be maintained (although the theory requires no counting).

The third candidate (1′.c) = ⟨V⟩.CV.C□́. combines underparsing with overparsing, violating both the PARSE and FILL constraints:

$$marks(\langle V\rangle.CV.C\acute{\square}) = \{*\textsc{Parse}, *\textsc{Fill}^{Nuc}\}$$

The final candidate (1′.d) = .□V.CV.⟨C⟩ violates $\textsc{Fill}^{Ons}$ rather than $\textsc{Fill}^{Nuc}$ (as well as PARSE):

$$marks(.\square V.CV.\langle C\rangle) = \{*\textsc{Parse}, *\textsc{Fill}^{Ons}\}$$

The marks incurred by these candidates are summarized in the following table:

(3″) **Marks incurred by Four Candidate Parses of /VCVC/**

Candidates		ONS	−COD	PARSE	$\textsc{Fill}^{Nuc}$	$\textsc{Fill}^{Ons}$
/VCVC/ →						
a.	.V.CVC.	*	*			
b.	⟨V⟩.CV.⟨C⟩			* *		
c.	⟨V⟩.CV.C□́.			*	*	
d.	.□V.CV.⟨C⟩			*		*

(The order in which the constraints are listed here is arbitrary.)

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, we can find the least marked, or optimal, one: this is the one and only well-formed parse that may be assigned to the input. 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 surface structure.[3]

[3] The notion of harmonic ordering defined here is identical with that of P&S (e.g., §5) but the formulation differs; the direct link between the two is provided by the Cancellation/Domination Lemma of P&S (Appendix).

(4) **Constraint Ranking, Harmonic Ordering, and Optimality.**
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, we write $\mathbb{C}_1 \gg \mathbb{C}_2$. 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 occuring 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 we write: $a \prec b$. (We also sometimes write *marks*(*a*) $\prec$ *marks*(*b*).) The harmonic ordering $\prec$ ranks all pairs of candidates. 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.[4]

We can illustrate harmonic ordering in the CV case by reexamining the table of marks (3″) under the assumption that the universal constraints have now been ranked by a particular grammar as follows:

(4.a) **Constraint Hierachy for L_1:**
ONS $\gg$ $-$COD $\gg$ FILL$^{\text{Nuc}}$ $\gg$ PARSE $\gg$ FILL$^{\text{Ons}}$

In the following *constraint tableau*, the most dominant constraints in L_1 appear to the left:

(4.b) **Constraint Tableau for $L_1 = \Sigma^{\text{CV}}_{\text{ep,del}}$**

Candidates	ONS	$-$COD	FILL$^{\text{Nuc}}$	PARSE	FILL$^{\text{Ons}}$
/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

[4] If two candidates are assessed exactly the same set of marks by the universal constraints, then they are equally harmonic (regardless of the constraint ranking). If it should happen that such a tie should occur for the most harmonic candidate output for an input, then the two outputs are both optimal, both well-formed, with the interpretation of free alternation.

optimal candidate is marked manually.[5] Starting at the bottom of the tableau, let us 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 $c \prec b$, we first cancel the common mark *PARSE; c then earns the worst mark of the two, *FILL$^{\text{Nuc}}$. We see the importance of retaining multiple marks in the comparison of b to d: one *PARSE mark cancels, leaving $marks'(b) = \{$*PARSE$\}$ and $marks'(d) = \{$*FILL$^{\text{Ons}}\}$. The worst mark is the uncancelled *PARSE incurred by b: so $b \prec d$.

This constraint hierarchy given in (4.a) and illustrated in (4.b) is that of a language in which all syllables have the surface form CV: onsets are required, codas are forbidden. In case of problematic inputs such as /VCVC/ where a faithful parse into CV syllables is not possible, this language uses overparsing to provide missing onsets and underparsing to avoid codas (it is the language denoted $\Sigma^{\text{CV}}_{\text{ep,del}}$ in P&S:§6.2.2.2). We will call this language L_1, and explore how the course of learning it differs from that of another CV-language, L_2.

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

(4.c) **Constraint Hierachy for L_2:**

ONS $\gg$ $-$COD $\gg$ FILL$^{\text{Ons}}$ $\gg$ PARSE $\gg$ FILL$^{\text{Nuc}}$

Now the tableau corresponding to (4.b) becomes (4.d); 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:

(4.d) **Constraint Tableau for L_2 = $\Sigma^{\text{CV}}_{\text{del,ep}}$**

Candidates	ONS	$-$COD	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 P&S's language $\Sigma^{\text{CV}}_{\text{del,ep}}$).

The relation between L_1 and L_2 illustrates the final central principle of Optimality Theory relevant here:

---

[5] Determining that this candidate is optimal requires demonstrating that it is more harmonic than *any* of the infinitely many competing candidates. A general technique for such demonstration, the Method of Mark Eliminability (P&S §7.3), proceeds by showing that any attempt to avoid the marks incurred by the putatively optimal output leads to alternatives which incur worse marks.

(5) **Typology by Re-Ranking.**
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.

In the CV theory, Universal Grammar imposes no restrictions on the ranking of the constraints we have discussed here (3). 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 P&S:§6.[6] In this typology, syllable structures may have required or optional onsets, and, independently, forbidden or optional codas. In a principles-and-parameters theory, such a typology might arise from a parameter which governs whether the principle 'Syllables have onsets' does or does not apply (unviolated on the surface) in the language. Here, we have a corresponding principle, ONS (3′), but it is a constraint present in all languages, violable when (and only when) dominated by conflicting constraints. By ranking the constraint differently in their hierarchies, languages differ in whether the constraint is surface-unviolated. (But even when ranked low enough to be violated in well-formed surface forms, ONS still functions: as P&S emphasize, /CVCV/ is universally parsed .CV.CV. rather than *.CVC.V. even in languages where the syllables .CVC. and .V. of the latter parse are both well-formed; this is because ONS—and/or −COD—is still operative in the language, even though low-ranked. A perhaps more dramatic case is provided by FILL; even in languages in which FILL is surface-violated—as manifest in epenthetic material—FILL still functions absolutely crucially to sharply limit possible epenthesis sites: P&S:§6.2.3.)

1.3 The Learning Problem

Substantive knowledge of language in Optimality Theory thus consists in knowledge of the universal set of possible inputs, outputs and *Gen* (2); of the universal constraints (3); of the ranking of those constraints particular to the language (4); 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.[7]

---

[6] Other explicit Optimality-Theoretic typologies derived from re-ranking universal constraints include that of sonority-based inventories of onset, nucleus and coda segments, including an explanation of the onset/coda licensing asymmetry (P&S:§8); feature-based segmental inventories (P&S:§9); and systems of case marking and grammatical voice (Legendre, Raymond, & Smolensky 1993). These phonological typologies all involve universal restrictions on constraint rankings, when the constraint set is enlarged beyond those of (3). By far the most striking case is McCarthy & Prince 1993's Optimality-Theoretic formulation of prosodic morphology, which consists largely in the principle that morphological constraints are universally dominated by prosodic constraints.

[7] It may well be that the Optimality-Theoretic constraints of universal grammar will turn out to need parameters; in this case, parameter-setting will also be part of the learning process. We limit our discussion to 'typology by re-ranking' because that is what has figured most prominently thus far in Optimality-Theoretic analyses of cross-linguistic variation, and because that is the novel aspect of the learning problem in comparison with principles-and-parameters theory.

It should also be observed that, while Optimality Theoretic analyses have to date succeeded to a significant degree in avoiding idiosyncratic, language-particular constraints, such constraints have still

Concerning acquisition of the lexicon, the reader is referred to discussion in P&S:§9. Here we assume that the learner has a hypothesized lexicon in hand, and faces the problem of learning the ranking of known (or, for that matter, merely hypothesized) constraints.

The initial data for the learning problem are pairs consisting of an input and its well-formed (optimal) parse. For example, the learner of the CV language L_1 might have as an initial datum the input /VCVC/ together with its well-formed parse .□V.CV.⟨C⟩ (4.b). Together with this single piece of explicit positive evidence comes a large mass of implicit negative evidence. Every alternative parse of this input is known to be ill-formed; for example, the faithful parse *.V.CVC. is ill-formed. Thus the learner knows that, with respect to the unknown constraint hierarchy,

$$\text{.V.CVC.} \prec \text{.□V.CV.⟨C⟩}$$

Furthermore, corresponding harmonic comparisons must hold for *every* sub-optimal parse.[8]

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

[sub-optimal parse *sub-opt* of input *i*] ≺ [optimal parse *opt* of input *i*]

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

2. The Recursive Constraint Demotion (RCD) Algorithm

Before giving a general description of the algorithm, we first illustrate its use. In §2.1 we show how the algorithm learns the language L_1 from the single positive example which we have discussed, /VCVC/ → .□V.CV.⟨C⟩; we then show how the algorithm learns the different language L_2 from its (different) parse of the same input. Then in §2.2 we give a general but somewhat informal statement of the algorithm, and in §2.3 an informal summary of the proof of its convergence and correctness, and discuss its computational complexity. More formal treatment is provided in the Appendix.

2.1 An Example: Learning CV Syllable Structure

In the first language we consider, L_1, the correct parse of /VCVC/ is .□V.CV.⟨C⟩. This fact forms our starting point, one initial datum. Table (3″) gives the marks incurred by the well-formed parse (labelled *d*) and by three ill-formed parses. From this table we form the following table of *mark-data pairs*:

---

sometimes proved necessary, and learning them may need to be part of a comprehensive learning theory for Optimality Theory.

[8] Unless the grammar leads two candidate outputs to tie for most harmonic, yielding two optimal outputs, all candidates other than the observed output can be inferred to be sub-optimal. The case of ties is taken up in §4.1 below.

(6) **Mark-data Pairs (L_1)**

	sub-opt ≺ *opt*			*loser-marks* = *marks*(*sub-opt*)	*winner-marks* = *marks*(*opt*)
a ≺ *d*	.V.CVC.	≺	.□V.CV.⟨C⟩	{*Ons, *−Cod}	{*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 output *sub-opt* be less harmonic than the optimal output *opt*, the marks incurred by the former, *marks*(*sub-opt*) must collectively be worse than *marks*(*opt*). According to (4), 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 (6):

(7) **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, *−Cod}	{*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 cancelled 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 cancelled 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 cancelled. 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 table (7) of mark-data after cancellation is 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:

(8) **Stratified Domination Hierarchy**

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

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 outputs for a given input, since it fails to specify the relative

ranking of constraints which may conflict. This has the consequence, discussed below in §4.1, that the output of the algorithm may fail to decide between candidates for which relevant training data is unavailable.

N.B.: 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 constrants are eliminated from the mark-data table (7), so that only not-yet-ranked constraints are left in the table.

The key observation is this:

(9) **Key observation.**

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 P&S: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 (7).

When the algorithm begins, the not-yet-ranked constraints comprise the entire universal set (3.a–e):

$$\textit{not-yet-ranked-constraints} = \{\textsc{Ons},\ -\textsc{Cod},\ \textsc{Parse},\ \textsc{Fill}^{\textsc{Nuc}},\ \textsc{Fill}^{\textsc{Ons}}\}$$

Examining the rightmost column of the mark-data table (7) we see that two marks, *PARSE and *$\textsc{Fill}^{\textsc{Ons}}$, appear in the list of uncancelled winner marks. So the two constraints PARSE and $\textsc{Fill}^{\textsc{Ons}}$ 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:

(9.a) $$\textit{highest-ranked-constraints} = \{\textsc{Ons},\ -\textsc{Cod},\ \textsc{Fill}^{\textsc{Nuc}}\}$$

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

(9.b) $$\textit{not-yet-ranked-constraints} = \{\textsc{Parse},\ \textsc{Fill}^{\textsc{Ons}}\}$$

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 *–COD appears), and also the pair $c \prec d$, because *$\textsc{Fill}^{\textsc{Nuc}}$ appears. The new mark-data table is thus:

(9.c) **Mark-data Pairs (L_1, After First Pass)**

sub-opt ≺ *opt*		*loser-marks*	*winner-marks*
b ≺ *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 (9.a), a reduced list of not-yet-ranked constraints (9.b), and a reduced mark-data table (9.c). 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 (9.c), we examine the rightmost column of the table, and observe that of the two *not-yet-ranked-constraints* (9.b), only one, FILL^{Ons}, appears. The remaining constraint, then, is output as the next stratum of highest-ranked constraints:

(9.a′) *highest-ranked-constraints* = {PARSE}

This leaves

(9.b′) *not-yet-ranked-constraints* = {FILL^{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.

(9.c′) **Mark-data Pairs (L_1, After Second Pass)**: none

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

{ONS, −COD, FILL^{Nuc}} ≫ {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^{Ons}.

(9.a″) *highest-ranked-constraints* = {FILL^{Ons}}

This leaves

(9.b″) *not-yet-ranked-constraints* = {}

so the algorithm terminates, with the final result:

(10) **Learned Stratified Hierarchy for L_1:**

{ONS, −COD, FILL^{Nuc}} ≫ {PARSE} ≫ {FILL^{Ons}}

This result represents a class of all 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 (10) is the particular total ranking given in (4.a): this is but one of the correct hierachies 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 parse of the same input, /VCVC/, which is now ⟨V⟩.CV.C☐́. (candidate *c* in (1′); see the tableau (4.d)). The mark-data table used by the algorithm, containing the marks of *sub-opt* ≺ *opt* pairs after cancellation, is now:

(11) **Mark-data Pairs after Cancellation (L_2)**

	sub-opt ≺ *opt*		*loser-marks*	*winner-marks*
a ≺ *c*	.V.CVC. ≺	⟨V⟩.CV.C☐́.	{*ONS, *−COD}	{*PARSE, *FILL^{Nuc}}
b ≺ *c*	⟨V⟩.CV.⟨C⟩ ≺	⟨V⟩.CV.C☐́.	{~~*PARSE~~ *PARSE}	{~~*PARSE~~ *FILL^{Nuc}}
d ≺ *c*	.☐V.CV.⟨C⟩ ≺	⟨V⟩.CV.C☐́.	{~~*PARSE~~ *FILL^{Ons}}	{~~*PARSE~~ *FILL^{Nuc}}

This table is identical to its L_1 counterpart (7) except that the marks *FILL^{Nuc} and *FILL^{Ons} are interchanged. The result of the algorithm is therefore the same as before, (10), with this exchange made.

(12) **Learned Stratified Hierarchy for L_2:**

{ONS, −COD, FILL^{Ons}} ≫ {PARSE} ≫ {FILL^{Nuc}}

Again, this stratified hierarchy is correct: its further refinements into totally-ranked hierarchies, including the one we singled out in (4.c), 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.

2.2 General Statement of the RCD Algorithm (Informal Version)

Given:

universal-constraints = a set of universal constraints
initial-data = a set of well-formed outputs of the target language *L*

We assume that *L* arises from some (not necessarily unique) total ranking *R* of the universal constraints.

To Find:

A stratified hierarchy in which these are the optimal parses of their corresponding inputs.

We proceed via an intermediate step in which *initial-data* is prepared for the algorithm.

2.2.1 Data Preparation

Given:

universal-constraints = a set of universal constraints
initial-data = a set of well-formed outputs of the target language *L*

Generate:

a. For each output in *initial-data* (*opt*), choose a set of sub-optimal competitors (*sub-opt*)
b. Make a table of these pairs, *sub-opt* ≺ *opt*, listing the marks incurred by the *sub-opt*s in one column, *loser-marks*, and the marks incurred by *opt*s in another, *winner-marks*.
c. The result is *mark-data* =

sub-opt ≺ *opt*	*loser-marks* = *marks*(*sub-opt*)	*winner-marks* = *marks*(*opt*)
...	...	...

2.2.2 Recursive Constraint Demotion (RCD)

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 appear 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 reduced *mark-data* and the reduced *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*).

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 Initialization 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 **C**; if it is *sub-opt*, then a token of the mark ***C** appears in the column *loser-marks*; if it is *opt*, then the token of ***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 form. 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.

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 rows in the initial *mark-data* table, i.e., the number of pairs *sub-opt* $\prec$ *opt* used.

So the number of steps required for all the passes can't exceed

$$(N_{constr})^2 N_{pairs}$$

In the worst case, the RCD algorithm is quadratic in the number of constraints, and linear in the number of mark-data pairs. This makes the algorithm quite efficient, although further work is required to investigate efficiency issues in the choice of *sub-opt* $\prec$ *opt* pairs.

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.

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* $\prec$ *winner-marks*

To Find:

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

Prior to use of CD, as for RCD (see §2.2.1) *mark-data* must be prepared as appropriate for the version of CD which will be used (see §3.2).

Core CD Algorithm

I. Mark Cancellation: As for RCD (§2.2.2)

II. Constraint Demotion (repeat this step until no demotions occur)
for each pair *loser-marks* $\prec$ *winner-marks* in *mark-data*
i. find the mark $*\mathbf{C}_{\text{loser}}$ in *loser-marks* which is highest-ranked in $\mathcal{H}$
ii. for each $*\mathbf{C}_{\text{winner}}$ in *winner-marks*
if $\mathbf{C}_{\text{loser}}$ does not dominate $\mathbf{C}_{\text{winner}}$ in $\mathcal{H}$, then demote constraint $\mathbf{C}_{\text{winner}}$ to the stratum in $\mathcal{H}$ immediately below that of $\mathbf{C}_{\text{loser}}$ (creating such a stratum if it does not already exist)

Output $\mathcal{H}$

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.

(13) **Batch CD**
a. Compile all available output forms into *initial-data*.
b. Perform Data Preparation, generating *mark-data*.
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 (31) below): all the data is processed simultaneously by Core CD.

(14) **On-line CD**

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

Execute repeatedly:

b. Select one optimal output *opt* and one suboptimal competitor *sub-opt*.

c. Set *mark-data* = {*marks*(*sub-opt*) ≺ *mark*(*opt*)}.

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

On-line CD operates by applying Core CD separately to each *sub-opt* ≺ *opt* pair.

(15) **I/O CD**

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

Execute repeatedly:

b. Select one optimal output *opt*.

c. Perform Data Preparation, generating *mark-data.*

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 form *opt* from the language is processed at a time, but a number of competitors *sub-opt* to *opt* may be processed at once.

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

$$\mathcal{H} = \mathcal{H}_0 = \{\textsc{Parse}, \textsc{Fill}^{\textsc{Nuc}}, \textsc{Fill}^{\textsc{Ons}}, \textsc{Ons}, -\textsc{Cod}\}$$

Step b has us pick a *sub-opt* ≺ *opt* pair; suppose we pick *b* ≺ *d* of (6). Step c has us generate the corresponding pair of mark lists, as shown in the second row of (6). Step d has us execute Core CD.

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

(16) **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$^{\textsc{Ons}}$}

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

$$\mathcal{H} = \{\textsc{Parse}, \textsc{Fill}^{\textsc{Nuc}}, \textsc{Ons}, -\textsc{Cod}\} \gg \{\textsc{Fill}^{\textsc{Ons}}\}$$

This completes the execution of Core CD, so we return to step b of the On-Line CD procedure: picking another *sub-opt* ≺ *opt* pair. Suppose this is *a* ≺ *d* of (6):

(17) **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, *−COD}	{*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 −COD 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:

$$\mathcal{H} = \{\text{FILL}^{Nuc}, \text{ONS}, -\text{COD}\} \gg \{\text{PARSE}, \text{FILL}^{Ons}\}$$

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

(18) **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/:

(19) **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:

$$\mathcal{H} = \{\text{FILL}^{Nuc}, \text{ONS}, -\text{COD}\} \gg \{\text{PARSE}\} \gg \{\text{FILL}^{Ons}\}$$

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, (10).

3.4. Analysis

Here's a sketch of the proof that the procedures of §3.3 based on the CD algorithm converge to a correct stratified hierarchy for the target language L.

The basic idea is this: If at any point CD places a constraint $\mathbf{C}_3$ in the third stratum, it is only because the data seen so far entail a sequence of constraint dominations in which two other constraints $\mathbf{C}_1$ and $\mathbf{C}_2$ form a domination chain over $\mathbf{C}_3$:

$$\mathbf{C}_1 \gg \mathbf{C}_2 \gg \mathbf{C}_3.$$

Constraints start at the top of the hierarchy and are only demoted by CD when they have to be in order to accomodate the data currently being evaluated. Each constraint is demoted until, eventually, it arrives in the highest stratum permitted it by the language. Each demotion brings the constraint demoted at least one stratum closer to its final destination; since constraints are never promoted, the hierarchy converges monotonically, in a sense defined precisely below, to a correct hierarchy.

We present a few definitions and then a series of propositions leading to the convergence result. In this section, we assume that all constraint hierarchies involve the same set of constraints.

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

(21) **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 $\mathcal{H}_1$.

This concept must be clearly distinguished from a more obvious relation between hierarchies:

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

(23) **Prop.** For any language L generated by an underlying total constraint ranking, there exists a unique stratified hierarchy $\mathcal{H}_L$ which generates L and which has the property that each constraint is ranked as high as possible; that is, for any other hierarchy $\mathcal{H}'$ which generates L, every constraint is in the same stratum or a lower stratum in $\mathcal{H}'$ relative to $\mathcal{H}_L$. We term $\mathcal{H}_L$ the *h-dominant target stratified hierarchy* for L, or simply the *target* for L.

(24) **Cor.** The target for L h-dominates every total constraint ranking which generates L.

(25) **Prop.** The hierarchy output by CD is h-dominated by the input hierarchy.

This holds because CD only demotes constraints.

(26) **Prop.** If the input hierarchy h-dominates the target, so does the output hierarchy.

This holds because CD will never demote a constraint lower than necessary; the constraint is as low as necessary in the target.

(27) **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 $\mathbf{C}$, of the difference between the offset of $\mathbf{C}$ in $\mathcal{H}_2$ and in $\mathcal{H}_1$.

(28) **Lemma.** The h-distance between the current hierarchy and the target decreases by at least 1 for each demotion/informative pair.

(29) **Lemma.** The h-distance from $\mathcal{H}_0$ to the target cannot exceed $\frac{1}{2}(N_{constr}-1)N_{constr}$.

(30) **Thm.** Starting with $\mathcal{H}_0$ and presenting the CD algorithm with data according to any of the procedures Batch CD, On-Line CD, or I/O CD, the procedure converges on the target hierarchy $\mathcal{H}_L$ for L after at most $\frac{1}{2}(N_{constr}-1)N_{constr}$ informative examples.

(31) **Prop.** The relation of Batch CD to RCD is this. The first iteration of Batch CD runs once through all the data, demoting a certain number of constraints. The contraints 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. This same property then holds recursively for subsequent iterations. In general, 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 $n > 1$).

In other words, the real difference between Batch CD and RCD is slight: Batch CD demotes constraints to a number of levels simultaneously, never demoting any constraints in the top n strata after iteration n, while the n^{th} call to RCD in effect just demotes constraints from the n^{th} to the $n-1^{\text{st}}$ stratum. This is why we have used the term Recursive Constraint Domination even though, unlike the other algorithms we have discussed, RCD does not explicitly employ Core CD.

4. Consequences and Extensions

4.1 Ties

The way we have approached the learning problem 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 to assume 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' form selected is in fact a second optimal form, then all marks cancel and this pair is discarded.

Two forms which do not incur identical marks can only tie for optimality if the underlying constraint hierarchy is not totally ranked, for two different marks must in this case have the same rank: neither can dominate the other. In this case, observing one of the optimal forms and assuming that all others are sub-optimal can lead to errors.

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. We might, for example, suppose that the target language derives from an underlying stratified hierarchy rather than a totally ranked hierarchy. We 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. Much of the analysis extends to this setting, although the Core CD Algorithm needs to be extended with an addition step to handle pairs $opt_1 \sim opt_2$ of tying optima. In this step, 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. Indeed, achieving ties for optimality between forms which incur different marks is a delicate matter. It may be that learning languages which do not derive from a totally-ranked hierarchy is in general much more difficult than the case we have studied here. In that case, demands of learnability would suggest that universal grammar rules out such languages.

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. That learning can be much less difficult when the hypothesis space is larger than the target space is a theme of recent work in Computational Learning Theory. While we cannot at this point rule out the possibility of an equally efficient learning algorithm which operates purely in the smaller space of totally-ranked hierachies, it seems quite unlikely to us.

Learning in the context of initial data which contains outright errors is almost certainly a problem of much greater complexity than that addressed here. It is worth noting, however, that if the initial data presented to the RCD algorithm are not consistent with any total ranking of the given constraints, the algorithm discovers this fact: At some stage of its execution, it fails in its attempt to find a constraint which does not appear among the *winner-marks* (Step II.a; see the discussion of convergence in §2.3).

4.2 The Superset Principle

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). Regarded as a model of language acquisition, the CD algorithm suggests a *Superset Principle* in which the languages hypothesized earlier in learning are larger than later languages. The initial state $\mathcal{H}_0$ is a completely unrefined stratified hierarchy in which all constraints form a single stratum. For the CV syllable structure example considered above, in this initial state, *all* the candidate parses we considered are tied for

optimality, therefore all are accepted initially as well-formed. Data is then used to successively refine this hierarchy — in a somewhat loose sense: the output of CD at any stage is not in general literally a refinement of its input, but the number of strata is strictly non-decreasing. The end result in the syllable structure example is that all but one of the initially accepted outputs is rejected by the final grammar.

Appendix. Formal Statement of the RCD Algorithm and Proof of Its Correctness

A.1 The RCD Algorithm

Given:

A list of constraints $\mathcal{H}$

A list of mark-data pairs:

$mark\text{-}data = \{(loser\text{-}marks_i , winner\text{-}marks_i)\}$

where each $loser\text{-}marks_i$ and $winner\text{-}marks_i$ is a list of marks = {*m}

Part I: Mark Cancellation

for each pair *i* in *mark-data*
    for each occurrence of a mark *m in both $loser\text{-}marks_i$ **and** $winner\text{-}marks_i$
        remove *m from $winner\text{-}marks_i$; remove *m from $loser\text{-}marks_i$
    if $winner\text{-}marks_i$ is empty, remove ($loser\text{-}marks_i$, $winner\text{-}marks_i$) from *mark-data*
endfor

Part II: Recursive Ranking
Set *not-yet-ranked-constraints* = $\mathcal{H}$
while *not-yet-ranked-constraints* not empty do

    set *highest-ranked-constraints* to empty
    copy *not-yet-ranked-constraints* to *highest-ranked-constraints*

    for each pair *i* in *mark-data*
        for each constraint $\mathbf{C}_j$ in *highest-ranked-constraints* matching a mark *m in $winner\text{-}marks_i$
        remove $\mathbf{C}_j$ from *highest-ranked-constraints*
    endfor

    for each constraint $\mathbf{C}_j$ in *highest-ranked-constraints*
        remove $\mathbf{C}_j$ from *not-yet-ranked-constraints*
    endfor

    for each pair *i* in *mark-data*
        if a mark *m in $loser\text{-}marks_i$ matches a constraint $\mathbf{C}_j$ in *highest-ranked-constraints*
            then remove ($loser\text{-}marks_i$, $winner\text{-}marks_i$) from *mark-data*
    endfor

    OUTPUT *highest-ranked-constraints*

endwhile

This algorithm sequentially outputs lists of constraints, with the first list containing the constraints ranked the highest, the second list containing the constraints ranked second highest, etc.

A.2 Proof of Correctness of the Algorithm

The algorithm is correct if it outputs a partial ranking of the constraints which assigns strictly higher harmony to *opt* than to *sub-opt* for each pair *sub-opt* ≺ *opt*. Thus, any totally ranked hierarchy consistent with the output (partially ranked) stratified hierarchy will correctly evaluate all of the data presented.

Part I of the algorithm, where common marks are cancelled within mark-data pairs, is justified by the Cancellation Lemma (P&S:139) which says precisely that the relative harmony of two structures will be correctly determined if all common marks are cancelled, and only the remaining marks are considered. Further, if cancellation results in the winner of a pair having no remaining marks, then the winner will be assigned higher harmony than the loser no matter which way the constraints are ranked. Consequently, the mark-data pair is guaranteed to be satisfied by the algorithm's output, and it contains no useful information to constrain the ranking, so it may be discarded.

Part II of the algorithm may be proven recursively by demonstrating that:

(a) an iteration correctly finds and outputs the subset of the given constraints that may be ranked highest;

(b) at the end of an iteration, the remaining mark-data pairs may only be explained on the basis of the constraints not yet output. Thus, what is left over is itself a well-defined problem, and the next set of constraints to be output is precisely the set of constraints that may be highest ranked in the remaining problem;

(c) each iteration is guaranteed to output a list with at least one constraint, so long as a solution exists that is consistent with all of the mark-data pairs. This ensures that the algorithm converges.

First, we prove that an iteration outputs those constraints which may be ranked highest (of the given constraints). From the set of constraints not yet output, a constraint is removed if a corresponding mark is found in at least one of the winners' lists of marks. The Cancellation/Domination Lemma (P&S:130,148,221) states that one parse is assigned higher relative Harmony than another if and only if every uncancelled mark incurred by the winner is dominated by an uncancelled mark of the loser. The first part of the algorithm insures that only uncancelled marks are considered. If an uncancelled mark incurred by the winner is dominated by one incurred by the loser, then the mark incurred by the winner cannot be among the highest ranked. Thus, all constraints eliminated by this step cannot be among the highest ranked.

Each of the constraints remaining at the end of the step may or may not appear in any particular loser's list of marks. If it doesn't, then the constraint is neutral with respect to that mark-data pair, so ranking it highest is (vacuously) consistent with that mark-data pair. If it does, then ranking the constraint highest guarantees that the winner will be more harmonic, as the constraint's mark dominates all uncancelled marks of the winner, so ranking the constraint highest is (non-vacuously) consistent with the mark-data pair. Thus, ranking all constraints remaining in the list highest is consistent with all of the mark-data pairs.

Next, we prove that, at the end of an iteration, the remaining mark-data pairs may only be explained on the basis of the constraints not yet output. First observe that no constraints that were output during the iteration may have marks appearing in any of the *winner-marks* lists, because any constraint with a mark appearing in any *winner-marks* was automatically disqualified from being output. Further, the step of the algorithm preceding the output step removes all mark-data pairs in which a *top-ranked-constraint* has a mark in the *loser-marks* list. Thus, the remaining mark-data pairs have no marks corresponding to constraints already output.

The constraints and mark-data pairs remaining at the end of an iteration may then be viewed as a problem set of their own. The next set of constraints to be output is exactly the set of constraints that may be ranked highest given the remaining constraints and the remaining mark-data pairs. This recursion

bottoms out when no mark-data pairs are left. Because mark-data pairs with empty *winner-marks* were removed in Part I of the algorithm, it is guaranteed that while there are mark-data pairs in the datalist, there will be constraints not yet output, including each constraint with a mark in at least one of the win lists of the remaining mark-data pairs.

If no mark-data pairs remain in the list, then no constraints will be removed from the list of remaining constraints, and all of them will be output. This is correct, because each of the constraints (vacuously) satisfies the remaining data (since there is none).

Finally, we prove that the algorithm converges whenever a solution exists. Suppose, to the contrary, that during an iteration, all (not yet output) constraints were disqualified from being highest-ranked. This means that each of the constraints has a mark appearing in the *winner-marks* list of at least one mark-data pair. This means that the mark-data pairs collectively require each constraint to be dominated by another constraint, which violates the requirement that constraint domination be non-circular. Thus, no solution exists which will satisfy all of the mark-data pairs. So, if a solution does exist, then each iteration will output a list containing at least one constraint, and reduce the list of constraints not yet output. This guarantees that the algorithm will converge.

Acknowledgements

We are most grateful for important comments and encouragement from Alan Prince, Géraldine Legendre, Jane Grimshaw, and Bruce Hayes. Tesar acknowledges the support of an NSF Graduate Fellowship, and both authors acknowledge the support of NSF grant IRI-9213894. We are grateful to the Institute of Cognitive Science at the University of Colorado at Boulder for partial support of the First Rutgers Optimality Theory Workshop where this work was first presented.

References

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

Archangeli, Diana. 1984. *Underspecification in Yawelmani phonology and morphology*. Doctoral Dissertation. MIT, Cambridge, MA. [Garland Press, 1988.]

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

Broselow, Ellen. 1982. On the interaction of stress and epenthesis, *Glossa* 16, 115-132.

Clements, G. N and S.J. Keyser. *CV Phonology*. Cambridge, MA: MIT Press.

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

Itô, Junko. 1986. *Syllable Theory in Prosodic Phonology*, Ph. D. Dissertation, UMass, Amherst.

Itô, Junko. 1989. A Prosodic Theory of Epenthesis, *Natural Language & Linguistic Theory* 7, 217-260.

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

Kaye, Jonathan and Jean Lowenstamm. 1984. De la syllabicité, in F. Dell, D. Hirst, and J.-R. Vergnaud, ed., *Forme sonore du langage*, Hermann, Paris.

LaPointe, Steven and Mark Feinstein. 1982. The Role of Vowel Deletion and Epenthesis in the Assignment of Syllable Structure, in *The Structure of Phonological Representations.Part II*, ed. H.v.d.Hulst and N. Smith, 69-120. Foris, Dordrecht, .

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

McCarthy, John. 1979. *Formal Problems in Semitic Phonology and Morphology*, Doctoral dissertation, MIT, Cambridge, MA.

McCarthy, John and Alan Prince. 1993. *Prosodic Morphology I: constraint interaction and satisfaction*. Ms. University of Massachusetts, Amherst, and Rutgers University, New Brunswick, NJ. To appear as Linguistic Inquiry Monograph, MIT Press, Cambridge, MA.

Piggott, Glyne and Raj Singh. 1985. The phonology of epenthetic segments. *Canadian Journal of Linguistics* 30, 415-453.

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

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

Prince, Alan and Paul Smolensky. 1993. *Optimality Theory: Constraint Interaction in Generative Grammar*. Ms. Rutgers University, New Brunswick, and University of Colorado, Boulder. NJ. To appear as Linguistic Inquiry Monograph, MIT Press, Cambridge, MA.

Selkirk, Elisabeth. 1981. Epenthesis and degenerate syllables in Cairene Arabic. In *Theoretical Issues in the Grammar of the Semitic Languages*, ed. H. Borer and J. Aoun. MIT, Cambridge.

Steriade, Donca. 1982. Greek prosodies and the nature of syllabification. Doctoral Dissertation. MIT, Cambridge, MA.

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