COMMITMENT-BASED LEARNING OF HIDDEN LINGUISTIC STRUCTURES

By

CRYSTAL GAYLE AKERS

A Dissertation submitted to the

Graduate School – New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Linguistics

written under the direction of

Bruce Tesar

and approved by

_____

_____

_____

_____

May 2012

ABSTRACT OF THE DISSERTATION

COMMITMENT-BASED LEARNING OF HIDDEN LINGUISTIC STRUCTURES

by CRYSTAL GAYLE AKERS

Dissertation Director:
Bruce Tesar

Learners must simultaneously learn a grammar and a lexicon from observed forms, yet some structures that the grammar and lexicon reference are unobservable in the acoustic signal. Moreover, these "hidden" structures interact: the grammar maps an underlying form to a particular interpretation. Learning one structure depends on learning the structures it interacts with, but if the learner commits to one structure, its interactions can be exploited to learn others. The Commitment-Based Learner (CBL) employs this strategy using error-driven learning (Gold 1967, Wexler and Culicover 1980) and inconsistency detection (Tesar 1997) to determine when to make commitments and what kinds of commitments to make.

The CBL overcomes structural ambiguity by extending branches from a hypothesis and committing to a separate structural interpretation in each branch, as in the Inconsistency Detection Learner (Tesar 2004). It resolves lexical ambiguity by making piecewise commitments to feature values, following the Output-Driven Learner (Tesar, to appear). Each branch has its own lexicon whose values reflect the interactions of underlying forms with the branch's structural commitments.

In computer simulations, the CBL learns all 97 languages in a constructed typology whose linguistic system includes 370 million grammar and lexicon combinations. For each language learned, the CBL takes far fewer steps than needed to exhaustively search for a consistent and restrictive combination. Employing inconsistency detection with Multi-Recursive Constraint Demotion (Tesar 1997) makes the CBL highly efficient, and it compares favorably in success and efficiency to its major stochastic competitors (Apoussidou 2007, Jarosz 2006, to appear).

The dissertation also introduces a previously unrecognized global lexical ambiguity defined by paradigmatic equality. Paradigmatic equals (PEs) have different grammars, but because their morpheme behaviors are identical, their learning data are equivalent and foil learning by inconsistency detection. To distinguish PEs, the CBL finds consistent mappings derived from words with unset features set to mismatch their surface values. A mapping with an error by the current ranking contributes new ranking information, allowing the learner to derive the hypothesis consistent with the PE that includes the mapping. In the system investigated, there are always two such mappings, each corresponding to a different PE.

ACKNOWLEDGEMENTS

It is a true pleasure to write these acknowledgements, both for the opportunity to recognize all of those who have helped me reach this point and for the conclusion signified to this long but rewarding process. I begin at the end of this experience, with the faculty who advised this dissertation. Bruce Tesar has been a remarkable chair and this dissertation owes much to him, not only for the works that have informed the learner presented here, but for his rigorous attention to detail, careful questioning, and discussions that have at every stage strengthened my work and writing. I cannot give thanks enough for the time and patience he has devoted to this project. I am also grateful to Alan Prince for his discussions of this work and for what was, for me, surprising but heartening encouragement. More generally, I feel fortunate to have worked with both Bruce and Alan on this dissertation and on a qualifying paper at Rutgers, and to have witnessed their interactions with me, with other students, and with each other during classes and reading group meetings; their intellectual curiousity and pursuit of precision and clarity are inspiring.

Shigeto Kawahara read drafts of this dissertation with the fresh eyes of someone who has not been immersed in learnability, and I thank him for providing that outside perspective, and especially for helping me to identify those parts of this work that could appeal to a broader audience. The dissertation was also improved by Gaja Jarosz's keen and critical draft review, and I thank her too for the discussions of her work that helped me to better understand her learner.

The faculty and students at Rutgers in my time there helped make this commuter feel like part of a community. Jane Grimshaw and Veneeta Dayal served on the committee of my first qualifying paper at Rutgers, and their warm yet exacting attention to that work made it better and vastly improved the experience of writing it. I was always glad to have them in my corner. I learned much from Viviane Déprez and Paul de Lacy and thank them for their efforts toward my qualifying papers as well. I must also thank my fellow students, for sharing insights, laughs, rides, and their futons, especially Paula, Patrick, Jimmy, Carlos, Aaron, Will, Jeremy, Mike, Seunghun and Věra.

My interest in linguistics began at Swarthmore College, where I took a semantics course only because it counted toward the philosophy major, my intended goal. That course was all it took to change my path. Kari Swingle, the instructor, was a model of careful class preparation and amazing recall, and I eagerly took every seminar I could with her. Ted Fernald and Donna Jo Napoli were also valuable influences at Swarthmore, but it was my last seminar, an introduction to Optimality Theory with John Alderete, that sealed my decision to come to graduate school for more linguistics, and in particular, for more OT and phonology.

Because I have the opportunity to do so, I also want to recognize the public schools of Pinellas County, Florida, and particularly the MEGSSS, IMAST, and CAT programs. After teaching for several years in middle and high schools myself, I know a small piece of the hard work and dedication that individual teachers bring to the classroom, and I know how special it is to find a group of talented and dedicated teachers working together at the same school with the freedom to design demanding and exciting curricula.

Imagine a bus ride home from school, several sixth graders bouncing in the back rows with heads bent over their math homework, proofs in symbolic logic that had them reminding each other of the difference between *modus ponens* and *modus tollens:* that was my experience. The math, science and technology magnet programs I was fortunate to attend in middle and high school had a lasting impact on my life, as the learnability simulations of this dissertation can attest. I remember with fondness and sincere gratitude the teachers I have learned from over the years, but especially Mrs. Chipelo, Mrs. Quarterman, Mrs. Ranieri, Mr. Lindsay, Ms. Terry, and Mr. Epstein.

Many family and friends have supported and encouraged me throughout graduate school. My in-laws, Carol and Ted Kneeland, were my cheerleaders in the frozen north. Colleen Hazelton was always ready with a sympathetic ear and, just as importantly, a fenced yard for our puppy play-dates. Melissa Marvel organized the coffees, dinners, and walks that helped keep me and many other friends beyond graduate school sane through the long winters. How can I begin to thank my parents, Joe and Judy Akers? They have always believed in me and enabled me to pursue every opportunity I could. Though they did not attend college themselves, they have never stopped learning. I am thankful that they are both able to see this dissertation be completed. And to Teresa, Kevin, and Nate, who promised as teenagers when I was born that they would see me through college if anything should happen to our parents: you're officially off the hook.

Finally, to Doug Kneeland, much love and gratitude for all your support and good-humor over the years. Thank you for being my partner in adventures past, present, and future, now with someone new whose presence is known but only just felt.

TABLE OF CONTENTS

1 ISSUES IN LEARNING HIDDEN STRUCTURE

Linguistic theories posit a variety of structures taken to be part of the adult speaker's knowledge of the language. Some of these structures are revealed in the physical linguistic signal itself, while others must be inferred from the speaker's knowledge of the grammar. For the language learner, unobservable structure poses a significant challenge because of its relationship to the grammar. Knowing the grammar would help the learner identify this "hidden" structure, but the learner does not yet know the grammar. In turn, knowing the hidden structure would help the learner infer the grammar, but of course the learner cannot yet identify the correct structure. This dissertation presents the Commitment-Based Learner (CBL) for overcoming these challenges to learn a grammar and two kinds of hidden structure simultaneously. Learning simulations of the CBL demonstrate that this learner can successfully and efficiently learn a language, including its grammar and lexicon, from its overt forms.

While what counts as hidden or unobservable structure is itself a topic for investigation, the intent of the concept can be illustrated by a sentence such as "*The dogs want to fetch sticks."* Each word has a morphological composition that cannot be determined simply by its sound. For example, both *dogs* and *sticks* contain the plural morpheme, but its pronunciation varies depending on its context: [z] in *dogs* and [s] in *sticks;* relatedly, each morpheme has an associated underlying phonological representation that may differ from what the learner actually hears. Syntactic structure is also hidden in this sentence: *the dogs* is the subject of the verb *want,* and it is also the implicit subject of the verb *fetch.*

In fact, hidden structure is quite common, occurring across linguistic components and at both the input and output levels of representation. Within semantics and syntax, quantifier raising, wh-movement, raising, control, and anaphoric binding are all types of hidden structure. These examples show that the general problem of learning hidden structures and the grammar at once is not limited to any one aspect of the language, and therefore must be central to any theory of language learning.

This dissertation focuses on learning two kinds of hidden phonological structure: foot structure and underlying forms. While the learner may discern stress by attention to its phonetic correlates, like loudness or vowel duration, laboratory experiments suggest that similar phonetic correlates for a foot boundary may not be available to the learner. For example, in a production experiment, Ota, Ladd and Tsuchiya (2003) find that foot-final moras in Japanese are no longer than other moras in the same foot. On the perception side, tendencies to group tones as trochees based on intensity contrasts and iambs on durational contrasts, as formulated in the Iambic/Trochaic Law (Hayes 1995), turn out to depend on the listener's native language. In particular, Kusumoto and Moreton (1997) find that while both English and Japanese speakers parse non-speech sounds differing in intensity as trochees, Japanese speakers also have a tendency to parse durational contrasts as trochees. Iversen, Patel, and Ohgushi (2008) replicate the study with a larger sample of native Japanese speakers in Japan with similar results: almost half grouped durational contrasts into long-short sequences. These results suggest that language learners do not have an acoustic reference for foot boundaries and must learn the structure instead.

Based on physical observation alone, then, a three-syllable word with primary stress (Y) between two unstressed syllables (s) is ambiguous between the interpretations [(sY)s], [s(Ys)], and [s(Y)s]. Determining the correct interpretation matters for the learner because different interpretations correspond to different languages. Yet, simply identifying the correct interpretation will not suffice to determine the language, because different constraint rankings can yield the same interpretation.

Determining underlying forms poses a similar challenge for the learner. Each observed form is the output of some input to the grammar, but the input itself cannot be directly observed. When the learner hears *sYs*, nothing about this form alone tells the learner the underlying stress feature value of any syllable. Again, learning the grammar goes hand-in-hand with identifying this hidden structure: the grammar contains constraints on input-output correspondence that must be ranked, but the learner must somehow do so knowing neither the input nor the actual structure of the output.

As the preceding paragraphs have hinted, the problem of inferring grammar is intertwined with the problem of discerning hidden structure. The grammar itself is not observable except by its effects as represented in the observed forms. The overarching problem therefore involves the interrelatedness of hidden structure and the grammar: the learner cannot fully know any one aspect of the language without knowing something about at least one of the others.

Further complicating the issue of learning hidden structure and the grammar together is the requirement that the learning method be computationally tractable. Executing an exhaustive search over the full range of possibilities is impossible, and the search space

remains enormous even when limits are imposed such as disallowing unbounded insertion and deletion. For example, an exhaustive search over a set of 20 constraints will include 2.4×10^{18} total rankings. Though an exhaustive search of this space might appear manageable, the set of constraints needed to account for the full range of human language phenomena must be far greater than twenty. Now consider an exhaustive search over all possible underlying forms. Suppose all segments have five binary features. If inputs and outputs can differ only by the settings of features on corresponding segments in the same order, then a ten-segment monomorphemic output like [CV.CV.CV.CV.CV] would have $(2^5)^{10}$, or over 10^{15}, possible underlying forms. To look at the problem another way, suppose that a language has ten morphemes, each of which contains three segments with five binary features. In total, there are $(2^5)^3$, or 32768, different ways to set the feature values of each morpheme, and $(2^{15})^{10}$ or 10^{45} different possible lexica. These numbers, already quite large, do not take into account numerous other ways that inputs and outputs can vary, such as by having different numbers or orders of segments.

Finally, the learner also has to determine the correct structure of the outputs. For an Optimality-theoretic grammar, the number of possible structural descriptions for an input depends on the assumptions about the candidate generator GEN (Prince and Smolensky 1993); for the assumptions given in section 2.3.1, a five-syllable input will have 300 candidate structural descriptions, each one way of parsing the five syllables. After the preceding numbers, this seems trivial, but learning the language means determining, for each observed form, the correct underlying form, the correct structural interpretation, and the constraint ranking that will map the underlying form to the structural interpretation.

For more complex systems, exhaustively searching the range of possibilities for each kind of hidden structure and the grammar is simply not feasible.

Significant progress has already been made toward learning hidden structure and grammar together, including by using error-driven learning (Gold 1967, Wexler and Culicover 1908) and inconsistency detection (Tesar 2000), which tests if combinations of hypothetical structures are consistent with each other and whatever the learner knows about the grammar. Inconsistency detection has proven to be an efficient means of successfully learning a grammar and one kind of hidden structure at a time, whether structural interpretations (Tesar 2000) or underlying forms (Tesar 2009).

The natural development from these preceding learning algorithms is to apply them to the problem of how to simultaneously learn the hidden structures of both inputs and outputs with the grammar. The Commitment-Based Learner banks on the power of error-driven learning and inconsistency detection to learn successfully and efficiently. The key feature of this learner is its use of committed information, which enables the learner to interleave inconsistency detection across both types of hidden structure. The learner maintains multiple language hypotheses, each containing commitments to different lexical information and structural representations and to ranking conditions consistent with those commitments. The interdependence of hidden structures and the ranking becomes an asset with this approach, enabling the learner to narrow the space of the grammar hypotheses until one or more is found that is consistent with everything the learner observes in the data.

For example, each commitment to a structural interpretation has implications for the ranking through the conditions required to make the hypothesized interpretation optimal. Inconsistency detection allows the learner to determine which combinations of interpretations cannot be correct for the language, excluding hypotheses whose interpretations require contradictory ranking conditions. In turn, each combination of consistent structural interpretations provides the learner with a ranking and a set of output forms that can be used to make commitments about underlying forms.

This chapter reviews prior work done on the topic of learning hidden structures and grammar, focusing on the learners that inform the CBL and some of their key stochastic competitors, including those based on the Gradual Learning Algorithm (GLA) (Boersma 1997) and Maximum Likelihood Learning of Lexicons and Grammars (MLG) (Jarosz 2006). The use of error-driven learning (section 1.1.1) will be a common thread among some works, while the potential for and use of inconsistency detection (1.1.2) will emerge as a key difference between learning approaches, arising from different assumptions about what kind of information the learner can retain. The role of these concepts in learning algorithms will be emphasized in the sections on deriving rankings (1.2.), managing structural ambiguity (1.3) and learning the lexicon (1.4).

Chapter 2 introduces the Commitment-Based Learner with a discussion of the informative potential of mutual dependency among structures, followed by illustrations of the CBL's component procedures at critical learning points. Chapter 3 expands the view of the CBL to cover a complete simulation, following along as the learner successfully processes a data set from start to finish. Chapter 4 presents some of the

relationships found between languages in the typology used to test the CBL and discusses the implications of those relationships for learning. Finally, chapter 5 concludes the dissertation.

1.1 ERRORS AND INCONSISTENCIES

This section provides a review of error- and inconsistency detection, along with some examples of how both can be useful for learning. More specific examples will occur in the remainder of this chapter and indeed, throughout the dissertation, as the CBL is an error-driven learner that also relies on inconsistency detection.

1.1.1 ERRORS

In *error-driven learning* (Gold 1967, Wexler and Culicover 1980), an "error" refers to the mismatch that occurs when the learner's grammar parses an input to an output different from the observed form produced by adult speakers. Errors detected by the learner motivate changes to the learner's current grammar hypothesis, but whether the learner detects the error depends on the hypothesis used in the parsing attempt, including any knowledge the learner has about inputs, outputs, and rules for mapping inputs to outputs. Detecting an error alerts the learner that some aspect of the hypothesis must change, something new must be learned. Ideally, the learner should have a mechanism for using a detected error to identify the particular aspect of the grammar that should change and to change it in a way that prevents the error.

Tesar and Smolensky (1994, 2000) capture the information relayed by detected errors in the form of *mark-data,* or *winner-loser (W-L) pairs,* which record the violation profiles of the desired winner – W, the observed form – and a loser – L, any other candidate. The

W-L pair can be written in the form of a *comparative tableau* (Prince 2000) as in (1), where the winner is the trochaic candidate /ss/[(Ys)], and the loser is /ss/[(sY)].[1] An "L" indicates that a constraint prefers the loser by assessing fewer violations for the loser than for the winner; similarly, a "W" indicates that the winner is preferred. The cells associated with constraints which do not prefer either candidate are blank here, although the standard notation of the comparative tableau would assign each of these cells an "*e,*" indicating that the constraint in question assigns equal violations to the desired winner and the desired loser.

(1) A winner-loser pair[2]

/ss/	PARSE-σ	FT-BIN	IAMB	FNF
[(Ys)] ~ [(sY)]			L	W

Tesar (1998a) puts W-L pairs to use in error-driven learning, recognizing that the pairs can provide the learner with critical information about how the ranking must change in order to render the desired winner optimal in comparison to the loser. Specifically, at least one winner-preferring constraint must dominate every loser-preferring constraint. Based on (1), PARSE-σ and FT-BIN do not prefer either candidate and can be ranked anywhere, but FNF must dominate IAMB. This ranking restriction is the *Elementary Ranking Condition*, or ERC, associated with the W-L pair (Prince 2000, 2002a, Brasoveanu and Prince 2011); an ERC is conveyed by the "W" and "L" notations in a comparative tableau row.

[1] Input-output mappings included within the text of this dissertation are written in the shorthand form /input/[output], without an arrow between the two, following Tesar (to appear). Mappings included in examples may have the traditional form /input/ → [output].

[2] The constraints used in this tableau are defined as follows. PARSE-σ: syllables must be parsed into feet. FT-BIN: feet must be disyllabic. IAMB: feet must be right-headed. FOOT-NONFINALITY (FNF): a foot must not be right-headed. Definitions and references for all constraints can be found in section 2.3.1.

While not every combination of desired winner and losing candidate will reveal new ranking information, error-driven learning can allow the learner to focus just on informative W-L pairs. If the learner's current ranking hypothesis generates an optimum different from the observed form or desired winner, then this competitor must be beaten by the desired winner – it is an *informative loser* (Tesar 1998a). The ERC for the W-L pair that includes this candidate as loser allows the learner to alter the ranking so that the desired winner is optimal. The W-L pair thus not only identifies what must change, it identifies what the change must be.

1.1.1.1 Error detection for stratified hierarchies

Grammars in Optimality Theory require that constraints be organized into a *strict dominance hierarchy* (Prince and Smolensky 1993), or total ranking. However, the methods to presented in section 1.2.1 for interpreting a list of W-L pairs can produce hierarchies that are not total rankings, but instead allow for one or more constraints to occupy the same stratum; these are *stratified hierarchies* (Tesar and Smolensky 1998, 2000). This section explains how stratified hierarchies can be used for error-driven learning. Unless otherwise specified, in the remainder of this dissertation the terms "hierarchy" and "stratified hierarchy" will refer to a hierarchy that is not a total ranking.

Continuing the example from above, a stratified hierarchy can be generated from the ranking restriction conveyed by the W-L pair in (1) – that PARSE-σ and FT-BIN can be ranked anywhere, but FNF must dominate IAMB. This restriction is satisfied by the hierarchy in (2).

(2) {PARSE-σ, FT-BIN, FNF} >> IAMB

According to this hierarchy, PARSE-σ, FT-BIN and FNF each dominate IAMB, but otherwise have no specified order with respect to each other as they all occupy the first stratum. Total rankings, or refinements (Tesar and Smolensky 2000), of a stratified hierarchy can be derived by freely re-ordering constraints within the same stratum while respecting the overall ordering of the original strata. The stratified hierarchy in (2) produces the six refinements in (3) below. The ordering of PARSE-σ, FT-BIN and FNF varies across these refinements, yet in each refinement all three constraints dominate IAMB, just as they do in (2).

(3) Refinements of stratified hierarchy {PARSE-σ, FT-BIN, FNF} >> IAMB

a. PARSE-σ >> FT-BIN >> FNF >> IAMB
b. PARSE-σ >> FNF >> FT-BIN >> IAMB
c. FT-BIN >> PARSE-σ >> FNF >> IAMB
d. FT-BIN >> FNF >> PARSE-σ >> IAMB
e. FNF >> PARSE-σ >> FT-BIN >> IAMB
f. FNF >> FT-BIN >> PARSE-σ >> IAMB

The ranking commitments of stratified hierarchies and of their refinements differ in several important ways from the ranking commitments of the W-L pairs from which they derive. For one, these hierarchies can suffer a loss of ranking information from the W-L pairs that generate them (Tesar 1997, Brasoveanu and Prince 2011). For example, while (2) is sufficient to express the ranking restriction of the W-L pair in (1), the necessary restriction that FNF dominate IAMB is obscured because this same stratified hierarchy can satisfy other ranking restrictions derived from different data. One of the many alternative sources for this hierarchy could be a W-L pair that requires PARSE-σ to dominate IAMB but allows FNF and FT-BIN to be ranked anywhere. Loss of information is a motivation for storing a list of the W-L pairs created during learning and allowing their ranking

commitments to be continually reflected in revised stratified hierarchies, as will be described in section 1.2.1.2.

Stratified hierarchies also impose ranking commitments beyond those entailed by the original W-L pairs, with the consequence that it is generally not possible to express the total rankings of a set of W-L pairs as all and only the refinements of a single stratified hierarchy. For example, W-L pair (1) only requires that FNF dominate IAMB, and therefore it is consistent with rankings in which IAMB dominates both PARSE-σ and FT-BIN, such as FNF >> IAMB >> PARSE-σ >> FT-BIN. The stratified hierarchy in (2) does not allow such a ranking because it requires both PARSE-σ and FT-BIN to dominate IAMB. As total rankings, the refinements of (2) each impose ranking relations in addition to those of the stratified hierarchy. Thus, refinement (3)a further requires that PARSE-σ dominate both FT-BIN and FNF and that FT-BIN also dominate FNF.

Although a stratified hierarchy typically cannot duplicate the exact ranking information of the set of W-L pairs, it sufficiently represents that information while imposing fewer additional ranking relations than any one of the total rankings consistent with the W-L pairs. Using a stratified hierarchy for error-driven learning offers a compromise that is more computationally tractable than evaluating all total rankings consistent with the set of W-L pairs without arbitrarily selecting any one total ranking and committing to the particular relations of that ranking. However, error detection requires some special considerations for stratified hierarchies.

For a total ranking of constraints, identifying errors and informative losers is a straightforward process: an error is detected if the ranking selects an optimum other than

the desired winner, and the informative loser is that optimum. In Optimality Theory, the optimum for a total ranking is that candidate which, in pairwise competitions with each of the other candidates, receives fewer violations of the highest-ranked constraint that distinguishes the pair; the optimum is the most *harmonic* candidate (Prince and Smolensky 1993) with respect to the ranking. Determining the optimum for a total ranking therefore amounts to filtering out less harmonic candidates – that is, all those which do not tie for fewest violations of the sole constraint in the stratum – beginning with the highest-ranked constraint and working downward until only one candidate remains or until all remaining candidates receive the same number of violations for each constraint, producing a tie.

Because a stratified hierarchy represents a range of possible total orderings of constraints, what counts as an error in a stratified hierarchy can depend on how the learner interprets violations within a stratum. For example, the tableau in (4) includes violation profiles for two candidates for the input /sYs/. The candidates tie for the first three strata, but receive nearly complementary violations for constraints in the lowest stratum. If (4)a is the desired winner, does the learner detect an error given the stratified hierarchy in (5)?

(4) Violation profiles of two competitors for /sYs/[3]

/sYs/	FT-BIN	PARSE-σ	MAXSTR	LMOST	RMOST	AFL	FNF	IAMB	*LAPSE
a. [(sY)s]	0	1	0	0	1	0	1	0	0
b. [s(Ys)]	0	1	0	1	0	1	0	1	0

(5) FT-BIN >> PARSE-σ >> MAXSTRESS >> {LMOST, RMOST, AFL, FNF, IAMB, *LAPSE}

The answer to this question depends on how the learner interprets the information of the violations in the lowest stratum. One option is to detect errors based on total quantity of violations within a stratum, in a technique called "mark pooling" (Tesar 1995). By the mark pooling technique, there is no error if candidate (4)a is the desired winner. That candidate receives only two violations of constraints in that stratum, whereas its competitor receives three.

However, pooling the violation marks obscures an unresolved conflict between constraints in the bottom stratum. While candidate (4)a receives fewer violations than (4)b, not all constraints prefer (4)a. Under some total rankings consistent with the stratified hierarchy, (4)b would be optimal. In particular, RMOST and FNF prefer (4)b to (4)a, and if either of these constraints dominates each of the constraints that prefers (4)a, then (4)b will win the competition. Selecting (4)b as a loser would provide the ranking information to resolve this conflict.

[3] MAXSTRESS: for each stressed syllable in the input, assign a violation if the corresponding output syllable does not bear primary stress. LEFTMOST/RIGHTMOST: ALIGN (PRWD, L/R, HEAD-FT, L/R). ALL FEET LEFT (AFL):∀ foot ∃ prosodic word such that the left edge of the prosodic word and the left edge of the foot coincide. *LAPSE: rhythm is alternating; no two adjacent unstressed syllables. See section 2.3.1 for all definitions and references.

The resulting W-L pair is shown (6). In (7) is given a total ranking of the constraints that meets the conditions imposed by this W-L pair by subordinating the two loser-preferring constraints to all other constraints; section 1.2 will discuss specific strategies for deriving rankings from ERCs.

(6) W-L pair /sYs/[(sY)s] ~ [s(Ys)]

/sYs/	FT-BIN	PARSE-σ	MAXSTR	LMOST	RMOST	AFL	FNF	IAMB	*LAPSE
[(sY)s] ~ [s(Ys)]				W	L	W	L	W	

(7) FT-BIN >> PARSE-σ >> MAXSTRESS >> LMOST >> AFL >> IAMB >> *LAPSE >> RMOST >> FNF

The *Conflicts Tie* (CTie) technique identifies informative losers like (4)b which can reveal ranking information about unresolved conflicts (Tesar 2000). CTie judges the competition in (4) as a tie. Two candidates tie by the CTie criterion if, in the highest stratum in which they do not have identical violations, neither candidate *harmonically bounds* the other; that is, one candidate does not receive fewer violations than the other for all constraints on which they accrue different numbers of violations (Prince & Smolensky 1993, Samek-Lodovici & Prince 1999, 2005).[4]

For (4), the candidates receive the same violations through the first three strata, but in the bottom stratum, they receive different violations and, crucially, the winner (4)a does not harmonically bound (4)b. Because CTie judges that (4)a is not the sole optimum, the

[4] Harmonic bounding is easily identified in a comparative tableau. If one row includes only "W" and "*e*" notations, the desired winner in that row's W-L pair harmonically bounds the loser; if the row includes only "L" and "*e*" notations, the loser of the W-L pair harmonically bounds the desired winner.

learner can construct the W-L pair with (4)b and produce a new ranking, such as (7). Note that this conflict does not get resolved from this competition if the learner uses the mark pooling technique instead. Identifying and resolving the conflict at this point enables the learner to use the ranking information for subsequent learning, and for this reason the CBL will use the CTie technique to identify informative losers.

1.1.2 INCONSISTENCIES

Detecting an error enables the learner to rule out a single incorrect grammar and, by using the information provided by the W-L pairs, to pick a new one instead. However, if the learner retains a list of all W-L pairs generated in the course of learning, as proposed by Tesar (1997), then it becomes possible for the learner to rule out a space of grammars through *inconsistency detection* (Tesar 2000, 2004a). Whereas an error identifies a mismatch between the winning output of a hypothesis and the observed form of adult production, inconsistency detection identifies conflict between hypothesized structures. Two structures are inconsistent with each other if no grammar can include both. Within OT, a structure is inconsistent with a grammar hypothesis if there is no way to honor the ranking commitments imposed by the W-L pairs of that hypothesis while including the structure.

Error and inconsistency detection differ in their consequences. An error arises because a single desired winner cannot win under a specific ranking, but it might still win under another. The learner who detects an error may be able to construct a new ranking using the information the error provides from the resulting W-L pair. An inconsistency arises because a set of desired winners cannot simultaneously win under any ranking. The

learner who detects an inconsistency cannot productively construct a new ranking, because no ranking can honor the conditions imposed by each separate winner.

For example, the comparative tableau in (8) includes the W-L pair from (6) and a new pair whose winner is /ssY/[(Xs)(Y)], with an initial secondary stress (X). The ranking conditions of these pairs are inconsistent: each constraint that prefers the winner for one of the W-L pairs prefers the loser in the other pair, so that no constraint prefers only winners. Therefore, no language can include both /sYs/[(sY)s] and /ssY/[(Xs)(Y)].

(8) /sYs/[(sY)s] and /ssY/[(Xs)(Y)] are inconsistent

Input	W~L	FT-BIN	PARSE-σ	MAXSTR	LMOST	RMOST	AFL	FNF	IAMB	*LAPSE
a. /sYs/	[(sY)s] ~ [s(Ys)]				W	L	W	L	W	
b. /ssY/	[(Xs)(Y)] ~ [(X)(sY)]				L		L	W	L	

Inconsistencies provide the learner with valuable information about the set of structures in the current language hypothesis. If the learner has observed the overt forms *sYs* and *XsY*, the inconsistency in (8) reveals that either one or both of the hypothesized structural interpretations is incorrect. A language hypothesis that includes /sYs/[(sY)s] and /ssY/[(Xs)(Y)] can therefore be rejected on the basis of this inconsistency. The Inconsistency Detection Learner (IDL) is an algorithm for learning the correct interpretations of structurally ambiguous overt forms, using inconsistency detection to rule out incorrect combinations of structures (Tesar 2004a). The IDL, which is employed by the CBL for phonotactic learning, is discussed further in 1.3.4. In addition to

eliminating some combinations of structural interpretations, inconsistency detection can also be used to learn underlying forms, as section 1.4.1 describes.

In sum, error and inconsistency detection are distinguished by the type of problem the learner detects and by the hypothetical component causing the problem. An incorrect ranking hypothesis causes the traditional error of error-driven learning: a mismatch between a calculated optimum and an observed form. Incorrect structural hypotheses trigger inconsistency: a conflict between a set of desired winners and the ranking conditions separately imposed by each winner.

1.2 LEARNING RANKINGS FROM ERRORS

This section reviews two major alternatives for using errors to learn a constraint ranking. Section 1.2.1 covers Recursive Constraint Demotion (Tesar and Smolensky 1994, 2000) and several of its variants. Section 1.2.2 reviews the stochastic Gradual Learning Algorithm (Boersma 1997). Two non-error-driven approaches to learning a grammar are covered in later sections, which discuss learning the ranking as part of the broader goals of managing structurally ambiguous data (1.3.3) and learning the lexicon (1.4.3).

1.2.1 RECURSIVE CONSTRAINT DEMOTION AND VARIANTS

1.2.1.1 Recursive Constraint Demotion (RCD)

Recursive Constraint Demotion (RCD) is an algorithm for converting the ranking information from W-L pairs to a stratified constraint hierarchy (Tesar and Smolensky 1994, 2000). As explained in section 1.1.1, a ranking that selects the desired winner as

optimal will have all L-preferring constraints dominated by at least one W-preferring constraint. RCD applies this reasoning to construct a ranking from a list of W-L pairs. Beginning with the highest stratum, RCD makes passes through the W-L pair list, in each pass demoting any L-preferring constraints to the next lowest stratum and repeating until all constraints have been ranked, if possible.

Two key properties of RCD make it very effective for successfully deriving a constraint ranking. First, if there exists a ranking that will make the desired winners optimal, RCD will find one such ranking. Second, if there is no ranking that will make the desired winners optimal, RCD terminates. Application of RCD to the W-L pair list in (9) will illustrate the first property.

(9) Consistent W-L pair list – before the first pass

/ss/	PARSE-σ	FT-BIN	IAMB	FNF	AFL	*LAPSE
a. [s(Y)] ~ [(Ys)]	L	L	W	L	L	
b. [s(Y)] ~ [(X)(Y)]	L	W		W		

The constraints PARSE-σ, FT-BIN, FNF, and AFL all prefer a losing candidate at least once and therefore must be demoted below the stratum containing IAMB, which only prefers a winner. Although *LAPSE does not prefer a winner, it does not prefer any losers either, and it can be ranked in the first stratum as well. The tableau in (10) shows the results of RCD's first pass through the W-L pair list.

(10) Consistent W-L pair list – after one pass

/ss/	*LAPSE	IAMB	PARSE-σ	FT-BIN	FNF	AFL
(9)a [s(Y)] ~ [(Ys)]		W	L	L	L	L
(9)b [s(Y)] ~ [(X)(Y)]			L	W	W	

After this pass, W-L pair (9)a is ignored because demoting all the L-preferring constraints below IAMB is enough to make its desired winner optimal. However, the ranking restriction of W-L pair (9)b, below the bolded line, remains to be satisfied after the first pass through the list, as the optimality of this pair's winner is not yet ensured by the current ranking. In order for (9)b to be optimal, PARSE-σ, the sole L-preferring constraint in the row, must be demoted to the stratum below FT-BIN and FNF, the W-preferring constraints. The ordering of constraints in the tableau in (11) reflects the new stratified hierarchy, (12).

(11) Consistent W-L pair list – after two passes

/ss/		*LAPSE	IAMB	FT-BIN	FNF	AFL	PARSE-σ
(9)a	[s(Y)] ~ [(Ys)]		W	L	L	L	L
(9)b	[s(Y)] ~ [(X)(Y)]			W	W		L

(12) {*LAPSE , IAMB} >> {FT-BIN, FNF, AFL} >> PARSE-σ

1.2.1.2 Multi-Recursive Constraint Demotion (MRCD)

Applying RCD to a list of W-L pairs can result in information loss. The problem, as described in section 1.1.1.1, is that stratified hierarchies obscure the necessary ranking restrictions, or ERCs, entailed by a set of W-L pairs. This problem is solved by a later development of RCD, *Multi-Recursive Constraint Demotion* (MRCD) (Tesar 1997), which preserves each W-L pair in a permanent list called the *support* (Tesar & Prince 2003). Whenever error-driven learning generates a new W-L pair, the pair is added to the permanent list, and then RCD applies to the whole list to construct a new ranking.

Importantly, retaining the W-L pairs and their ERCs allows the learner to detect when desired winners are inconsistent with each other.

The following example shows how MRCD uses the second property of RCD – its ability to terminate when necessary – to reveal that a set of desired winners cannot all be optimal. Consider the support below. For the two-syllable input, the desired winner is an iambic foot, but for the three-syllable input the desired winner is the right-aligned trochaic foot.

(13) Inconsistent support – before the first pass

Input	W~L	PARSE-σ	FT-BIN	*LAPSE	AFL	IAMB	FNF
a. /ss/	[(sY)]~ [(Ys)]					W	L
b. /sss/	[s(Ys)]~ [s(sY)]				L	L	W
c. /sss/	[s(Ys)]~ [(Ys)s]			W	L		

Demoting the L-prefering constraints AFL, IAMB and FNF to the stratum below PARSE-σ, FT-BIN and *LAPSE takes care of W-L pair (13)c, shown above the bolded line in tableau (14) below. Here RCD terminates, detecting a problem: in the second stratum, every constraint is L-preferring. No ranking can make the two desired winners of these remaining pairs optimal.

(14) Inconsistent support - after the first pass

Input	W~L	PARSE-σ	FT-BIN	*LAPSE	AFL	IAMB	FNF
(13)c /sss/	[s(Ys)]~ [(Ys)s]			W	L		
(13)a /ss/	[(sY)]~ [(Ys)]					W	L
(13)b /sss/	[s(Ys)]~ [s(sY)]				L	L	W

As (14) demonstrates, maintaining a support that stores W-L pairs makes inconsistency detection possible. A learner that maintains a support is able to determine

what kinds of structures can coexist in the language hypothesis and can use this information both to resolve structurally ambiguous data (1.3.4) and to learn the lexicon (1.4.1).

1.2.1.3 Biased Constraint Demotion (BCD)

As explained above, one of the essential properties of RCD is that if there is at least one ranking that will satisfy all of the given W-L pairs, RCD will find one such ranking. In particular, RCD finds the stratified hierarchy in which each constraint is ranked as high as possible. Two later modifications to RCD, Biased Constraint Demotion (BCD) (Prince and Tesar 2004) and Low-Faithfulness Constraint Demotion (LFCD) (Hayes 2004), bias the learner toward the construction of more restrictive grammars and are intended to address the "subset problem" (Angluin 1980 and Baker 1979). The subset problem refers to the situation of a learner who wrongly adopts a grammar that allows more structures than the language actually permits. Because the correct grammar is a subset of this grammar, the learner will never encounter the evidence needed to reject the wrong assumption: the only evidence is positive evidence, and everything permitted by the correct grammar is also permitted by the incorrect, superset grammar.

Within Optimality Theory, relative rankings of markedness and faithfulness constraints determine the range of linguistic structures permitted by a language. A grammar that permits the broadest range of structures will rank all faithfulness constraints above all markedness constraints, allowing any structure that appears in inputs to appear in outputs, within the bounds permitted by GEN. At the other extreme, the grammar with the narrowest range of structures will have all markedness constraints dominating the

faithfulness constraints. Thus, the basic insight of Biased Constraint Demotion: the learner who waits for positive evidence of a marked structure to rank a faithfulness constraint above the pertinent markedness constraints will derive the more restrictive grammar, admitting the observed marked structures that motivate the ranking and as few others as possible.

Prince and Tesar illustrate the use of BCD for learning phonotactic ranking information from the distributional data of observed forms. Any observed form has been selected as optimal by the adult grammar, and therefore the learner knows that any structures present in that form are perfectly permissible in the language.[5] Consequently, if the observed form is used as an input, the optimal output should be the observed form itself. This mapping from observed form as input to observed form as output is the *identity map* (Prince and Tesar 2004). Because the identity map of an observed form will best satisfy all faithfulness constraints, it can fail to be optimal only due to its satisfaction of markedness constraints. Error-driven learning in this case will produce a W-L pair in which W is the identity map and L includes a less marked output than the identity map. As a result, the W-L pair reveals conflicts between markedness constraints alone or between both markedness and faithfulness constraints.

The learner derives a new ranking by applying BCD to the support generated by error-driven learning – that is, new rankings are derived by MRCD, using BCD in place of RCD. At the outset of learning, there will be no W-L pairs in the support; applying BCD to this empty list derives the learner's initial constraint ranking in which all

[5] Error-driven learning treats all observed data as evidence of grammatical forms. Thus, speech errors are also processed as though they are grammatical.

markedness constraints dominate all faithfulness constraints: {M} >> {F}. Deriving each new ranking through the application of BCD to the support ensures that the ranking bias toward markedness constraints persists throughout the learning process.

The constraint set in the example below includes all nine constraints used in the simulations of the CBL, including the faithfulness constraint MAXSTRESS, which assigns a violation if an underlyingly +stress syllable does not correspond to an output syllable bearing primary stress. The initial ranking after applying BCD to the empty support appears in (15). Suppose the learner has constructed the support in (16) for the winners /sYs/[(sY)s] and /Yss/[(Ys)(X)].

(15) Initial ranking by BCD

{PARSE-σ, FT-BIN, IAMB, FNF, AFL, *LAPSE, LMOST, RMOST} >> MAXSTRESS

(16) Support for /Yss/[(sY)s] and /Yss/[(Ys)(X)]

Input	W~L	PARSE-σ	FT-BIN	IAMB	FNF	AFL	*LAPSE	LMOST	RMOST	MAXSTR
a. /sYs/	[(sY)s] ~ [s(Ys)]			W	L	W		W	L	
b. /sYs/	[(sY)s] ~ [(Y)(sX)]	L	W		W	W			W	W
c. /sYs/	[(sY)s] ~ [(Ys)(X)]	L	W	W		W				W
d. /Yss/	[(Ys)(X)] ~ [(sY)s]	W	L	L		L				W

*LAPSE, LMOST, and MAXSTRESS do not prefer losers in any of the W-L pairs. RCD would rank all of these in the first stratum, but BCD ranks only the markedness constraints and waits for at least another pass through the support to rank MAXSTRESS. The updated support appears in (17), followed by the ranking after this first pass.

Ranking *LAPSE and LMOST in the first stratum satisfies W-L pair (16)a, shown above the bolded line.

(17) Support after first pass of BCD

Input	W~L	*LAPSE	LMOST	PARSE-σ	FT-BIN	IAMB	FNF	AFL	RMOST	MAXSTR
(16)a /sYs/	[(sY)s] ~ [s(Ys)]		W			W	L	W	L	
(16)b /sYs/	[(sY)s] ~ [(Y)(sX)]			L	W		W	W	W	W
(16)c /sYs/	[(sY)s] ~ [(Ys)(X)]			L	W	W		W		W
(16)d /Yss/	[(Ys)(X)] ~ [(sY)s]			W	L	L		L		W

(18) {*LAPSE, LMOST} >> {PARSE-σ, FT-BIN, IAMB, FNF, AFL, RMOST, MAXSTRESS}

Of the remaining unranked constraints, FNF, RMOST and MAXSTRESS prefer only winners in the rows below the bolded line. BCD again ranks just the markedness constraints, producing the ranking for the updated tableau in (19). This ranking, in (20), now satisfies (16)b also.

(19) Support after second pass of BCD

Input	W~L	*LAPSE	LMOST	FNF	RMOST	PARSE-σ	FT-BIN	IAMB	AFL	MAXSTR
(16)a /sYs/	[(sY)s] ~ [s(Ys)]		W	L	L			W	W	
(16)b /sYs/	[(sY)s] ~ [(Y)(sX)]			W	W	L	W		W	W
(16)c /sYs/	[(sY)s] ~ [(Ys)(X)]					L	W	W	W	W
(16)d /Yss/	[(Ys)(X)] ~ [(sY)s]					W	L	L	L	W

(20) {*LAPSE, LMOST} >> {FNF, RMOST} >> {PARSE-σ, FT-BIN, IAMB, AFL, MAXSTRESS}

Five unranked constraints remain. Because the markedness constraints are all L-preferring, now the W-preferring MAXSTRESS must finally be ranked. The updated support appears in (21). All of the W-L pairs are satisfied by the resulting ranking, (22). By comparison, applying MRCD to the support in (16) would yield the ranking in (23). This ranking has just two strata because all the W-L pairs are satisfied if *LAPSE, LMOST, and MAXSTRESS are ranked highest.

(21) Support after third pass of BCD

Input	W~L	*LAPSE	LMOST	FNF	RMOST	MAXSTR	PARSE-σ	FT-BIN	IAMB	AFL
(16)a /sYs/	[(sY)s] ~ [s(Ys)]		W	L	L				W	W
(16)b /sYs/	[(sY)s] ~ [(Y)(sX)]			W	W	W	L	W		W
(16)c /sYs/	[(sY)s] ~ [(Ys)(X)]					W	L	W	W	W
(16)d /Yss/	[(Ys)(X)] ~ [(sY)s]					W	W	L	L	L

(22) Final ranking by BCD

{*LAPSE, LMOST} >> {FNF, RMOST} >> MAXSTRESS >> {PARSE-σ, FT-BIN,IAMB, AFL}

(23) Final ranking by MRCD

{*LAPSE, LMOST, MAXSTRESS } >> {PARSE-σ, FT-BIN, IAMB, FNF, AFL, RMOST}

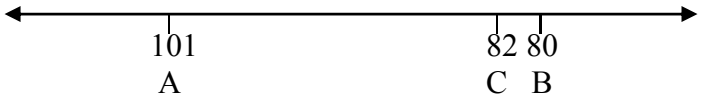
The more restrictive ranking in (22) is consistent with language L14 in the typology of the Stress system, the system used to illustrate the CBL (see section 2.3.1). In this language, all inputs map to either [(sY)s] or [(Ys)(X)]. L14 contains a subset of the forms of language L15, which includes a third form, [(X)(sY)]. L15 is consistent with the less restrictive ranking in (23).

BCD includes other criteria for ranking faithfulness constraints when the constraint set includes more than one, such as by ranking a set of faithfulness constraints (an *F-Gang)* that enables a markedness constraint to be ranked next. Because the Stress system used to illustrate the CBL includes just one faithfulness constraint, this example provides a sufficient look at how BCD will operate in the discussions to follow.

1.2.2 THE GRADUAL LEARNING ALGORITHM

The *Gradual Learning Algorithm,* or GLA, (Boersma 1997) is a primary alternative to the RCD-inspired learning algorithms described in 1.2.1. Designed to handle optionality in a language, the GLA presupposes a variant of Optimality Theory called *stochastic OT* (Boersma 1997, Boersma and Hayes 2001). Whereas constraints are strictly ranked in classic OT, in stochastic OT constraints are ranked along a continuous number line. For example, while constraints A and C both dominate B below, C is much closer to B.

(24) GLA number line



The numbers marked on the line represent the means of normal (Gaussian) distributions of values for the constraints listed and are called *ranking values* (Boersma and Hayes 2001). When a candidate set is evaluated, each constraint's ranking value is temporarily perturbed by a small random noise value that shifts the value of the constraint slightly higher or lower to derive the *selection point* (Boersma and Hayes 2001) for that constraint, which is the value used for the evaluation; this is *stochastic* evaluation

(Boersma 1997). Because stochastic evaluation randomly selects values from the normal distributions of individual constraints, it amounts to the selection of a complete ranking from a probability distribution over the possible rankings.

Optionality derives from the overlap between the normal distributions of ranking values. The closer the ranking values of two constraints, the greater the overlap, and the greater the likelihood that stochastic evaluation will use a ranking different from that of the ranking values. For example, in one evaluation C's ranking could decrease to 80.7 while B's increases to 81, resulting in B dominating C for that evaluation. The greater distance between A and C makes it far less likely – although not impossible – that C will dominate A in an evaluation.

Similar to classic OT, different grammars in stochastic OT correspond to different ranking values of constraints; the GLA is the stochastic counterpart of RCD, providing an algorithm for setting these ranking values. Like MRCD, the GLA employs error-driven learning, but instead of constructing a new ranking for each error detected, the GLA simply modifies the existing ranking by nudging all W-preferring constraints slightly higher and all L-preferring constraints slightly lower. The size of the nudge is defined by a *plasticity* value that grows smaller as learning advances (Boersma 1997). The small shifts in ranking values regulated by the plasticity make errors less likely over time, whereas RCD immediately corrects the ranking to eliminate the error.

The GLA does not maintain a support of W-L pairs. Once ranking values are shifted by the plasticity, the GLA has no more use for the W-L pair derived from error-detection. Both because W-L pairs are discarded after each ranking modification and because

modifications involve such small movements, the GLA is forgiving of noisy data such as speech errors and re-rankings based on such data do not severely interfere with learning. For a learner that uses MRCD to learn the ranking, adding to the support an ERC that results from observing a speech error has a permanent effect on the ranking and may lead to inconsistency. On the other hand, the GLA lacks MRCD's ability to perform inconsistency detection because it does not store W-L pairs from prior errors.

1.3 MANAGING STRUCTURAL AMBIGUITY

A structure is ambiguous if it has more than one interpretation. For the stress data under discussion here, a structural interpretation consists of any parsing matching the observed form in its pattern of stressed and unstressed syllables. A two syllable word with initial stress, *Ys,* has the possible interpretations [(Y)s] and [(Ys)], but not [s(Y)] or [(sY)]; the latter two parsings are simply candidate structural descriptions for a two syllable input.

Different interpretations of an overt form will correspond to different languages, and how the learner interprets one overt form can have consequences for what the learner infers about the grammar and the interpretations of other overt forms. This section reviews two basic responses to the problem of structural ambiguity.[6] In the first, the learner takes a flexible approach to interpretations: either the interpretations can vary over the course of learning (sections 1.3.1 and 1.3.2) or the interpretations are simply ignored (section 1.3.3). The second response is more rigid: an overt form receives a permanent interpretation once it becomes clear that its interpretation cannot be ignored

[6] See also section 1.4.2 and 1.4.3 for two learners that manage structurally ambiguous data while learning a lexicon.

(section 1.3.4). Both kinds of responses can be error-driven, but only the second kind can exploit inconsistency detection for learning.

1.3.1 ROBUST INTERPRETIVE PARSING/ CONSTRAINT DEMOTION

The *Robust Interpretative Parsing/ Constraint Demotion* (RIP/CD) algorithm is an early, error-driven approach to resolving the problem of structural ambiguity (Tesar 1998b, Tesar and Smolensky 2000). Robust interpretative parsing takes an observed form as input and assigns it the most harmonic structural interpretation, regardless of whether the ranking selects that interpretation as optimal. The learner then checks whether the underlying form of the observed form maps to the same interpretation. If it does, the learner makes no changes to the ranking, but if it does not, the learner performs error-driven learning. The ranking is altered using *online constraint demotion,* or "online CD" (Tesar and Smolensky 1998). For online CD, the W-L pair supplied by error detection is immediately used to update the ranking and is then discarded, with the next round of error-driven learning applying to the revised ranking. This process repeats until the assigned structural representation is the optimal output for the underlying form.

Because the RIP/CD algorithm chooses the most harmonic structural interpretation as judged by the learner's current ranking hypothesis, it has the potential to choose the wrong interpretation, and consequently to learn the wrong grammar. The algorithm can sometimes right itself, but there are several cases in which the algorithm fails because of its rigid selection of the most harmonic interpretation at the time (Tesar and Smolensky 2000); however, when RIP/CD succeeds, it is quite efficient. In simulations of RIP/CD performed over 124 languages generated from a set of 12 constraints, Tesar and

Smolensky find that the algorithm typically succeeds in fewer than 10 learning steps, each an application of constraint demotion.

1.3.2 ROBUST INTERPRETIVE PARSING/ GRADUAL LEARNING ALGORITHM

Apoussidou and Boersma (2003) implement robust interpretative parsing with the GLA to learn foot structure in Latin. The differences between RIP/GLA and RIP/CD are minimal. As with RIP/CD, the learner uses an observed form as input, determines which of the possible interpretations of the observed form is most harmonic, and assigns this interpretation to the observed form. Because the GLA assumes stochastic OT, the ranking used to select the most harmonic interpretation of the observed form is in effect randomly chosen from the probability distribution over all possible rankings, given the current ranking values. The learner then uses this same ranking to check whether the underlying form of the observed form maps to the same interpretation; if not, then error-driven learning occurs, with ranking values changed as described in section 1.2.2.

Importantly, because RIP/GLA shares with RIP/CD the selection of the most harmonic interpretation at the time of observation, it also shares the possibility of getting stuck by selecting the wrong interpretation. In comparison to RIP/CD, which requires very few learning steps when it is successful, RIP/GLA generally processes thousands of data before succeeding. GLA simulations are typically evaluated by assessing the ranking values after the learner has processed a round of data. In simulations where the learners process overt forms, as in the RIP/CD simulations, Apoussidou and Boersma report that the learners succeed after processing 3 to 35 rounds of 1000 data.

1.3.3 THE NAÏVE PAIRWISE RANKING LEARNER AND RANDOM SEARCH

Like the preceding learners, the *Naïve Pairwise Ranking Learner*, or NPRL, (Jarosz to appear) takes an online approach, but it is not error-driven. Moreover, NPRL ignores structural interpretations altogether and avoids the problem of structural ambiguity by comparing overt forms only. Simulations of the NPRL show that it is successful and efficient, at least as compared to exhaustive search and RIP/GLA, but its performance mirrors qualities of random search. Random search is just as successful and requires even fewer learning steps than NPRL.

Based on Naïve Parameter Learning (Yang 2002), NPRL derives a total ranking from pairwise relative rankings of constraints. Thus, the ranking CON1 >> CON2 >> CON3 is derived from three pairwise rankings: CON1 >> CON2, CON1 >> CON3, CON2 >> CON3. Each relative ranking of a pair of constraints has a probability specified by the grammar. The total ranking is created by a repeating process in which two unranked constraints are ranked relatively given the probability assigned by the grammar, along with whatever other pairwise rankings follow from their ranking, until all constraints have been ranked with respect to each other. As in the GLA, the total ranking represents a selection from the probability distribution over all possible rankings.

NPRL learns by comparing the overt form of the current ranking's optimum against the observed form. If these overt forms match, then all pairwise rankings are rewarded using a learning rate that increases the probability associated with the ranking: the higher the learning rate, the greater the increase in probability. Similarly, if the overt forms do not match, the probabilities associated with each ranking are penalized. NPRL is not

error-driven because it updates the ranking in some way with each observed form and it is "naïve" because it updates every pairwise ranking rather than finding a way to reward just those that favor the desired outcome.

In simulations of the NPRL applied to the metrical stress system used by Tesar and Smolensky (2000), Jarosz finds that given a high learning rate, the NPRL is efficient at learning a language as compared to exhaustive search and RIP/GLA, but not RIP/CD. The NPRL is also completely successful, where success means converging on a ranking that generates all the overt forms of the language. With twelve constraints used in the simulations, there are nearly five-hundred million different total rankings, but the NPRL requires on average 16004 iterations (each the processing of a single observed form) to successfully learn the 124 languages in the system. However, the learning rate is a key factor in both success and efficiency. At low rates, NPRL is unsuccessful after one million iterations, and at high rates, success is achieved essentially through random search, as updating after the mismatch between predicted and observed forms produces a far different ranking for the next evaluation.

Jarosz also evaluates a random search learner which eliminates both the reward and penalty schemes of the NPRL. The learner simply picks a new ranking at random, checks whether the ranking can generate each observed overt form, and if not, selects a new ranking. This learner is error-driven, as a new ranking is selected only if the current ranking cannot generate a given overt form. The learner is successful if a ranking is chosen that generates all of observed forms. In simulations, random search proves

successful in all trials and more efficient than NPRL, requiring 10,025 iterations on average to learn a language.

Compared to RIP/CD and RIP/GLA, random search appears to be a surprising winner: it requires fewer learning steps than RIP/GLA, and though nowhere near as few as RIP/CD can require, it succeeds where that algorithm fails. However, there are two key areas where random search is unsatisfying. First, the simulations evaluating random search do not take into account restrictiveness. The simulation is deemed successful if its ranking can generate all of the overt forms in a language and unsuccessful if it does not. Restrictiveness is irrelevant to these simulations because no neutralizations occur in the target languages, but if they did, on this criterion for success a "successful" ranking could overgenerate. The random search learner, while error-driven, cannot respond to errors in a way that will prevent their future occurrence, much less drive the learner toward a ranking more restrictive than the last. In contrast, an error-driven learner that builds from its accumulated store of knowledge from errors can incorporate a ranking bias to derive more restrictive rankings. Section 1.3.4 describes such a learner, which incorporates BCD and MRCD to efficiently derive restrictive rankings

Second, random search is unlikely to be a computationally plausible strategy for learning both a lexicon and a ranking to generate the observed forms. The linguistic system presented in section 2.3.1 and used to evaluate the Commitment-Based Learner produces a typology of 97 languages but includes over 370 million grammar and lexicon combinations. It is true that all of these languages can be generated by multiple total rankings and even true that, as described in chapter 4, multiple lexicon and grammar

combinations can generate the observed forms of a language. Therefore, it might be reasonable to expect that it would take about the same number of iterations (roughly 10,000) to find a lexicon and ranking to fit the data as it took to find a ranking fitting the data for a language in the Tesar and Smolensky typology, with its 500 million possible total rankings. However, this system is remarkably modest, having nine constraints, six morphemes, and only trisyllabic words. As the space of grammar and lexicon combinations grows for more complex systems, the number of iterations required to find a combination that generates the data is certain to explode – and the inability to address restrictiveness will remain.

1.3.4 THE INCONSISTENCY DETECTION LEARNER

The *Inconsistency Detection Learner*, or IDL (Tesar 2004), is an error-driven learner that avoids the tendencies of RIP/CD and RIP/GLA to get stuck on the wrong interpretations by creating separate hypotheses containing different interpretations. A support in each hypothesis enables the learner to reap the benefits of MRCD, including identifying and rejecting inconsistent hypotheses. In comparison to random search, the most successful of the preceding learners reviewed, IDL both successfully and efficiently learns restrictive rankings to generate the observed data.

While the IDL can assign stable interpretations to overt forms, it begins first by determining whether the current ranking hypothesis can map an input to some structural interpretation of the observed form, employing error-driven learning as described in section 1.1.1. If the ranking maps the input to one interpretation of the observed form, the learner makes no changes to the hypothesis and learns nothing new. If an error is detected

– if the input does not map to a structural interpretation or if it maps to more than one interpretation in a tie – then the learner follows the familiar pattern of error-driven learning for constraint demotion, constructing a W-L pair in which the loser is the candidate description selected as optimal by the current ranking.

The IDL differs substantially from RIP/CD in allowing each interpretation to be the winner of its own W-L pair instead of selecting the most harmonic interpretation of the current ranking to be the winner. If an observed form has five structural interpretations, an error on this form will cause the learner to generate five separate W-L pairs that have the same loser, L, since the error has shown that each interpretation has separately lost to this candidate. Each of these W-L pairs is stored in the support of its own branch hypothesis which inherits the support of the original hypothesis. A branch's support therefore includes everything that was in the parent's list plus the W-L pair from the error that caused the branch. The learner derives a new ranking for each branch hypothesis by applying RCD to the branch's support.[7]

Storing a W-L pair in the support entails a commitment to its winner every time MRCD applies. If MRCD finds no constraint ranking that will allow all desired winners in a W-L pair list to win, then the structural interpretations of the winners are inconsistent with each other, and the learner rejects the hypothesis containing this list. Otherwise, the hypothesis remains active and under consideration. The learner will test the next

[7] Because Tesar's focus is on illustrating the use of the IDL in learning structurally ambiguous representations, all inputs match the observed forms completely, no Faithfulness constraints are included, and there is no benefit to applying BCD to the W-L pair list. BCD becomes useful again when Faithfulness violations are included.

observed form in each hypothesis that remains and will perform error-driven learning described above when errors are detected.

The example from (8) in section 1.1.2 offers a partial example of the IDL at work. In that example, the learner has observed *sYs* and *XsY* and interpreted these forms as [(sY)s] and [(Xs)(Y)], respectively. As shown in tableau (8) and repeated below in (25), the identity mappings for these forms – /sYs/[(sY)s] and /ssY/[(Xs)(Y)] – are inconsistent with each other.[8]

(25) /sYs/[(sY)s] and /ssY/[(Xs)(Y)] are inconsistent

Input	W~L	FT-BIN	PARSE-σ	MAXSTR	LMOST	RMOST	AFL	FNF	IAMB	*LAPSE
a. /sYs/	[(sY)s] ~ [s(Ys)]				W	L	W	L	W	
b. /ssY/	[(Xs)(Y)] ~ [(X)(sY)]				L		L	W	L	

The support in (25) represents one branch from a prior hypothesis which found that [(sY)s] is a possible – that is, consistent – interpretation of the first observed form, *sYs.* The original support for that hypothesis can be found in (6). The inconsistency here proves that [(sY)s] and [(Xs)(Y)] cannot both be correct interpretations of the observed forms, and any hypothesis that includes both of these interpretations can be rejected. However, this combination of interpretations is only one of those checked by the IDL. An alternative interpretation of *XsY* parses the head-foot as an iamb at the right edge: [(X)(sY)]. As shown in (26), this alternative interpretation is consistent with the candidate /sYs/[(sY)s] for the first form. The comparative tableau below represents the

[8] As described in 2.3, for the examples and simulations described in this dissertation, secondary stress is a parameter on -stress output syllables. Therefore, in the identity mapping for *XsY,* the input correspondent of the secondary stressed syllable is unstressed:/ssY/[(Xs)(Y)].

support for these candidates and includes all of the W-L pairs created by error-driven learning. The stratified hierarchy in (27) is derived by BCD from this support.

(26) /sYs/[(sY)s] and /ssY/[(X)(sY)] are consistent

Input	W~L	IAMB	*LAPSE	MAXSTR	LMOST	RMOST	AFL	FT-BIN	FNF	PARSE-σ
a. /sYs/	[(sY)s] ~ [s(Ys)]	W			W	L	W		L	
b. /sYs/	[(sY)s] ~ [(Y)(sX)]			W		W	W	W	W	L
c. /ssY/	[(Xs)(Y)] ~ [(sY)s]			W	L	W	L	L	L	W
d. /sYs/	[(sY)s] ~ [(X)(sY)]			W	W	L	W	W	W	L
e. /sYs/	[(sY)s] ~ [(sY)(X)]						W	W	W	L

(27) {IAMB, *LAPSE} >> MAXSTRESS >> {LMOST, RMOST, AFL, FT-BIN, FNF, PARSE-σ}

The support in (26) represents a second branch from the hypothesis that assigns *sYs* the interpretation [(sY)s]. There is also a third branch, which is inconsistent because the identity mapping for this interpretation, /sYs/[s(Y)s], is harmonically bounded by the iambic candidate /sYs/[(sY)s]. In all, the IDL extends three branches from the initial hypothesis in which *sYs* is interpreted as [s(Ys)]. If the learner is committed to the interpretation [(X)(sY)] for overt form *XsY,* then only the branch hypothesis whose support is in (26) is consistent, and any new forms the learner observes must be evaluated against that support only, with the same branching procedure repeating as necessary.

In simulations involving the same kind of metrical stress data[9] used to evaluate the NPRL and the random search learners in section 1.3.4, the IDL proves successful and

[9] The system used by Tesar (2004) omits the Word-Foot-Left/Right constraints used by Tesar and Smolensky (2000), whose system is used in the NPRL and random search learner simulations.

highly efficient (Tesar 2004). An IDL simulation successfully learns a language if it learns a ranking that generates the overt forms of the language. The constraints and candidate sets used in the simulations define a typology of 1527 languages.[10] The IDL learns each language and, on average, succeeds in just 62 applications of RCD (one application for each ERC stored). This average includes all applications of RCD in all hypothesis branches created in the course of learning; for most languages, the learner maintains at most two consistent hypotheses simultaneously.

Retaining a support for active hypotheses has a severe effect on the number of hypotheses the learner must retain. As a hypothesis increases in the number of interpretations and concomitant ranking restrictions that it is committed to, it also increases the potential that a candidate for a newly-observed form will be inconsistent with those ranking restrictions. Inconsistency here leads to hypothesis rejection. Consequently, while the maximum number of active hypotheses the learner would have to maintain at once would be equal to the product of the number of structural interpretations for all the observed forms in the language, restrictions placed by the ERCs of the W-L pairs mean that many combinations of interpretations will be inconsistent, and hypotheses containing these combinations are rejected.

The IDL fares very well in comparison with the learners discussed in the preceding sections. It matches the total success rate of random search, but where random search fails to derive restrictive rankings, the IDL can easily incorporate a ranking bias such as

[10] Tesar notes that there are in fact 2140 distinct languages, if full structural descriptions are included, but many have identical sets of overt forms. Languages with identical sets of overt forms are represented by only a single data set, leaving 1527 distinct sets of overt forms. These languages are globally surface-ambiguous, as defined in section 4.1.

BCD to do so. The efficiency claims of each learner are harder to compare, as they use different metrics and involve different-sized constraint sets, but the IDL appears favorable nonetheless. The random search learner averages around 10,000 iterations, where an iteration is the observation of an overt form and the subsequent ranking update that occurs. Each IDL simulation includes 62 words, and on average the learner requires 62 applications of RCD – counting those in all hypotheses created – to successfully learn the language. To observe 10,000 overt forms during the course of learning, the IDL would have to see each word 161 times before learning the language. Even if the learner had to observe the entire list of 62 words before detecting an error and applying RCD, making 62 updates would require observing only 3844 (=62*62) overt forms.

The IDL owes much to the supports of its hypotheses. Maintaining multiple hypotheses, each with its own support, adds complexity and more machinery to this learner but guarantees success: somewhere, one of the hypotheses has the correct interpretations for the observed forms. Moreover, the number of hypotheses maintained at any given time is far fewer than the product of the interpretations of each overt form because having a support enables the learner to detect and reject inconsistent hypotheses. Finally, unlike the random search learner, the IDL displays gradual learning due to its use of supports: storing the W-L pair created after detecting an error ensures that this particular error will not happen again. Each error therefore brings the learner closer and closer to either learning the target grammar or detecting an inconsistency.

All learners have disadvantages. As a learner that relies on a support, the major disadvantage of the IDL is its difficulty in handling noisy data such as speech errors and

free variation. In attempting to learn from these potentially contradictory data, the learner may find that every hypothesis is inconsistent and ultimately fail to learn the language. Random search experiences a similar failing: the learner cannot converge on a ranking if free variation creates a unending cycle of match and mismatch. By contrast, GLA-based learners tolerate noisy data well, but RIP/GLA is not guaranteed success.

1.4 LEARNING THE LEXICON

The preceding section presented learners using a variety of strategies for learning a grammar from structurally ambiguous data. These learners vary by whether or not they are error-driven and whether or not they maintain a support to enable inconsistency detection. Those divides continue in this section, which presents several different approaches to learning a lexicon and ranking. Two of these are error-driven and follow the pattern seen in the last section: the RCD-based approach utilizes inconsistency detection (1.4.1) while the GLA-based approach does not (1.4.2). The third approach, Maximal Likelihood Learning of Lexicons and Grammars (Jarosz 2006, section 1.4.3), also uses a stochastic, online learning algorithm but is not error-driven. The latter two learners also can handle structurally ambiguous data, but they are discussed in this section because of their ability to learn the lexicon as well.

1.4.1 INCONSISTENCY DETECTION AND THE OUTPUT-DRIVEN LEARNER

Learning about underlying forms requires that the learner know the morphological composition of words in the language. Once the learner has this knowledge, paradigmatic information – the surface realizations of morphemic contrasts and alternations (Alderete

et al. 2005 and Tesar 2004) – makes it possible to learn underlying forms through inconsistency detection.

A morphemic contrast consists of a pair of surface-distinct words differing by only a single morpheme in the same morphological environment. An example from Modern Hebrew is the pair *gàmadím* 'dwarf' (pl.) and *tírasim* 'corn' (pl). In these words the plural suffix /-im/ provides the morphological environment for the contrast morphemes, the roots /gamad/ and /tíras/. Since the context is the same for the contrasting morphemes in a contrast pair, the different surface realizations of the pair must be due to the underlying forms of the contrast morphemes.

Tesar (2006) describes how the learner can infer individual features of the underlying forms by applying inconsistency detection to lexical hypotheses, or pairs of possible underlying forms for the contrast pair. Each lexical hypothesis includes a different combination of feature values for the as-yet unset features. If there are *n* binary features whose underlying values are unknown, the learner must construct 2^n local lexical hypotheses (Merchant and Tesar 2008). In the contrast pair *gàmadím* and *tírasim* each syllable has a single feature with two possible settings: +stress or –stress. The contrast pair therefore has five feature values that the learner must set, one for each syllable of the disyllabic contrast morphemes, the roots, and one for the monosyllabic environment morpheme, the shared suffix. In total, there are 32 ($=2^5$) lexical hypotheses, whose underlying stress values are summarized by the chart in (28).

(28) Lexical hypotheses for the contrast pair *gàmadím* and *tírasim*

gamad	*tíras*	*-im*	*gamad*	*tíras*	*-im*	*gamad*	*tíras*	*-im*	*gamad*	*tíras*	*-im*
/--/	/--/	/-/	/+-/	/--/	/-/	/-+/	/--/	/-/	/++/	/--/	/-/
/--/	/--/	/+/	/+-/	/--/	/+/	/-+/	/--/	/+/	/++/	/--/	/+/
/--/	/+-/	/-/	/+-/	/+-/	/-/	/-+/	/+-/	/-/	/++/	/+-/	/-/
/--/	/+-/	/+/	/+-/	/+-/	/+/	/-+/	/+-/	/+/	/++/	/+-/	/+/
/--/	/-+/	/-/	/+-/	/-+/	/-/	/-+/	/-+/	/-/	/++/	/-+/	/-/
/--/	/-+/	/+/	/+-/	/-+/	/+/	/-+/	/-+/	/+/	/++/	/-+/	/+/
/--/	/++/	/-/	/+-/	/++/	/-/	/-+/	/++/	/-/	/++/	/++/	/-/
/--/	/++/	/+/	/+-/	/++/	/+/	/-+/	/++/	/+/	/++/	/++/	/+/

In the method described by Tesar (2006) and Merchant and Tesar (2008), the learner tests each local lexical hypothesis for inconsistency, then examines the consistent hypotheses for common feature settings. Since the correct underlying forms must be in one of the consistent lexical hypotheses, any feature value that appears in all of the consistent hypotheses necessarily appears in the correct one. The common feature values of consistent lexical hypothesis provide the learner with information about underlying forms.

Tesar (2008, 2009, to appear) improves the computational efficiency of testing lexical hypotheses significantly with the concept of *output-driven maps*, defined as below.

(29) "A map is output-driven if, for every grammatical candidate A→X of the map, if candidate B→X has greater similarity than A→X, then B→X is also grammatical (it is part of the map)." (Tesar 2009)

The map for a language consists of the set of grammatical candidates, with each candidate being a mapping from an input to an output. Similarity is judged according to *disparities* (Tesar 2008, 2009, to appear). Each disparity is one difference, such as in feature value, between corresponding segments of the input and output. For example, the

input /pa/ and the output [pá:] have two disparities: the length and stress of the second segment, the vowel. This mapping and an alternative mapping containing the output [pá:] are given with their disparities in (30) below.

(30) Mappings and their disparities

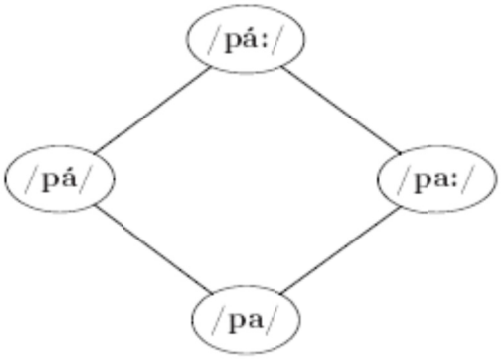
a.	/pa/ → [pá:]	*Disparities:*	segment 2 stress; segment 2 length
b.	/pá/ → [pá:]		segment 2 length

The mappings above contain the same output and can be compared with reference to their disparities. To begin, the mapping in (30)b contains a subset of the disparities in mapping (30)a, making its input /pá/ more similar to the output than the unstressed input /pa/ of (30)a is. Therefore, if (30)a, the mapping /pa/ → [pá:], is grammatical in a language where the grammatical candidates form an output-driven map, then the mapping in (30)b must also be grammatical. This example serves to illustrate the property of output-driven maps described in (29). Now consider the contrapositive of this property: if the mapping in (30)b is not grammatical in the language, and the language has an output-driven map, then the mapping in (30)a also cannot be grammatical because it contains a superset of the disparities in (30)b. The *Output-Driven Learner,* or ODL, (Tesar 2008, 2009, to appear) exploits this property.

The ODL uses the structure of output-driven maps to apply inconsistency detection to just the lexical hypotheses that could provide information about the underlying form instead of to all local lexical hypotheses. In particular, the ODL tests only the hypotheses that include one disparity. If a lexical hypothesis with a single disparity is inconsistent, the learner knows that the feature tested by the disparity must be set as in the output.

The space of lexical hypotheses for the observed form [pá:] is illustrated in (31) by a *relative similarity lattice,* with each node representing an input for the observed output (Tesar to appear). The input /pá:/ in the topmost node is identical to the observed form; in an output-driven map, this input must map to [pá:]. In the second row, the inputs /pá/ and /pa:/ each contain one disparity. The most dissimilar input is in the bottommost node; /pa/ has two disparities from the observed form.

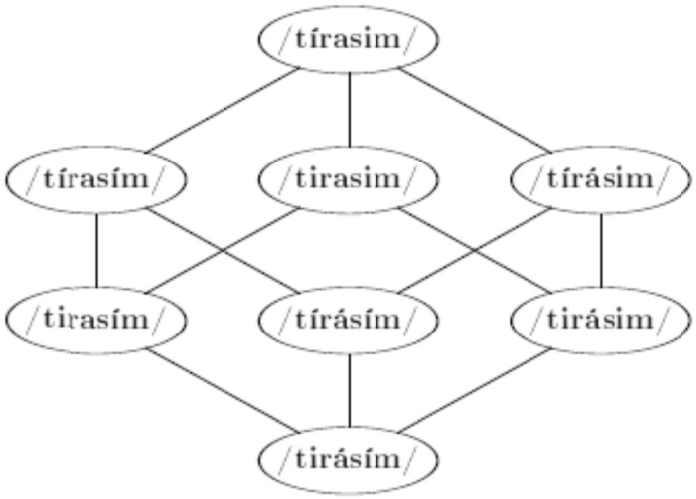
(31) Relative similarity lattice for [pá:]



To learn the underlying stress and length values of [pá:], just the candidates with the single-disparity inputs need be evaluated for consistency with the current support. If the single-disparity candidate /pá/ → [pá:] is inconsistent with the current ranking information, the learner knows that it must be because the input /pá/ is too dissimilar to the output, specifically because the candidate has a length disparity on the second segment. Therefore, determining that the candidate /pá/ → [pá:] is inconsistent means learning that the underlying form includes the setting [+length] on the second segment. The candidate /pa/→ [pá:] never has to be evaluated, because if either of the single-disparity candidates are inconsistent, it will be also. Similarly, it will be consistent if both of the single-disparity candidates are consistent.

For a slightly more complicated example, consider the relative similarity lattice for *tírasim* in (32). This word contains three syllables with one binary stress feature in each, producing eight local lexical hypotheses for the underlying form, and the lattice includes nodes for inputs with zero, one, two, and three disparities from the observed form. Again, the pertinent inputs for setting features, the single-disparity inputs, appear in the second row from the top. For any of the candidates including these inputs, determining that the candidate is inconsistent will enable the ODL to set a stress feature. The candidates with the more dissimilar inputs from the third and fourth rows of the lattice do not have to be evaluated.

(32) Relative similarity lattice for [(tíra)sim]



The computational savings afforded by exploiting the structure of output-driven maps can be considerable. In the above example, only three of the eight local lexical hypotheses for this single form need be tested to determine if any of its features can be set. For contrast pairs, the efficiency gains are even greater. While the learner must still test 2^x lexical hypotheses for x unset features that alternate in the surface forms of a

contrast pair, only *y* hypotheses need be tested for *y* non-alternating unset features. In general, for binary feature values, if there are *y* non-alternating unset features and *x* alternating unset features, the learner must test 2^x *$*(x+y)$* pairs for inconsistency. For the contrast pair *gàmadím* and *tírasim*, there are four non-alternating unset features and one alternating unset feature (the stress feature of the suffix /-im/), yielding 10 lexical hypotheses (2^1*(4+1)) out of the 32 total hypotheses. The unshaded cells of the chart in (33) represent all the single disparity lexical hypotheses for this contrast pair.[11]

(33) Single disparity lexical hypotheses for contrast pair *gàmadím* and *tírasim*

gamad	*tíras*	*-im*	*gamad*	*tíras*	*-im*	*gamad*	*tíras*	*-im*	*gamad*	*tíras*	*-im*
/--/	/--/	/-/	/+-/	/--/	/-/	/-+/	/--/	/-/	/++/	/--/	/-/
/--/	/--/	/+/	/+-/	/--/	/+/	/-+/	/--/	/+/	/++/	/--/	/+/
/--/	/+-/	/-/	/+-/	/+-/	/-/	/-+/	/+-/	/-/	/++/	/+-/	/-/
/--/	/+-/	/+/	/+-/	/+-/	/+/	/-+/	/+-/	/+/	/++/	/+-/	/+/
/--/	/-+/	/-/	/+-/	/-+/	/-/	/-+/	/-+/	/-/	/++/	/-+/	/-/
/--/	/-+/	/+/	/+-/	/-+/	/+/	/-+/	/-+/	/+/	/++/	/-+/	/+/
/--/	/++/	/-/	/+-/	/++/	/-/	/-+/	/++/	/-/	/++/	/++/	/-/
/--/	/++/	/+/	/+-/	/++/	/+/	/-+/	/++/	/+/	/++/	/++/	/+/

Detecting an inconsistency depends on having access to necessary ranking conditions. Consequently, the ranking and the lexicon have a symbiotic relationship. As the learner determines more about the ranking, typically more features can be set by inconsistency detection.[12] In turn, setting features can produce more ranking information. Morphemic alternations, in which the same morpheme has different surface realizations in different environments, indicate that at least one of the surface realizations must be unfaithful. A

[11] A joint relative similarity order can be constructed for contrast pairs (Tesar to appear). If the environment morpheme alternates, as *–im* does in this example, the joint relative similarity order is constituted of two separate suborders, one for each value of the alternating feature. Each suborder is a lattice.

[12] There are some cases, however, where inconsistency detection fails to set features even if the learner has full knowledge of the target grammar; see section 4.2.

set feature that surfaces unfaithfully suggests that a markedness constraint must dominate some faithfulness constraint[13]; therefore, this unfaithful mapping may yield non-phonotactic ranking information if, having set the feature, an error is detected on the resulting mapping (Tesar 2006, to appear).

Section 1.3.4 has shown that inconsistency detection can allow for extremely efficient, successful learning from data with ambiguous structural interpretations. This section has now shown that inconsistency detection can be used to learn a lexicon as well, by efficiently evaluating the space of local lexical hypotheses. Judging by the number of lexical hypotheses that must be evaluated to set a feature (if one can be set), it is clear that the Output-Driven Learner (ODL) is highly efficient: for single forms, the number of lexical hypotheses to evaluate only increases linearly with the number of features in the form. Contrast pairs require more evaluations, but far fewer than otherwise if the structure of output-driven maps is exploited.

However, while the ODL can identify where to focus learning efforts, there is no guarantee on any evaluation that a feature will get set or that the feature can ever be set, even if it is contrastive in the target language. The grammar itself is one factor: a certain body of knowledge is required to set features by inconsistency detection. The same lexical hypotheses may be evaluated repeatedly until the support contains just the right ERCs to produce an informative inconsistency. The map of the target language is another factor. As chapter 4 explains, the paradigmatic properties of the map may prohibit successful applications of inconsistency detection, even if the map is output-driven.

[13] An unfaithful mapping does not always result from an M >>F violation; there may also be restrictions on GEN that force unfaithful mappings, as in the Stress system described in 2.3.1; however, even these unfaithful mappings can reveal new ranking information, as described in section 2.4.4.2.

Setting features for these languages can require appealing to special strategies for deriving additional ranking information (see section 4.1.2). Finally, if the map is not output-driven, the ODL will not succeed. However, Tesar (to appear) suggests some directions towards modifications for achieving benefits of the ODL with non-output-driven maps.

1.4.2 LEXICAL CONSTRAINTS AND THE GLA

In the RCD-based inconsistency detection learners described in 1.3.4 and 1.4.1, the support is the added structure to a hypothesis that makes inconsistency detection possible and ensures that errors do not recur; its presence is a major distinguishing characteristic for these learners. In contrast, while the GLA is also error-driven, it makes no use of a support. Apoussidou (2007) proposes a learner in the GLA-framework that also eschews the added structure of a separate lexicon and instead learns underlying forms via the ranking of lexical constraints.

For this learner, each candidate for evaluation consists of an underlying form, surface representation, and meaning. Lexical constraints assign violations for pairing particular underlying forms and meanings in the same candidates (following Boersma 2001). For example, (34) below shows some of the lexical constraints for the root /tíras/ 'corn' in Modern Hebrew following Apoussidou's model.

(34) Lexical constraints on the root *tíras*

a) */tiras/ 'corn'
b) */tírás/ 'corn'
c) */tirás/ 'corn'
d) */tíras/ 'corn'

Informally, the meaning of each constraint is that the underlying form and meaning should not be associated. If these constraints were ranked according to their order in (34), the most harmonic candidates with respect to this constraint would associate /tíras/ and 'corn,' and would vary only by surface representation. Deciding between these candidates would then fall to the markedness and faithfulness constraints.

The constraints listed in (34) include just the lexical constraints whose underlying form component includes possible settings of the stress features in the root. Since there are two binary stress features that must be set, there are only four ($=2^2$) hypotheses for the settings; however, the complete list of lexical constraints that matter for learning the underlying form of /tíras/ is far larger. In fact, there must be a lexical constraint against each pairing of a hypothetical underlying form and a meaning – and even pairings that include the meanings of other morphemes (Apoussidou 2007, p. 176). For just /tíras/ and the root /gamad/ 'dwarf' from Modern Hebrew, the constraint set must include all of the following lexical constraints.

(35) Lexical constraints on the root *tíras* and *gamad*

a.	*/tiras/ 'corn'	b.	*/tírás/ 'corn'	c.	*/tirás/ 'corn'	d.	*/tíras/ 'corn'
e.	*/tiras/ 'dwarf'	f.	*/tírás/ 'dwarf'	g.	*/tirás/ 'dwarf'	h.	*/tíras/ 'dwarf'
i.	*/gamad/ 'corn'	j.	*/gámád/ 'corn'	k.	*/gamád/ 'corn'	l.	*/gámad/ 'corn'
m.	*/gamad/ 'dwarf'	n.	*/gámád/ 'dwarf'	o.	*/gamád/ 'dwarf'	p.	*/gámad/ 'dwarf'

In one of Apoussidou's simulations, the learner tackles the problem of learning structural interpretations as well as underlying forms. The learning data, from Modern Greek, include six words, comprising three different roots and three monosyllabic suffixes. The constraint set contains four faithfulness constraints (MAX and DEP for root

and affix stress), six markedness constraints to account for structural interpretations, and two lexical constraints for each morpheme.[14] The learner randomly observes each form repeatedly (in descriptions of the other simulations involving learning underlying forms only, the learner observes 1,000,000 forms in each simulation). The ranking is updated by error-driven learning according to the GLA strategy explained in 1.2.2.

Results of ten simulations indicate that the overt forms generated by the final ranking always match the observed forms, but underlying forms and surface interpretations can vary across simulations.[15] Additionally, within a simulation the underlying form of the same morpheme can also vary due to the interaction of faithfulness and lexical constraints.

The crucial example in the simulation involves the faithfulness constraint MAX(AFFIX), the lexical constraint */-on/ 'Gen. Pl.' and the suffix /-ón/ 'Gen. Pl.,' which is analyzed as underlyingly +stress. In the desired analysis, this suffix surfaces faithfully if the root is underlyingly unstressed, as in *θalasón* 'sea-Gen.Pl', but unfaithfully if the root is stressed underlyingly, as in *γóndolon* 'gondola-Gen.Pl.'. Thus, the desired analysis of the underlying forms is /θalas+ón/ and /γóndol+ón/, respectively.

For *θalasón,* either ranking of MAX(AFFIX) and */-on/ 'Gen. Pl.' achieves the desired analysis because the candidate with the underlying +stress suffix harmonically bounds

[14] The constraint sets in all simulations do not contain the full set of possible lexical constraints as represented by the example in (35). Apoussidou explains that because the learning data include meanings (e.g. "*tíras* 'corn'"), lexical constraints like */tíras/ 'dwarf' will be ranked very highly, with the implication that those constraints are not pertinent for learning underlying forms.

[15] Such variance in outcomes is not necessarily a problem because languages can be globally ambiguous with respect to their surface forms and lexica, as explained in chapter 4. In these cases, it is possible for different constraint rankings and lexica to generate the same forms and morpheme behaviors.

the –stress candidate, as shown in (36). In all simulations *θalasón* ‘sea-Gen.Pl’ therefore has the underlying form /θalas+ón/.

(36) MAX(AFFIX) and */-on/ ‘Gen. Pl.’ do not conflict for *θalasón* ‘sea-Gen.Pl.’

‘sea-Gen.Pl.’			MAX(AFFIX)	*/-on/ ‘Gen. Pl.’
Meaning	UF	Surface Rep.		
a. ‘sea-Gen.Pl.’	/θalas +ón/	θalasón	0	0
b. ‘sea-Gen.Pl.’	/θalas +on/	θalasón	0	1

But if the learner derives a ranking in which MAX(AFFIX) dominates */-on/ ‘Gen. Pl.’, then the underlying form of this suffix could vary with its environment. The tableau in (37) shows that MAX(AFFIX) and */-on/ ‘Gen. Pl.’ conflict if the suffix surfaces as unstressed: satisfying one entails a violation of the other. Under the ranking MAX(AFFIX) >> */-on/ ‘Gen. Pl.’, the genitive plural suffix is underlyingly unstressed in the word *γóndolon.*

(37) MAX(AFFIX) and */-on/ ‘Gen. Pl.’ conflict for *γóndolon* ‘gondola-Gen.Pl.’

‘gondola-Gen.Pl.’			MAX(AFFIX)	*/-on/ ‘Gen. Pl.’
Meaning	UF	Surface Rep.		
a. ‘gondola-Gen.Pl.’	/γóndol+ón/	γóndolon	1	0
b. ‘gondola-Gen.Pl.’	/γóndol+on/	γóndolon	0	1

Allowing for different underlying forms for the same morpheme is not necessarily a problem. If suffix stress were neutralized, then regardless of underlying form the grammar would ensure the correct surface form. But suffix stress in these data is contrastive. In the nominative singular forms, the first syllable bears the main stress: *θálasa* ‘sea-Nom.Sg.’ and *γóndola* ‘gondola-Nom.Sg.’. It must be the underlying stress

value of the suffix that causes main stress to alternate between first and final syllables of surface forms of the root /*θalas-*/, and therefore the underlying form of the genitive plural suffix should have a stable stress value.

To review, while this learner has eliminated the added structure of the support and lexicon, it remains quite complex. The explosion of lexical constraints, and the increased effort that results from evaluating these constraints, comes as the cost of learning about underlying forms through the ranking. Within this model the right ranking values, determined using error-driven learning, reveal both underlying forms and surface representations; the learner has no record of nor commitment to any structure outside of the ranking. Additionally, the interactions between faithfulness and lexical constraints lead to unusual conclusions, such as the same morpheme having a different underlying form according to its environment – and this result occurs even without the additional lexical constraints that have been ignored for this simulation. How different does the underlying form in the lexical constraint have to be from the observed form to ensure that the constraint does not have a crucial effect on a competition?

As for the relative efficiency of this learner, it is not clear just how many forms must be processed before converging on a ranking that will generate the observed forms. It is clear, however, that as the number of lexical constraints increases, the amount of computation will increase as well. In fact, Apoussidou explicitly notes that not including all the possible lexical constraints in the simulations reduces computation time (2007, p. 176). It is reasonable to expect that these constraints would affect not only the time

needed to compute violation profiles, but also how many forms must be processed before converging on a ranking.

1.4.3 Maximum Likelihood Learning of Lexicons and Grammars

Maximum Likelihood Learning of Lexicons and Grammars, or MLG (Jarosz 2006), is a stochastic but not error-driven model for learning a grammar and lexicon. As with the GLA learners, an MLG grammar is a probability distribution over constraint rankings, but additionally, each underlying form is a probability distribution over possible underlying forms. For each observed form, the learner updates the grammar and lexicon by rewarding the ranking and underlying form combinations that maximize the likelihood of the observed form.

The MLG represents a particular kind of approach to learning, with no particular set algorithmic implementation. For one implementation, Jarosz (2006, 2007) uses the Expectation-Maximization (EM) algorithm (Dempster et al. 1977) to illustrate MLG's ability to derive restrictive grammars and simulate the effects of ranking biases. In the simulations performed by this implementation, the grammar is a probability distribution of all total constraint rankings, and the lexicon a probability distribution over all possible underlying forms. During phonotactic learning, probabilities in the lexicon remain fixed with a uniform value, while the probabilities associated with different total rankings change according to the data. The more frequently an overt form is observed, the more rankings that maximize the likelihood of that form are rewarded.

For an extreme example of restrictiveness, one target language in the simulation has one overt form for all words (Jarosz 2007). As this form is observed repeatedly, the

probabilities associated with the rankings change: those associated with rankings that cannot generate this form at all shrink, while the others grow. Over time, as only this one form is observed, probabilities for rankings that generate other forms besides this one shrink as well, because the form is most likely to be generated by rankings that can only generate it and no other forms – that is, the most restrictive rankings. Having learned these rankings (or stratified hierarchy), the probabilities associated with different underlying forms can remain uniform: each is equally likely to be the underlying form that maps to the observed output, because the right ranking ensures that mapping. In simulations without such extensive neutralization, the lexicon is learned by a similar process, rewarding grammar and lexicon combinations that make the observed forms most likely.

This simulation of MLG reveals very appealing qualities about its approach. Its ability to derive restrictive rankings is elegant, it models gradual learning and because it is sensitive to the frequency of observed forms, it shares the GLA learners' robustness with noisy data. However, the simulation is tractable only because the constraint set and rich base are small. Rankings are not rewarded on the basis of a particular input mapping to the observed overt form (the identity map has no role in this approach), but rather on the likelihood of generating the overt form given the rich base. This approach does not scale well as constraint sets and underlying forms increase.

To improve the efficiency of MLG learners, Jarosz (to appear) proposes several sampling variants using the EM algorithm. The specific implementations of these variants differ, but the general idea is the same. The learner maintains a single, stochastic

grammar hypothesis and for each observed form, samples a ranking compatible with the grammar. Again, if the ranking generates the observed form, it is rewarded; however, in these variants the grammar is a stochastic partial order updated according to the probabilities of pairwise rankings. Similarly, the lexicon is represented with probabilistic binary features, and feature values that generate the observed form are rewarded.

In simulations of three sampling variants with the same data used to evaluate the Naïve Pairwise Ranking Learner in 1.3.3, none of the variants successfully learned all 124 of the languages, although the worst performing variant only failed to learn seven languages. In another set of simulations, the variants successfully learned a grammar and lexicon to generate the forms of the "paka" language (Tesar 2006). While MLG is a promising approach to learning a grammar and lexicon, it is clear that more work is needed to develop implementations that efficiently achieve complete success.

1.5 Conclusion

The challenge of learning hidden structure and the grammar simultaneously lies in the fact that the two affect each other. Knowing the hidden structure would yield information about the grammar, and vice versa, but a single observed form and an initial constraint hierarchy offer little solid information. The learners described in the preceding sections address this uncertainty in various ways, from bypassing problematic structures altogether (NPRL, MLG) to finding ways to safely explore alternatives (IDL, ODL). In particular, the error-driven learners that incorporate inconsistency detection have demonstrated the benefit of exploiting the relationships between observed data, inferred

structural knowledge, and the grammar. Their strategies provide the foundation for the Commitment-Based Learner, introduced in chapter 2.

2 THE COMMITMENT-BASED LEARNER

This chapter introduces the Commitment-Based Learner (CBL) for learning structural interpretations and underlying forms. First, section 2.1 provides a closer look at how these hidden structures can interact to pose challenges that any learner must somehow overcome. Section 2.2 then uses the mutual dependency among structures to motivate the commitment-based strategy, showing that if the learner can uncover one hidden structure, it can be used to learn another in turn; sections 2.2.1 and 2.2.2 provide a basic description of how the learner makes structural and lexical commitments.

Before delving any more deeply into the details of the CBL, section 2.3 introduces the Stress system used to evaluate this learner in simulations. This system provides the examples for the remainder of the dissertation, beginning with those illustrating the CBL's actual implementation in section 2.4, which constitutes the majority of this chapter. Section 2.4 explains the CBL's actions at critical points as the learner develops a language hypothesis corresponding to target L5 from the Stress system typology. The focus of this section is identifying what the critical issues are at different stages, explaining when the learner decides to handle them, and describing how they are handled.

The complete simulation from which these pieces are drawn appears in chapter 3, which includes the learner's progress beginning with the initial data and ending with the set of all final, consistent language hypotheses. In illustrating the simulation in its entirety, chapter 3 will demonstrate how the sets of commitments made by different

language hypotheses can affect when and how the learner confronts the issues identified in 2.4.

2.1 MUTUAL DEPENDENCY AMONG HIDDEN STRUCTURES

Each word the learner observes includes information about the grammar of the language that generated the word. Some information is explicit, such as the phonotactic information conveyed in an overt form. Other information is indirect and implicit. For example, the overt form represents an output with a particular prosodic interpretation, and whatever that output is, the grammar has mapped some input to it. The overt form therefore reflects an input-output mapping, but what the input and output are and what ranking determines that mapping are questions that cannot be resolved directly from the overt form. The crux of the learner's task is to use the direct evidence to infer the unobservable information, yet attempting to learn about both hidden structures at once involves untangling complicated interactions between them, within single forms as well as in combination with other forms.

First, for a single form the hidden structures of inputs and outputs can interact to make some input-output mappings possible and others not. For example, suppose that the learner observes the word *tírasim* 'corn pl.' (Modern Hebrew).[16] Only two structural

[16] Modern Hebrew was chosen for examples here because its properties can be modeled by the Stress system. Stress assignment in Modern Hebrew is quantity-insensitive, and in the nominal system primary stress is by default word final, with secondary stress on alternate syllables to the left (Bolozky 1982, Bat-El 1993, Graf 2000); this default pattern typically appears in native words (Becker 2003a). I have assumed the inclusion of secondary stresses, following Bolozky 1982, Bat-El 1993, Graf 2000, and Graf and Ussishkin 2003, but note that Becker (2003b), citing laboratory experiments and data from a radio talk show, reports that only the primary stressed syllable exhibits phonetic evidence for stress, in the form of increased vowel duration. Finally, against previous analyses (Bolozky 1982, Bat-El 1993, Graf 2000, Becker 2003b), I assume that the language parses iambic feet from right to left; however, in contrast with Graf and Ussishkin (2003), who derive this effect without recourse to constraints on foot

interpretations match the observed form in its pattern of stresses: [(tíra)sim] and [(tí)rasim]. The learner must determine an underlying form and constraint ranking that selects one of these interpretations as optimal against all other candidates. To simplify this illustration, consider only the possible underlying forms for which candidates incur no violations of MAXSTRESS[17], namely /tiras+im/ and /tíras+im/. While there are rankings that will map the input /tíras+im/ to either possible structural interpretation, no ranking will map the input /tiras+im/, with no lexical stress, to the interpretation [(tí)rasim]; only the binary trochaic interpretation, [(tíra)sim], can be optimal. As the tableau below shows, the degenerate interpretation is harmonically bounded by a structural description parsing an iambic foot at the left edge.

(38) Harmonic bounding of /tiras+im/[(tí)rasim]

/tiras+im/	PARSE-σ	FT-BIN	IAMB	FNF	*LAPSE	AFL	LMOST	RMOST	MAXSTR
[(tí)rasim] ~ [(tirá)sim]	L	L			L			L	

Second, hidden structures can also interact across forms via the ranking conditions they separately impose on the grammar, making some combinations of structures possible and others not. To illustrate, suppose that the learner observes another word, *mèvugár* 'adult sg.', which has two non-harmonically bounded structural interpretations: [(mèvu)(gár)] and [(mè)(vugár)].[18] Again, two lexical hypotheses incur no MAXSTRESS violations: /mevugar/ and /mevugár/. The learner must determine the correct underlying

structure, I derive it via the rhythm constraints of the Stress system. The data for Modern Hebrew are from Bolozky 1982 and Graf 2000.

[17] All constraints are defined in 2.3.

[18] A third interpretation, [(mè)vu(gár)], is harmonically-bounded by [(mè)(vugár)], regardless of the underlying stress of the input.

forms and structural interpretations of both *tírasim* and *mèvugár*, where doing so requires finding a ranking that makes both input-output mappings optimal. Each word has two possible faithful inputs and two valid structural interpretations, creating sixteen different combinations of faithful mappings, shown in the chart in (39).

(39) All faithful mappings of interpretations of *tírasim* and *mèvugár*

Underlying Forms		**Structural Interpretations**	
a. /tiras+im/	/mevugar/	[(tíra)sim]	[(mè)(vugár)]
b. /tiras+im/	/mevugar/	[(tíra)sim]	[(mèvu)(gár)]
c. /tiras+im/	/mevugar/	[(tí)rasim]	[(mè)(vugár)]
d. /tiras+im/	/mevugar/	[(tí)rasim]	[(mèvu)(gár)]
e. /tiras+im/	/mevugár/	[(tíra)sim]	[(mè)(vugár)]
f. /tiras+im/	/mevugár/	[(tíra)sim]	[(mèvu)(gár)]
g. /tiras+im/	/mevugár/	[(tí)rasim]	[(mè)(vugár)]
h. /tiras+im/	/mevugár/	[(tí)rasim]	[(mèvu)(gár)]
i. /tíras+im/	/mevugar/	[(tíra)sim]	[(mè)(vugár)]
j. /tíras+im/	/mevugar/	[(tíra)sim]	[(mèvu)(gár)]
k. /tíras+im/	/mevugar/	[(tí)rasim]	[(mè)(vugár)]
l. /tíras+im/	/mevugar/	[(tí)rasim]	[(mèvu)(gár)]
m. /tíras+im/	/mevugár/	[(tíra)sim]	[(mè)(vugár)]
n. /tíras+im/	/mevugár/	[(tíra)sim]	[(mèvu)(gár)]
o. /tíras+im/	/mevugár/	[(tí)rasim]	[(mè)(vugár)]
p. /tíras+im/	/mevugár/	[(tí)rasim]	[(mèvu)(gár)]

First consider the combinations in (39)a-(39)d, which have unstressed inputs for both words. The tableau in (40) reveals that the candidates in (39)a, /tiras+im/[(tíra)sim] and /mevugar/[(mè)(vugár)], have contradictory ranking requirements. Because these candidates therefore are impossible in combination, at least one of the underlying forms or structural interpretations in these mappings must be incorrect.

(40) /tiras+im/[(tíra)sim] and /mevugar/[(mè)(vugár)] are inconsistent

Input	W~L	PARSE-σ	FT-BIN	IAMB	FNF	*LAPSE	AFL	LMOST	RMOST	MAXSTR
/tiras+im/	[(tíra)sim] ~ [(tì)(rasím)]	L	W	L	W	L	W	W	L	
/mevugar/	[(mè)(vugár)] ~ [(mévu)gar]	W	L	W	L	W	L	L	W	

Changing the structural interpretations offers no improvement. The combination of mappings in (39)b retains /tiras+im/[(tíra)sim] but includes the alternative interpretation of *mèvugár*: /mevugar/[(mèvu)(gár)]. Yet this combination cannot be correct either, as (41) demonstrates that /mevugar/[(mèvu)(gár)] is harmonically bounded. As a result, no combination that includes /mevugar/[(mèvu)(gár)] can be correct. Therefore, (39)b and (39)d are impossible, as are (39)j and (39)l.

(41) Harmonic bounding of /mevugar/[(mèvu)(gár)]

Input	W~L	PARSE-σ	FT-BIN	IAMB	FNF	*LAPSE	AFL	LMOST	RMOST	MAXSTR
/mevugar/	[(mèvu)(gár)] ~ [(mè)(vúgar)]						L	L		

Likewise, (38) has already shown that /tiras+im/[(tí)rasim] is harmonically bounded; therefore, (39)c is not a valid combination of mappings, and neither are (39)g and (39)h. These facts together prove that the underlying forms /tiras+im/ and /mevugar/ cannot both be correct, as (39)a-(39)d are all inconsistent. Additionally, the harmonic bounding of /tiras+im/[(tí)rasim] and /mevugar/[(mèvu)(gár)] has ruled out all combinations that includes either one of these candidates, thereby eliminating half of the combinations included in (39). All inconsistent combinations to this point are shaded in (42) below.

(42) Combinations with /tiras+im/[(tí)rasim] or /mevugar/[(mèvu)(gár)] are inconsistent

Underlying Forms		Structural Interpretations	
a. /tiras+im/	/mevugar/	[(tíra)sim]	[(mè)(vugár)]
b. /tiras+im/	/mevugar/	[(tíra)sim]	[(mèvu)(gár)]
c. /tiras+im/	/mevugar/	[(tí)rasim]	[(/mè)(vugár)]
d. /tiras+im/	/mevugar/	[(tí)rasim]	[(mèvu)(gár)]
e. /tiras+im/	/mevugár/	[(tíra)sim]	[(mè)(vugár)]
f. /tiras+im/	/mevugár/	[(tíra)sim]	[(mèvu)(gár)]
g. /tiras+im/	/mevugár/	[(tí)rasim]	[(mè)(vugár)]
h. /tiras+im/	/mevugár/	[(tí)rasim]	[(mèvu)(gár)]
i. /tíras+im/	/mevugar/	[(tíra)sim]	[(mè)(vugár)]
j. /tíras+im/	/mevugar/	[(tíra)sim]	[(mèvu)(gár)]
k. /tíras+im/	/mevugar/	[(tí)rasim]	[(mè)(vugár)]
l. /tíras+im/	/mevugar/	[(tí)rasim]	[(mèvu)(gár)]
m. /tíras+im/	/mevugár/	[(tíra)sim]	[(mè)(vugár)]
n. /tíras+im/	/mevugár/	[(tíra)sim]	[(mèvu)(gár)]
o. /tíras+im/	/mevugár/	[(tí)rasim]	[(mè)(vugár)]
p. /tíras+im/	/mevugár/	[(tí)rasim]	[(mèvu)(gár)]

While *tírasim* and *mèvugár* cannot both have unstressed inputs, it remains possible that just one has an unstressed input. In fact, (42)e,f,i,k, are all consistent. First, /tiras+im/[(tíra)sim] is consistent with both interpretations of *mèvugár* as long as the input is /mevugár/. As proof, the tableau in (43) includes the W-L pairs that determine a ranking for the combination in (42)e, /tiras+im/[(tíra)sim] and /mevugár/[(mè)(vugár)]; the ranking itself appears in (44).

(43) /tiras+im/[(tíra)sim] and /mevugár/[(mè)(vugár)] are consistent

Input	W~L	MaxStr	AFL	Lmost	Parse-σ	Rmost	FNF	Ft-Bin	Iamb	*Lapse
a. /mevugár/	[(mè)(vugár)] ~ [(mevú)gar]	W	L	L	W	W	L	L		
b. /tiras+im/	[(tíra)sim] ~ [ti(rasím)]		W	W		L	W		L	
c. /tiras+im/	[(tíra)sim] ~ [(tirá)(sìm)]		W		L		W	W	L	L
d. /mevugár/	[(mè)(vugár)] ~ [me(vugár)]				W		L	L		W
e. /tiras+im/	[(tíra)sim] ~ [(tirá)sim]						W		L	L

(44) MAXSTRESS >> {AFL, LMOST} >> {PARSE-σ, RMOST} >> {FNF, FT-BIN} >> {IAMB, *LAPSE}

The combinations of mappings in (42)f,i,k, are also all consistent. Rankings for each of these combinations appear below.

(45) Ranking for (42)f: /tiras+im/[(tíra)sim] and /mevugár/[(mèvu)(gár)]

MAXSTRESS >> FNF >> PARSE-σ >> {IAMB, FT-BIN, AFL, LMOST} >> {RMOST, *LAPSE}

(46) Ranking for (42)i: /tíras+im/[(tíra)sim] and /mevugar/[(mè)(vugár)]

MAXSTRESS >> RMOST >> {IAMB, AFL, LMOST} >> {PARSE-σ, *LAPSE} >> {FNF, FT-BIN}

(47) Ranking for (42)k: /tíras+im/[(tí)rasim] and /mevugar/[(mè)(vugár)]

{MAXSTRESS, IAMB} >> RMOST >> {AFL, LMOST} >> {PARSE-σ, *LAPSE} >> {FNF, FT-BIN}

As a brief aside, the W-L pairs in tableau (43) constitute the support for the *skeletal basis* for that combination of mappings (Brasoveanu 2003, Brasoveanu and Prince 2011, Prince 2002a). For comparison, the skeletal basis itself is given in (48). For each row of the support tableau in (43), the skeletal basis includes only the W's in the highest stratum represented in the row and the L's that they immediately dominate; any information that can be derived by transitivity with other rows has been removed. Thus, the "W" under *LAPSE in (43)d is absent from the corresponding row of the skeletal basis because (43)d and (43)e jointly entail that PARSE-σ dominate *LAPSE.

(48) Skeletal basis for /tiras+im/[(tíra)sim] and /mevugár/[(mè)(vugár)]

ERC	MAXSTR	AFL	LMOST	PARSE-σ	RMOST	FNF	FT-BIN	IAMB	*LAPSE
(43)a	W	L	L						
(43)b		W	W		L				
(43)c		W		L					
(43)d				W		L	L		
(43)e						W		L	L

The support of a skeletal basis provides clear and direct evidence of the minimum W-L pairs required to produce a ranking that ensures the optimality of a set of candidates. For this reason, in the remainder of this dissertation the support of the skeletal basis will serve as evidence for any ranking that is not otherwise supported by a set of ERCs derived from error-driven learning, such as the rankings for the target languages of the Stress system. The supports of the skeletal bases for the combinations in (42)f,i,k above can be found Appendix A.

Finally, to finish the discussion of the combinations in (42), those in (42)m-p include lexically-stressed inputs for both words. The combinations in (42)m-o are consistent. Again, rankings for these combinations appear below, while the supports for their skeletal bases are included in Appendix A.

(49) Ranking for (42)m: /tíras+im/[(tíra)sim] and /mevugár/[(mè)(vugár)]

MAXSTRESS >> {AFL, LMOST, RMOST} >> {IAMB, PARSE-σ, * LAPSE} >> {FNF, FT-BIN}

(50) Ranking for (42)n: /tíras+im/[(tíra)sim] and /mevugár/[(mèvu)(gár)]

MAXSTRESS >> {FNF, RMOST} >> {PARSE-σ, *LAPSE} >> {IAMB, FT-BIN, AFL, LMOST}

(51) Ranking for (42)o: /tíras+im/[(tí)rasim] and /mevugár/[(mè)(vugár)]

{MAXSTRESS, IAMB} >> {AFL, LMOST, RMOST} >> {PARSE-σ, *LAPSE} >> {FNF, FT-BIN}

The only inconsistent combination is in (42)p. As shown in (52), the ranking restrictions imposed by the candidates /tíras+im/[(tí)rasim] and /mevugár/[(mèvu)(gár)] contradict for *LAPSE, IAMB, PARSE-σ, and AFL: no ranking can make both candidates optimal.

(52) /tíras+im/[(tí)rasim] and /mevugár/[(mèvu)(gár)] are inconsistent

Input	W~L	MAXSTR	FT-BIN	LMOST	RMOST	FNF	*LAPSE	IAMB	PARSE-σ	AFL
/tíras+im/	[(tí)rasim] ~ [(tí)(ràsim)]						L	W	L	W
/mevugár/	[(mèvu)(gár)] ~ [me(vugár)]		L	L			W	L	W	L

Although there are sixteen faithful combinations of mappings for *tírasim* and *mèvugár*, only seven can be correct for these observed forms. The following chart briefly summarizes the possible faithful input-output mappings for the observed forms *tírasim* and *mèvugár*. It divides horizontally according to the hypothesized underlying form of *tírasim* first, and each part divides horizontally again for the hypothesized underlying forms of *mèvugár,* then again for the structural interpretations of each word. Thus, the topmost row of mappings in the chart indicates that there are no consistent interpretations of *tírasim* and *mèvugár* if both are unstressed underlyingly.

(53) Faithful input-output mappings for *tírasim* and *mèvugár*

Underlying Forms		**Structural Interpretations**	
/tiras+im/	/mevugar/	*no consistent combinations*	
	/mevugár/	[(tíra)sim]	[(mèvu)(gár)]
			[(mè)(vugár)]
/tíras+im/	/mevugar/	[(tíra)sim]	[(mè)(vugár)]
		[(tí)rasim]	
	/mevugár/	[(tíra)sim]	[(mèvu)(gár)]
			[(mè)(vugár)]
		[(tí)rasim]	[(mè)(vugár)]

To review, this section has shown the effect of mutual dependency on the range of possible input-output mappings for two observed forms. Out of sixteen combinations of mappings, nine are inconsistent. If a third form were introduced, the total number of combinations would increase as well, but again, the number of consistent combinations would be far fewer than the total. Only combinations that include the consistent mappings in (53) have the potential to be consistent at all, and among those, it is likely that many would be inconsistent due to interactions with the candidate for the new observed form.

This power of mutual dependency to reduce the space of hypotheses has been seen before, in use with the Inconsistency Detection Learner (IDL) in section 1.3.4. The IDL uses inconsistency detection to manage structurally ambiguous data like *tírasim* and *mèvugár* in order to learn the grammar for the observed forms, but unlike the Commitment-Based Learner (CBL), the IDL does not attempt to learn underlying forms.

Section 2.2 will show how the mutual dependency among structural interpretations and underlying forms can yield the information to simultaneously learn both kinds of hidden structure. This section provides the fundamental motivation for the CBL's approach to learning.

2.2 LEARNING FROM COMMITTED INFORMATION

The chart in (53) of the preceding section shows that there is no one underlying form or structural interpretation that is always correct; instead, what one structure can be depends on what the other structures are. As a result, an assumption about one structure can affect how the learner interprets another. For example, the shading in chart (54) below indicates that both consistent combinations of mappings that include [(tí)rasim] also include [(mè)(vugár)]. Therefore, if the learner is committed to interpreting *tírasim* as [(tí)rasim], then *mèvugár* must be interpreted as [(mè)(vugár)], simply because it the only interpretation of that form consistent with [(tí)rasim].

(54) Interpreting *tírasim* as [(tí)rasim] entails interpreting *mèvugár* as [(mè)(vugár)]

<table>
<tr><th colspan="2">Underlying Forms</th><th colspan="2">Structural Interpretations</th></tr>
<tr><td rowspan="2">/tiras+im/</td><td>/mevugar/</td><td colspan="2">no consistent combinations</td></tr>
<tr><td>/mevugár/</td><td>[(tíra)sim]</td><td>[(mèvu)(gár)]
[(mè)(vugár)]</td></tr>
<tr><td rowspan="2">/tíras+im/</td><td>/mevugar/</td><td>[(tíra)sim]
[(tí)rasim]</td><td>[(mè)(vugár)]</td></tr>
<tr><td>/mevugár/</td><td>[(tíra)sim]</td><td>[(mèvu)(gár)]
[(mè)(vugár)]</td></tr>
<tr><td colspan="2"></td><td>[(tí)rasim]</td><td>[(mè)(vugár)]</td></tr>
</table>

Mutual dependency therefore presents two key, related challenges. First, if learning these structures requires having learned something about a different structure, how does the learner ever start learning? Second, how does the learner ensure that each step taken is correct, or at least that it can recover from any missteps?

The GLA and MLG learners in 1.4.2 and 1.4.3, which share the Commitment-Based Learner's goal of learning a grammar and lexicon from structurally ambiguous data,

answer these questions in different ways. The GLA learner incorporates the lexicon into the grammar by way of lexical constraints. Because both structural interpretation and underlying form are evaluated by constraints, no prior knowledge of either structure is required; determining the most harmonic candidate determines both structures for a given observed form on a given evaluation. The MLG learner eliminates the need for prior knowledge by evaluating all possible underlying forms and structural interpretations. Both of these approaches to mutual dependency rely on the flexibility of probabilistic hypotheses, allowing these learners to recover from the use of incorrect values for hidden structure.

Yet, as explained in 1.4.2 and 1.4.3, these learners are both unsatisfactory. In the first case, the GLA approach introduces an explosion of lexical constraints which must have a non-trivial effect on the number of learning steps required to converge on a ranking; additionally, these lexical constraints interact with faithfulness constraints to yield an odd conclusion about the lexicon – namely, that the underlying form of a morpheme can vary with its context. The MLG approach is promising but its implementation is a work in progress, and in simulations the more efficient implementations, the sampling variants, fail to learn some languages.

Finally, in gaining flexibility both learners sacrifice the ability to capitalize on mutual dependency: knowing that one structure entails another can greatly reduce the space of possible grammars to account for the data. The Commitment-Based Learner employs mutual dependency to its benefit by using commitments to underlying forms and structural interpretations to winnow all possible hidden structures to just those possible in

combination. The interactions across kinds of hidden structures become an asset in this approach. With each newly observed form, the learner can develop more articulated hypotheses about the hidden structures of the language and its ranking conditions, and the space of possible grammars shrinks.

To see how this can work, consider *tírasim* and *mèvugár* again, now from the perspective of the CBL. Upon observing *tírasim* the learner, assuming that the identity map is optimal in the target language, can know that the target's grammar must generate a word whose input matches *tírasim* in its stress pattern and whose output is a valid interpretation of the overt form. There are two such mappings, /tírasim/[(tíra)sim] and /tírasim/[(tí)rasim], and each entails certain ranking restrictions that will affect what inferences the learner later draws from new data. Importantly, knowing something about the ranking conditions of the target language will help the learner to discern the correct hidden structures in new information. If the learner picks the right mapping here, its ranking conditions may help the learner determine the structural interpretation or underlying form of another word later. It therefore pays to learn this information now, but how? What kinds of commitments need to be made, and how does the learner insure against missteps while learning?

2.2.1 MAKING STRUCTURAL COMMITMENTS

To exploit mutual dependency effectively, the learner needs a base of information from which to draw conclusions. Commitments to structural interpretations provide a portion of that base. Adopting the approach of Prince and Tesar (2004), described in 1.2.1.3, the Commitment-Based Learner attempts to learn phonotactic ranking

information from observed forms. The potential for this information is indicated by two methods of error detection.

First, an error is detected if the current ranking does not map the identity input to exactly one output, it being any one of the structural interpretations of the observed form. Because the CBL uses the ranking information derived from error-driven learning throughout the course of learning, now there must be a particular, stable identity mapping associated with this overt form in order not to derive conflicting ranking information later. Therefore, detecting an error on the overt form leads the learner to commit to a particular structural interpretation to ensure that the interpretation will be the same every time the learner observes that overt form. If the learner commits to the interpretation [(tí)rasim] for the overt form *tírasim*, then any word that has the overt form *tírasim* will thereafter be interpreted as [(tí)rasim], even if it has a different morphemic composition from the word that initially spurred the commitment. Second, once the overt form has a committed interpretation, an error is detected if the current ranking does not map the identity input to that committed interpretation. The learner will add W-L pairs to the support and adjust the ranking until no more errors are detected. In this way, committing to a structural interpretation also commits the learner to an identity map and its entailed ranking conditions. Subsequent learning data will be interpreted through the lens of this ranking information.

To revisit the example in the preceding section, the ranking requirements of /tíras+im/[(tí)rasim] are inconsistent with those of /mevugár/[(mèvu)(gár)], as illustrated by the chart in (53). Committing to this identity mapping for *tírasim* will therefore

require the learner to interpret *mèvugár* as [(mè)(vugár)] instead. As a result, this information leads the learner to search for the target grammar in the space of grammars that generates the set of mappings /tíras+im/[(tí)rasim] and /mevugár/[(mè)(vugár)]. These are the broad strokes of phonotactic learning by the CBL: the learner finds sets of consistent identity maps for the observed data, committing to the structural interpretations and attendant ranking conditions of those mappings along the way.

Yet, how does the learner choose which structural interpretation to commit to? This is the first problem of handling mutual dependency: how does the learner choose a structural interpretation if picking the correct one requires knowing something more about the target language? Furthermore, the consequences of committing to a particular interpretation extend far beyond the interpretation itself: each interpretation commitment amounts to the decision taken between exploring a space of grammars that permits the mapping containing that interpretation and the space that does not. Because the learner cannot know which space of grammars the target language occupies, the safe tactic is to explore both by committing to each structural interpretation in its own language hypothesis.

The Inconsistency Detection Learner (IDL; section 1.3.4) uses just this strategy to learn structural interpretations from overt forms. The learner investigates multiple interpretations, certain that one of these interpretations is the correct one. The interactions between committed structural interpretations help to limit the proliferation of hypotheses over time, as the learner rules out combinations of interpretations with conflicting ranking requirements. By incorporating the IDL at this point, the CBL not only learns

about the structural interpretations, but amasses ranking knowledge that will inform later learning steps, including for learning underlying forms.

2.2.2 MAKING LEXICAL COMMITMENTS

When morphophonemic learning begins, the learner will already have a substantial foundation of knowledge about the grammar of the target language derived from the phonotactic information of the observed forms' identity mappings. Continuing to employ the IDL's strategy of pursuing multiple possibilities now would be unwise, both because it would amount to evaluating all possible underlying forms and structural interpretations, just as the probabilistic learners do, and because it is possible, through neutralization, that multiple underlying forms could generate the same data. Moreover, the IDL's strategy is unnecessary because the relationships among the accumulated commitments can reveal quite a lot about the space of possible underlying forms.

To exploit these relationships effectively, it is essential that an observed form have a committed interpretation. This structural commitment identifies what the word's input must map to, even though the precise underlying form remains unknown. Continuing the example from the preceding section, suppose the learner has committed to the following interpretations and identity mappings: /tírasim/[(tí)rasim] and /mevugár/[(mè)(vugár)]. The underlying forms in these mappings do not represent the learner's knowledge of the lexicon – at least, not yet. They only indicate some valid mappings in the hypothesized grammar, but the ranking requirements of these mappings will nonetheless enable the learner to infer knowledge of the underlying forms.

Given this set of consistent, committed identity mappings, the learner can ask, what are the underlying forms of these words? The CBL answers this question by asking another, slightly different question: how can the inputs for these words differ from their surface forms? Recall from 2.1 that the mapping from the unstressed input, /tirasim/[(tí)rasim], is harmonically bounded. Because the learner is committed to this structural interpretation, the harmonic bounding of /tirasim/[(tí)rasim] is evidence that the first syllable of the morpheme *tíras* cannot differ from its surface form and therefore must be stressed underlyingly.[19] The shaded cells in chart (55) illustrate this point. Observe that [(tí)rasim] is only a consistent interpretation if the input is /tírasim/, which matches the stress contour of the observed form.

(55) The underlying form of [(tí)rasim] must have initial stress

Underlying Forms		**Structural Interpretations**	
/tiras+im/	/mevugar/	*no consistent combinations*	
	/mevugár/	[(tíra)sim]	[(mèvu)(gár)]
			[(mè)(vugár)]
/tíras+im/	/mevugar/	[(tíra)sim]	[(mè)(vugár)]
		[(tí)rasim]	
	/mevugár/	[(tíra)sim]	[(mèvu)(gár)]
			[(mè)(vugár)]
		[(tí)rasim]	[(mè)(vugár)]

In contrast, the mapping /mevugar/[(mè)(vugár)] is not only *not* harmonically bounded, just like the faithful mapping it is consistent with the committed identity mapping for *tírasim,* as shown by the shaded cells in (56). The learner therefore cannot make any inferences about the underlying form of *mèvugár* based on the mapping with an unstressed input.

[19] This conclusion assumes that the target language is output-driven, as are all languages in the Stress system typology.

(56) Both /mevugar/[(mè)(vugár)] and /mevugár/[(mè)(vugár)] are consistent with /tírasim/[(tí)rasim]

Underlying Forms		**Structural Interpretations**	
/tiras+im/	/mevugar/	*no consistent combinations*	
	/mevugár/	[(tíra)sim]	[(mèvu)(gár)]
			[(mè)(vugár)]
/tíras+im/	/mevugar/	[(tíra)sim]	[(mè)(vugár)]
		[(tí)rasim]	
	/mevugár/	[(tíra)sim]	[(mèvu)(gár)]
			[(mè)(vugár)]
		[(tí)rasim]	[(mè)(vugár)]

These examples illustrate lexical learning by the CBL in its broadest strokes: if changing the value of a single feature from its surface correspondent's value makes the resultant mapping inconsistent with the other committed structures and mappings, then the feature must be set in the lexicon to match its surface value. Importantly, this strategy relies on the target language having an output-driven map, and the CBL therefore incorporates the Output-Driven Learner (section 1.4.1), which learns underlying forms by using inconsistency detection as described here.

2.2.3 CONCLUSION

Section 2.2 identified two problems for learning mutually dependent structures. The first, how the learner can begin to make headway if all structures depend on each other, has been answered for the CBL already: by using the multiple language hypothesis strategy of the IDL, the learner can simultaneously but separately commit to all interpretations of one overt form, knowing that one of these must be correct. Applying this strategy during the phonotactic learning stage provides a foundation for further learning, and it is also employed during the morphophonemic stage, when attempting to learn underlying forms requires words to have full structural descriptions. The second,

and more fundamental, question is, how does the learner protect against missteps and make progress in the right direction? The IDL's branching strategy offers one answer – pursue all options – and inconsistency detection itself offers another – commit to a structure because no other structure could follow from the prior commitments.

Underlying these strategies is the basic fact that the mutual dependency among inputs, outputs, and the ranking is less problematic if any two of the three are known, because the interactions themselves are informative. Learning structural interpretations by branching is a brute force response to knowing only one of the three components – the identity input. Setting features by inconsistency detection is more subtle, but successful because the learner has at least partial knowledge of two components – a committed output and whatever ranking conditions are recorded in the support. Learning the ranking derives from committed input-output mappings, first from identity mappings and later from mappings that include learned lexical information. By committing to these components – underlying forms, structural interpretations, and mappings – the learner builds a store of knowledge used to tease out informative interactions between the committed structures.

Finally, note that the piecewise commitments to single feature values stand in marked contrast to the CBL's treatment of structural ambiguity, where commitments are made to entire structural interpretations and incite hypothesis branching. Whereas parsing a syllable into a foot may affect where the boundaries of other feet lie, feature values are independent of each other, making it permissible to commit to one value without reference to the others. Moreover, the structure of output-driven maps makes it *possible*

to identify single feature values for commitments. In the example of the preceding section, the harmonic bounding of /tirasim/[(tí)rasim] is not only evidence that this particular lexical hypothesis is incorrect, but that all lexical hypotheses that include the -stress value on the first syllable's stress feature are also incorrect because they include the same disparity as this initial inconsistent mapping. There is no similar way to determine the optimal placement of feet independently of one another. These observations define a broader learning strategy for the CBL: branch only when piecewise commitments are impossible.

In the remainder of this chapter, section 2.3 introduces the Stress system, used to evaluate the CBL, while section 2.4 describes the CBL's use and storage of commitments for learning a language in the Stress system and focuses on the learner's actions at several key points. The illustration in 2.4 shows what information the CBL knows at each learning step, how the CBL learns from this information, and what conclusions it draws. The objective is to highlight the details of the CBL in examples of its basic learning procedures, those the CBL applies to learn most languages in the Stress system typology. Chapter 3 then follows the CBL step-by-step as it processes a set of learning data to completion.

2.3 The Stress System Typology and Simulation Details

Computer learning simulations of languages in the Stress system were conducted to evaluate the Commitment-Based Learner (CBL). This section describes the Stress system and explains how the data for the learning simulations were produced. The languages of

the Stress system are used for all examples of the Commitment-Based Learner throughout the remainder of the dissertation.

2.3.1 THE STRESS SYSTEM

Words in the Stress system consist of disyllabic roots and monosyllabic suffixes. Each syllable has a binary stress feature with the value stressed (+) or unstressed (-). In the input, stressed syllables will be indicated by *Y*, unstressed syllables by *s*. The Stress system contains four unique roots and two unique suffixes, shown in (57). These morphemes combine to form eight morphological words, in (58).

(57) Morphemes in the Stress system

Roots				Suffixes	
r1	r2	r3	r4	s1	s2
ss	Ys	sY	YY	s	Y

(58) Morphological words in the Stress system

Words	r1s1	r2s1	r3s1	r4s1	r1s2	r2s2	r3s2	r4s2
UFs	/ss-s/	/Ys-s/	/sY-s/	/YY-s/	/ss-Y/	/Ys-Y/	/sY-Y/	/YY-Y/

GEN makes the following restrictions on candidates. Each candidate has exactly one syllable with primary stress. Each stressed syllable is the head of a foot; therefore, each candidate has at least one foot: the head-foot, bearing primary stress. A foot may contain one or two syllables, and exactly one must be stressed. Therefore, a foot cannot be headless, unbounded or ternary. A syllable cannot be a member of more than one foot. Last, each candidate must have the same number of syllables as its input. These restrictions generate 24 possible outputs for each of the possible three-syllable inputs in the Stress system.

Note that in the Stress system, secondary stress is a property of -stress syllables in the output. The values +stress and "False" for the secondary-stress property represent the head of the head-foot, while -stress and "True" represent all other foot-heads; syllables that are not foot-heads are represented with the values –stress and "False". In the outputs described in this section and elsewhere, *Y* denotes an output syllable bearing primary stress, and *X* one bearing secondary stress. The output forms of the Stress system therefore appear as in (59).

(59) Output forms in the Stress system

[s(sY)]	[(sX)(Y)]	[s(X)(Y)]	[(Y)ss]	[(X)s(Y)]	[(X)(Y)s]
[ss(Y)]	[s(Ys)]	[(Ys)s]	[(Y)(sX)]	[(Y)(Xs)]	[(Y)(X)(X)]
[(sY)s]	[s(Y)s]	[(Ys)(X)]	[(X)(sY)]	[(X)(Ys)]	[(X)(Y)(X)]
[(sY)(X)]	[s(Y)(X)]	[(Xs)(Y)]	[(Y)s(X)]	[(Y)(X)s]	[(X)(X)(Y)]

The constraint set for the Stress system includes eight Markedness constraints and a single Faithfulness constraint, defined below.

(60) Constraints

a. FT-BIN Feet must be disyllabic. (Prince and Smolensky 1993)
b. PARSE-σ Syllables must be parsed into feet. (Prince and Smolensky 1993)
c. *LAPSE Rhythm is alternating; no two adjacent unstressed syllables.[20] (Alber 2005)
d. IAMB Feet must be right-headed.
e. FOOT-NONFINALITY (FNF) A foot must not be right-headed. (Tesar 2000)
f. ALL FEET LEFT (AFL) ∀ foot ∃ prosodic word such that the left edge of the prosodic word and the left edge of the foot coincide. (McCarthy and Prince 1993)
g. LEFTMOST (LMOST) ALIGN (PRWD, L, HEAD-FT, L) ∀ prosodic word ∃ head-foot of the prosodic word such that the right/left edge of the prosodic word and the right/left edge of the head-foot coincide. (EDGEMOST in Prince and Smolensky 1993)
h. RIGHTMOST (RMOST) ALIGN (PRWD, R, HEAD-FT, R)
i. MAXSTRESS For each stressed syllable in the input, assign a violation if the corresponding output syllable does not bear primary stress. (MAX-HEAD-FOOT in Graf 2000)

[20] The basic insight of this constraint can be found in Selkirk 1984. The constraint has been used in many more recent analyses, including Alber 2005, Gordon 2002, and Kager 2001, 2006.

2.3.2 THE LEARNING DATA

Data for the learning simulations were produced from all languages in the typology. For each language, the output of each word was converted to its overt form. The resulting data set contains the overt form of each word in association with its morpheme identity; this is the learning data for the language. For example, the map of language L5 appears in (61). This language, which will be discussed further in section 2.4 and in chapter 3, produces the learning data in (62). Note that although the data include the morphemic composition of the overt forms, the learner does not access that information during the phonotactic learning stage.

(61) L5

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(sY)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-Y/

(62) L5 learning data

ssY r1s1	*ssY* r1s2	*Yss* r2s1	*ssY* r2s2	*sYs* r3s1	*ssY* r3s2	*sYs* r4s1	*ssY* r4s2

After each data set was constructed, it was compared to all other stored data sets and discarded if it identically matched any of those stored. The data of L5 match those of L4, whose map appears in (63). Only the two shaded forms differ from L5, and these have the same overt form, *sYs*, as they do in L5, making L4 and L5 globally surface-ambiguous, as defined in the introduction to chapter 3. Because they cannot be distinguished by their overt forms and thus yield the same learning data, the learner is expected to learn both L4 and L5 from the data in (62), as well as any other language that is globally surface-ambiguous with them. This process is illustrated in chapter 3.

(63) L4

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(sY)]	[(Ys)s]	[(sY)s]	[(sY)s]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-Y/

The 97 languages in the Stress system typology yield 61 unique data sets like (62). The CBL processed each of these data sets in separate computer learning simulations. In every simulation, the CBL successfully learned all the globally surface-ambiguous languages associated with the data.

2.4 FUNDAMENTAL ISSUES AND PROCEDURES FOR THE COMMITMENT-BASED LEARNER

This section provides examples of the Commitment-Based Learner at key points in the learning process. These points are presented sequentially to show how the learner handles the data and language hypotheses at different stages and to highlight how commitments accumulate and inform later learning steps. An outline of the learner's actions appears in (117) in section 2.4.5.

2.4.1 THE TARGET LANGUAGE

The typology of the Stress system, described in 2.3, includes language L5, shown in (64) and generated by the stratified hierarchy in (65). L5 has lexical stress and by default, feet are iambic and the head-foot is right-aligned. The following sections illustrate the elements of the CBL as the learner attempts to learn L5. The learning data for this language are shown in (66); forms are listed in the order in which they are observed in this illustration.

(64) L5

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(sY)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-Y/

(65) FT-BIN >> PARSE-σ >> MAXSTRESS >> RMOST >> {IAMB, AFL, LMOST} >> {FNF, *LAPSE}

(66) Learning data corresponding to L5

ssY r1s1	*ssY* r1s2	*Yss* r2s1	*ssY* r2s2	*sYs* r3s1	*ssY* r3s2	*sYs* r4s1	*ssY* r4s2

2.4.2 PHONOTACTIC LEARNING

The CBL begins learning with an initial language hypothesis, Hyp0, which contains an empty support, an empty lexicon, and an empty set of structural interpretation commitments for overt forms. The initial stratified hierarchy is derived by applying BCD to the support. The result, (67), simply ranks all markedness constraints together, above the sole faithfulness constraint.

(67) {FNF, IAMB, PARSE-σ, FT-BIN, AFL, LMOST, RMOST, *LAPSE} >> MAXSTRESS

The objective of the phonotactic learning stage is to learn as much as possible about the target's constraint hierarchy from the phonotactic information of the observed overt forms; learning underlying forms will wait until the learner knows the observed forms' morphemic decomposition. The CBL's committed ranking information – the support ERCs – derives from error-detection, which requires that the learner be able to identity an error by knowing which input-output mappings the ranking should generate. Each input

will simply match the observed overt form at this stage, while the output will be some interpretation of that overt form.

Initially, error-driven learning will apply without committing to any particular interpretation for the output. The learner will wait to commit to interpretations until certain that commitments are needed to yield new information. Finding that the current ranking selects an optimum besides one of the possible identity maps of the overt form, or that it selects more than one of the possible identity maps as optimal, indicates that there is ranking information to be learned and that it is time for the overt form to receive a committed interpretation using the same branching strategy employed by the Inconsistency Detection Learner (IDL). These phonotactic learning steps are described in (68).

(68) Phonotactic learning[21]

```
Def phonotactic_learning(overt_forms, l_hyp_list)
    WHILE any l_hyp in l_hyp_list has changed
        Move l_hyp to changed_hyps list if l_hyp.hyp_change is True[22]
        Move l_hyp to lang_hyp_list if l_hyp.hyp_change is False
        FOR each overt in overt_forms
            Set l_hyp_list = changed_hyps
            Set changed_hyps to empty list
            UNTIL l_hyp_list is empty
                Remove the first l_hyp from l_hyp_list
                Set l_hyp.hyp_change to False
                Find optima for the identity input of overt given ranking in l_hyp
                IF overt has a committed interpretation in l_hyp THEN
                    IF any optimum does not match the committed interp. THEN
                        Error detected: perform error-driven learning
                        Set l_hyp.hyp_change to True
                    ENDIF
                    Add l_hyp to changed_hyps list if lang_hyp is consistent
                    Discard l_hyp if it is inconsisent
                ELSE  // No committed interpretation for overt
                    IF there is only one optimum AND it matches overt THEN
                        Add l_hyp to changed_hyps  list
                    ELSE
                        Error detected: extend branches from l_hyp
                        Add each consistent branch to changed_hyps list
                    ENDIF
                ENDIF
            ENDUNTIL
        ENDFOR
        Shift all hypotheses in changed_hyps list to l_hyp_list
    ENDWHILE
    Shift all hypotheses in l_hyp_list to lang_hyp_list
    Return lang_hyp_list
END
```

To illustrate, suppose the learner is just beginning to learn from the data in (66) and observes the overt form *ssY* first. Because Hyp0 has not yet committed to any interpretation of this form, the learner simply checks whether the initial BCD hierarchy in

[21] The actual Ruby code used to implement the CBL is included in Appendix B. Pseudocode included in the text omits code used for recordkeeping, such as maintaining the list of discarded hypotheses.

[22] All language hypotheses are initialized with this value set to True.

(67) allows for a tie or an optimum that is not an interpretation of *ssY*. The violation tableau below includes the two interpretations of the overt form, (69)a and (69)b, and the remainder of the possible optima – that is, all candidates which are not individually or collectively harmonically bounded. The degenerate interpretation in (69)b is harmonically bounded by (69)a. The remaining candidates CTie (see 1.1.1.1) with (69)a on the constraints in the first stratum, resulting in an error.

(69) Initial ranking produces an error on *ssY*

Input	Output	FNF	IAMB	PARSE-σ	FT-BIN	AFL	LMOST	RMOST	*LAPSE	MAXSTR
a. /ss-Y/	[s(sY)]	1	0	1	0	1	1	0	1	0
b.	[ss(Y)]	1	0	2	1	2	2	0	1	0
c.	[(sY)s]	1	0	1	0	0	0	1	0	1
d.	[(sY)(X)]	2	0	0	1	2	0	1	0	1
e.	[s(Ys)]	0	1	1	0	1	1	0	0	1
f.	[(Ys)s]	0	1	1	0	0	0	1	1	1
g.	[(Ys)(X)]	1	1	0	1	2	0	1	0	1
h.	[(Xs)(Y)]	1	1	0	1	2	2	0	0	0
i.	[(Y)(sX)]	2	0	0	1	1	0	2	0	1
j.	[(X)(sY)]	2	0	0	1	1	1	0	0	0
k.	[(Y)(Xs)]	1	1	0	1	1	0	2	0	1
l.	[(X)(Ys)]	1	1	0	1	1	1	0	0	1

This error indicates unresolved conflicts in the ranking, but to correctly resolve the conflicts, the learner needs to know which candidate should win. In this case, making the choice appears trivial because one candidate interpretation harmonically bounds the other; however, the learner will not reach this conclusion until separately evaluating each interpretation in its own language hypothesis. Here the CBL calls on the IDL to resolve the structural ambiguity of the overt form.

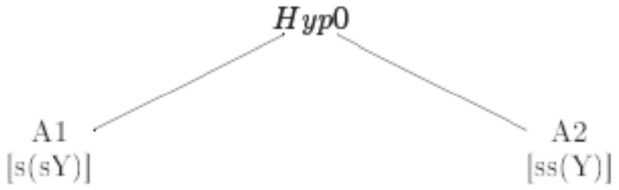
Each language hypothesis in the CBL includes a support, a set of committed structural interpretations, and a lexicon. When the IDL applies to these language hypotheses, they split into language hypothesis branches according to the method described in (70). Each branch is created as a copy of its parent hypothesis and inherits all of the committed information of the parent, including its lexicon, structural commitments, and W-L pairs, and adds a new structural commitment and any W-L pairs resulting from its addition. For this first error, each branch simply inherits the empty support and empty lexicon of Hyp0.

(70) Branch method for language hypotheses

```
Def branch(overt_form, l_hyp)
  Create empty branch_list
  Determine the interpretations of overt_form
  FOR each interpretation
        lang_hyp = copy of l_hyp
        Add a commitment to interpretation for overt in lang_hyp
        Perform error-driven learning in lang_hyp
        IF lang_hyp is consistent THEN
              Set lang_hyp.hyp_change to True
              Add lang_hyp to branch_list
        ENDIF
  ENDFOR
  Return branch_list
END
```

GEN provides two interpretations for *ssY*: [s(sY)] and [ss(Y)]. The IDL directs the initial language hypothesis Hyp0, on which the error occurred, to extend two branches to separately accommodate each interpretation. The diagram in (71) charts the learner's progress as a tree. The two new branches, A1 and A2, commit to [s(sY)] and [ss(Y)], respectively.

(71) Branching for first structural commitment



As noted in (70), in each of these branches the learner performs error-driven learning for a mapping from the input /ss-Y/ to the committed structural interpretation of that branch. A committed interpretation here entails a committed identity mapping, and the learner now will be checking that the current ranking makes this mapping the sole optimum. The learner adds W-L pairs, employing MRCD with BCD as needed until either there are no more errors, as judged by CTies, or an inconsistency is detected. When there is a choice of more than one informative loser, as in the example above where any candidate could be informative about the ranking, the learner could adopt any; in this computer simulation, it simply adopts the first one it encounters.

A1, the branch with the binary, iambic commitment, adds the three W-L pairs listed below in (72) during the course of error-driven learning and remains consistent. Applying BCD to the support produces the ranking in (73). A2 commits to the unary interpretation. It adds two W-L pairs to its support, (74), and the second reveals that its interpretation is harmonically bounded. In the support tableaux, the order in which W-L pairs are added is indicated in the leftmost column, together with a "P" to indicate that that the addition occurred during phonotactic learning.

(72) A1 support; committed to [s(sY)]

ERC#	Morph. word	Input	Winner	Loser	RMOST	FT-BIN	IAMB	LMOST	AFL	PARSE-σ	FNF	*LAPSE	MAXSTR
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	W		L		L	W	L	W
b. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]	W			L	L			L	W
c. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W				L	L	W

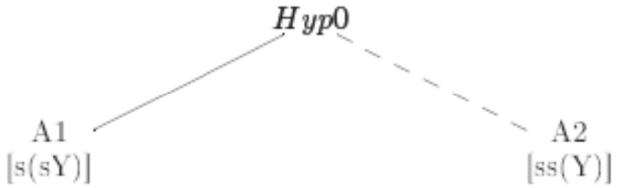
(73) {RMOST, FT-BIN, IAMB} >> {LMOST, AFL, PARSE-σ, FNF, *Lapse} >> MAXSTRESS

(74) A2 support; commitment to [ss(Y)] is inconsistent

ERC#	Morph. word	Input	Winner	Loser	RMOST	IAMB	FNF	*LAPSE	MAXSTR	LMOST	AFL	PARSE-σ	FT-BIN
a. 1P	r1s1	/ss-Y/	ss(Y)	[s(Ys)]		W	L	L	W	L	L	L	L
b. 2P	r1s1	/ss-Y/	ss(Y)	[s(sY)]						L	L	L	L

In the tree in (75), the dashed line connecting Hyp0 to A2 indicates that A2 has been rejected for inconsistency, while the solid line to A1 indicates that that language hypothesis is consistent and remains for processing with further data. Because language hypothesis A1 contains a commitment to the structural interpretation [s(sY)], it is also committed to the ranking conditions entailed by the identity mapping /ss-Y/[s(sY)], recorded in the support in (72). Whatever other commitments the learner makes for the remainder of the data, they must be consistent with the commitments made thus far in A1.

(75) One consistent branch after first commitment



The next new overt form observed is *Yss*. The learner detects an error: the current ranking of A1 does not select any interpretation of this overt form as optimal. The violation tableau in (76) reveals the problem: candidate (76)a, /Yss/[s(sY)], is not an eligible interpretation, but it incurs no violations of constraints in the first stratum. The best an eligible interpretation could do is tie, which is an error in itself, but in this case both candidate interpretations of the overt form, (76)e and (76)g, do worse, with each incurring multiple violations in the first stratum. This error indicates that the ranking of A1 has wrongly resolved conflicts, and the learner must commit to structural interpretations in order to determine how to adjust the ranking.

(76) Error in A1 for r2s1 *Yss*

Input	Output	RMOST	FT-BIN	IAMB	LMOST	AFL	PARSE-σ	FNF	*LAPSE	MAXSTR
a. /Ys-s/	[s(sY)]	0	0	0	1	1	1	1	1	1
b.	[(sY)s]	1	0	0	0	0	1	1	0	1
c.	[(sY)(X)]	1	1	0	0	2	0	2	0	1
d.	[s(Ys)]	0	0	1	1	1	1	0	0	1
e.	[(Ys)s]	1	0	1	0	0	1	0	1	0
f.	[(Ys)(X)]	1	1	1	0	2	0	1	0	0
g.	[(Y)ss]	2	1	0	0	0	2	1	1	0
h.	[(Y)(sX)]	2	1	0	0	1	0	2	0	0
i.	[(X)(sY)]	0	1	0	1	1	0	2	0	1
j.	[(Y)(Xs)]	2	1	1	0	1	0	1	0	0
k.	[(X)(Ys)]	0	1	1	1	1	0	1	0	1

A1 now extends two branches to accommodate the two interpretations of the overt form *Yss*. Branch A1B1 commits to [(Ys)s] and inherits from A1 its commitment to [s(sY)] and its support. Candidate (76)a is adopted as the informative loser for a new W-L pair, included as (77)d below. The ranking in (78) derived by BCD makes both committed interpretations optimal.

(77) A1B1 support; committed to [s(sY)] and [(Ys)s]

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	LMOST	RMOST	AFL	FNF	IAMB	*LAPSE
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	L	W		W		L
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W				L	W	L
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	L	W	L			L
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	W	L	W	W	L	

(78) FT-BIN >> PARSE-σ >> MAXSTRESS >> {LMOST, RMOST, AFL, FNF, IAMB, *LAPSE}

Branch A1B2 commits to the unary interpretation [(Y)ss] and inherits the commitment and support of A1 also. Like A1B1, it adopts candidate (76)a as a loser for the new W-L pair 4P, included as (79)d in the support below. The resulting ranking in (80) makes both committed interpretations optimal.

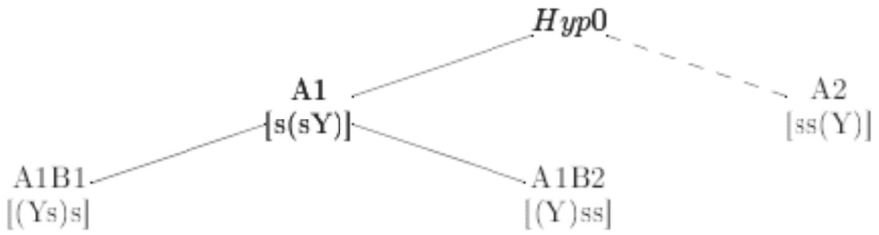
(79) A1B2 support; committed to [s(sY)] and [(Y)ss]

ERC#	Morph. word	Input	Winner	Loser	IAMB	FNF	MAXSTR	LMOST	RMOST	AFL	PARSE-σ	FT-BIN	*LAPSE
a. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]	W	L	W						L
b. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]		W	W	L	W		L	W	L
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	L	W	L			L
d. 4P	r2s1	/Ys-s/	[(Y)ss]	[s(sY)]			W	W	L	W	L	L	

(80) IAMB >> FNF >> MAXSTRESS >> {LMOST, RMOST, AFL, PARSE-σ, FT-BIN, *LAPSE}

Both language hypotheses are consistent, and the learner will evaluate the next observed form against the ranking of each. The new branches are included in the diagram in (81).

(81) Branching for second structural commitment



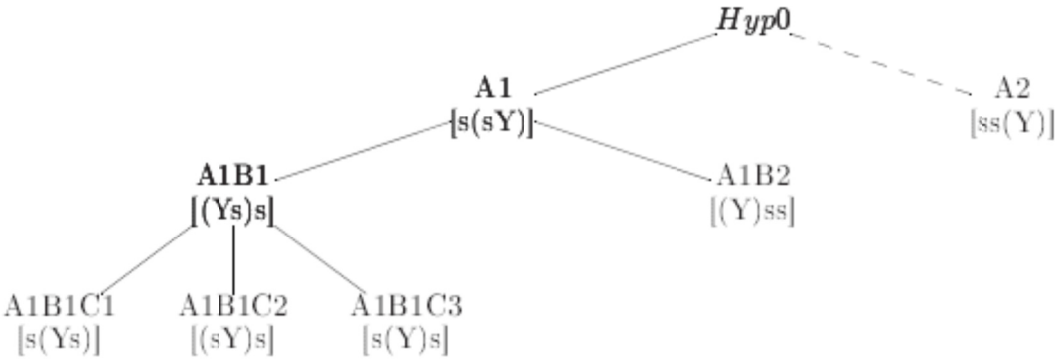
The third and final new overt form observed is *sYs*. The learner checks this form for errors against the rankings of both remaining language hypotheses, but detects an error only in A1B1. The violation tableau in (82) includes all potentially optimal candidates. The iambic interpretation in (82)b ties with the trochaic interpretation in (82)d through the top three strata, but they conflict in the shaded bottom stratum. LMOST, AFL and IAMB prefer the iambic interpretation, RMOST and FNF prefer the trochaic, and the candidates tie on *LAPSE. Although the trochaic interpretation [s(Ys)] incurs more total violations in the bottom stratum, by the CTie criterion it ties with [(sY)s].

(82) Error in A1B1 for *sYs*

Input	Output	FT-BIN	PARSE-σ	MAXSTR	LMOST	RMOST	AFL	FNF	IAMB	*LAPSE
a. /sYs/	[s(sY)]	0	1	1	1	0	1	1	0	1
b.	[(sY)s]	0	1	0	0	1	0	1	0	0
c.	[(sY)(X)]	1	0	0	0	1	2	2	0	0
d.	[s(Ys)]	0	1	0	1	0	1	0	1	0
e.	[(Ys)s]	0	1	1	0	1	0	0	1	1
f.	[(Ys)(X)]	1	0	1	0	1	2	1	1	0
g.	[(Y)(sX)]	1	0	1	0	2	1	2	0	0
h.	[(X)(sY)]	1	0	1	1	0	1	2	0	0
i.	[(Y)(Xs)]	1	0	1	0	2	1	1	1	0
j.	[(X)(Ys)]	1	0	0	1	0	1	1	1	0

Based on the error above, the learner must extend a branch from A1B1 for each of the three interpretations of *sYs,* producing the tree in (83). The outcome of each branch is described in section 3.1; however, the remainder of this chapter concerns only the branch that commits to the trochaic interpretation [s(Ys)]. The support tableau for this branch, A1B1C1, is provided in (84). It adds W-L pair 5P to the inherited support from A1B1. The updated ranking appears in (85).

(83) Branching from A1B1 for third commitment



(84) A1B1C1 support after committing to [s(Ys)]

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	FNF	*LAPSE	LMOST	AFL	IAMB
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	W	L	L		
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W		L	L			W
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W		L	L	L	
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	W		W	W	L
e. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W	W		L	L	L

(85) FT-BIN >> PARSE-σ >> MAXSTRESS >> {RMOST, FNF, *LAPSE} >> {LMOST, AFL, IAMB}

Language hypothesis A1B2 does not have to extend branches at this time because it processes *sYs* without error. The violation tableau in (86) shows that there is only one optimum for /sYs/ given the current ranking, and that optimum includes the trochaic interpretation of the observed form. The shaded cells indicate that this candidate, in (86)d, is the most harmonic because it incurs no violations of the constraints IAMB and FNF in the highest strata. Phonotactic learning will end without language hypothesis A1B2 ever making a structural interpretation commitment for *sYs*.

(86) No error in A1B2 for *sYs*

Input	Output	IAMB	FNF	MAXSTR	LMOST	RMOST	AFL	PARSE-σ	FT-BIN	*LAPSE
a. /sYs/	[s(sY)]	1	1	1	0	1	1	0	1	0
b.	[(sY)s]	1	0	1	1	0	0	0	0	0
c.	[(sY)(X)]	2	0	0	1	0	0	1	2	0
d.	[s(Ys)]	0	0	1	0	0	1	0	1	1
e.	[(Ys)s]	0	1	1	1	1	0	0	0	1
f.	[(Ys)(X)]	1	1	0	1	0	0	1	2	1
g.	[(Y)(sX)]	2	1	0	2	0	0	1	1	0
h.	[(X)(sY)]	2	1	0	0	0	1	1	1	0
i.	[(Y)(Xs)]	1	1	0	2	0	0	1	1	1
j.	[(X)(Ys)]	1	0	0	0	0	1	1	1	1

Constructing simultaneous language hypotheses to accommodate each possible interpretation of an overt form guarantees that the learner will construct the correct structural interpretations of the target language; the only question is how many other language hypotheses the learner will have to construct and consider also. Applying inconsistency detection reduces the number of language hypotheses to evaluate for

further learning and prevents the learner from wasting effort on combinations of interpretations that cannot be correct. Although there are twelve logically possible hypotheses containing interpretations of these three overt forms, the learner creates just five during phonotactic learning: A1B1C1, A1B1C2, A1B1C3, A1B2, and A2. Eliminating A2 early on enables the learner to avoid considering any of the six possible branches that contain its committed interpretation, [ss(Y)].

Error-driven learning helps to limit the number of simultaneous hypotheses as well by ensuring that hypotheses branch only when necessary. A1B2 does not immediately branch when the learner observes the third overt form, *sYs*, because its current ranking makes one of the valid interpretations the sole optimum. Until branching is required, the learner can evaluate all data against the commitments made in A1B2, and will be able to set a number of features before branching for the third time.

2.4.3 IS LEARNING COMPLETE?

For every data set, the learner will derive as much phonotactic ranking information as possible using the error-driven learning procedures described in section 2.4.2. This learning stage is complete when the rankings of the consistent language hypotheses can process all overt forms without error. For some data sets, this will be enough to account for the target language, and the learner will not have to set the underlying values of features. For languages with predictable stress, for example, no features need be set at all as the appropriate ranking will ensure that, whatever the input, the correct output is optimal.

After ending the phonotactic learning stage, then, the learner pauses to evaluate whether each of the remaining consistent language hypotheses fully account for the data. There are two criteria the learner checks. First, does any word fail error detection in this hypothesis? Second, for any overt form still without a committed interpretation, would making a commitment lead to new information? If the answer is "yes" to either question, there is more to learn in this language hypothesis.

All targets in this simulation have an output-driven map, a fact which allows the learner to easily determine whether any word in a language hypothesis fails error detection. Following Tesar (to appear), for each word the learner constructs an input in which each unset feature is assigned the opposite of its surface value and checks the resultant mapping for errors against the ranking derived by BCD. Just as in section 2.4.2, if the word has a committed interpretation, an error is detected if the ranking does not select this interpretation as the sole optimum.

It is possible for a language hypothesis to successfully derive the grammar of a target language without making a structural interpretation commitment for every overt form observed. To evaluate the second criterion of language hypothesis completion, then, it is necessary to have an error detection procedure for uncommitted overt forms. In the CBL, this procedure has several parts. First, the identity mapping of the overt form cannot yield an error under the current ranking derived by BCD; there must be exactly one optimal candidate whose output is an interpretation of the overt form. This part is the standard error detection procedure illustrated in (69) for the uncommitted overt form *ssY*. Second, the ranking must map the maximal mismatch input, with all unset features set to

mismatch their surface forms, to exactly one output with the same overt form as observed. This is the counterpart to the error detection procedure for words with committed interpretations. Finally, the optima selected in these previous tests must be identical.

Whether or not it has a committed interpretation, if a word fails error detection, then the current grammar is missing some crucial piece of information, whether about the ranking or the value of an unset feature in this word. If all words in a language hypothesis pass error detection, then the CBL judges that the language hypothesis is complete. For the CBL, learning is complete when, for each consistent language hypothesis remaining, every word passes error detection.

To continue the example from the previous section, the learner performs error detection on the words in A1B1C1 and determines that this language hypothesis is incomplete. Note that because the learner has not observed the words during the non-phonotactic stage, morphemic identities are as yet unknown; however, all that matters for error detection at this point is the identity mapping. Suppose, then, that the learner performs the error detection test on a word with the mapping /ssY/[s(sY)]. This word has a committed interpretation, [s(sY)], and all three stress features are unset; therefore, the error detection test evaluates the candidate /YY-s/[s(sY)] against the current ranking, repeated in (87).

(87) FT-BIN >> PARSE-σ >> MAXSTRESS >> {RMOST, FNF, *LAPSE} >> {LMOST, AFL, IAMB}

The violation tableau below reveals an error. The error detection candidate, (88)a, ties

with the candidates in (88)b,d,e on the two highest ranked constraints and incurs more violations of MAXSTRESS in the next highest stratum. The pertinent violations of these candidates are shaded.

(88) A1B1C1 – /ssY/[(s(sY)] fails error detection

Input	Output	FT-BIN	PARSE-σ	MAXSTR	RMOST	FNF	*LAPSE	LMOST	AFL	IAMB
a. /YY-s/	[s(sY)]	0	1	2	0	1	1	1	1	0
b.	[(sY)s]	0	1	1	1	1	0	0	0	0
c.	[(sY)(X)]	1	0	1	1	2	0	0	2	0
d.	[s(Ys)]	0	1	1	0	0	0	1	1	1
e.	[(Ys)s]	0	1	1	1	0	1	0	0	1
f.	[(Ys)(X)]	1	0	1	1	1	0	0	2	1
g.	[(Y)(sX)]	1	0	1	2	2	0	0	1	0
h.	[(X)(sY)]	1	0	2	0	2	0	1	1	0
i.	[(Y)(Xs)]	1	0	1	2	1	0	0	1	1
j.	[(X)(Ys)]	1	0	1	0	1	0	1	1	1

In addition to demonstrating that a language hypothesis as a whole lacks some crucial bit of information, performing error detection tests over the entire data set indicates for which words in particular the language hypothesis requires more information. Keeping track of these words allows the learner to focus effort where it is needed and where it is likely to be most fruitful. Note that as the language hypothesis grows with new committed information, which words pass error detection will change as well. Therefore, it is important to repeat the error detection tests over the entire data set – including for words which have previously passed error detection – to evaluate whether learning is complete.

On the basis of just one word's failure to pass error detection, the learner can know that the current grammar is incomplete. What happens next depends on what procedures the learner has just completed and whether they were successful. In this example for A1B1C1, the learner has successfully completed the phonotactic learning stage and the error here means that the learner will next attempt to set the features of individual words. If some features are set, yet after one pass through the learning data some word still fails error detection, the learner will attempt another round of learning from individual words. But if no features can be set from single forms, the learner will appeal to contrast pairs in the next round of learning. Therefore, after every pass through the learning data hereafter, the learner will perform error detection and determine whether to continue learning and by what methods.

2.4.4 NON-PHONOTACTIC LEARNING

During the phonotactic learning stage, the learner commits to structural interpretations and to input-output mappings – as W-L pairs in the support – to learn ranking information. While this next stage focuses on learning underlying forms, the learner can still make any kind of commitment. In fact, reliance on an inconsistency detection strategy to set features compels the learner to assign structural interpretations to uncommitted overt forms if those forms are to be used to set features. The CBL incorporates the Output-Driven Learner (ODL) for learning underlying forms in languages with output-driven maps. As explained in 1.4.1, the ODL sets features by evaluating test candidates in which one unset feature at a time is set to mismatch its corresponding output value. If the resulting test candidate is inconsistent with the current support, then the underlying value of the feature must match its surface value. Using this

method, the ODL can more efficiently set features from single words and contrast pairs than a learner that must evaluate every local lexical hypothesis for consistency, such as the learners of Merchant (2008), and Merchant and Tesar (2008).

This section describes how the CBL increases both ranking and lexical commitments during the non-phonotactic learning stage in language hypothesis A1B1C1 from the preceding section. This branch already includes committed interpretations for each of the three overt forms in the data set for L5. The learning data are repeated in (89) along with their interpretations in this language hypothesis; all words currently fail to pass the error detection. The current support for A1B1C1 and its ranking by BCD are repeated in (90) and (91).

(89) Learning data for L5, showing committed interpretations

[s(sY)] r1s1	[s(sY)] r1s2	[(Ys)s] r2s1	[s(sY)] r2s2	[s(Ys)] r3s1	[s(sY)] r3s2	[s(Ys)] r4s1	[s(sY)] r4s2

(90) A1B1C support

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	FNF	*LAPSE	LMOST	AFL	IAMB
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	W	L	L		
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W		L	L			W
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W		L	L	L	
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	W		W	W	L
e. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W	W		L	L	L

(91) FT-BIN >> PARSE-σ >> MAXSTRESS >> {RMOST, FNF, *LAPSE} >> {LMOST, AFL, IAMB}

Section 2.4.4.1 describes how the learner sets features using a contrast pair when single forms prove uninformative. In section 2.4.4.2, the learner uses the new lexical commitments to derive new ranking information, and section 2.4.4.3 shows how the commitments of the previous two sections finally enable the learner to set features from single forms.

2.4.4.1 Setting features by contrast pair

Both single words and contrast pairs can be used to set features using the ODL. For A1B1C1, however, single words fail to set any features. To understand why, suppose the learner attempts to set features in r1s1 [s(sY)] using the test candidates shown below. As no features have yet been set in the lexicon, there is a test candidate for each of the three stress features in the word. The syllable whose stress value is being tested in each candidate is indicated by outlining.

(92) Test candidates for r1s1

a. /**Y**s-Y/ → [s(sY)]
b. /s**Y**-Y/ → [s(sY)]
c. /ss-**s**/ → [s(sY)]

Again, this language hypothesis corresponds to target L5, and therefore its support is consistent with L5, whose map appears below. The reason why the test candidates for r1s1 cannot set a feature should be clear from this map: each is a mapping in L5 and cannot be inconsistent with the support. The pertinent mappings are shaded below.

(93) L5

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(sY)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-Y/

Because r1s2, r2s2, r3s2, and r4s2 have the same interpretation as r1s1, and therefore the same test candidates at this point as well, they too are unable to set features. Similarly, two of the three test candidates for r3s1 – and thus r4s1, which shares the interpretation [s(Ys)] – are mappings in L5 and must be consistent with the support; these are (94)a and (94)b below. The third test candidate for r3s1, (94)c, as well as those for r2s1 in (95), are not mappings in the target language, and their consistency with the support must be explained by a lack of crucial ranking information. Contrast pairs prove vital for uncovering this information. Because the members of a contrast pair must be consistent with each other as well as with the support, processing in pairs provides more opportunity to detect an inconsistency and therefore to set a feature.

(94) Test candidates for r3s1

a. /**Y**Y-s/ → [s(Ys)]
b. /s**Y**-s/ → [s(Ys)]
c. /sY-**Y**/ → [s(Ys)]

(95) Test candidates for r2s1

a. /**s**s-s/ → [(Ys)s]
b. /Y**Y**-s/ → [(Ys)s]
c. /Ys-**Y**/ → [(Ys)s]

The informative contrast pair identified by the learner is r1s1 [s(sY)] and r2s1 [(Ys)s]. There are five unset features – in r1, r2, and s1 – and because the surface value of s1 alternates, the learner must evaluate a total of 10 different test candidates. The values assigned to each feature to construct the candidates are listed in the chart below, together with the outcome of inconsistency detection for each pair of candidates evaluated.

(96) Test candidates for contrast pair r1s1 [s(sY)], r2s1 [(Ys)s].

Feature value Disparity		**Input**		**Consistent?**
a.	r1 σ1 +stress	r1s1	/**Y**s-Y/	No *same input, different output*
		r2s1	/Ys-Y/	
		r1s1	/**Y**s-s/	
		r2s1	/Ys-s/	
b.	r1 σ2 +stress	r1s1	/s**Y**-Y/	No *inconsistent with support*
		r2s1	/Ys-Y/	
		r1s1	/s**Y**-s/	
		r2s1	/Ys-s/	
c.	r2 σ1 –stress	r1s1	/ss-Y/	No *same input, different output*
		r2s1	/**s**s-Y/	
		r1s1	/ss-s/	
		r2s1	/**s**s-s/	
d.	r2 σ2 +stress	r1s1	/ss-Y/	Yes
		r2s1	/Y**Y**-Y/	
		r1s1	/ss-s/	
		r2s1	/Y**Y** -s/	
e.	s1 stress alternates	r1s1	/ss-**Y**/	Yes
		r2s1	/Ys-**Y**/	
		r1s1	/ss-**s**/	
		r2s1	/Ys-**s**/	

The contrast pair produces several inconsistencies, two of which are quite easy to see. The pairs in (96)a test the value of stress in the first syllable of r1 and are inconsistent because they involve the same input mapping to different outputs. This inconsistency allows the learner to set r1's first syllable to –stress. For the same reason, the pairs in

(96)c, which test the value of stress in the first syllable of r2, are inconsistent and allow the learner to set that syllable's feature to +stress.

A third inconsistency arises from the pairs in (96)b, which test the stress value of the second syllable of r1 by setting it to +stress. Tableau (97) includes the support from (90) as well as, in (97)f,g, W-L pairs constructed for this contrast pair (labeled "test" as these are test mappings for the contrast pair). FT-BIN, PARSE-σ and MAXSTRESS can be ranked in the first three strata, but none of the remaining unranked constraints prefer only winners. For W-L pair 5 of the support, given in (97)e, only FNF and RMOST prefer the winner, but FNF prefers the loser to the test candidate for r1s1 in (97)f, and RMOST prefers the loser to r2s1 in (97)g. Setting r1 to /sY/ is therefore inconsistent with the support.

(97) A1B1C is inconsistent with r1s1 /**s**Y-Y/[s(sY)]

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	FNF	*LAPSE	LMOST	AFL	IAMB
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	W	L	L		
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W		L	L			W
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W		L	L	L	
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	W		W	W	L
e. 5	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W	W		L	L	L
f. test	r1s1	/sY-Y/	[s(sY)]	[s(Ys)]					L	L			W
g. test	r2s1	/Ys-Y/	[(Ys)s]	[s(sY)]				L	W		W	W	L

The other pair in (96)b is also inconsistent. This pair tests the stress value of the second syllable of r1 as well, but sets s1 to –stress, its surface value in r2s1. The resulting test candidate for r1s1, /**s**Y-s/[s(sY)], now includes two disparities from its output: one in

r1, and one in s1. The crucial ERC is in (98)f. For this W-L pair, MAXSTRESS prefers the loser, leaving no way to rank the remaining constraints after ranking FT-BIN and PARSE-σ: all the remaining constraints prefer the loser in at least one W-L pair. Based on these inconsistencies, the updated lexicon for A1B1C1 in (99) now includes three set features.

(98) A1B1C1 is inconsistent with r1s1 /**s**Y-s/[s(sY)]

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	FNF	*LAPSE	LMOST	AFL	IAMB
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	W	L	L		
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W		L	L			W
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W		L	L	L	
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	W		W	W	L
e. 5	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W	W		L	L	L
f. test	r1s1	/sY-s/	[s(sY)]	[s(Ys)]			L	W		L	L	L	

(99) A1B1C1 lexicon

r1	r2	r3	r4	s1	s2
/ss/	/Y?/	/??/	/??/	/-?/	/-?/

As explained earlier in this section, the learner could not set any features in this language hypothesis using single forms alone. However, because contrast pairs generally require more effort than single forms in terms of the number of forms that must be evaluated, the learner appeals to them only after the latest pass through the data fails to set any features from single forms. For single forms the learner constructs one test candidate for each unset feature. The maximum number of tests therefore equals the number of unset features in the word. In contrast pairs, an unset feature can alternate, as it does for s1 in r1s1 and r2s1 here, and both values must be evaluated in each test. Each

alternating unset binary feature therefore doubles the number of tests the learner must evaluate. For r1s1 and r2s1, there are five unset features, and one alternates, causing the learner to evaluate the ten tests shown in (96).

Just how much more effort the contrast pair requires depends on how many of its unset features alternate in value. In the Stress system, where all words contain just three features, at most the learner would have to evaluate 10 tests for a contrast pair, as in (96). In this worst-case scenario the environment morpheme is a monosyllabic suffix and the contrast morphemes are disyllabic roots. This pair has a total of five unset features, the feature of the environment morpheme alternates, so that the learner must evaluate 10 tests. The CBL does not compare all potentially informative contrast pairs to determine which involves the fewest tests, and instead simply attempts to learn from single forms first before evaluating any contrast pair.

In total, the contrast pair r1s1 [s(sY)] and r2s1 [(Ys)s] enables the learner to set three features that could not be set by evaluating either form individually. As 2.4.4.3 shows, the information learned from this pair allows the learner to use r2s1 to set the remaining stress features in that word.

2.4.4.2 Learning ranking information from set features

The task of learning ranking information occurs continually, in both stages of learning. During the phonotactic learning stage, what committed ranking information the support contains determines whether a language hypothesis survives for another round of learning or is rejected due to inconsistency. While this role for the support continues to be

vital, in the non-phonotactic learning stage the support also becomes essential for setting features. It is imperative, then, that the support contain crucial ranking information, balanced as always with the need to gather that information efficiently. The low-faithfulness ranking bias of BCD enables the learner to detect errors and recover phonotactic ranking information even when no features have been set; however, set features themselves are a potential source of new ranking information, if they ever surface unfaithfully. To draw out informative errors from set features the learner benefits from employing a low-markedness ranking bias, following Tesar (to appear). This section illustrates how one of the features set by the contrast pair in 2.4.4.1 contributes new ranking information to the language hypothesis.

The current support for A1B1C1 from (90) is repeated below in (100). The tableau shows the support after applying BCD, whose low-faithfulness bias causes MAXSTRESS to occupy the third stratum, despite it preferring only winners. This ranking is given in (101).

(100) A1B1C1 support – BCD ranking

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	FNF	*LAPSE	LMOST	AFL	IAMB
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	W	L	L		
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W		L	L			W
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W		L	L	L	
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	W		W	W	L
e. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W	W		L	L	L

(101) FT-BIN >> PARSE-σ >> MAXSTRESS >> {RMOST, FNF, *LAPSE} >> {LMOST, AFL, IAMB}

The BCD ranking is extremely useful for deriving phonotactic ranking information, but a different bias can uncover different errors. In an attempt to learn non-phonotactic ranking information, the learner applies a low-markedness bias to the support to produce the ranking in (102), in which MAXSTRESS now occupies the highest stratum. Unfaithful mappings of set features are checked for errors against this ranking.

(102) A1B1C1 low-markedness ranking

MAXSTRESS >> {FT-BIN, PARSE-σ, RMOST, FNF, *LAPSE} >> {LMOST, IAMB, AFL}

Usually an unfaithful mapping of a set feature means that a markedness constraint crucially dominates some faithfulness constraint(s). For the Stress system, this would mean that one of the eight markedness constraints must dominate MAXSTRESS. The low-markedness ranking allows MAXSTRESS to be ranked as high as possible and increases the likelihood that it will conflict with a markedness constraint to produce an error. Learning this non-phonotactic ranking information is the intention of the low-markedness bias.

However, in the Stress system an unfaithful mapping can also arise as a consequence of the restrictions on GEN. While it is possible for every feature in a word to be underlyingly +stress, only one can surface with primary stress; the others will each incur one violation of MAXSTRESS. Therefore, the low-markedness ranking may not produce an informative error about the relative ranking of MAXSTRESS and a markedness constraint, but it nonetheless could prove informative for sorting out relations among the markedness

constraints themselves. If each candidate must incur at least one violation of MAXSTRESS, then it falls to the markedness constraints to determine which feature surfaces faithfully. With the majority of the markedness constraints clumped into the second stratum of the ranking in (102), the conditions are set to produce errors from CTies. This is just what happens when the learner checks for errors in A1B1C1 using the ranking in (102).

The map of A1B1C1 is shown in (103) with as-yet unset features indicated by "?". From the contrast pair used in section 2.4.4.1, the learner has set r1 to /ss/ and r2 to /Y?/. Only an underlyingly +stress feature in the Stress system can surface unfaithfully, and the +stress feature in r2 surfaces unfaithfully in r2s2 [s(sY)], shaded. The learner will check this form for errors against the low-markedness ranking.

(103) A1B1C1

r1 = /ss/	r2 = /Y?/	r3 = /??/	r4 = /??/	
[s(sY)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s1 = /-?/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-?/

The learner checks for errors on r2s2 using the mapping /Ys-Y/[s(sY)], whose input includes the value of the set feature in r2 and matches the remaining unset feature values to their surface forms. Because the input has two underlyingly stressed syllables, every candidate will violate MAXSTRESS at least once. The violation tableau below includes just the candidates that incur a single MAXSTRESS violation. The shaded cells indicate the candidates that are most harmonic through the second stratum; observe that the desired winner, (104)a, is in a CTie with two other candidates. The learner adopts the first of these as a loser and adds W-L pair 6 to the support in (105). Applying the low-markedness bias to the updated support produces the ranking in (106).

(104) Error in A1B1C1 on r2s2 /Ys-Y/[s(sY)] using ranking (102)

Input	Output	MAXSTR	FNF	PARSE-σ	FT-BIN	RMOST	*LAPSE	IAMB	AFL	LMOST
a. /Ys-Y/	[s(sY)]	1	1	1	0	0	1	0	1	1
b.	[(Ys)s]	1	0	1	0	1	1	1	0	0
c.	[(Ys)(X)]	1	1	0	1	1	0	1	2	0
d.	[(Xs)(Y)]	1	1	0	1	0	0	1	2	2
e.	[(Y)ss]	1	1	2	1	2	1	0	0	0
f.	[(Y)(sX)]	1	2	0	1	2	0	0	1	0
g.	[(X)(sY)]	1	2	0	1	0	0	0	1	1
h.	[(Y)(Xs)]	1	1	0	1	2	0	1	1	0

(105) A1B1C updated support – low-markedness ranking

ERC#	Morph. word	Input	Winner	Loser	MAXSTR	PARSE-σ	FT-BIN	RMOST	*LAPSE	FNF	IAMB	AFL	LMOST
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	L	W			L
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]	W				L	L	W		
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]	W			W	L			L	L
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]	W			L		W	L	W	W
e. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W		W	L	L	L
f. 6	r2s2	/Ys-Y/	[s(sY)]	[(Ys)s]				W		L	W	L	L

(106) A1B1C1 updated low-markedness ranking

MAXSTRESS >> { PARSE-σ, FT-BIN, RMOST, *LAPSE} >> {FNF, IAMB, AFL, LMOST}

The learner now reevaluates r2s2 /Ys-Y/[s(sY)] and detects a second error under the updated ranking from (106). As before, the violation tableau below includes just the candidates with the minimal MAXSTRESS violation, and the shaded cells emphasize the violations in the second stratum for /Ys-Y/[s(sY)] and the most harmonic competitors.

There are two informative losers that CTie with the desired optimum, and the learner adopts (107)g for the seventh W-L pair. The resulting support and low-markedness ranking appear in (108) and (109).

(107) Second error in A1B1C1 on r2s2 /Ys-Y/[s(sY)] using low-markedness bias

Input	Output	MAXSTR	PARSE-σ	FT-BIN	RMOST	*LAPSE	FNF	IAMB	AFL	LMOST
a. /Ys-Y/	[s(sY)]	1	1	0	0	1	1	0	1	1
b.	[(Ys)s]	1	1	0	1	1	0	1	0	0
c.	[(Ys)(X)]	1	0	1	1	0	1	1	2	0
d.	[(Xs)(Y)]	1	0	1	0	0	1	1	2	2
e.	[(Y)ss]	1	2	1	2	1	1	0	0	0
f.	[(Y)(sX)]	1	0	1	2	0	2	0	1	0
g.	[(X)(sY)]	1	0	1	0	0	2	0	1	1
h.	[(Y)(Xs)]	1	0	1	2	0	1	1	1	0

(108) A1B1C1 updated support – low- markedness bias

ERC#	Morph. word	Input	Winner	Loser	MAXSTR	RMOST	FT-BIN	PARSE-σ	*LAPSE	LMOST	AFL	FNF	IAMB
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	**W**	**W**	**W**	**L**	**L**	**L**		**W**	
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]	**W**				**L**			**L**	**W**
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]	**W**	**W**			**L**	**L**	**L**		
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]	**W**	**L**				**W**	**W**	**W**	**L**
e. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]		**W**				**L**	**L**	**W**	**L**
f. 6	r2s2	/Ys-Y/	[s(sY)]	[(Ys)s]		**W**				**L**	**L**	**L**	**W**
g. 7	r2s2	/Ys-Y/	[s(sY)]	[(X)(sY)]			**W**	**L**	**L**			**W**	

(109) A1B1C1 updated low-markedness ranking

MAXSTRESS >> {RMOST, FT-BIN} >> {PARSE-σ, *LAPSE, LMOST, AFL, FNF, IAMB}

No further errors are detected on r2s2 using the new ranking above. If there were any other unfaithful mappings of set features, the learner would continue to perform error-driven learning using the low-markedness ranking. Once all unfaithful mappings are processed without error, the learner returns to error-driven learning using the ranking derived by BCD. The updated support for A1B1C1 derived by applying BCD is given in (110), with ranking in (111).

(110) A1B1C1 current support – BCD ranking

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	*LAPSE	LMOST	AFL	FNF	IAMB
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	L	L		W	
b. 7	r2s2	/Ys-Y/	[s(sY)]	[(X)(sY)]	W	L			L			W	
c. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W		L			L	W
d. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W	L	L	L		
e. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L		W	W	W	L
f. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W		L	L	W	L
g. 6	r2s2	/Ys-Y/	[s(sY)]	[(Ys)s]				W		L	L	L	W

(111) A1B1C1 updated BCD ranking
FT-BIN >> PARSE-σ >> MAXSTRESS >> {RMOST, *LAPSE} >> {LMOST, AFL, FNF, IAMB}

The low-markedness bias has served here to flesh out the relationships between the markedness constraints. This round of error-driven learning focusing on unfaithful mappings of set features produces two new W-L pairs. Based on these pairs, the learner now knows that RMOST must dominate all of LMOST, AFL, FNF, and IAMB. Before, the support had only revealed the less-informative disjunction that either RMOST or FNF must dominate the other three.

2.4.4.3 Setting features by single form

The learner can often set some, if not all, features from single forms before ever appealing to a contrast pair. A1B1C1 is an unusual exception, as it requires a contrast pair to set some features initially; thereafter, all remaining features that must be set can be set from single forms. Having seen already how features can be learned from contrast pairs, understanding how the learner uses single forms for that purpose should be simple. The process follows the same principles as for using contrast pairs: the learner varies the value of one unset feature at a time and checks whether the resulting mapping is inconsistent with the support. If so, the feature must be set in the lexicon to match its surface value. This section shows how the learner sets the remaining unset features in r2s1, one of the members of the contrast pair used earlier.

The current support for A1B1C1 remains as in (110). The current lexicon, in (112), includes only the set features learned from the contrast pair in 2.4.4.1.

(112) A1B1C lexicon

r1	r2	r3	r4	s1	s2
/ss/	/Y?/	/??/	/??/	/-?/	/-?/

The learner may now attempt to learn from r2s1 [(Ys)s], which has two unset features. The stress feature of the second syllable of r2 is unset, and its surface value is –stress. To test the value of this feature, the learner constructs a candidate in which that feature is set to +stress. The test candidate, /Y**Y**-s/[(Ys)s] in (113)h, is inconsistent with A1B1C1. The problematic ranking conditions appear in (113)f-h.

(113) A1B1C1 is inconsistent when r2 = /YY/

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	*LAPSE	LMOST	AFL	FNF	IAMB
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	L	L		W	
b. 7	r2s2	/Ys-Y/	[s(sY)]	[(X)(sY)]	W	L			L			W	
c. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W		L			L	W
d. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W	L	L	L		
e. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L		W	W	W	L
f. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W		L	L	W	L
g. 6	r2s2	/Ys-Y/	[s(sY)]	[(Ys)s]				W		L	L	L	W
h. test	r2s1	/YY-s/	[(Ys)s]	[s(Ys)]				L	L	W	W		

After FT-BIN, PARSE-σ and MAXSTRESS are ranked, the ERCs in (113)f-h must explain the rankings of the other constraints. W-L pairs 5P and 6 contradict each other in all but RMOST, which prefers the winner in both pairs; however, RMOST prefers the loser in (113)h. With all unranked constraints preferring losers for at least one pair, the ranking conditions are inconsistent. The learner updates the lexicon, (114), so that r2 is set to /Ys/.

(114) A1B1C1 lexicon updated for r2

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/??/	/??/	/-?/	/-?/

The process repeats for the unset feature in s1, setting it to +stress in the test candidate: /Ys-**Y**/[(Ys)s]. The tableau in (115) shows that the test candidate is inconsistent because its ranking conditions contradict those of W-L pair 6. The learner can now update the lexicon as in (116), with s1 set to /-s/.

(115) A1B1C1 is inconsistent when s1 = /-Y/

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	*LAPSE	LMOST	AFL	FNF	IAMB
a. 6	r2s2	/Ys-Y/	[s(sY)]	[(Ys)s]				W		L	L	L	W
b. test	r2s1	/Ys-Y/	[(Ys)s]	[s(sY)]				L		W	W	W	L

(116) A1B1C1 lexicon updated for s1

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/??/	/??/	/-s/	/-?/

2.4.5 PUTTING THE PIECES TOGETHER

The preceding sections have provided a close look at each component of the CBL, from its initial passes through the data during phonotactic learning to how it sets features and how it judges whether learning is complete. These sections have provided a glimpse into how these components can fit together, but not yet a full outline of how the learner uses and re-uses them throughout the course of learning; this is now provided in (117).

(117) Preliminary outline of the Commitment-Based Learner

Within each language hypothesis, beginning with Hyp0, and for each observed form:

Phonotactic Learning

1. Check for errors
    a. If the form lacks a committed structural interpretation and yields an error, apply the IDL to extend branches. Repeat step 1 for each branch.
    b. If a form has a committed structural interpretation and produces an error, perform error-driven learning. Repeat step 1.
    c. If the form does not produce an error, process the next overt form.
2. Phonotactic learning ends when no errors are detected on any observed forms.

Non-phonotactic Learning – first pass through data

3. Perform error-driven learning over all known words.
    a. Reject hypothesis if it is inconsistent
4. Does the form have a committed interpretation?
    a. Yes – apply the ODL to set features from the single form.
        i. If features are set, seek non-phonotactic ranking information from unfaithful mappings using the low-markedness ranking bias.
        ii. If no features are set, observe the next form.
    b. No – perform error detection on the overt form.
        a. If the overt form passes error detection, observe the next form. Go to step 3.
        b. If it does not pass error detection, apply the IDL to assign interpretations and extend branches.
            i. Continue learning in the resulting branches, beginning with the first observed form in the data set. Go to step 3.
5. Perform error detection on the list of known words.
    a. If all words pass error detection, this language hypothesis is complete.
        i. Are all consistent language hypotheses are complete?
            1. Yes – stop. Learning is complete.
            2. No – continue learning in the incomplete language hypotheses.
    b. If some words fail error detection, go to step 6.

Non-phonotactic Learning – after the first pass through the data

6. Were any features set by single-form learning in the last pass through the data?
    a. Yes – repeat steps 3-5 for each word that currently fails error detection.
    b. No – apply the ODL to set features from contrast pairs in the list of known words. Go to step 7.
7. Were any features set by contrast pairs in this pass?
    a. Yes – repeat steps 3-5 for each word that currently fails error detection.
    b. No – wait for new information in the language hypothesis and repeat steps 3-5 for each word that currently fails error detection.. (See chapter 4).

2.5 CONCLUSION

The mutual dependency between hidden structures can provide valuable information about the target grammar. The Commitment-based Learner (CBL) exploits that potential and uses commitments to some hidden structures to illuminate others. The CBL lays a foundation of knowledge during phonotactic learning with commitments to structural interpretations and their entailed ranking conditions, then builds on that foundation during the non-phonotactic stage with commitments to underlying feature values. To do so, the CBL incorporates procedures and learners that individually have proved successful at solving intermediate learning problems. In particular, error-driven learning yields ranking information and inconsistency detection indicates whether combinations of structures are permissible. These techniques are incorporated by the Inconsistency Detection Learner and the Output-Driven Learner, both of which are in turn incorporated by the CBL.

In addition to describing the motivations for the CBL, this chapter has illustrated how the CBL makes commitments and manipulates the incorporated learners at critical learning points. Chapter 3 will expand the scope of the illustration to follow the learner's progress from start to finish, through each step described in (117), as the learner processes this same learning data.

3 A COMPLETE LEARNING SIMULATION

The preceding chapter described the component procedures of the Commitment-Based Learner (CBL) for learning hidden structures of the surface and underlying forms. This chapter follows the CBL as it learns, from start to finish, showing where and when it uses each component and finally providing a successful outcome from the learning data. This chapter uses the same data as in chapter 2 and refers to that chapter for some explanations of the learner's actions.

The learning data in (118) are from set 15. The forms are listed left to right in the order in which the learner observes them in this illustration.

(118) Learning data set 15

ssY r1s1	*ssY* r1s2	*Yss* r2s1	*ssY* r2s2	*sYs* r3s1	*ssY* r3s2	*sYs* r4s1	*ssY* r4s2

Because the CBL rejects only inconsistent language hypotheses, it is possible to end learning with more than one consistent language hypothesis. In fact, this chapter will show that the CBL will learn all three targets associated with the learning data above, including L5, introduced previously, as well as L4 and L6, whose overt forms match those of L5. These languages are globally-surface ambiguous, as defined in (119).

(119) Global surface ambiguity (map-based definition)

> Languages L*A* and L*B* are globally surface-ambiguous if their maps are identical with respect to overt forms.

Each target is described below, along with a stratified hierarchy derived from the support of its skeletal basis. As expected by their global ambiguity, there are a number of similarities between these languages. The shaded cells highlight the key differences.

(120) L4

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(sY)]	[(Ys)s]	[(sY)s]	[(sY)s]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-Y/

(121) FT-BIN >> PARSE-σ >> MAXSTRESS >> IAMB >> {FNF, RMOST} >> {AFL, LMOST, *LAPSE}

(122) L5

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(sY)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-Y/

(123) FT-BIN >> PARSE-σ >> MAXSTRESS >> RMOST >> {IAMB, AFL, LMOST} >> {FNF, *LAPSE}

(124) L6

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(sY)]	[(Y)ss]	[(sY)s]	[(sY)s]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-Y/

(125) IAMB >> FNF >> MAXSTRESS >> {FT-BIN, RMOST} >> {PARSE-σ, AFL, LMOST, *LAPSE}

All the languages are sensitive to lexical stress but are by default iambic with rightmost primary stress. The languages group together according to two major

properties. First, L4 and L6 sacrifice right-alignment of the head-foot in order to parse iambs, unlike L5, which makes the opposite sacrifice. Therefore, r3s1 and r4s1 both surface as [(sY)s] in L4 and L6, but as [s(Ys)] in L5. Second, when faithfulness to an underlying stress pushes the head-foot from the right edge, as for r2s1 /Ys-s/, L4 and L5 pattern alike by parsing a binary trochaic head-foot to reduce the distance from the right edge, so that r2s1 surfaces as [(Ys)s]. L6 instead prefers to avoid trochaic feet altogether at a further expense to right-alignment. In that language, r2s1 surfaces with a degenerate foot: [(Y)ss].

Stress is contrastive in the suffixes of each language. Each language has three root behaviors, with r3 and r4 behaving alike. Stress is therefore contrastive for the second syllable of the root, and stress in the first root syllable is neutralized if the second syllable is stressed underlyingly. If structural interpretations are excluded, the three languages have the same morpheme behaviors, as illustrated by the table in (126), which groups r3 and r4 together to show that these roots behave alike.

(126) L4, L5, and L6: like morphemes grouped together

r1 =/ss/	r2 = /Ys/	r3 = /sY/ r4 = /YY/	
ssY	Yss	sYs	s1 = /-s/
ssY	ssY	ssY	s2 = /-Y/

This observation suggests the revised definition of global surface ambiguity given in (127), which specifically compares morpheme behaviors.

(127) Global surface ambiguity (morpheme behavior definition)

> Language L*A* and L*B* are globally surface-ambiguous if they have the same morpheme behaviors, excluding structural interpretations.

Although the map-based definition of global surface ambiguity in (119) is useful for quickly identifying some globally-surface ambiguous language, section 4.4 will show that comparing the overt realizations of morpheme behavior instead will identify cases of global surface ambiguity that the earlier definition misses. Additionally, reference to morpheme behaviors will be useful for providing a unified definition of global ambiguity, encompassing both global surface amibiguity and global lexical ambiguity. For more on global ambiguity and the CBL's response to globally ambiguous languages, see chapter 4.

3.1 PHONOTACTIC LEARNING

Some of the actions taken during phonotactic learning for this data set have been described already in chapter 2. The remainder of this section reviews each step taken by the learner for all the data, but for the details of the errors and inconsistencies related to the overt forms *ssY* and *Yss,* including the support updates for these forms, see section 2.4.2.

As explained in the preceding chapter, the CBL begins learning with an initial, empty language hypothesis, Hyp0, containing no structural commitments, W-L pairs, or lexical entries. The learner observes the first overt form, without its morpheme identity, and applies error-driven learning to check for new ranking information. If an error is detected, the learner applies the IDL and extends new branches. After this point, use of error-

driven learning and the IDL applies within each separate, consistent language hypothesis, and it is possible for an overt form to yield a branch-inducing error in one language hypothesis but not another. The CBL repeatedly cycles through the learning data until all overt forms are processed in all consistent language hypotheses without error. The outline for phonotactic learning in (128) is extracted from the more complete learning outline in 2.4.5; a fuller description of phonotactic learning specifically appears in (68) in section 2.4.2.

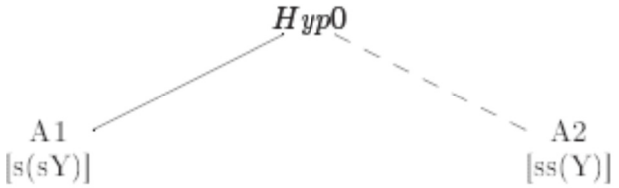
(128) Phonotactic learning outline

Within each language hypothesis, beginning with Hyp0, and for each observed form:

1. Check for errors
    a. If the form lacks a committed structural interpretation and yields an error, apply the IDL to extend branches. Repeat step 1 for each branch.
    b. If a form has a committed structural interpretation and produces an error, perform error-driven learning. Repeat step 1.
    c. If the form does not produce an error, process the next overt form.
2. Phonotactic learning ends when no errors are detected on any observed forms.

The diagram in (129) summarizes the outcome of observing the first form, *ssY*. An error detected on this form causes Hyp0 to branch into language hypotheses A1 and A2. The dashed line to A2 indicates that this branch is inconsistent. Again, the learner rejects inconsistent language hypotheses and never evaluates them again for learning.

(129) Branches extended from initial error on *ssY* in Hyp0



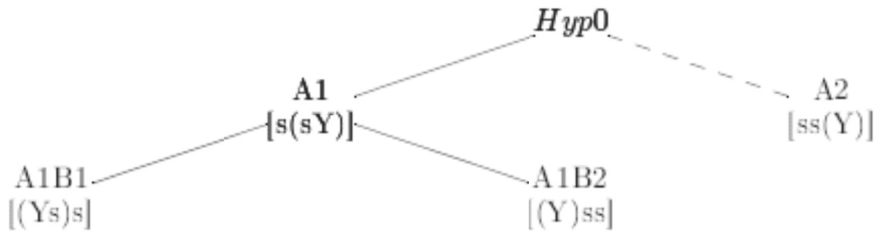
According to the data set in (118), repeated below, the learner observes *ssY* again, this time as the overt form of the word r1s2. As explained in 2.4.2, this overt form is processed without error in the only remaining language hypothesis, A1, which was just altered to accommodate *ssY* when the form was observed for the word r1s1. The learner must evaluate each overt form in each consistent language hypothesis, but the remainder of this section will avoid discussing overt forms processed in branches that have not changed since the last time the overt form was processed.

(130) Learning data set 15 (Lgs. 4, 5, 6)

ssY r1s1	*ssY* r1s2	*Yss* r2s1	*ssY* r2s2	*sYs* r3s1	*ssY* r3s2	*sYs* r4s1	*ssY* r4s2

The next new overt form observed is *Yss*. An error on this overt form in A1 induces branching into A1B1 and A1B2. These language hypotheses inherit all of the stored information in A1, as indicated by the repetition of A1 in their labels. Additionally, each makes its own commitment to a structural interpretation for *Yss*. Both of the branches are consistent.

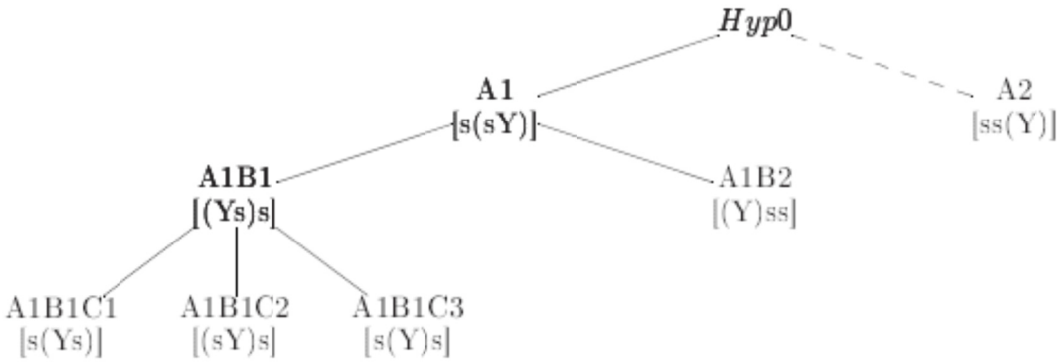
(131) Branches extended from error on *Yss* in A1



The fourth overt form observed from the data set is *ssY*, for the word r2s2. This overt form was last observed for r1s2, when it was processed in the parent language hypothesis A1; this time it is processed in both A1B1 and A1B2. The additional ranking information included in these branches make it plausible that a new error could be detected on *ssY*, but the rankings of both branches process the form without error.

Finally, the learner observes a new overt form, *sYs*, for r3s1. Section 2.4.2 showed that the learner detects an error on this form in A1B1 but not A1B2; therefore, only A1B1 extends branches now, shown in the tree in below.

(132) A1B1 branches for error on *sYs*



Branch A1B1C1 commits to the trochaic interpretation [s(Ys)]. Section 2.4.2 explains that the iambic interpretation [(sY)s] is the informative loser for this branch, and the appropriate W-L pair is added to the support for A1B1C1, which appears now in (133). Following the format established in chapter 2, the order in which W-L pairs are added is indicated in the leftmost column, with a "P" to indicate pairs added during phonotactic learning; thus, the newly added W-L pair is 5P. The updated ranking is shown in (134).

(133) A1B1C1 support; committed to [s(sY)], [(Ys)s], and [s(Ys)]

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	FNF	*LAPSE	LMOST	AFL	IAMB
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	W	L	L		
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W		L	L			W
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W		L	L	L	
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	W		W	W	L
e. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W	W		L	L	L

(134) FT-BIN >> PARSE-σ >> MAXSTRESS >> {RMOST, FNF, *LAPSE} >> {LMOST, AFL, IAMB}

The learner then checks for new ranking information, evaluating the identity mappings of the committed interpretations – that is, all the winners in the W-L pairs – to determine whether they yield errors under the updated ranking. No further errors are detected in this branch. Because each of the three unique overt forms in the data set now has a committed interpretation in A1B1C1, detecting no more errors means that this language hypothesis has acquired all the information it can without knowing the morpheme identities of the learning data: its phonotactic learning stage has effectively

ended after observing the first five overt forms in the learning data. The learner, however, does not have this insight. Therefore, as the learner observes the remaining data – *ssY* for r3s2, *sYs* for r4s1, and *ssY* again for r4s2 – each form will by duly processed in turn. Each will be assigned a structural interpretation from the stored commitment list – [s(sY)] and [s(Ys)], as appropriate – and then its identity mapping will be evaluated against the current ranking. No errors can arise, as the support in (133) already includes the ranking conditions necessary to ensure the optimality of /ssY/[s(sY)] and /sYs/[s(Ys)].

The second branch arising from the error on the overt form *sYs* is A1B1C2, which commits to the iambic interpretation, and now the trochaic interpretation is the informative loser. The appropriate W-L pair is labeled 5P in the updated support for A1B1C2, (135). Again, the learner processes all the committed identity mappings according to the updated ranking, (136), and detects no further errors.

(135) A1B1C2 support; committed to [s(sY)], [(Ys)s], and [(sY)s]

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	IAMB	LMOST	AFL	*LAPSE	RMOST	FNF
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W		L		L	W	W
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W	W			L		L
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W		L	L	L	W	
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	W	W		L	W
e. 5P	r3s1	/sY-s/	[(sY)s]	[s(Ys)]				W	W	W		L	L

(136) FT-BIN >> PARSE-σ >> MAXSTRESS >> {IAMB, LMOST, AFL, *LAPSE} >> {RMOST, FNF}

Finally, branch A1B1C3 commits to the degenerate interpretation [s(Y)s], but this interpretation is harmonically bounded by the iambic interpretation, shown in (137) by W-L pair 5P. Because no constraints prefer the winner for that W-L pair, the commitment to [s(Y)s] makes A1B1C3 inconsistent. This language hypothesis is rejected.

(137) A1B1C3 is inconsistent; /sYs/[s(Y)s] is harmonically-bounded

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	IAMB	LMOST	AFL	*LAPSE	RMOST	FNF
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W		L		L	W	W
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W	W			L		L
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W		L	L	L	W	
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	W	W		L	W
e. 5P	r3s1	/sY-s/	[s(Y)s]	[(sY)s]	L	L			L	L			

The last branch to review now is A1B2. This language hypothesis does not yield an error on the overt form *sYs* r3s1 because, as 2.4.2 explains, its current ranking information is sufficient to ensure the optimality of exactly one interpretation of the overt form. For this reason, it is unnecessary at this time to extend branches from A1B2 and commit to any particular interpretation. Its current support, which will remain unchanged, is given in (138), with ranking in (139).

(138) A1B2 support; committed to [s(sY)] and [(Y)ss]

ERC#	Morph. word	Input	Winner	Loser	IAMB	FNF	MAXSTR	LMOST	RMOST	AFL	PARSE-σ	FT-BIN	*LAPSE
a. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]	W	L	W						L
b. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]		W	W	L	W		L	W	L
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	L	W	L			L
d. 4P	r2s1	/Ys-s/	[(Y)ss]	[s(sY)]			W	W	L	W	L	L	

(139) IAMB >> FNF >> MAXSTRESS >> {LMOST,RMOST, AFL, PARSE-σ, FT-BIN, *LAPSE} >> {RMOST, FNF}

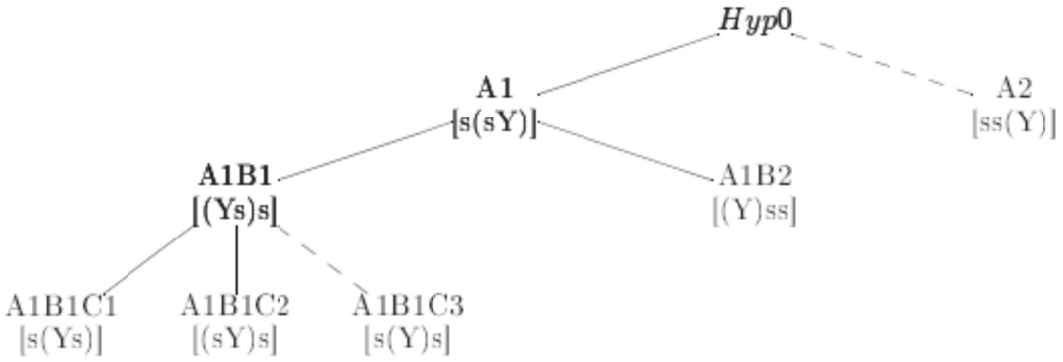
To review, at this point the learner has observed the first five overt forms in the learning data set and created four language hypotheses: A1B1C1, A1B1C2, A1B1C3, and A1B2. Of these, A1B1C1, A1B1C2, and A1B2 are consistent. The learner next cycles through the remaining three overt forms in data set (130), observing *sYs* once more and *ssY* twice more to complete one pass through the data. These forms are processed without error in each of the three consistent language hypotheses. There are no further changes.

The phonotactic learning stage ends when no more ranking information can be extracted from the overt forms of the learning data, which in turn means that every overt form can be processed in every consistent language hypothesis without error. A1B1C1 and A1B1C2 have in effect already completed phonotactic learning; having a commitment for each of the three unique overt forms and then rechecking the stored commitments for errors as this first pass through the data continues ensures that no new information can arise from revisiting the data in these language hypotheses. But A1B2 does not have a committed interpretation for *sYs*, and therefore the repeated check of

stored commitments cannot ensure that this form continues to be processed without error. If the ranking of a language hypothesis changes after processing an uncommitted overt form, it is possible for the new ranking to yield an error on that form later. For this reason, in this simulation the CBL simply keeps track of whether any error occurred in any language hypothesis during a pass through the data. If so, the CBL keeps evaluating the whole data set against every language hypothesis, until at last it passes through the entire data set without detecting any errors. Consequently, in this example the CBL makes a second complete pass through the learning data before completing phonotactic learning.

The tree in (140) includes all the language hypotheses created during phonotactic learning, with dashed lines indicating the inconsistent ones. A1B1C1 has committed to all the structural interpretations included in L5 and will ultimately derive a grammar for that target during the non-phonotactic learning stage. Similarly, A1B1C2 will derive a grammar consistent with the target L4. Finally, A1B2 has made structural commitments for just two overt forms. This language hypothesis will eventually branch and yield the grammar for the target L6.

(140) All branches created during phonotactic learning



A total of 21 ERCs were stored (and 21 RCD applications were made) during phonotactic learning. Combined, the three consistent language hypotheses that remain store 14 ERCs in their supports. The maximum number of hypotheses stored at any one time is four, when the learner has extended branches from A1B1 for the form *sYs* and lasting only until inconsistency detection reveals that the commitment for this form in branch A1B1C3 is inconsistent.

3.1.1 IS LEARNING COMPLETE?

Before the non-phonotactic learning stage begins, and before each pass through the learning data thereafter, the CBL checks each consistent language hypothesis to see if it is complete using the error detection procedure described in 2.4.3. Because the non-phonotactic learning stage has not yet begun, the learner does not know the morphemic identities of any of the words observed during phonotactic learning. All features are currently unset, and therefore all words with the same overt forms will have the same error detection candidates, regardless of their morphemic identity. This section will refer to the morphological word labels, such as r1s1, but what matters for error detection now is simply the identity mapping of that word: /ssY/[s(sY)].

As illustrated in 2.4.3, r1s1 /ss-Y/[s(sY)] fails error detection for A1B1C1, whose current ranking is repeated in (141).

(141) A1B1C1 current ranking

FT-BIN >> PARSE-σ >> MAXSTRESS >> {RMOST, FNF, *LAPSE} >> {LMOST, AFL, IAMB}

All three features of r1s1 are unset, and as a consequence the error detection candidate is /YY-s/[s(sY)], with all features set to mismatch the values of their surface correspondents. As 2.4.3 explains, although this candidate, (142)a in the violation tableau below, ties for most harmonic through the first two strata, it incurs two violations of MAXSTRESS in the third stratum, making it less harmonic than the candidates in (142)b,d,e that remain faithful to one of the underlying stresses. This error is sufficient to demonstrate that the language hypothesis is as yet incomplete.

(142) A1B1C1 – r1s1 fails error detection

Input	Output	FT-BIN	PARSE-σ	MAXSTR	RMOST	FNF	*LAPSE	IAMB	LMOST	AFL
a. /YY-s/	[s(sY)]	0	1	2	0	1	1	0	1	1
b.	[(sY)s]	0	1	1	1	1	0	0	0	0
c.	[(sY)(X)]	1	0	1	1	2	0	0	0	2
d.	[s(Ys)]	0	1	1	0	0	0	1	1	1
e.	[(Ys)s]	0	1	1	1	0	1	1	0	0
f.	[(Ys)(X)]	1	0	1	1	1	0	1	0	2
g.	[(Y)(sX)]	1	0	1	2	2	0	0	0	1
h.	[(X)(sY)]	1	0	2	0	2	0	0	1	1
i.	[(Y)(Xs)]	1	0	1	2	1	0	1	0	1
j.	[(X)(Ys)]	1	0	1	0	1	0	1	1	1

The same word will demonstrate that A1B1C2 and A1B2 are also incomplete. As branches from the original A1 parent, each of these language hypotheses commits to [s(sY)] for r1s1, and therefore all use the error detection candidate /YY-s/[s(sY)]. In fact, the violation tableau in (142) above also serves to illustrate the error on r1s1 in A1B1C2. In the current ranking for this branch, repeated in (143), the first three strata are identical

to those of A1B1C1. Consequently, an error is detected on r1s1 [s(sY)] in A1B1C2 for the same reason as in A1B1C1.

(143) A1B1C2 current ranking

FT-BIN >> PARSE-σ >> MAXSTRESS >> {IAMB, LMOST, AFL, *LAPSE} >> {RMOST, FNF}

Finally, in A1B2 MAXSTRESS also occupies the third stratum, and its relatively high rank causes r1s1 to fail error detection in this language hypothesis as well. The current ranking of A1B2 is repeated in (144). As for the other language hypotheses, the error detection candidate, (145)a in the violation tableau below, fails because it incurs more violations of MAXSTRESS than a competitor, here (145)b.

(144) A1B2 current ranking

IAMB >> FNF >> MAXSTRESS >> {LMOST,RMOST, AFL, PARSE-σ, FT-BIN, *LAPSE} >> {RMOST, FNF}

(145) A1B2 – r1s1 fails error detection

Input	Output	IAMB	FNF	MAXSTR	LMOST	RMOST	AFL	PARSE-σ	FT-BIN	*LAPSE
a. /YY-s/	[s(sY)]	0	1	2	1	0	1	1	0	1
b.	[(sY)s]	0	1	1	0	1	0	1	0	0
c.	[(sY)(X)]	0	2	1	0	1	2	0	1	0
d.	[s(Ys)]	1	0	1	1	0	1	1	0	0
e.	[(Ys)s]	1	0	1	0	1	0	1	0	1
f.	[(Ys)(X)]	1	1	1	0	1	2	0	1	0
g.	[(Y)(sX)]	0	2	1	0	2	1	0	1	0
h.	[(X)(sY)]	0	2	2	1	0	1	0	1	0
i.	[(Y)(Xs)]	1	1	1	0	2	1	0	1	0
j.	[(X)(Ys)]	1	1	1	1	0	1	0	1	0

Because all the consistent language hypotheses fail error detection for at least one word, the learner will seek to set features in each of them during the non-phonotactic learning stage.

3.2 LEARNING UNDERLYING FORMS

Now that the non-phonotactic learning stage has begun, the learner will receive the morphological information associated with each observed form, but the data continue to be processed in each consistent language hypothesis according to the order represented in (118). The CBL assumes that the learner can recall all previously observed words and can perform error-driven learning over them at any time. To model this assumption in this implementation of the CBL, each word observed is added to a list of known words unless it already appears in the list, and the learner performs error-driven learning over the list to check for new ranking information. Note that keeping a record of these known words is not essential to the CBL. An implementation that checks for errors only on the current observed word rather than on all known words will ultimately determine the same ranking information as this one, with the only difference being precisely when errors are detected. While the learner might have to complete a pass through the data to cycle back to a form that produces an error, such an implementation would not be dramatically slower than this one: each error could delay detection only for as many words as there are between the current observed word and a word that yields the error.

After checking for new ranking information, the learner will attempt to set features by employing the Output-Driven Learner (ODL) as long as the word already has a committed structural interpretation. Otherwise, the learner will check if the word passes error detection with all unset features set to mismatch their surface values. If it does not,

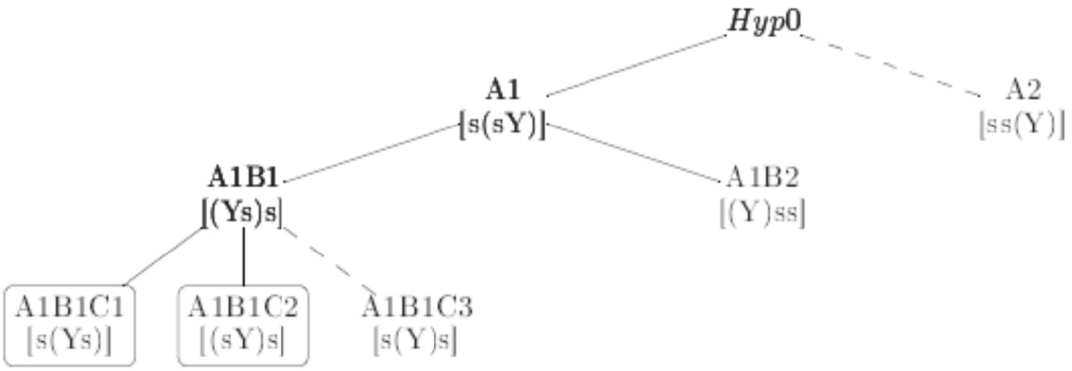
then the learner will apply the Inconsistency Detection Learner (IDL) to assign structural interpretations and extend branches. See 2.4.5 for the outline of these learning steps.

The paths of the branches from A1B1 and A1B2 diverge further in this stage. In order to best show how the language hypotheses evolve, the following sections will illustrate each in isolation.

3.2.1 BRANCHES FROM A1B1

This section follows the learner's progress in the two consistent language hypotheses that branched from A1B1. These are framed in the tree below.

(146) Consistent branches from A1B1



3.2.1.1 A1B1C1

A1B1C1 is the language hypothesis that ultimately will yield the grammar of the target L5. The map of L5 and a stratified hierarchy consistent with its skeletal basis appear in (147) and (148).

(147) L5

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(sY)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-Y/

(148) FT-BIN >> PARSE-σ >> MAXSTRESS >> RMOST >> {IAMB, AFL, LMOST} >> {FNF, *LAPSE}

A1B1C1 begins the non-phonotactic learning stage with the support and ranking below.

(149) A1B1C1 support

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	FNF	*LAPSE	LMOST	AFL	IAMB
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	**W**	**L**	**W**	**W**	**W**	**L**	**L**		
b. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			**W**		**L**	**L**			**W**
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			**W**	**W**		**L**	**L**	**L**	
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			**W**	**L**	**W**		**W**	**W**	**L**
e. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				**W**	**W**		**L**	**L**	**L**

(150) FT-BIN >> PARSE-σ >> MAXSTRESS >> {RMOST, FNF, *LAPSE} >> {LMOST, AFL, IAMB}

3.2.1.1.1 First pass through the data fails to set features

The learner adds each of the eight observed words to the list of known words but can derive no new grammatical information from them. Each of these words already has a committed interpretation, and there are no errors because the current ranking is unchanged from phonotactic learning. As explained in 2.4.4.1, initially no features can be

set from single forms. Because all the words fail error detection and single form learning has proved fruitless, the learner appeals to contrast pairs to set a feature.

3.2.1.1.2 Second pass through the data sets features with contrast pair r1s1, r2s1

A1B1C1 is the language hypothesis of the example in 2.4.4.1, and as detailed there, it takes a contrast pair to set the first few features. The first contrast pair identified is r1s1 [s(sY)], r2s1 [(Ys)s]. In this pair, the environment morpheme s1 alternates between –stress in r1s1 and +stress in r2s1, making this a plausible pair for learning. As previously described, the pair is informative and enables the learner to set features in r1 and r2, and to update the lexicon in (151). Additionally, 2.4.4.2 explains how the unfaithful mapping of r2's first syllable in r2s2 [s(sY)] yields new ranking information as well. The resulting W-L pairs 6 and 7 are added to the support in (152); application of BCD yields the ranking in (153).

(151) A1B1C1 – Contrast pair r1s1/r2s1 sets three features

r1	r2	r3	r4	s1	s2
/ss/	/Y?/	/??/	/??/	/-?/	/-?/

(152) A1B1C1 support updated using unfaithful mappings of r2s2

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	*LAPSE	LMOST	AFL	FNF	IAMB
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	L	L		W	
b. 7	r2s2	/Ys-Y/	[s(sY)]	[(X)(sY)]	W	L			L			W	
c. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W		L			L	W
d. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W	L	L	L		
e. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L		W	W	W	L
f. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W		L	L	W	L
g. 6	r2s2	/Ys-Y/	[s(sY)]	[(Ys)s]				W		L	L	L	W

(153) FT-BIN >> PARSE-σ >> MAXSTRESS >> {RMOST, *LAPSE} >> {LMOST, AFL, FNF, IAMB}

The contrast pair has set some features, but all words continue to fail error detection. Having added new lexical and ranking information, it is plausible that single forms which could not be used to set a feature in the first pass through the data might be able to now. In this simulation, the learner ceases this second pass through the learning data and starts a third pass, beginning with the first word in the data set, r1s1. The progress of this pass through the data continues below.

3.2.1.1.3 Third pass begins: r1s1 [s(sY)], r1s2 [s(sY)], and r2s1 [(Ys)s]

The test candidates for the first two words observed in this pass are consistent with the current support for A1B1C1. Because r1 was set to /ss/ in the preceding section using a contrast pair, both r1s1 and r1s2 each have just one unset feature, in the suffix. These words surface alike as [s(sY)], and therefore they have the same test candidate, shown in (154).

(154) Test candidate for r1s1 and r1s2

/ss-**s**/ → [s(sY)]

In spite of the new lexical and ranking information obtained by the contrast pair in the preceding section, this test candidate remains consistent with the support. Just as section 2.4.4.1 explains, language hypothesis A1B1C1 corresponds to target L5, which includes /ss-s/[s(sY)] in its map. Any mapping that is consistent with L5 is consistent with

A1B1C1, whose ranking conditions are less stringent than L5's. This test candidate will always be consistent with A1B1C1, and therefore it will always be uninformative for lexical learning.

The third word, r2s1 [(Ys)s], finally allows the learner to set a feature. With the stress feature of the first syllable already set to +stress, there are two candidates to test, below.

(155) Test candidates for r2s1 [(Ys)s]

a. /Y**Y**-s/ → [(Ys)s]
b. /Ys-**Y**/ → [(Ys)s]

A1B1C1 is the language hypothesis described in 2.4.4.3, and the candidates above are the same ones used in that section. As explained there, both test candidates are inconsistent with the support for A1B1C1. The learner has now determined that r2 must be /Ys/ underlyingly and s1 /-s/. Three morphemes now have complete entries in the lexicon, below.

(156) A1B1C1 lexicon updated for r2 and s1

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/??/	/??/	/-s/	/-?/

Having set a feature, the learner seeks non-phonotactic ranking information. Although s1 surfaces as +stress in r1s1 [s(sY)], this does not count as an unfaithful mapping. MAXSTRESS assigns violations only for +stress syllables that surface as –stress, not the reverse. In this case, r1s1 is not identified as potentially informative for non-phonotactic ranking information.

3.2.1.1.4 Third pass continues: r2s2 [s(sY)]

The next word processed in the third pass through the data is r2s2 [s(sY)], in which only the suffix s2 remains unset. Each time a word is processed, the learner checks that the current ranking makes all the known words optimal. In this example, r2s2 /Ys-Y/[s(sY)] is optimal according to the current ranking, but one word in the list, r1s1 /ss-s/[s(sY)], is not.

Because all features of r1s1 have already been set, the learner evaluates the mapping /ss-s/[s(sY)] against the current ranking derived by BCD, in (153). The violation tableau in (157) includes the candidate for r1s1 in (157)a and all other potential optima that satisfy undominated FT-BIN by parsing only binary feet. All candidates tie in the first three strata, but the desired winner (157)a loses to the right-aligned trochaic candidate, (157)c, in the fourth stratum, as the shaded cells indicate. The learner adopts (157)c as a loser and adds the resulting W-L pair 8 to the support in (158); the updated ranking appears in (159).

(157) Error in A1B1C1 for r1s1 /ss-s/[s(sY)]

Input	Output	FT-BIN	PARSE-σ	MAXSTR	RMOST	*LAPSE	LMOST	AFL	FNF	IAMB
a. /ss-s/	[s(sY)]	0	1	0	0	1	1	1	1	0
b.	[(sY)s]	0	1	0	1	0	0	0	1	0
c.	[s(Ys)]	0	1	0	0	0	1	1	0	1
d.	[(Ys)s]	0	1	0	1	1	0	0	0	1

(158) A1B1C1 support updated after error on r1s1

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	LMOST	AFL	IAMB	FNF	*LAPSE
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	L			W	L
b. 7	r2s2	/Ys-Y/	[s(sY)]	[(X)(sY)]	W	L						W	L
c. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W				W	L	L
d. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W	L	L			L
e. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	W	W	L	W	
f. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W	L	L	L	W	
g. 6	r2s2	/Ys-Y/	[s(sY)]	[(Ys)s]				W	L	L	W	L	
h. 8	r1s1	/ss-s/	[s(sY)]	[s(Ys)]							W	L	L

(159) FT-BIN >> PARSE-σ >> MAXSTRESS >> RMOST>> {LMOST, AFL, IAMB} >> {FNF, *LAPSE}

With no errors detected on the other words, the learner returns to r2s2 and now detects a lexically-informative inconsistency using the test candidate /Ys-**s**/[s(sY)] to set the feature in s2. The tableau in (160) includes the current support and one W-L pair created for the test candidate, (160)i. After FT-BIN and PARSE-σ are ranked in the first two strata, each of the remaining constraints prefers a loser at least once. Based on this inconsistency the learner can set s2 to +stress. The updated lexicon appears in (161).

(160) A1B1C1 is inconsistent with r2s2 when s2 = /-s/

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	LMOST	AFL	IAMB	FNF	*LAPSE
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	L			W	L
b. 7	r2s2	/Ys-Y/	[s(sY)]	[(X)(sY)]	W	L						W	L
c. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W				W	L	L
d. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W	L	L			L
e. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	W	W	L	W	
f. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W	L	L	L	W	
g. 6	r2s2	/Ys-Y/	[s(sY)]	[(Ys)s]				W	L	L	W	L	
h. 8	r1s1	/ss-s/	[s(sY)]	[s(Ys)]							W	L	L
i. test	r2s2	/Ys-s/	[s(sY)]	[(Ys)s]			L	W	L	L	L		

(161) A1B1C1 lexicon updated for s2

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/??/	/??/	/-s/	/-Y/

3.2.1.1.5 Third pass concludes: r3s1 [s(Ys)], r3s2 [s(sY)], r4s1 [s(Ys)], r4s2 [s(sY)]

The learner continues this pass through the data and processes r3s1, which has unset features in the root only. Because no errors are detected on any of the known words for the current ranking in (159), the learner can now attempt to set the features of r3 using the candidates in (162).

(162) Test candidates for r3s1 [s(Ys)]

a. /**Y**Y-s/ → [s(Ys)]
b. /s**s**-s/ → [s(Ys)]

The first candidate is consistent with the support for a familiar reason. Recall that although the learner does not know it, A1B1C1 corresponds to target L5. Test candidate

(162)a is a mapping in L5, as shaded in the map repeated below, and therefore it is consistent with the support of A1B1C1.

(163) L5

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(sY)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-Y/

However, test candidate (162)b is inconsistent because it makes contradictory ranking requirements with W-L pair 8, as shown in (164). The learner therefore can set the second syllable of r3 to +stress in the lexicon, (165).

(164) /ss-s/[s(Ys)] is inconsistent with W-L pair 8

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	LMOST	AFL	IAMB	FNF	*LAPSE
a. 8	r1s1	/ss-s/	[s(sY)]	[s(Ys)]							W	L	L
b. test	r3s1	/ss-s/	[s(Ys)]	[s(sY)]							L	W	W

(165) A1B1C1 lexicon updated for r3

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/?Y/	/??/	/-s/	/-Y/

Although r3 now has only one unset feature, the next word, r3s2 [s(sY)], will not set it. The test candidate shown in (166) again matches a mapping in the target L5: it is the canonical mapping of r2s2.

(166) Test candidates for r3s2 [s(sY)]

/**Y**s-Y/ → [s(sY)]

Finally, the learner processes r4s1 [s(Ys)] and r4s2 [s(sY)]. Because r4 behaves like r3, the test candidates for these words will be identical to those in (162) and (166), with the same outcomes. Thus, the learner can set the second syllable of r4 to +stress also. Only the initial syllables of these roots remain unset in the lexicon, (167).

(167) A1B1C1 lexicon updated for r3 and r4

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/?Y/	/?Y/	/-s/	/-Y/

The set features in r3 and r4 surface unfaithfully in r3s2 and r4s2, making these words potential sources of non-phonotactic ranking information. The learner applies a low-markedness bias to the support to produce ranking (168), used in the violation tableau in (169). Candidate (169)a corresponds to both r3s2 and r4s2. It ties with the right-aligned trochaic candidate in (169)d through the first two strata, and does better in the third because it incurs no violations of IAMB. As a result, there is no error, and no new ranking information to be learned.

(168) MAXSTRESS >> {FT-BIN, RMOST} >> {PARSE-σ, LMOST, AFL, IAMB} >> {FNF, *LAPSE}

(169) No error on r3s2 /sY-Y/[s(sY)] under low-markedness ranking

Input	Output	MAXSTR	FT-BIN	RMOST	PARSE-σ	LMOST	AFL	IAMB	FNF	*LAPSE
a. /sY-Y/	[s(sY)]	1	0	0	1	1	1	0	1	1
b.	[(sY)s]	1	0	1	1	0	0	0	1	0
c.	[(sY)(X)]	1	1	1	0	0	2	0	2	0
d.	[s(Ys)]	1	0	0	1	1	1	1	0	0
e.	[(Ys)s]	2	0	1	1	0	0	1	0	1
f.	[(Ys)(X)]	2	1	1	0	0	2	1	1	0
g.	[(Y)(sX)]	2	1	2	0	0	1	0	2	0
h.	[(X)(sY)]	1	1	0	0	1	1	0	2	0
i.	[(Y)(Xs)]	2	1	2	0	0	1	1	1	0
j.	[(X)(Ys)]	1	1	0	0	1	1	1	1	0

3.2.1.1.6 A1B1C1 is complete

R3 and r4 still have unset features, but now all words pass error detection. In L5, underlying stress in the first syllable of the root is neutralized if the second syllable is also stressed underlyingly. R3 and r4 therefore have the same phonological behaviors, and the feature values of their first syllables do not have to be set in the lexicon, as their error detection tests indicate: the error detection candidates, shown below, match the canonical mappings of r4s1 and r4s2 in L5 and are necessarily consistent with the support.

(170) Final error detection candidates

a. r3s1, r4s1 /**Y**Y-s/ → [s(Ys)]
b. r3s2, r4s2 /**Y**Y-Y/ → [s(sY)]

The final lexicon for A1B1C1, in (171), leaves r3 and r4 each with one unset feature. The final support is repeated in (172), with its ranking derived by BCD in (173).

(171) A1B1C1 – Final lexicon

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/?Y/	/?Y/	/-s/	/-Y/

(172) A1B1C1 – Final support

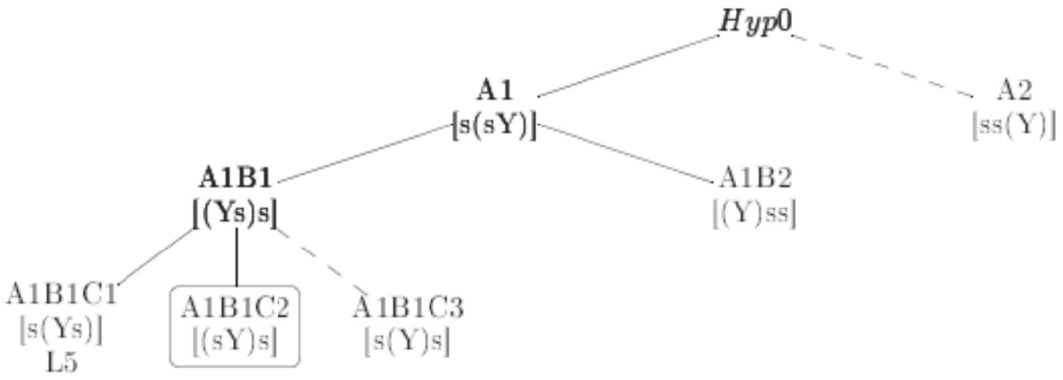
ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	RMOST	LMOST	AFL	IAMB	FNF	*LAPSE
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W	W	L			W	L
b. 7	r2s2	/Ys-Y/	[s(sY)]	[(X)(sY)]	W	L						W	L
c. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W				W	L	L
d. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W	L	L			L
e. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	W	W	L	W	
f. 5P	r3s1	/sY-s/	[s(Ys)]	[(sY)s]				W	L	L	L	W	
g. 6	r2s2	/Ys-Y/	[s(sY)]	[(Ys)s]				W	L	L	W	L	
h. 8	r1s1	/ss-s/	[s(sY)]	[s(Ys)]							W	L	L

(173) FT-BIN >> PARSE-σ >> MAXSTRESS >> RMOST>> {LMOST, AFL, IAMB} >> {FNF, *LAPSE}

3.2.1.2 A1B1C2

Language hypothesis A1B1C2, framed in (174), is the second consistent branch that survives when A1B1 branches to commit to interpretations of *sYs* during phonotactic learning. A1B1C2 corresponds to target L4, whose map appears in (175) followed by a stratified hierarchy that will generate it.

(174) A1B1C2 in the hypothesis tree



(175) L4

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(sY)]	[(Ys)s]	[(sY)s]	[(sY)s]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-Y/

(176) FT-BIN >> PARSE-σ >> MAXSTRESS >> IAMB >> {FNF, RMOST} >> {AFL, LMOST, *LAPSE}

L4 differs from L5 just in r3s1 and r4s1, where L4 parses iambs at the cost of shifting the head-foot away from the right edge. While the precise details of the learner's progress through the data in A1B1C2 differ from those described in A1B1C1, the overview is quite similar. Both language hypotheses must first use the contrast pair r1s1 [s(sY)], r2s1 [(Ys)s] to set features in r1 and r2. Then, non-phonotactic ranking information following from setting r2 enables the rest of the features to be set from single forms. Because this language hypothesis offers no new insight into the CBL, the learning steps are described in the outline in (177), followed by the final support, (178), ranking (179), and lexicon, (180).

(177) Outline of learning A1B1C2

1. Single form learning fails to set any features
2. Contrast pair learning
    a. Set r1 to /ss/ and r2 to /Y?/ using contrast pair r1s1 [s(sY)], r2s1 [(Ys)s]
    b. Add W-L pairs 6 and 7 from unfaithful mapping of r2 in r2s2 [s(sY)], using markedness-low ranking.
3. Single form learning.
    a. Add W-L pair 8 from error on r2s2 using BCD ranking.
    b. Set r2 to /Ys/ and s1 to /s/ using r2s1 [(Ys)s]
    c. Add W-L pair 9 from unfaithful mapping of s1 in r1s1 [s(sY)], using markedness-low ranking.
    d. Set r3 to /?Y/ using r3s1 [(sY)s].
    e. Set r4 to /?Y/ using r4s1 [(sY)s].
4. All words pass error detection.

(178) A1B1C2 final support; committed to [s(sY)], [(Ys)s] and [(sY)s]

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	PARSE-σ	MAXSTR	IAMB	RMOST	FNF	LMOST	AFL	*LAPSE
a. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W		W	W	L		L
b. 6	r2s2	/Ys-Y/	[s(sY)]	[(Y)ss]	W	W			W		L	L	
c. 7	r2s2	/Ys-Y/	[s(sY)]	[(Y)(sX)]	W	L			W	W	L		L
d. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]			W	W		L			L
e. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W		W		L	L	L
f. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(sY)]			W	L	L	W	W	W	
g. 5P	r3s1	/sY-s/	[(sY)s]	[s(Ys)]				W	L	L	W	W	
h. 8	r2s2	/Ys-Y/	[s(sY)]	[(Ys)s]				W	W	L	L	L	
i. 9	r1s1	/ss-s/	[s(sY)]	[(sY)s]					W		L	L	L

(179) FT-BIN >> PARSE-σ >> MAXSTRESS >> IAMB >> {RMOST, FNF} >> {LMOST, AFL, *LAPSE}

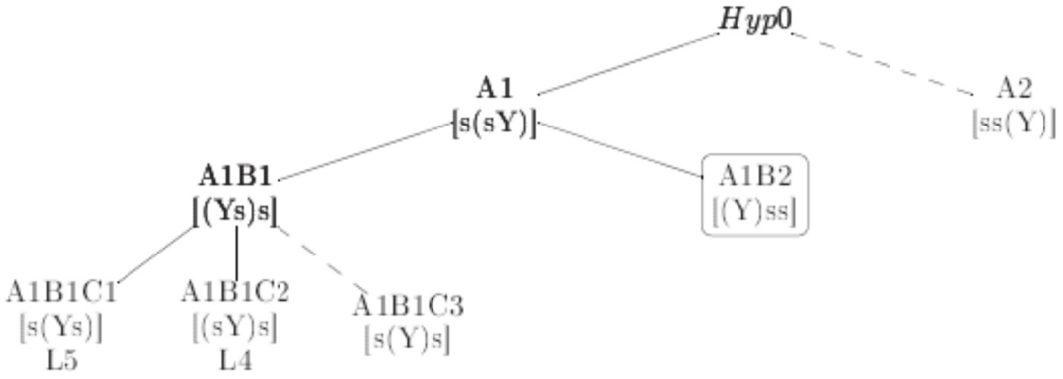
(180) A1B1C2 – Final lexicon

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/?Y/	/?Y/	/-s/	/-Y/

3.2.2 BRANCHES FROM A1B2

The preceding sections describe the learner's progress through two branches extended from A1B1 after commitments to r3s1 *sYs*. Both survive as complete language hypotheses, A1B1C1 and A1B1C2, corresponding to target languages L5 and L4, respectively. This section follows the progress of A1B2, framed in (181). It is the third language hypothesis surviving when phonotactic learning ends, and it will ultimately yield a language hypothesis corresponding to target L6.

(181) A1B2 in the hypothesis tree



3.2.2.1 First pass: r1s1 [s(sY)], r1s2 [s(sY)], r2s1 [(Y)ss], r2s2 [s(sY)], r3s1 sYs

When the learner first begins to learn underlying forms, this language hypothesis has committed to [s(sY)] and [(Y)ss] – its point of difference with A1B1 – but has not yet made a commitment for *sYs.* Its support and current ranking by BCD are repeated below.

(182) A1B2 support; committed to [s(sY)] and [(Y)ss]

ERC#	Morph. word	Input	Winner	Loser	IAMB	FNF	MAXSTR	LMOST	RMOST	AFL	PARSE-σ	FT-BIN	*LAPSE
a. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]	W	L	W						L
b. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]		W	W	L	W		L	W	L
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	L	W	L			L
d. 4P	r2s1	/Ys-s/	[(Y)ss]	[s(sY)]			W	W	L	W	L	L	

(183) IAMB >> FNF >> MAXSTRESS >> {LMOST, RMOST, AFL, PARSE-σ, FT-BIN, *LAPSE}

A1B2 is notably different from A1B1C1 and A1B1C2 because it allows for the learner to set features as soon as single form learning begins, even before committing to an interpretation for the third overt form. First, however, the attempts to set features in r1s1 [s(sY)] and r1s2 [s(sY)] fail, just as in the other language hypotheses and for the same reason: A1B2 will eventually branch and yield the grammar of a target language, L6, and it happens that the test candidates for these words are mappings in that target. As the current support of A1B2 is consistent with L6, the test candidates must be consistent also. But at last the learner processes r2s1, and here A1B2's commitment to [(Y)ss] instead of the trochaic [(Ys)s] results in a lexically informative inconsistency. The three test candidates of r2s1 appear below.

(184) Test candidates for r2s1 [(Y)ss]

a. /**s**s-s/ → [(Y)ss]
b. /Y**Y**-s/ → [(Y)ss]
c. /Ys-**Y**/ → [(Y)ss]

The first candidate is harmonically bounded by the candidate parsing a left-aligned iamb. The comparative tableau in (185) shows that no constraint prefers the test candidate to that competitor; likewise, (186) shows that the second test candidate is harmonically bounded by the same iambic competitor. These inconsistencies enable the learner to set r2 to /Ys/ and to update the lexicon, (187).

(185) /ss-s/[(Y)ss] is harmonically bounded; sets r2 to /Y?/

Input	Winner	Loser	MAXSTR	IAMB	LMOST	AFL	FNF	PARSE-σ	FT-BIN	RMOST	*LAPSE
/ss-s/	[(Y)ss]	[(sY)s]						L	L	L	L

(186) /YY-s/[(Y)ss] is harmonically bounded; sets r2 to /Ys/

Input	Winner	Loser	IAMB	MAXSTR	LMOST	AFL	FNF	PARSE-σ	FT-BIN	RMOST	*LAPSE
/YY-s/	[(Y)ss]	[(sY)s]						L	L	L	L

(187) A1B2 lexicon updated for r2

r1	r2	r3	r4	s1	s2
/??/	/Ys/	/??/	/??/	/-?/	/-?/

These features do not ever surface unfaithfully, but the learner receives new ranking information anyway from an error detected on r2s2 [s(sY)] under ranking (183). The violation tableau below includes the two candidates which best satisfy constraints in the top three strata. The candidates conflict on the constraints of the bottom stratum, with LMOST and AFL preferring the competitor (188)b to (188)a, which includes the

committed output. By the CTie criterion, these candidates tie despite the fact that (188)b incurs more total violations than (188)a. The learner adopts (188)b as an informative loser and adds W-L pair 5 to the support, (189). The updated ranking appears in (190)

(188) Error in A1B2 for r2s2 /Ys-Y/[s(sY)]

Input	Output	IAMB	FNF	MAXSTR	LMOST	RMOST	AFL	PARSE-σ	FT-BIN	*LAPSE
a. /Ys-Y/	[s(sY)]	0	1	1	1	0	1	1	0	1
b.	[(Y)ss]	0	1	1	0	2	0	2	1	1

(189) A1B2 support updated after error on r2s2

ERC#	Morph. word	Input	Winner	Loser	IAMB	FNF	MAXSTR	RMOST	PARSE-σ	FT-BIN	*LAPSE	LMOST	AFL
a. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]	W	L	W				L		
b. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]		W	W	W	L	W	L	L	
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W			L	L	L
d. 4P	r2s1	/Ys-s/	[(Y)ss]	[s(sY)]			W	L	L	L		W	W
e. 5	r2s2	/Ys-Y/	[s(sY)]	[(Y)ss]				W	W	W		L	L

(190) IAMB >> FNF >> MAXSTRESS >> {RMOST, PARSE-σ, FT-BIN, *LAPSE} >> {LMOST, AFL}

Because the features of the root r2 have already been set, there is only one test candidate to evaluate for r2s2, with the feature value disparity in the suffix: /Ys-**s**/[s(sY)]. The tableau in (191) shows that this test candidate is inconsistent with W-L pair 4P, which makes contradictory ranking requirements for its winner /Ys-s/[(Y)ss]. The inconsistency allows the learner to set s2 to +stress, updating the lexicon to (192).

(191) /Ys-s/[s(sY)] is inconsistent with A1B2

ERC#	Morph. word	Input	Winner	Loser	IAMB	FNF	MAXSTR	RMOST	PARSE-σ	FT-BIN	*LAPSE	LMOST	AFL
a. 4P	r2s1	/Ys-s/	[(Y)ss]	[s(sY)]			W	L	L	L		W	W
b. test	r2s2	/Ys-s/	[s(sY)]	[(Y)ss]			L	W	W	W		L	L

(192) A1B2 lexicon updated for s2

r1	r2	r3	r4	s1	s2
/??/	/Ys/	/??/	/??/	/-?/	/-Y/

Halfway into this first pass through the data in language hypothesis A1B2, the learner has already set three features from single forms and added new ranking information. The learner next observes r3s1 *sYs*, for which no structural commitment has yet been made and now error detection reveals that a committed interpretation is finally warranted. The error detection input is /Ys-Y/, with all features set to mismatch their surface values. The violation tableau in (193) includes the most harmonic candidates through the first two strata. Candidate (193)a has the overt form *sYs,* matching the observed form*,* but it is less harmonic than candidates (193)b,c because it receives an additional violation of MAXSTRESS.

(193) R3s1 *sYs* fails error detection

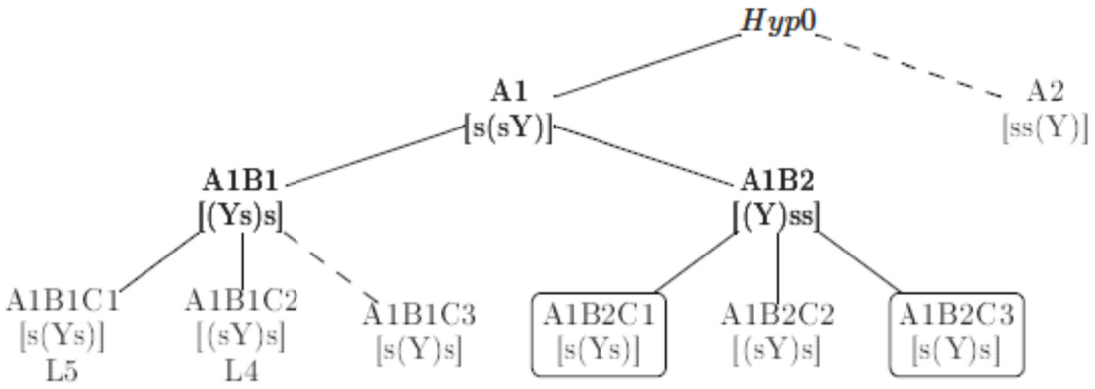
Input	Output	IAMB	FNF	MAXSTR	RMOST	PARSE-σ	FT-BIN	*LAPSE	LMOST	AFL
a. /Ys-Y/	[(sY)s]	0	1	2	1	1	0	0	0	0
b.	[s(sY)]	0	1	1	0	1	0	1	1	1
c.	[(Y)ss]	0	1	1	2	2	1	1	0	0

Now the learner extends branches from A1B2. Each branch will inherit the information gleaned from this round of single form learning. As the following sections will show, the inherited support plus the ranking conditions imposed by each new commitment will cause two branches to die immediately and allow only the branch corresponding to L6 to survive.

3.2.2.2 A1B2C1 and A1B2C3

Language hypotheses A1B2C1 and A1B2C3, framed in the tree below, are the two branches that prove inconsistent because of their commitments for *sYs*.

(194) A1B2C1 and A1B2C3 in the hypothesis tree



A1B2C1 commits to the right-aligned trochaic interpretation of *sYs:* [s(Ys)]. Together, the three commitments of this language hypothesis run the gamut of possibilities: right-aligned iambic, [s(sY)]; left-aligned degenerate, [(Y)ss]; and now right-aligned trochaic, [s(Ys)]. It is perhaps no surprise that this third and final commitment makes the language hypothesis inconsistent.

Tableau (195) includes the W-L pairs added by error-driven learning after committing to [s(Ys)]. The commitment initially leads to adding W-L pairs 6 and 7 using r3s1. Next, the learner reviews the rest of the forms, which now all have committed structural interpretations, checking for errors on the new ranking. Detecting an error on r1s1, the learner adds W-L pair 8, then detects an error on r2s1. The resulting W-L pair 9 produces the final inconsistency. MAXSTRESS prefers only winners, leaving the remaining markedness constraints to be ranked. Of these, only RMOST, FT-BIN, and FNF prefer the winners of pairs 5-8, but they prefer the loser in pair 9.

(195) A1B2C1 support; commitments to [s(sY)], [(Y)ss] and [s(Ys)] are inconsistent

ERC#	Morph. word	Input	Winner	Loser	MAXSTR	LMOST	RMOST	AFL	PARSE-σ	FT-BIN	FNF	IAMB	*LAPSE
a. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]	W						L	W	L
b. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]	W	L	W		L	W	W		L
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]	W	L	W	L					L
d. 4P	r2s1	/Ys-s/	[(Y)ss]	[s(sY)]	W	W	L	W	L	L			
e. 5	r2s2	/Ys-Y/	[s(sY)]	[(Y)ss]		L	W	L	W	W			
f. 6	r3s1	/sY-s/	[s(Ys)]	[(sY)s]		L	W	L			W	L	
g. 7	r3s1	/sY-s/	[s(Ys)]	[(X)(Ys)]					L	W	W		
h. 8	r1s1	/ss-Y/	[s(sY)]	[(Xs)(Y)]		W		W	L	W		W	L
i. 9	r2s1	/Ys-s/	[(Y)ss]	[(Ys)s]			L		L	L	L	W	

Whereas A1B2C1 is inconsistent due to the final combination of its structural commitments, A1B2C3 is doomed simply on the basis of its commitment to [s(Y)s]. The parsing of a degenerate foot may be optimal when right- or left-aligned, but not otherwise; thus, this interpretation is harmonically bounded, and the learner rejects A1B2C3. The pertinent W-L pair is the same as W-L pair 5P in (137) of section 3.1,

which shows that this interpretation is harmonically bounded. The learner rejects A1B2C3 after adding only five ERCs to its support.

3.2.2.3 A1B2C2

Finally, A1B2C2 is the branch that commits to [(sY)s], making it the language hypothesis for target L6, repeated with a corresponding stratified hierarchy below. In contrast to L4 and L5, this target prefers iambs over rightmost main stress. It parses a degenerate foot at the left edge of r2s1, [(Y)ss], to satisfy this preference without violating MAXSTRESS.

(196) L6

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(sY)]	[(Y)ss]	[(sY)s]	[(sY)s]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(sY)]	[s(sY)]	s2 = /-Y/

(197) IAMB >> FNF >> MAXSTRESS >> {FT-BIN, RMOST} >> {PARSE-σ, AFL, LMOST, FNF, *LAPSE}

During phonotactic learning in A1B1 and A1B2, there was no commitment made for *sYs* because the rankings always made at least one interpretation of the overt form optimal. Within language hypothesis A1B2, that optimal interpretation was [(sY)s]. This should be evident from reviewing the conclusions of the previous section. The degenerate interpretation [s(Y)s] is harmonically bounded, and (195) shows that the trochaic interpretation [s(Ys)] is inconsistent with commitments to [s(sY)] and [(Y)ss]. Consequently, when the learner commits to [(sY)s] now, the current ranking already makes the interpretation optimal. The support and lexicon for this branch are inherited

from A1B2 without any additions, and are included below along with the ranking derived by applying BCD to the support.

(198) A1B2C2 support; committed to [s(sY)], [(Y)ss], and [(sY)s]

ERC#	Morph. word	Input	Winner	Loser	IAMB	FNF	MAXSTR	RMOST	PARSE-σ	FT-BIN	*LAPSE	LMOST	AFL
a. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]	W	L	W				L		
b. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]		W	W	W	L	W	L	L	
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W			L	L	L
d. 4P	r2s1	/Ys-s/	[(Y)ss]	[s(sY)]			W	L	L	L		W	W
e. 5	r2s2	/Ys-Y/	[s(sY)]	[(Y)ss]				W	W	W		L	L

(199) A1B2C2 lexicon

r1	r2	r3	r4	s1	s2
/??/	/Ys/	/??/	/??/	/-?/	/-Y/

(200) IAMB >> FNF >> MAXSTRESS >> {RMOST, PARSE-σ, FT-BIN, *LAPSE} >> {LMOST, AFL}

Because A1B2C2 remains consistent after branching from A1B2, the learner can begin the second pass through the learning data, starting again with r1s1. The first form to set a feature in the newly extended branch is r2s1, whose unset feature is in the suffix. The test candidate, /Ys-**Y**/[(Y)ss], is the loser of W-L pair 5, (198)e, and therefore inconsistent with the support. The lexicon is updated with s1 set to –stress.

(201) A1B2C2 lexicon updated for s1

r1	r2	r3	r4	s1	s2
/??/	/Ys/	/??/	/??/	/-s/	/-Y/

Recall that although s1 surfaces as +stress in r1s1 [s(sY)], this mapping does not count as unfaithful according to MAXSTRESS and the learner does not select it to learn non-phonotactic ranking information from the low-markedness ranking; however, the word does yield an error on the BCD ranking. The violation tableau in (202) includes the most harmonic candidates through the first two strata: the two iambic candidates that also best satisfy FNF by avoiding degenerate feet. The shaded cells reveal an unresolved conflict between RMOST and *LAPSE in the fourth stratum, with RMOST preferring the committed desired winner and *LAPSE its competitor. By the CTie criterion, these candidates tie, and the learner adopts (202)b as a loser, adding the new W-L pair 6 to the support for A1B2C2 in (203). In the updated ranking in (204), *LAPSE now occupies the bottom stratum.

(202) Error in A1B2C2 for r1s1 /ss-s/[s(sY)]

Input	Output	IAMB	FNF	MAXSTR	RMOST	PARSE-σ	FT-BIN	*LAPSE	LMOST	AFL
a. /ss-s/	[s(sY)]	0	1	0	0	1	0	1	1	1
b.	[(sY)s]	0	1	0	1	1	0	0	0	0

(203) A1B2C2 support after error on r1s1

ERC#	Morph. word	Input	Winner	Loser	IAMB	FNF	MAXSTR	RMOST	PARSE-σ	FT-BIN	LMOST	AFL	*LAPSE
a. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]	W	L	W						L
b. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]		W	W	W	L	W	L		L
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W			L	L	L
d. 4P	r2s1	/Ys-s/	[(Y)ss]	[s(sY)]			W	L	L	L	W	W	
e. 5	r2s2	/Ys-Y/	[s(sY)]	[(Y)ss]				W	W	W	L	L	
f. 6	r1s1	/ss-s/	[s(sY)]	[(sY)s]				W			L	L	L

(204) IAMB >> FNF >> MAXSTRESS >> {RMOST, PARSE-σ, FT-BIN} >> {LMOST, AFL, *LAPSE}

Finally, the new ranking allows the learner to set features in the rest of the roots. First, continuing to process the data in order, the learner can set r3 and r4 using r3s1 and r4s1, which now have a committed interpretation. These roots have the same phonological behaviors, and the test candidates for r3 in (205) will suffice to demonstrate those for r4. The first test candidate is consistent with the support, an unsurprising conclusion as this is the canonical mapping of r4s1 in the target L6. The second test candidate is inconsistent with the support, as it is the loser of W-L pair 6. R3, and by the same reasoning r4, must have its second syllable set to +stress in the lexicon, (206).

(205) Test candidates for r3s1

a. /**Y**Y-s/ → [(sY)s]
b. /s**s**-s/ → [(sY)s]

(206) A1B2C2 – Lexicon updated for r3 and r4

r1	r2	r3	r4	s1	s2
/??/	/Ys/	/?Y/	/?Y/	/-s/	/-Y/

Before continuing, note here the importance of r3s1 and r4s1 for learning L6. The roots r3 and r4 behave alike, and differently from either r1 or r2. This difference must be reflected in the lexicon to explain the observed behaviors, but only a word containing the unstressed suffix s1 can be informative about the underlying form of the roots. In L6 the preference to right-align the head-foot means that the rightmost underlyingly stressed syllable will surface faithfully with primary stress. Therefore, all words containing s2 surface as [s(sY)]. The test candidates for the roots in these words will themselves be mappings in L6 and are necessarily consistent with the support. Branching and committing to an interpretation of *sYs* for r3s1 and r4s1 is not simply a consequence of the learner encountering an uncommitted overt form during the non-phonotactic learning stage, it is a crucial part of the whole learning process. Without a committed interpretation, the learner cannot use these words to set features in r3 and r4 by inconsistency detection, and without distinguishing r3 and r4 from the other roots in the lexicon, the learner cannot successfully learn the target.

After setting these features, only features in r1 remain to be set. The learner returns to the beginning of the data set, again observing r1s1, which has two unset features, both in the root. The test candidate with the disparity in the first syllable, /**Y**s-s/[s(sY)], is inconsistent because it is the loser in W-L pair 4P. This inconsistency allows the learner to set the first syllable of r1 now to –stress. The second test candidate has its disparity in the second syllable of r1: /s**Y**-s/[s(sY)]. The following comparative tableau shows that

this candidate, in (207)g, is inconsistent with the support because it has contradictory ranking requirements with W-L pair 4P, in (207)d. Together, these two inconsistencies enable the learner to update the lexicon in (208) by setting r1 to /ss/.

(207) A1B2C2 is inconsistent for r1s1 /s**Y**-s/[s(sY)]

ERC#	Morph. word	Input	Winner	Loser	IAMB	FNF	MAXSTR	RMOST	PARSE-σ	FT-BIN	LMOST	AFL	*LAPSE
a. 1P	r1s1	/ss-Y/	[s(sY)]	[s(Ys)]	W	L	W						L
b. 2P	r1s1	/ss-Y/	[s(sY)]	[(Y)(sX)]		W	W	W	L	W	L		L
c. 3P	r1s1	/ss-Y/	[s(sY)]	[(sY)s]			W	W			L	L	L
d. 4P	r2s1	/Ys-s/	[(Y)ss]	[s(sY)]			W	L	L	L	W	W	
e. 5	r2s2	/Ys-Y/	[s(sY)]	[(Y)ss]				W	W	W	L	L	
f. 6	r1s1	/ss-s/	[s(sY)]	[(sY)s]				W			L	L	L
g. test	r1s1	/sY-s/	[s(sY)]	[(sY)s]			L	W			L	L	L

(208) A1B2C2 lexicon updated for r1

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/?Y/	/?Y/	/-s/	/-Y/

To be sure that the language hypothesis is complete, the learner evaluates the error detection candidates for the words containing unset features: r3s1, r3s2, r4s1, and r4s2. These candidates, shown below, correspond to canonical mappings for r4s1 and r4s2 in the target language and are therefore consistent; again, the stress in the first syllable of a root is not contrastive if the second syllable is stressed underlyingly. These features can remain unset in the lexicon.

(209) Error detection candidates for r3 and r4

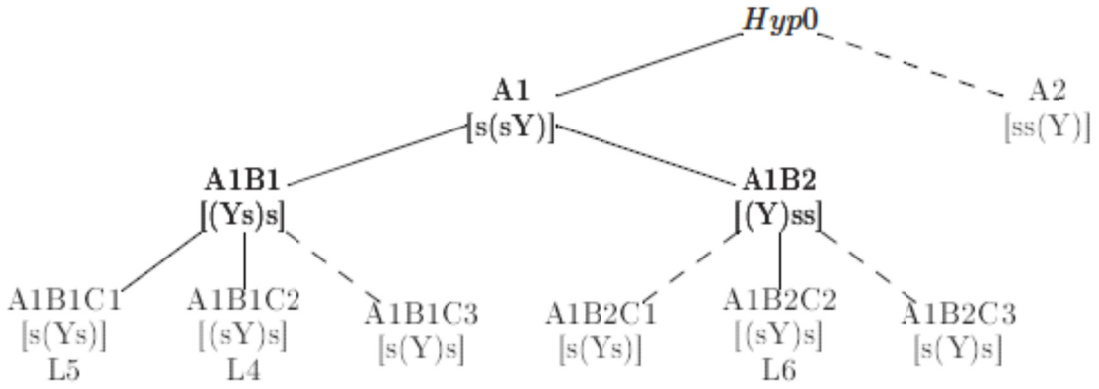
a. /**Y**Y-s/ → [(sY)s] r3s1, r4s1
b. /**Y**Y-Y/ → [s(sY)] r3s2, r4s2

This language hypothesis has concluded learning. The final support and lexicon remain as in (203) and (208), respectively.

3.3 CONCLUSION

The learning data in set 15 correspond to three target languages whose maps are identical in their overt forms. Over the course of this simulation, the learner successfully derives a consistent language hypothesis for each target. Four additional language hypotheses are created and rejected for being inconsistent. The tree in (210) shows the outcomes of all of the language hypotheses created during the course of learning.

(210) All language hypotheses



The simulation demonstrates how relationships among commitments enable the learner to simultaneously learn two kinds of hidden structure. Phonotactic learning provides a solid base of ranking information derived when the learner commits to

particular structural interpretations. In that learning stage, committing to an interpretation in effect means committing to an identity mapping containing that interpretation, which in turn means committing to the ranking conditions entailed by that mapping. The interpretation commitment has significant and lasting consequences. For example, the ranking information provided by structural commitments can affect when and how features get set. A1B1C2 and A1B2C2 make the same commitments for *ssY* and *sYs* (the A1 and C2 commitments), but their different commitments for *Yss* means that only the latter, which commits to [(Y)ss], can set all features by single forms. Because A1B1C2 lacks the ranking conditions entailed by that committed interpretation, the learner first must use a contrast pair to draw out a lexically informative inconsistency.

Making structural commitments continues to be important in the non-phonotactic learning stage. In order to set features of a morpheme in a given word using inconsistency detection, that word must have a committed interpretation. A1B2 illustrates this point. The lexicon must sufficiently distinguish the three root behaviors observed in the target language, but only words containing the underlyingly unstressed suffix s1 will enable the learner to set features in roots. Additionally, using inconsistency detection to set features requires a full structural description: an input and a specific output, not just an overt form. Therefore, all words containing s1 must have committed structural interpretations in order for the learner to set the necessary root features and, ultimately, to successfully learn the language. For A1B2, this requirement forces *sYs* to finally receive a committed interpretation during the non-phonotactic learning stage.

The Stress system generates a typology of 97 languages represented by 61 sets of learning data, including set 15, and the learner's progress through this data set is typical of the progress through the majority of the others. This simulation has demonstrated the CBL's successful use of committed information to exploit the mutual dependency between structural interpretations and underlying forms. Moreover, the CBL has succeeded at learning having processed and stored a reasonably small amount of information. For the seven language hypotheses created, RCD applies only 44 times (once for each ERC stored), and only 26 times within the three consistent language hypotheses.

4 PARADIGMATIC RELATIONSHIPS AND GLOBAL AMBIGUITIES

Languages can relate to one another in ways which may make it difficult for a learner to distinguish between them. Identifying and understanding the relationships between languages is crucial for ensuring that the learner has the means to successfully learn every language in the typology. For example, a language that contains a subset of the forms of another language gives rise to the "subset problem" mentioned in 1.2.1.3: positive evidence cannot distinguish between the two such languages. Biased Constraint Demotion (BCD) (Prince and Tesar 2004) or Low-Faithfulness Constraint Demotion (LFCD) (Hayes 2004) ranking biases arise as a response to this problem. By applying a low-faithfulness ranking bias, the learner enforces a more restrictive ranking until positive evidence supports a less restrictive one and derives different rankings for the subset language and its superset.

This chapter focuses on three other relationships between languages, including global ambiguities and the heretofore unrecognized relationship of paradigmatic equality. The chapter examines both how the CBL handles these relationships and what consequences follow from the learner's methods.

Sections 4.1 and 4.2 examine how paradigmatic relationships can interfere with the Commitment-Based Learner's standard learning procedures described in chapter 3. Section 4.1 introduces paradigmatic equality, in which two languages share all of the same morpheme behaviors. Paradigmatic equals are problematic because the learner cannot derive the ranking information required to distinguish one language from another using error-driven learning. In turn, inconsistency detection fails because of the missing

ranking information. Once again, it is the interactions between hidden structures that pose trouble for the learner, but paradigmatic equality presents a new twist: the learning data alone cannot decide between competing language hypotheses. This section proposes a procedure called ERC by Consistent Mismatch (ECM), which uses consistent surface-mismatched candidates to extract additional ranking information, allowing the learner to untangle the paradigmatic equals that both explain the data.

Section 4.2 discusses learning paradigmatic subsets, which contain a proper subset of the morpheme behaviors of another language in the typology (Tesar, to appear). Paradigmatic subsets likewise pose a problem for setting features by inconsistency detection, because although the learner derives a restrictive ranking by applying BCD, the support is nonetheless consistent with the less-restrictive superset language. Section 4.2 further shows that even if the learner acquires the ERCs to support the restrictive ranking, inconsistency detection may still fail to set features in some paradigmatic subsets. For these cases, the CBL learns the lexicon using the Fewest Set Features procedure (Tesar, to appear).

With section 4.3, the focus turns to the learner's response to languages that relate in multiple ways to other languages in the typology. Section 4.3 presents a language that is both a paradigmatic subset and a paradigmatic equal. This section demonstrates that the separate learning complications of each relationship need and can be overcome using the procedures described in 4.1 and 4.2. Section 4.4 is concerned with global ambiguities, in particular with the interactions of global lexical ambiguity, evinced by paradigmatic equals, and global surface ambiguity, in which two languages share the overt forms of all

words. Section 4.4 shows that the Commitment-Based Learner will learn all globally ambiguous languages from a data set, so that learning data for a language with a paradigmatic equal will also yield language hypotheses corresponding to all globally surface ambiguous counterparts of that paradigmatic equal.

Finally, section 4.5 considers some reasons and criteria for selecting a single language hypothesis out of all the consistent branches when learning ends.

4.1 PARADIGMATIC EQUALS AND GLOBAL LEXICAL AMBIGUITY

Paradigmatic relationships are defined by the morpheme behaviors evinced between two languages. This section will begin by first examining the maps and morpheme behaviors of two languages that are paradigmatic equals, before launching into more specific definitions of this relationship and global lexical ambiguity.

The central languages for this section are L75 and L76, which appear in (211) and (213), each followed by a ranking that will generate it. Both languages require exhaustive parsing, and their preference for left-alignment leaves the degenerate foot always at the left edge. Suffixes are contrastive in these languages. Observe that one language is no more restrictive than the other, as each has a phonotactic inventory of three forms: [(X)(sY)], [(Y)(sX)], and [(X)(Ys)].

(211) L75

r1 =/ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(X)(sY)]	[(Y)(sX)]	[(X)(Ys)]	[(Y)(sX)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

(212) {PARSE-σ, *LAPSE} >> {AFL, FT-BIN} >> MAXSTRESS >> IAMB >> {RMOST, FNF} >> LMOST

(213) L76

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(X)(sY)]	[(Y)(sX)]	[(X)(Ys)]	[(X)(Ys)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

(214) { PARSE-σ, *LAPSE} >> {AFL, FT-BIN} >> MAXSTRESS >> RMOST >> {IAMB, LMOST} >> FNF

R4s1 manifests the only overt difference between the two languages. Its input, /YY-s/, entails a violation of MAXSTRESS, and the languages crucially differ on which syllable of the root r4 is realized faithfully. For L75, the ranking IAMB >> {RMOST, FNF} causes the first syllable to surface faithfully in a unary head-foot at the left edge, and the requirement for exhaustive parsing forces an iambic secondary foot to the right, yielding [(Y)(sX)]. In this language, stress in the second syllable of the root is neutralized if the first syllable is underlyingly +stress.

Morphemes that behave alike in L75 are grouped together in the chart below. L75 evinces two suffix behaviors and three root behaviors. Roots with an unstressed initial syllable contrast for the values of the second syllable, yielding two of the three behaviors. The third behavior is evinced by roots with a stressed initial syllable. Thus, r2 /Ys/ and r4 /YY/ behave alike, differently from both r1 /ss/ and r3 /sY/.

(215) L75 – like morpheme grouped together

r1 =/ss/	r2 = /Ys/ r4 = /YY/	r3 = /sY/	
[(X)(sY)]	[(Y)(sX)]	[(X)(Ys)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

By contrast, in L76 the ranking RMOST >> IAMB allows the output of /YY-s/ to preserve stress on the second root syllable by parsing the head-foot as a trochee at the right edge, with a secondary degenerate foot aligned to the left: [(X)(Ys)]. Like L75, L76 distinguishes two suffix behaviors and three root behaviors, shown in (216). The key difference is that in L76 stress in the first syllable of the root is neutralized if the second syllable is +stress underlyingly; thus, roots r3 /sY/ and r4 /YY/ behave alike, and differently from both r1 /ss/ and r2 /Ys/.

(216) L76 – like morpheme behaviors grouped together

r1 = /ss/	r2 = /Ys/	r3 = /sY/ r4 = /YY/	
[(X)(sY)]	[(Y)(sX)]	[(X)(Ys)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

A comparison of (215) and (216) reveals that the languages exhibit all the same morpheme behaviors. In fact, if (215) were laid over (216), the languages would look identical, as shown in (217).

(217) L75 and L76 overlaid

/ss/	/Ys/ (/YY/ L75)	/sY/ (/YY/ L76)	
[(X)(sY)]	[(Y)(sX)]	[(X)(Ys)]	/-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	/-Y/

There is one final, and crucial, observation to be made about the morpheme behaviors in these languages: they are identical in structural interpretations as well as overt realizations. For this reason, L75 and L76 are paradigmatic equals.

(218) Paradigmatic equal

> Languages L*A* and L*B* are paradigmatic equals if the set of morpheme behaviors, including structural interpretations, in L*A* is identical to the set of morpheme behaviors in L*B*.

What is remarkable about paradigmatic equals, and problematic for using inconsistency detection to set features, is that from the learner's perspective the data for these different languages appear to be the same. This similarity is deeper than the similarity evinced in chapter 3 by the globally surface ambiguous languages L4, L5, and L6. Those languages have the same overt form for each morphological word, but the overall morpheme behaviors differ when structural interpretations are included. For paradigmatic equals, what differs are the specific input-output mappings associated with each behavior. The problem for the learner using inconsistency detection now becomes clearer: when different input-output mappings can produce the same behavior, there will be no inconsistency. In short, paradigmatic equals are problematic because they exhibit global lexical ambiguity, defined as in (219).

(219) Global lexical ambiguity

> Languages L*A* and L*B* are globally lexically-ambiguous if they have the same morpheme behaviors, including structural interpretations, but differ only by which underlying forms of the rich base produce which behaviors.

Thus, L75 and L76 are globally lexically ambiguous because, while particular inputs behave differently in the languages due to the differing rankings, the set of behaviors themselves remains the same. This characteristic distinguishes global lexical ambiguity as the term is applied here from other conceivable uses of the term. For comparison, consider a language with fully predictable stress, like L78[23], which maps all inputs of the rich base to [(X)(sY)]. There is a sense in which the local lexical ambiguity of each word – the uncertainty of the word's underlying form – is a global ambiguity, because the ranking permits each word to have any underlying form. L78 therefore could have a lexicon that reflects the rich base or a lexicon in which every input simply matches the surface form, or any lexicon between these extremes. However, L78 is not globally lexically-ambiguous. It is the only language in which all outputs are [(X)(sY)]; there is no way to achieve the same neutralization behavior with the ranking from a different skeletal basis, regardless of the underlying forms assigned in the lexicon. The opposite is true for L75 and L76: the ranking of one language will produce exactly the same behaviors as the other language, as long as the lexicon changes also. It is this characteristic that so complicates learning a paradigmatic equal.

As paradigmatic equals, L75 and L76 have a very special relationship to one another. To review, they are not distinguished by restrictiveness, as they have the same phonotactic inventory. For the same reason, they are not globally surface ambiguous: L75 and L76 do not include different interpretations of the same overt forms, they have the same interpretations of the same overt forms. Yet, the simple identity of the phonotactic inventories does not sufficiently characterize the paradigmatic equality relationship.

[23] L78 is discussed further in section 4.5.

Consider language L65 below, which like L75 and L76 has an inventory consisting of [(X)(sY)], [(Y)(sX)], and [(X)(Ys)].

(220) L65

r1 =/ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(Y)(sX)]	[(Y)(sX)]	[(X)(Ys)]	[(Y)(sX)]	s1 = /-s/
[(X)(sY)]	[(Y)(sX)]	[(X)(sY)]	[(Y)(sX)]	s2 = /-Y/

L65 has contrastive stress and distinguishes three root behaviors, illustrated in (221). Stress is contrastive for the initial root syllables, and contrastive for the second root syllable only if the first is stressed. Thus, /Ys/ and /YY/ behave alike, differently from both /ss/ and /sY/.

(221) L65 – like morpheme grouped together

r1 =/ss/	r2 = /Ys/ r4 = /YY/	r3 = /sY/	
[(Y)(sX)]	[(Y)(sX)]	[(X)(Ys)]	s1 = /-s/
[(X)(sY)]	[(Y)(sX)]	[(X)(sY)]	s2 = /-Y/

L65 therefore resembles L75 and L76 superficially, with its two suffix behaviors and three root behaviors, and the same phonotactic inventory. Moreover, its morphemes group together in just the same way as those in L75, with r2 and r4 behaving alike and r1 and r3 each evincing a different behavior. However, L65 is not the paradigmatic equal of L75 or L76. Contrast the compressed morpheme behaviors of L75 and L76 repeated below in (222), with those of L65 in (223). Differences in the surface alternation behaviors, indicated by the shaded cells, clearly reveal that these languages are not paradigmatic equals.

(222) L75 and L76 compressed for r4

/ss/	/Ys/	/sY/	
[(X)(sY)]	[(Y)(sX)]	[(X)(Ys)]	/-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	/-Y/

(223) L65 compressed for r4

/ss/	/Ys/	/sY/	
[(Y)(sX)]	[(Y)(sX)]	[(X)(Ys)]	/-s/
[(X)(sY)]	[(Y)(sX)]	[(X)(sY)]	/-Y/

In order to successfully learn either L75 or L76, the learner must set the values of all contrastive features in addition to deriving an appropriate ranking. The features that must be set for both targets include all the stress features in the suffixes and in r1. Additionally, learning L75 further requires distinguishing r3 from r2 and r4, such as by setting both features in r3 and the first syllable's stress feature in r2 and r4 to produce the lexicon in (224). Similarly, learning L76 requires distinguishing r2 from r3 and r4, which can be accomplished by setting both features in r2 and the second syllable's stress feature in r3 and r4, as in (225).

(224) Idealized lexicon for L75

r1	r2	r3	r4	s1	s2
/ss/	/Y?/	/sY/	/Y?/	/-s/	/-Y/

(225) Idealized lexicon for L76

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/?Y/	/?Y/	/-s/	/-Y/

To derive rankings that will generate each target, the learner needs to determine the relative rankings of RMOST and IAMB, the conflict which distinguishes L75 from L76. The only mappings that are informative about this conflict include the input /YY-s/. Therefore, in order to learn the ranking, the learner must detect an error for a candidate with the input /YY-s/.

This mandate seems simple enough, but for paradigmatic equals, uncertainty about the ranking and lexicon may not be so easily unraveled. Here the problem of global lexical ambiguity emerges in full force. In this example, learning that /YY-s/ is the input for one of the observed outputs would lead to an informative error resolving the conflict between RMOST and IAMB; however, the inconsistency-detection strategies for setting features cannot ever learn this input. Because the test candidates evaluated to set the remaining unset features all include the input /YY-s/, the learner needs to know the relative ranking of RMOST and IAMB to detect the lexically-informative inconsistency. Thus, the learner of this paradigmatic equal is trapped in a cycle of persistent uncertainty: the correct underlying form is needed in order to detect the error that will illuminate the missing ranking information, but that same ranking information is needed to detect the inconsistency that will reveal the correct underlying form.

This cycle is a consequence of global lexical ambiguity: the data will support either paradigmatic equal as an explanation and therefore cannot single out just one language as correct. Completing learning requires that the learner commit to information that supports one language hypothesis over another, but what kind of commitment, and on what grounds? The remainder of this section works through the details of learning L75 to

expose where global lexical ambiguity impedes further progress by inconsistency detection. This section also proposes handling persistent uncertainty by using the procedure ERC by Consistent Mismatch (ECM) to derive ranking information that will make inconsistency detection again viable for learning.

4.1.1 LEARNING L75

The learning data for L75 are shown below in the order the learner observes them in this simulation. LgHyp75 is the corresponding language hypothesis for this target. The support produced for LgHyp75 from the phonotactic data only is shown in (227), with the ranking derived by BCD given in (228).

(226) L75 learning data

XsY r1s1	*XsY* r1s2	*YsX* r2s1	*XsY* r2s2	*XYs* r3s1	*XsY* r3s2	*YsX* r4s1	*XsY* r4s2

(227) LgHyp75 support from phonotactics

ERC#	Morph. word	Input	Winner	Loser	PARSYL	*LAPSE	AFL	FTBIN	MAXSTR	LMOST	RMOST	IAMB	FNF
a. 1P	r1s1	/ss-Y/	[(X)(sY)]	[s(Ys)]	W			L	W			W	L
b. 3P	r1s1	/ss-Y/	[(X)(sY)]	[(sY)s]	W		L	L	W	L	W		L
c. 2P	r1s1	/ss-Y/	[(X)(sY)]	[(Y)(sX)]					W	L	W		
d. 4P	r2s1	/Ys-s/	[(Y)(sX)]	[(X)(sY)]					W	W	L		
e. 5P	r3s1	/sY-s/	[(X)(Ys)]	[(Y)(sX)]					W	L	W	L	W
f. 6P	r1s1	/ss-Y/	[(X)(sY)]	[(Y)(Xs)]					W	L	W	W	L
g. 7P	r2s1	/Ys-s/	[(Y)(sX)]	[(Y)(Xs)]								W	L

(228) {PARSE-σ, *LAPSE} >> {AFL, FT-BIN} >> MAXSTRESS >> {LMOST, RMOST, IAMB} >> FNF

With seven W-L pairs added from phonotactic information, the learner has derived many of the crucial ranking conditions of the target. In fact, this support enables the learner to set the values of seven features in LgHyp75 from single forms once the phonotactic learning stage ends. The resulting lexicon appears in (229).

(229) LgHyp75 lexicon

r1	r2	r3	r4	s1	s2
/ss/	/Y?/	/?Y/	/Y?/	/-s/	/-Y/

During the process of setting features by single forms, the learner adds two new W-L pairs to the support, in (230). W-L pair 8 in (230)h is added after the learner sets s1 to /-s/ and detects an error on r1s1 /ss-s/[(X)(sY)] using the BCD ranking from (228). This W-L pair contributes the ranking condition RMOST >> LMOST. Then, after setting the first syllable of r2 to +stress, the learner detects an error on r2s2 /Ys-Y/[(X)(sY)]. This error is detected under the ranking derived by applying the low-markedness bias to the first eight W-L pairs. That low-markedness ranking appears in (231), and the resulting W-L pair 9 is shown in (230)c. Applying BCD to the support after adding these new W-L pairs produces the ranking in (232).

(230) LgHyp75 support updated during single form learning

ERC#	Morph. word	Input	Winner	Loser	PARSE-σ	*LAPSE,	AFL	FT-BIN	MAXSTR	RMOST	IAMB	LMOST	FNF
a. 1P	r1s1	/ss-Y/	[(X)(sY)]	[s(Ys)]	W			L	W		W		L
b. 3P	r1s1	/ss-Y/	[(X)(sY)]	[(sY)s]	W		L	L	W	W		L	L
c. 9	r2s2	/Ys-Y/	[(X)(sY)]	[s(sY)]	W	W		L					L
d. 2P	r1s1	/ss-Y/	[(X)(sY)]	[(Y)(sX)]					W	W		L	
e. 4P	r2s1	/Ys-s/	[(Y)(sX)]	[(X)(sY)]					W	L		W	
f. 5P	r3s1	/sY-s/	[(X)(Ys)]	[(Y)(sX)]					W	W	L	L	W
g. 6P	r1s1	/ss-Y/	[(X)(sY)]	[(Y)(Xs)]					W	W	W	L	L
h. 8	r1s1	/ss-s/	[(X)(sY)]	[(Y)(sX)]						W		L	
i. 7P	r2s1	/Ys-s/	[(Y)(sX)]	[(Y)(Xs)]							W		L

(231) Low-markedness ranking detects error on r2s2 /Ys-Y/[(X)(sY)]

MAXSTRESS >> {PARSE-σ, *LAPSE, AFL, FT-BIN, RMOST, IAMB} >> {LMOST, FNF}

(232) LgHyp75 updated ranking

{PARSE-σ, *LAPSE} >> {AFL, FT-BIN} >> MAXSTRESS >> {RMOST, IAMB} >> {LMOST, FNF}

W-L pair 9 arises from a conflict among the markedness constraints grouped together in the second stratum of the ranking in (231). The mapping tested, r2s2 /Ys-Y/[(X)(sY)], incurs one violation of MAXSTRESS, but no other candidate incurs fewer; thus, any error for this candidate must be due to conflict between markedness constraints. W-L pair 9 contributes a new disjunction – at least one of PARSE-σ and *LAPSE must dominate both FT-BIN and FNF – but this information does not alter the stratified hierarchy by BCD nor indicate a conflict involving MAXSTRESS. At the end of single-form learning, the learner

has acquired evidence from W-L pair 8 that RMOST must dominate LMOST, but, as expected, has not determined the relative ranking of RMOST and IAMB.

The learner must still distinguish r2 and r4, which behave alike, from r3 in the lexicon. Only words containing s1 can be informative about alternations in these roots, as root stress in the target L75 is neutralized when the suffix is underlyingly stressed. The pertinent test candidates for r2 and r3 are shown in (233), with the syllable containing the unset test feature outlined.

(233) Test candidates for unset features in r2 and r3

a. r2s1 /Y**Y**-s/ → [(Y)(sX)]
b. r3s1 /**Y**Y-s/ → [(X)(Ys)]

Unfortunately, these candidates are testing which syllable of the root gets neutralized for the input /YY-s/, and this is the very question whose answer distinguishes L75 from its paradigmatic equal, L76. Neither candidate is informative. Candidate (233)a is consistent with L75. Given the correct ranking, the stress feature of the second syllable of r2 could remain unset in the lexicon of LgHyp75, as stress in the second syllable of the root is contrastive in the target only if the first syllable is unstressed.

By the same reasoning, both stress features of r3 should be set in the lexicon to distinguish it from r1 on the one hand and r2 and r4 on the other, yet candidate (233)b is uninformative about r3's remaining unset feature. Despite being inconsistent with the target, this candidate is consistent with the support in (230) because the ranking of RMOST and IAMB remains at issue. In other words, there is a language in the space of

possible languages that meets the ranking conditions imposed by the support and by test candidate (233)b; that language is L76. It is the existence of this paradigmatic equal that prevents the learner from setting any of these unset features.

Leaving the features unset in the lexicon is not an option. R2s1 and r3s1 differ in their surface forms, and their underlying forms must reflect that difference. The learner determines that something more must be learned in this language hypothesis by performing error detection tests on r2s1, r3s1, and r4s1. The violation tableau below shows the result of error detection. While these words are different, the input that maximally differs from the surface form is the same for each word, /YY-s/, and each candidate with this input fails error detection. As shown in (234), RMOST and IAMB conflict in the fourth stratum, with each preferring a different structural interpretation, as expected.

(234) R2s1, r3s1, and r4s1 all fail error detection

Input	Output	PARSE-σ	*LAPSE	AFL	FT-BIN	MAXSTR	RMOST	IAMB	LMOST	FNF
a. /YY-s/	[(Y)(sX)]	0	0	1	1	1	**2**	**0**	0	2
b.	[(X)(Ys)]	0	0	1	1	1	**0**	**1**	1	1

The error detection test exposes the primary problem with leaving these features unset: what happens if the learner attempts to use the underlying form /YY-s/? The current ranking will yield an error if given this input. In order to distinguish these words, either r3 must be set to /sY/, allowing r2 and r4 to each leave one feature unset, or else both r2 and r4 must be fully set in the lexicon, allowing r3 to leave one feature unset.

The learner is at an impasse. Single-form learning cannot set a feature because the test candidates are either consistent with the target language or its paradigmatic equal. R2s1 and r3s1 form a contrast pair, but it is not informative either. Regardless of where the disparity is, the test candidate and its unchanged counterpart will be consistent with one of the targets. For example, in (235)a, r2 contains the disparity, and the resulting two candidates are consistent with L75. In (235)b, r3 contains the disparity, and the resulting candidates are consistent with L76.

(235) Test candidates for contrast pair r2s1, r3s1

a. r2s1 /Y**Y**-s/ → [(Y)(sX)]
   r3s1 /sY-s/ → [(X)(Ys)]

b. r2s1 /Ys-s/ → [(Y)(sX)]
   r3s1 /**Y**Y-s/ → [(X)(Ys)]

It is clear that inconsistency detection can do no more to set features until the language hypothesis receives new information.

4.1.2 ERC BY CONSISTENT MISMATCH

The learner has already extracted all of the ranking information available from error-driven learning and all of the lexical information available from inconsistency detection. The data cannot provide any other information by these methods. This section offers another source of ranking information, a consistent mismatch candidate, such as those in (233), and introduces ERC by Consistent Mismatch (ECM) as the procedure for procuring this information.

Using the test illustrated in (234), the learner has detected errors on the mismatch candidates of r2s1, r3s1, and r4s1, in which the sole unset feature of each word is set to mismatch its surface form. These are minimal mismatch candidates, as defined below.

(236) Minimal mismatch candidate

A minimal mismatch candidate includes exactly one unset feature whose underlying value mismatches its surface value; all other unset features in the candidate match their underlying values to their surface values.

Failing error detection indicates that the ranking must be refined, remaining unset features must be set in the lexicon, or both. Additionally, the learner has determined that these same mismatch candidates are consistent, as the candidates in (233) have failed to yield any inconsistency to support setting the remaining unset feature in either form. These facts offer a solution to the problem of persistent uncertainty. By adopting one of these mismatch candidates as a winner, the learner can be certain to identify new ranking information consistent with the current support. This information will allow the leaner to disambiguate the data of the paradigmatic equals enough to continue learning using the routine error- and inconsistency-detection procedures described in chapters 2 and 3. Pseudocode for the ECM procedure is given below.

(237) ERC by Consistent Mismatch (ECM) procedure[24]

```
DEF ECM(lang_hyp)
    FOR each word in lang_hyp that fails error detection
      Create the minimal mismatch candidates for the word[25]
      FOR each minimal mismatch candidate
        IF the minimal mismatch candidate is consistent THEN
          IF the minimal mismatch candidate adds a new ERC THEN
              Commit to this mapping
              Update lang_hyp
              BREAK
          ENDIF
        ENDIF
      ENDFOR
    ENDFOR
END
```

In LgHyp75, the words r2s1, r3s1, and r4s1 all fail error detection. Following the procedure in (237), the learner determines that the mismatch candidate /YY-s/[(Y)(sX)] derived from r2s1 (or r4s1) is consistent. The error anticipated by the CTie in (234) now contributes W-L pair 10 in (238)i, which resolves the ranking of RMOST and IAMB. The new stratified hierarchy, (239), matches the stratified hierarchy of L75, repeated in (240).

[24] This procedure is a simplification of several different methods included in the actual Ruby code given in Appendix B.

[25] See 4.1.3and 4.3.3 for discussion of using maximal versus minimal mismatch candidates.

(238) LgHyp75 support updated by consistent mismatch candidate /YY-s/[(Y)(sX)]

ERC#	Morph. word	Input	Winner	Loser	PARSE-σ	*LAPSE	AFL	FT-BIN	MAXSTR	IAMB	RMOST	FNF	LMOST
a. 1P	r1s1	/ss-Y/	[(X)(sY)]	[s(Ys)]	W			L	W	W		L	
b. 3P	r1s1	/ss-Y/	[(X)(sY)]	[(sY)s]	W		L	L	W		W	L	L
c. 9	r2s2	/Ys-Y/	[(X)(sY)]	[s(sY)]	W	W		L				L	
d. 2P	r1s1	/ss-Y/	[(X)(sY)]	[(Y)(sX)]					W		W		L
e. 4P	r2s1	/Ys-s/	[(Y)(sX)]	[(X)(sY)]					W		L		W
f. 5P	r3s1	/sY-s/	[(X)(Ys)]	[(Y)(sX)]					W	L	W	W	L
g. 6P	r1s1	/ss-Y/	[(X)(sY)]	[(Y)(Xs)]					W	W	W	L	L
h. 7P	r2s1	/Ys-s/	[(Y)(sX)]	[(Y)(Xs)]						W		L	
i. 10	r2s1	/YY-s/	[(Y)(sX)]	[(X)(Ys)]						W	L	L	W
j. 8	r1s1	/ss-s/	[(X)(sY)]	[(Y)(sX)]							W		L

(239) LgHyp75

{PARSE-σ, *LAPSE} >> {AFL, FT-BIN} >> MAXSTRESS >> IAMB >> {RMOST, FNF} >> LMOST

(240) L75

{PARSE-σ, *LAPSE} >> {AFL, FT-BIN} >> MAXSTRESS >> IAMB >> {RMOST, FNF} >> LMOST

This ranking allows the learner to complete the language hypothesis. The test candidate for r3s1, /**Y**Y-s/[(X)(Ys)], is inconsistent with the new support – in fact, it is the loser of W-L pair 10. The test candidates for r2s1 and r4s1 are consistent, as their mappings are identical to the winner of W-L pair 10. The updated lexicon, (241), now matches the idealized lexicon for L75 from (224), repeated below.

(241) LgHyp75 lexicon

r1	r2	r3	r4	s1	s2
/ss/	/Y?/	/sY/	/Y?/	/-s/	/-Y/

(242) Idealized L75 lexicon

r1	r2	r3	r4	s1	s2
/ss/	/Y?/	/sY/	/Y?/	/-s/	/-Y/

By adopting /YY-s/[(Y)(sX)] as a winner, the learner in effect resolves that the learning data are from target L75; however, the learner has no reason a priori to adopt one consistent mismatch candidate over another. These learning data are globally lexically-ambiguous between a solution corresponding to L75 and one corresponding to L76. Therefore, the learner could just as well have selected the other consistent mismatch candidate from (233), /YY-s/[(X)(Ys)], corresponding to r3s1. In that case, the final language hypothesis would correspond to L76.

The support in (243) derives from this alternative solution. The new W-L pair in this support, (243)h, commits to /YY-s/[(X)(Ys)] as a winner, and from the CTie error illustrated in (234) adopts /YY-s/[(Y)(sX)] as its loser. This alternative W-L pair 10 entails that RMOST must dominate IAMB. Applying BCD to the support produces a same stratified hierarchy, (244), for this alternative language hypothesis that matches the ranking of L76 given in (214).

(243) Alternative LgHyp75 support

ERC#	Morph. word	Input	Winner	Loser	PARSE-σ	*LAPSE	AFL	FT-BIN	MAXSTR	RMOST	IAMB	LMOST	FNF
a. 1P	r1s1	/ss-Y/	[(X)(sY)]	[s(Ys)]	W			L	W		W		L
b. 3P	r1s1	/ss-Y/	[(X)(sY)]	[(sY)s]	W		L	L	W	W		L	L
c. 9	r2s2	/Ys-Y/	[(X)(sY)]	[s(sY)]	W	W		L					L
d. 4P	r2s1	/Ys-s/	[(Y)(sX)]	[(X)(sY)]					W	L		W	
e. 2P	r1s1	/ss-Y/	[(X)(sY)]	[(Y)(sX)]					W	W		L	
f. 5P	r3s1	/sY-s/	[(X)(Ys)]	[(Y)(sX)]					W	W	L	L	W
g. 6P	r1s1	/ss-Y/	[(X)(sY)]	[(Y)(Xs)]					W	W	W	L	L
h. 10	r3s1	/YY-s/	[(X)(Ys)]	[(Y)(sX)]						W	L	L	W
i. 8	r1s1	/ss-s/	[(X)(sY)]	[(Y)(sX)]						W		L	
j. 7P	r2s1	/Ys-s/	[(Y)(sX)]	[(Y)(Xs)]							W		L

(244) { PARSE-σ, *LAPSE} >> {FT-BIN, AFL} >> MAXSTRESS >> RMOST >> {IAMB, LMOST} >> FNF

This alternative support leads to alternative lexical information. This time, the test candidates for r2s1 and r4s1, /Y**Y**-s/[(Y)(sX)], are inconsistent, enabling the learner to set both roots to /Ys/ in the lexicon. Now the unset feature in r3 can remain unset. The final lexicon appears in (245).

(245) Alternative LgHyp75 lexicon, corresponding to L76

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/?Y/	/Ys/	/-s/	/-Y/

4.1.3 ASSESSING ECM

To review, the learner is unable to set all necessary features by inconsistency detection because the crucial ranking and lexical information each depends on knowing which output the input /YY-s/ maps to in the target language. By committing to either

consistent minimal mismatch candidate as a winner, the learner takes an uninformed stand on the output and forces the acquisition of new ranking information that resolves the conflict between RMOST and IAMB. The final lexical information follows as a result, regardless of which way the conflict is decided.

ECM is successful here, but it makes a strong and potentially dangerous departure from other methods incorporated in the CBL by requiring that the learner commit to uncertain information. Consider how the CBL handles uncertainty otherwise. To resolve structural uncertainty, the CBL applies the Inconsistency Detection Learner, or IDL, and creates language hypothesis branches. Although the learner cannot be certain that a particular branch is correct for the target, as long as every possible interpretation is included among the branches the learner can be certain that some branch is correct for the target. To resolve lexical uncertainty, the CBL applies the Output-Driven Learner (ODL). The learner can be certain that a feature value set by inconsistency detection could not be set to any other value and remain consistent with the language hypothesis. By contrast, the learner cannot be certain that the consistent mismatch candidate is a winner in the target language. It is this very uncertainty – that the target could have a different winner for that mismatch input – which provokes the learner to commit to the ranking conditions imposed by a consistent mismatch candidate in the first place.

The danger of making an uncertain commitment is that it could be inconsistent with information learned later and, lacking the insurance of branching, the learner might fail to derive any language hypothesis at all consistent with the observed data. The risk depends on the unresolved conflicts remaining when the learner selects a consistent mismatch

candidate for inclusion in the support. The learner adds W-L pairs to the support only as needed to prevent errors, and as a result the support may not include all of the ranking conditions entailed by its winners. Based on this incomplete information, a mismatch candidate could be consistent with the current support even though it is inconsistent with the full set of ranking conditions entailed by the other words. A later round of error-driven learning would reveal the fatal inconsistency.

Although the CBL avoids branching as a method of learning underlying forms, branching could offer a solution to the problem described above. A consistent mismatch candidate adds lexical information indirectly, by adding ranking information that contributes to lexically-informative inconsistencies. Viewed in this way, adopting a consistent mismatch candidate as a winner is not unlike choosing a structural interpretation for an overt form and adopting its identity map as a winner during phonotactic learning: in both cases, the learner commits to an input-output mapping without knowing for certain that the mapping is consistent with the target language or even with all the ranking conditions entailed by the other committed mappings. Yet, this approach to phonotactic learning is tenable because the learner simultaneously evaluates all possible interpretations in separate language hypotheses, and if an interpretation is consistent with the prior commitments, the learner will find it. Similarly, extending branches for each consistent mismatch winner might enable the learner to derive at least one consistent language hypothesis for the data. For the example described in this

section, by branching the learner would ultimately derive language hypotheses corresponding to both L75 and its paradigmatic equal, L76.[26]

Setting aside this concern over making an uncertain commitment, there is also a question to ask about the implementation of ECM; specifically, what effect, if any, would choosing a maximal mismatch candidate have on learning with ECM? A maximal mismatch candidate is defined as in (246).

(246) Maximal mismatch candidate

> A maximal mismatch candidate has surface-mismatched underlying values for all unset features.

To learn L75, the learner commits to the consistent minimal mismatch candidate – one like a test candidate used to set a feature by inconsistency detection. In an output-driven map, committing to a mismatch candidate entails a commitment to all mappings with inputs more similar to the output. Committing to a minimal mismatch candidate is therefore a conservative choice, while the aggressive choice is to commit to the maximal mismatch candidate. The advantage of the aggressive choice is that it affords the learner all of the ranking information of these entailed mappings, but this could be a disadvantage if later information should reveal that only some of those mappings should be consistent.

[26] The branching procedure would need to be sensitive to the ranking information imparted by the ERCs to prevent the creation of identical branches. In LgHyp75, consistent mismatch candidates could be derived from r2s1, r3s1, and r4s1, but those derived from r2s1 and r4s1 would add identical W-L pairs and therefore yield identical branches. Instead of branching for each consistent mismatch candidate, then, the branching procedure could instead create branches for each unique ERC derived from consistent mismatch candidates.

For LgHyp75 and LgHyp76, these choices happen to be identical: the maximal mismatches have only the one disparity that the minimal mismatches have. However, there are nine pairs of paradigmatic equals generated in the Stress system, and for some of these, the aggressive and conservative options differ. The learner can safely commit to the aggressive, maximal mismatch candidate in each of these cases, but these are not strong evidence that the aggressive choice is always safe, nor even warranted.

For example, consider the paradigmatic equals L39 and L51. Both languages parse only binary trochees, but they differ on the default alignment of the head-foot: L39 aligns the head-foot to the right, L51 to the left. Maps and stratified hierarchies by BCD for each language appear below. The shaded cells indicate the default behavior in each map.

(247) L39

r1 =/ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(Ys)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s1 = /-s/
[s(Ys)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s2 = /-Y/

(248) {FNF, FT-BIN} >> {IAMB, PARSE-σ} >> MAXSTRESS >> {RMOST, *LAPSE} >> {AFL, LMOST}

(249) L51

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(Ys)s]	[(Ys)s]	[s(Ys)]	[(Ys)s]	s1 = /-s/
[(Ys)s]	[(Ys)s]	[s(Ys)]	[(Ys)s]	s2 = /-Y/

(250) {FNF, FT-BIN} >> {IAMB, PARSE-σ} >> MAXSTRESS >> {AFL, LMOST}>> { RMOST, *LAPSE}

Suppose the learner has received the data of L39.[27] After phonotactic learning, the support appears as in (251). At this point, the learner reaches the familiar impasse of paradigmatic equals. The support has an unresolved conflict among the constraints in the bottom stratum, but the known data cannot resolve it. The lexicon, in which all features are currently unset, (252), requires that both features of r2 be set to distinguish it from the other roots, and for the remaining roots to each set one feature to distinguish themselves from r2; however, the unresolved conflict in the ranking prevents the necessary lexically-informative inconsistencies. Because neither root behavior is yet distinguished in the lexicon for any root, all words currently fail error detection. The learner therefore can now apply ECM to find additional ranking information from a consistent mismatch candidate.

(251) LgHyp39 support

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	FNF	PARSE-σ	IAMB	MAXSTR	LMOST	RMOST	AFL	*LAPSE
a. 2P	r1s1	/sY-s/	[s(Ys)]	[(Y)(Xs)]	W	W	L		W	L	W		
b. 1P	r1s1	/sY-s/	[s(Ys)]	[s(sY)]		W		L	W				W
c. 3P	r1s1	/sY-s/	[s(Ys)]	[(Ys)s]					W	L	W	L	W
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(Ys)]					W	W	L	W	L

(252) LgHyp39 lexicon

r1	r2	r3	r4	s1	s2
/??/	/??/	/??/	/??/	/-?/	/-?/

[27] Because L39 is a paradigmatic subset as well as a paradigmatic equal, it is discussed in detail in 4.3.

Since all words fail error detection, consider first the maximal mismatch of r1s1, /Ys-Y/ → [s(Ys)], shown as the winner in (253)c. This mapping incurs an additional violation of MAXSTRESS than does the loser /Ys-Y/ → [(Ys)s]. As a result, the W-L pair is inconsistent with W-L pairs 3P and 4P, as shown in (253).

(253) Maximal mismatch r1s1 /Ys-Y/ → [s(Ys)] is inconsistent

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	FNF	PARSE-σ	IAMB	MAXSTR	LMOST	RMOST	AFL	*LAPSE
a. 3P	r1s1	/sY-s/	[s(Ys)]	[(Ys)s]					W	L	W	L	W
b. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(Ys)]					W	W	L	W	L
c. test	r1s1	/Ys-Y/	[s(Ys)]	[(Ys)s]					L	L	L	W	W

The maximal mismatch of r2s1 is /sY-Y/ → [(Ys)s], and the tableau in (254) demonstrates that this mapping is inconsistent with the current support for the same reason as above.

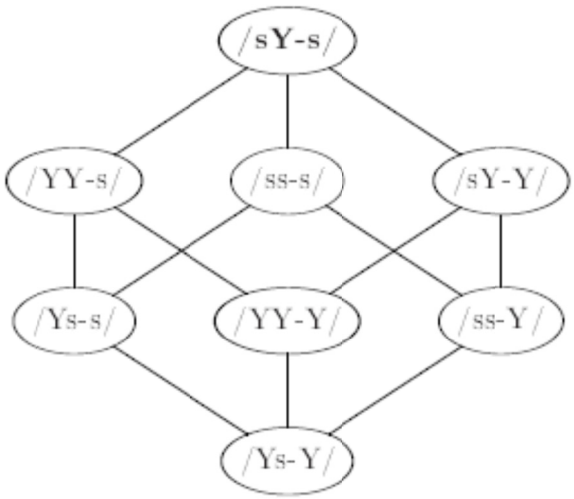
(254) Maximal mismatch r2s1 /sY-Y/ → [(Ys)s] is inconsistent

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	FNF	PARSE-σ	IAMB	MAXSTR	LMOST	RMOST	AFL	*LAPSE
a. 3P	r1s1	/sY-s/	[s(Ys)]	[(Ys)s]					W	L	W	L	W
b. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(Ys)]					W	W	L	W	L
c. test	r2s1	/sY-Y/	[(Ys)s]	[s(Ys)]					L	W	L	W	L

Because inconsistent W-L pairs cannot be productively added to the support, these maximal mismatch candidates have provided no help for learning the language, but the minimal mismatch candidates for these words turn out to be more informative. The complete lexical space for r1s1 appears below, with the minimal mismatch candidates in

the second row. By comparison with the map of L39 in (247), it is clear that any of the minimal mismatch candidates will be consistent: each of the minimal mismatch inputs maps to the output [s(Ys)] in L39.

(255) Lexical space of r1s1



The tableau in (256) includes the updated support with a minimal mismatch candidate, /ss-s/[s(Ys)], added as W-L pair 5 in (256)e. The ranking in (257) now matches the target's ranking; the minimal mismatch candidate has correctly resolved the conflict between RMOST, *LAPSE, LMOST and AFL.

(256) LgHyp39 support updated with minimal mismatch candidate

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	FNF	PARSE-σ	IAMB	MAXSTR	RMOST	*LAPSE	LMOST	AFL
a. 2P	r1s1	/sY-s/	[s(Ys)]	[(Y)(Xs)]	W	W	L		W	W		L	
b. 1P	r1s1	/sY-s/	[s(Ys)]	[s(sY)]		W		L	W		W		
c. 3P	r1s1	/sY-s/	[s(Ys)]	[(Ys)s]					W	W	W	L	L
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(Ys)]					W	L	L	W	W
e. 5	r1s1	/ss-s/	[s(Ys)]	[(Ys)s]						W	W	L	L

(257) {FNF, FT-BIN} >> {IAMB, PARSE-σ} >> MAXSTRESS >> {RMOST, *LAPSE} >> {AFL, LMOST}

Of course, there is a solution consistent with L39's paradigmatic equal, L51, which the learner could derive instead by also finding a consistent minimal mismatch candidate containing r2, such as r2s1 [(Ys)s]. The minimal mismatch candidates of r2s1 include the inputs /ss-s/, /YY-s/, and /Ys-Y/. Each of these candidates are consistent with L51, as illustrated by the shaded cells of the map of that language, repeated below.

(258) L51

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(Ys)s]	[(Ys)s]	[s(Ys)]	[(Ys)s]	s1 = /-s/
[(Ys)s]	[(Ys)s]	[s(Ys)]	[(Ys)s]	s2 = /-Y/

Adopting one of these minimal mismatch candidates will therefore result in a language hypothesis consistent with L51. In the alternative support for LgHyp39 below, W-L pair 5 in (259)e includes a minimal mismatch candidate for r2s1. Committing to this candidate resolves the conflict among the four lowest-ranked constraints so that AFL and

LMOST dominate RMOST and * LAPSE. The final stratified hierarchy for this alternative appears in (260).

(259) Alternative LgHyp39 updated support

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	FNF	PARSE-σ	IAMB	MAXSTR	AFL	LMOST	RMOST	*LAPSE
a. 2P	r1s1	/sY-s/	[s(Ys)]	[(Y)(Xs)]	W	W	L		W		L	W	
b. 1P	r1s1	/sY-s/	[s(Ys)]	[s(sY)]		W		L	W				W
c. 3P	r1s1	/sY-s/	[s(Ys)]	[(Ys)s]					W	L	L	W	W
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(Ys)]					W	W	W	L	L
e. 5	r2s1	/ss-s/	[(Ys)s]	[s(Ys)]						W	W	L	L

(260) {FNF, FT-BIN} >> {IAMB, PARSE-σ} >> MAXSTRESS >> {AFL, LMOST}>> { RMOST, *LAPSE}

Learning L39 and L51 provides some evidence in favor of the conservative choice of mismatch candidates, in that only the minimal mismatch is consistent with the information derived from phonotactic learning. In general, a strategy that commits the learner to fewer entailed mappings would be safer than the alternative, and for that reason, pursuing ranking information from minimal mismatch candidates is preferable to using maximal mismatch candidates instead. However, it may be that the choice makes no substantive difference. Section 4.3 will continue the discussion of learning L39, which is a paradigmatic subset in addition to being a paradigmatic equal. As a paradigmatic subset, learning L39 will require the use of Fewest Set Features, a procedure explicitly for setting features when inconsistency detection fails. Fewest Set Features must be employed to fully learn the lexicon, regardless of when the complete ranking conditions

are learned. Section 4.3 will show that the maximal mismatch candidate by ECM can be informative, if the learner has first set a feature in r1, r3, or r4 using Fewest Set Features.

This section has exposed the phenomenon of paradigmatic equality and illustrated the difficulties it poses for a learner that relies on the ODL's inconsistency detection procedures to set features, yet much more remains to be learned. The ECM procedure offered here may be a reliable last-resort mechanism for adding ranking information if it incorporates a branching procedure also, but without knowing more about paradigmatic equality in general it is unclear how successful it can be overall. How paradigmatic equality interacts with other language relationships is another important area for future investigation. A language can have multiple different relationships to other languages in a typology, and a solution aimed at untangling the paradigmatic equality relationship may not fully address these others, as the following section will show.

4.2 PARADIGMATIC SUBSETS

The Stress typology also contains a number of paradigmatic subset languages (Tesar, to appear). Like a paradigmatic equal, all of the paradigmatic subset's morpheme behaviors are shared with another language in the typology; however, the paradigmatic subset is a proper subset, making it a special example of restrictiveness.

The familiar "subset problem," briefly discussed in 1.2, concerns a language whose forms are a subset of another language in the typology. The learner lacks the negative evidence that a form in the superset is not allowed in the subset, and the more restrictive ranking conditions of the subset language cannot be derived by error-driven learning. As described earlier**,** one solution to this problem is to apply a low-faithfulness ranking bias,

such as BCD, to the support. The bias maintains a more restrictive ranking until positive evidence supports a less restrictive one.

The ranking bias is useful for error-driven learning of the ranking, but it cannot address the problem of learning the underlying forms of paradigmatic subsets. The Output-Driven Learner (ODL) sets a feature's value using inconsistency detection. Because the support derived from the subset's learning data is consistent with the superset language as well, the learner can only set features from test candidates that are inconsistent with both the subset target and its superset. The test candidates for other feature values may indeed be inconsistent with the subset target, but without more ranking knowledge to distinguish the subset from the superset, these features cannot be set by inconsistency detection. This is the problem of learning underlying forms in the paradigmatic subsets in Tesar's Stress/Length typology (to appear): the existence of a superset language stymies the ODL from setting features by inconsistency detection. The paradigmatic subsets in the Stress typology further demonstrate that it may not be possible to set some features by inconsistency detection even if all the ranking conditions of the subset language were known.

For example, L83 is a paradigmatic subset language in the Stress system whose features cannot all be set by inconsistency detection. The map of L83 appears in (261), and a stratified hierarchy derived by applying BCD to the skeletal basis of L83 is given in (262). L83 has leftmost main stress and exhaustive parsing, and it is by default trochaic. Suffixes in L83 neutralize in all environments, as do r1, r2, and r4, due to the ranking

LMOST >> MAXSTRESS. Because MAXSTRESS dominates both FNF and AFL, r3 contrasts with the other roots and surfaces faithfully in an iambic foot at the left edge.

(261) L83

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(Y)(Xs)]	[(Y)(Xs)]	[(sY)(X)]	[(Y)(Xs)]	s1 = /-s/
[(Y)(Xs)]	[(Y)(Xs)]	[(sY)(X)]	[(Y)(Xs)]	s2 = /-Y/

(262) {PARSE-σ, LMOST, *LAPSE} >> FT-BIN >> MAXSTRESS >> {FNF, AFL} >> {IAMB, RMOST }

L83 is a paradigmatic subset of L82, in (263). L82 includes the phonotactic inventory of L83 plus one additional form, for r1s2 [(X)(sY)], shaded below. This language also has leftmost main stress and exhaustive parsing and is by default trochaic, but it differs from L83 because it will allow a rightmost head-foot in order to satisfy MAXSTRESS. The stratified hierarchy in (264) shows that MAXSTRESS dominates LMOST, allowing the underlying stress of suffix s2 to surface faithfully in the context of r1, whose syllables are both underlyingly –stress. As a result, r1 behaves differently from r2 and r4, which have underlyingly +stress first syllables, and from r3, whose first syllable is –stress also but whose second syllable is +stress. Thus, L82 has two suffix behaviors and three root behaviors.

(263) L82

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(Y)(Xs)]	[(Y)(Xs)]	[(sY)(X)]	[(Y)(Xs)]	s1 = /-s/
[(X)(sY)]	[(Y)(Xs)]	[(sY)(X)]	[(Y)(Xs)]	s2 = /-Y/

(264) {PARSE-σ, *LAPSE} >> FT-BIN >> MAXSTRESS >> LMOST >> {FNF, AFL} >> {IAMB , RMOST }

The restrictiveness relation between L83 and L82 is evident by comparison of the phonotactic inventories, but the paradigmatic subset relation requires consideration of the morpheme behaviors in each language. In L83, s1 and s2 behave alike, as shown by their grouping in the description of the language in (265). This suffix behavior is identical to the behavior of s1 in L82, indicated by the shaded cells in (266). Thus, L83 is a paradigmatic subset of L82.

(265) L83 - one suffix behavior

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(Y)(Xs)]	[(Y)(Xs)]	[(sY)(X)]	[(Y)(Xs)]	s1 = /-s/ s2 = /-Y/

(266) L82 (L83 subset shaded)

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(Y)(Xs)]	[(Y)(Xs)]	[(sY)(X)]	[(Y)(Xs)]	s1 = /-s/
[(X)(sY)]	[(Y)(Xs)]	[(sY)(X)]	[(Y)(Xs)]	s2 = /-Y/

However, L83 can be compressed further from (265). It has only two root behaviors: one for r3, and one for every other root. The compression of these morpheme behaviors means that with the correct ranking, the stress features of the suffixes can remain unset in the learned lexicon and the roots r1, r2 and r4 each need only set the single feature that will distinguish them from r3. R3, then, must have both of its features set in the learned lexicon. These behaviors are shown in the compressed table below.

(267) L83 - two root behaviors, one suffix behavior

/?s/ /Y?/	/sY/	
[(Y)(Xs)]	[(sY)(X)]	/-?/

In L82, roots r2 and r4 behave alike, giving this language the three root behaviors and two suffix behaviors described in (268). The learner must set all the features of s1, s2, r1, and r3, and additionally must distinguish r2 and r4 from the other roots by setting their first syllables to +stress.

(268) L82 – three root behaviors, two suffix behaviors

/ss/	/Y?/	/sY/	
[(Y)(Xs)]	[(Y)(Xs)]	[(sY)(X)]	/-s/
[(X)(sY)]	[(Y)(Xs)]	[(sY)(X)]	/-Y/

Learning L82 presents no problem, as the data are sufficient to distinguish the language from its paradigmatic subset and all other languages in the typology. But learning L83 is another matter. As the remainder of this section will show, it is not just its paradigmatic superset, L82, which stands in the way of setting features, but the precise morpheme behaviors of L83 itself.

The phonotactic information for L83 enables the learner to derive the support in (269) and to set both features of r3; the ranking derived by BCD for this language hypothesis, LgHyp83, appears in (270). The lexicon, (271), includes no other set feature at this point, and because r3 always surfaces faithfully, it offers no potential for learning non-phonotactic ranking information.

(269) LgHyp83 support from phonotactics

ERC#	Morph. word	Input	Winner	Loser	LMOST	PARSE-σ	*LAPSE	FT-BIN	MAXSTR	AFL	FNF	RMOST	IAMB
a. 1P	r1s1	/Ys-s/	[(Y)(Xs)]	[s(Ys)]	W	W		L	W		L	L	
b. 3P	r1s1	/Ys-s/	[(Y)(Xs)]	[(sY)s]		W		L	W	L		L	L
c. 5P	r3s1	/sY-s/	[(sY)(X)]	[(Y)(Xs)]					W	L	L	W	W
d. 4P	r1s1	/Ys-s/	[(Y)(Xs)]	[(Ys)(X)]						W		L	
e. 2P	r1s1	/Ys-s/	[(Y)(Xs)]	[(Y)(sX)]							W		L

(270) {LMOST, PARSE-σ, *LAPSE} >> FT-BIN >> MAXSTRESS >> {AFL, FNF} >> {RMOST, IAMB}

(271) LgHyp83 lexicon

r1	r2	r3	r4	s1	s2
/??/	/??/	/sY/	/??/	/-?/	/-?/

The suffixes are not contrastive for stress and can remain unset, but the other three roots each must have one syllable's stress feature set to a value that will distinguish it from r3. The test candidates used for these roots to set features by inconsistency detection are identical because the roots behave alike in all words. Candidate (272)a attempts to set the stress feature of the first root syllable, candidate (272)b attempts to set the stress feature of the second.

(272) Test candidates for r1, r2, and r4

a. /**s**s-s/ → [(Y)(Xs)]
b. /Y**Y**-s/ → [(Y)(Xs)]

Both candidates are consistent with the support in (269) and therefore will not permit the learner to set any feature in the three roots. The support above is missing some of the ranking information of the target's skeletal basis; in particular, in the support LMOST dominates MAXSTRESS only by application of BCD – there are no explicit ranking arguments for this relationship otherwise. However, it is important to note that the consistency of the test candidates is not due to inadequate ranking information, but rather due to each of these candidates being a mapping in the target L83. Resolving the conflict between LMOST and MAXSTRESS would not produce the inconsistency required to set features.

To make this point more plainly, suppose the learner had access to the support for the skeletal basis of L83, in (273). The only mapping that distinguishes the paradigmatic subset L83 from its superset L82 is /ss-Y/[(Y)(Xs)]. This candidate is the winner in (273)b, and by transitivity this W-L pair and that in (273)d contributes the ranking LMOST >> MAXSTRESS .

(273) L83 skeletal basis support

Morph. word	Input	Winner	Loser	PARSE-σ	LMOST	* LAPSE	FT-BIN	MAXSTR	FNF	AFL	IAMB	RMOST
a. r3s1	/sY-s/	[(sY)(X)]	[(sY)s]	W			L		L	L		
b. r1s2	/ss-Y/	[(Y)(Xs)]	[(Xs)(Y)]		W			L		W		L
c. r3s1	/sY-s/	[(sY)(X)]	[(Ys)(X)]					W	L		W	
d. r3s1	/sY-s/	[(sY)(X)]	[(Y)(sX)]					W		L		W
e. r1s1	/ss-s/	[(Y)(Xs)]	[(Y)(sX)]						W		L	
f. r1s1	/ss-s/	[(Y)(Xs)]	[(Ys)(X)]							W		L

Even this support would not enable the learner to set features in r1, r2, or r4. No test candidate based purely on phonotactic knowledge could produce the inconsistency necessary to set a feature in these roots. The problem is that MAXSTRESS cannot prefer any desired loser when the test input differs from the observed output by only one feature. Either the single disparity must involve setting to – a feature that is + at the surface, in which case no candidate will incur a MAXSTRESS violation on that feature, or it will involve setting a feature to + that is – on the surface, in which case every candidate, including the desired winner, will incur at least one MAXSTRESS violation.

To better illustrate this point, consider [(Y)(Xs)], the output of all six words containing roots r1, r2, and r4. When the disparity is in the first syllable, forming the input /**s**s-s/, no candidate violates MAXSTRESS at all. When the disparity is in the second syllable, forming /Y**Y**-s/, then every candidate must have at least one MAXSTRESS violation. Since [(Y)(Xs)] itself would incur just one MAXSTRESS violation for this input, no other competitor could do better. Therefore, MAXSTRESS cannot prefer another candidate over either of the test candidates from (272), even with all crucial ranking information known.

Each of r1s1, r2s1, and r4s1 forms a contrast pair with r3s1, but again, the test candidates are not inconsistent. Because both features of r3 have already been set, there are three test candidates to evaluate: one for each syllable of the root and one for the suffix. The test candidates for the roots are those in (272), and evaluating them as contrast pairs yields no new information. Because the winner of W-L pair 5P in (269)c is r3s1 /sY-s/[(sY)(X)], the learner has in effect been evaluating the consistency of the test

candidates for the roots with r3s1 all along, even during single form learning. Finally, suffix stress is neutralized in the target, making it unsurprising that the test candidate for s1 is also consistent.

Just as with the paradigmatic equals, the learner has reached a point where inconsistency detection cannot set features, yet features must be set in order to distinguish the other roots from r3. For LgHyp83, however, even if new ranking information could be useful, there is no consistent mismatch winner to reveal that information. The minimal mismatch candidates derived from words containing r1, r2, and r4 are shown in (274), and they are the same candidates used to determine by inconsistency detection if a feature could be set in the roots. The candidates were uninformative then because they are consistent and they are uninformative now because the current ranking already makes them optimal.

(274) Minimal mismatch candidates for r1, r2, and r4

a. /**s**s-s/ → [(Y)(Xs)]
b. /Y**Y**-s/ → [(Y)(Xs)]

The maximal mismatch candidate is also uninformative. The maximal mismatch for all words containing r1, r2, or r4 is /sY-Y/[(Y)(Xs)]. It is not unexpected that this candidate is inconsistent, given that suffix stress neutralizes in the language and the support already includes the winner /sY-s/[(sY)(X)], with the same root and the unstressed suffix. These two candidates make contradictory ranking requirements, demonstrated in (275); note that although the maximal mismatch test candidate is labeled r1s1 here, the candidate will be the same for all words containing r1, r2, and r4 .

(275) Maximal mismatch candidate is inconsistent

ERC#	Morph. word	Input	Winner	Loser	LMOST	PARSE-σ	*LAPSE	FT-BIN	MAXSTR	AFL	FNF	RMOST	IAMB
a. 5P	r3s1	/sY-s/	[(sY)(X)]	[(Y)(Xs)]					W	L	L	W	W
b. test	r1s1	/sY-s/	[(Y)(Xs)]	[(sY)(X)]					L	W	W	L	L

The learner has exhausted the potential to set features by inconsistency detection, whether by single forms or contrast pairs, yet r1, r2, and r4 must each have the value of at least one feature set to distinguish them from r3. The learner cannot derive new ranking information, as all the mismatch candidates for words containing r1, r2, and r4 are uninformative. Moreover, having new ranking information would not help anyway, as the test candidates for the roots are consistent even with the skeletal basis of the target language.

Here the learner must employ the Fewest Set Features procedure, introduced by Tesar to complete the learning of a paradigmatic subset language. The Fewest Set Features procedure exploits the property of output-drivenness to determine, for a word that is currently failing error detection, the fewest features whose values must be set to match their surface correspondents' in order to create for that word a consistent mapping that passes error detection. The procedure is employed only on words that continue to fail error detection and only after single forms and contrast pairs fail to set any features and no new ranking information is derived by any means, including from a consistent mismatch candidate.

In the CBL's implementation of Fewest Set Features, the learner takes one of the words which has failed error detection – say, r1s1 – and systematically evaluates the candidates in its viable lexical subspace for consistency with the support and with the other words which have passed error detection. In order to set the fewest features possible, the learner begins by testing those candidates whose inputs contain only one unset feature that matches the value of its surface correspondent. If one of these is consistent, then the learner sets the value of that matching unset feature to its surface value and resumes learning as before. When more than one is consistent, the learner can choose either to inform the feature-setting. To learn languages in the Stress system, the CBL does not require Fewest Set Features to evaluate candidates containing two unset features, although in theory the procedure could be modified to do so if necessary. The pseudocode for Fewest Set Features, adapted for the CBL from the work of Tesar (to appear) is given below.

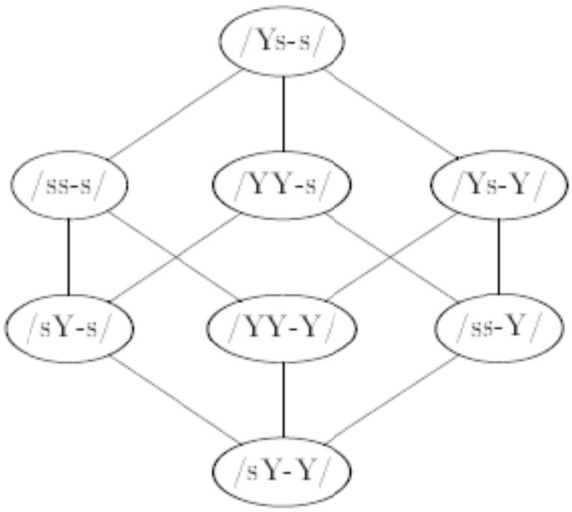
(276) Fewest Set Features – CBL adaptation

```
DEF Fewest_Set_Features(lang_hyp)
    FOR each word in lang_hyp that fails error detection
        Determine the feature value to pass error detection
            IF there is such a feature value THEN
                Set the feature to this value in the lexicon
                Check for ranking information from the set feature
                BREAK
            ENDIF
    ENDFOR
END
```

The lattice in (277) shows the entire lexical space for r1s1. The Fewest Set Features procedure works upward through this space. In effect, the procedure begins with the test for error detection on the input of the bottom node, containing the maximum three

disparities. If the word with this input passes error detection, then the fewest features to be set is zero. Here, however, r1s1 has failed error detection. The learner must evaluate the inputs in the next row up from the bottom. All of these contain two disparities, leaving only one feature set to match the surface form.

(277) Lexical space for r1s1



In the row of inputs with two disparities, the only inconsistent input is /sY-s/, in which the feature values of the root match those of r3. The other two inputs are consistent, leaving the learner with a choice. To distinguish r1 from r3, only one feature need be set: either the first syllable of the root must be set to + stress, as in /YY-Y/, or the second syllable must be set to –stress, as in /ss-Y/. In this computer simulation, the learner always selects the first option found, and here sets r1 to /Y?/ in the lexicon.

The newly set feature in r1 never surfaces unfaithfully, and therefore yields no new ranking information. With the updated lexicon and the support unchanged from (269), r1s1 and r1s2 now pass error detection, but the words containing r2 and r4 do not.

Because these morphemes behave like r1, the process for setting the fewest features in words containing them is identical to that for r1s1, with a similar outcome. The learner sets r2 and r4 to /Y?/, producing the lexicon in (278). The support remains unchanged.

(278) LgHyp83 lexicon – final

r1	r2	r3	r4	s1	s2
/Y?/	/Y?/	/sY/	/Y?/	/-?/	/-?/

4.2.1 FEWEST SET FEATURES AND PARADIGMATIC EQUALS

Fewest Set Features is also a potential solution for learning paradigmatic equals. If it were applied to learning L75, the final lexicon of LgHyp75 would appear as in (279). The features of r2, r3, and r4 which could not be set by inconsistency detection are set to match their surface values.

(279) LgHyp75 lexicon learned using Fewest Set Features

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/sY/	/Ys/	/-s/	/-Y/

Using Fewest Set Features would leave the support of LgHyp75 unchanged from (230) in section 4.1.1, when it was updated by single-form learning, and the conflict between RMOST and IAMB would remain unresolved. From the perspective of the learner, this version of LgHyp75 is acceptable, as it fully accounts for all of the learning data. Yet, in another sense, this language hypothesis is unsatisfactory because it will result in error when applied to the rich base. The ranking derived from (230) yields a CTie between [(Ys)(X)] and [(X)(Ys)] for the input /YY-s/; this is the error illustrated in (234). Fewest Set Features allows the learner to work around incomplete ranking information by

fixing the lexicon to include a subset of the underlying forms in the rich base, but because the features it sets never alternate in L75, no new non-phonotactic ranking information is ever learned. For this reason, the CBL employs ECM first when confronted by a situation where inconsistency detection fails to set required features.

4.2.2 CONCLUSION

The Fewest Set Features procedure was originally proposed as a solution for learning paradigmatic subsets, but L83 suggests that this procedure may have wider usage. Although L83 is a paradigmatic subset, it is not the existence of its paradigmatic superset, L82, that prevents the learner from setting features in r1, r2, and r4 by inconsistency detection. Instead, the problem is that the root behaviors of L83 compress into two groups: r3, and everything else. Each time the learner attempts to set a feature of r1, r2, or r4, the test candidate is consistent because it is a member of the non-r3 mappings of L83 itself. To put the matter differently, only a test candidate using /sY/ as the underlying form of r1, r2, or r4 could ever be inconsistent with L83, and constructing such a candidate would require that the learner have already set the stress feature of the first syllable to –stress or the second to +stress; however, this is impossible because both features are only ever set to match their surface correspondents' values. The features of r1, r2, and r4 never alternate, leaving no way for the learner to ever set the first syllable of one of these roots to –stress or the second to +stress and therefore no way to construct a test candidate using /sY/ as the underlying form of any of these three roots.

A procedure like Fewest Set Features is necessary for learning languages like L83, which cannot be fully learned by inconsistency detection alone, no matter how much

ranking information is known, and it also has value for learning languages like L75. Whether it should replace the ECM solution advocated in 4.1 for untangling global lexical ambiguity is unclear, as neither procedure is well understood at this point. It should be noted that of the nine pairs of paradigmatic equals produced in the Stress system, seven are also paradigmatic subsets of other languages and require the learner to apply Fewest Set Features regardless of learning from consistent mismatch candidates. Section 4.3 covers the topic of languages that are simultaneously paradigmatic equals and paradigmatic subsets using L39, a language first introduced in 4.1.3, to illustrate how Fewest Set Features and ECM can be interwoven to handle the complications of both relationships. Given that languages can participate in both paradigmatic relationships at once, the risks of committing to the ranking information of consistent mismatch candidates by ECM may appear too great. These relationships must be investigated further, along with the learning procedures described in sections 4.1 and 4.2.

4.3 WHEN THE PARADIGMATIC SUBSET IS A PARADIGMATIC EQUAL

The paradigmatic relationships described in the preceding sections can occur in combination, as they do for seven pairs of languages in the Stress system. One of these languages is L39, which was introduced in 4.1.3 to illustrate some consequences of committing to minimal versus maximal mismatch candidates. Learning a language like L39 requires the learner to overcome the common challenge of both kind of paradigmatic relationships – the inability of inconsistency detection to set all required features – but it also requires the learner to properly handle the relationships' unique characteristics: that the data for one paradigmatic equal can be inconsistent with the other given the right

ranking, but the data of a paradigmatic subset can never be inconsistent with its superset language.

The preceding sections have described alternative procedures for managing each paradigmatic relationship based on these properties. Thus, ERC by Consistent Mismatch (ECM) exploits the paradigmatic equal's potential for learning crucial ranking information by creating a consistent mismatch candidate, while Fewest Set Features provides a means of setting features when inconsistency detection fails – and even when inconsistency detection would fail with the target's ranking conditions fully known. This section demonstrates how the learner weaves together these different approaches to learn languages like L39, which participates in both kinds of paradigmatic relationships. It will also provide a fuller look at the relative benefit of pursuing minimal mismatch candidates by ECM.

4.3.1 THE PARADIGMATIC RELATIONSHIPS OF L39

The map of L39 and a stratified hierarchy derived from its skeletal basis by BCD are repeated below, along with the map and ranking of its paradigmatic equal, L51. These languages both parse binary trochees and differ only by the default alignment of the head-foot, with L39 aligning it to the right and L51 aligning it to the left. The shaded cells indicate the default behavior in each map.

(280) L39

r1 =/ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(Ys)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s1 = /-s/
[s(Ys)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s2 = /-Y/

(281) {FNF, FT-BIN} >> {IAMB, PARSE-σ} >> MAXSTRESS >> {RMOST, *LAPSE} >> {AFL, LMOST}

(282) L51

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(Ys)s]	[(Ys)s]	[s(Ys)]	[(Ys)s]	s1 = /-s/
[(Ys)s]	[(Ys)s]	[s(Ys)]	[(Ys)s]	s2 = /-Y/

(283) {FNF, FT-BIN} >> {IAMB, PARSE-σ} >> MAXSTRESS >> {AFL, LMOST}>> {RMOST, *LAPSE}

Suffixes in these languages are not contrastive, and the root behaviors display the same pattern shown by L83 in 4.2: there is a behavior for one specific root – r2 in L39, r3 in L51 – and then a default behavior for all the other roots. The map of L39 is shown compressed in (284) to highlight the single suffix behavior and two root behaviors; the compressed map of L51 would be similar.

(284) L39 - two root behaviors, one suffix behavior

/s?/ /?Y/	/Ys/	
[s(Ys)]	[(Ys)s]	/-?/

L39 is a paradigmatic subset of five other languages in the Stress system typology, including L37, whose map is shown in (285) with shaded cells indicating the L39 subset. L37 parses trochees by default just as L39 does, but r1s2 and r2s2 show that it will also parse iambs. For r1s2, the ranking FT-BIN >> {PARSE-σ, * LAPSE} forces the right-aligned iambic head-foot even at the cost of an initial stress lapse. With two underlyingly stressed syllables, r2s2 must violate MAXSTRESS once. For this form, the crucial ranking

is RMOST >> {FNF, AFL, LMOST}, which forces s2 to surface faithfully in the iambic head-foot instead of preserving stress on r2 with the left-aligned trochaic output [(Ys)s]. The stratified hierarchy for L37 derived by applying BCD to the support of the skeletal basis is given in (286).

(285) L37 (L39 subset shaded)

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(Ys)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s1 = /-s/
[s(sY)]	[s(sY)]	[s(Ys)]	[s(Ys)]	s2 = /-Y/

(286) FT-BIN >> PARSE-σ >> MAXSTRESS >> {RMOST, *LAPSE} >> {FNF, IAMB, AFL, LMOST}

4.3.2 COMBINING ECM AND FEWEST SET FEATURES

L37 is included here to demonstrate unequivocally that a paradigmatic equal, L39, can also be a paradigmatic subset of another language. However, the fundamental issue for this section – that there can be multiple reasons within a language for why inconsistency detection fails to set all necessary features – does not rely on L39 having a paradigmatic subset. The similarity of the chart of L39's morpheme behaviors in (284) with L83's chart in (267) makes it clear that the learner will face the same kind of problem learning L39 as learning L83: the test candidates for the roots with the default behavior will all be consistent simply because every single-disparity test will produce another of the default, non-r2 mappings, and not just because there is a superset language. What makes learning L39 different from learning L83 is that not a single feature can be set by standard inconsistency detection initially. Before delving into learning L39, then, a

question must be answered: when inconsistency detection fails, which paradigmatic relationship does the learner attempt to address first?

The CBL first tackles paradigmatic equality, attempting to learn by ECM on the grounds that it is preferable to produce new ranking information than not. Unlike Fewest Set Features, the objective of ECM is to uncover ranking information. ECM forces a resolution to a conflict exposed during error detection by committing to an input-output mapping which is not explicitly produced by the current commitments of the language hypothesis. For example, in section 4.1 error detection reveals an unresolved conflict between RMOST and IAMB, but it takes a mapping from the input /YY-s/, which ECM finds in a consistent mismatch candidate, to resolve the conflict.

As has been seen throughout the preceding chapters, learning more about the ranking can produce a range of benefits. First, the new ranking information can allow the learner to set features by inconsistency detection and consequently enable words to pass error detection. For LgHyp75 in 4.1, an ERC added by the consistent mismatch candidate allows for r3 to be set to /sY/ by inconsistency detection, causing r3s1 finally to pass error detection. Second, new ranking information can allow words which previously failed error detection to now pass without setting any additional features, just as in LgHyp75 features in r2 and r4 could remain unset once the new ERC was added. Finally, ranking information helps to reduce the potential for error when the ranking is applied to a rich base, just as adding the ERC produced by ECM to LgHyp75 expressly eliminates the error that otherwise would have occurred given the input /YY-s/. Section 4.1 shows that LgHyp75 derives all of these benefits from one application of ECM. While setting a

single feature can cause a word to finally pass error detection, the other benefits will not accrue without that feature inciting an error that leads to a ranking change.

The objective of Fewest Set Features is simply to set the fewest features needed for a single word to pass error detection, and any ranking information recovered is incidental. Although it is at least theoretically possible for new ranking information to arise if the newly set feature ever surfaces unfaithfully elsewhere, Fewest Set Features does not ever produce new ranking information when it is implemented for languages in the Stress system, regardless of whether ECM is implemented first. For these reasons, the CBL attempts to learn from ECM before employing Fewest Set Features.

The CBL bundles contrast pair learning, ECM, and Fewest Set Features together and employs them one after the other once a round of single-form learning passes without making any changes to a consistent but incomplete language hypothesis. As soon as one of these procedures sets a feature, the language hypothesis is updated and, if the hypothesis is not yet complete, a new round of single-form learning begins on the hope that this latest change will lead to more information. The pseudocode below describes the further learning attempts that occur.

(287) Further learning beyond single forms

```
DEF FURTHER_LEARNING(lang_hyp)
    IF learning is complete in lang_hyp THEN
        Store lang_hyp with completed language hypotheses
    ELSE
        SET learning_change to true
        WHILE learning_change is true
            Set learning_change to false
            IF features are set with a contrast pair THEN
                Set learning_change to true
            ELSE
                Employ ECM procedure
                IF ranking information is learned by ECM THEN
                  Set learning_change to true
                ENDIF
                IF learning_change is false THEN
                    Employ Fewest Set Features
                    IF a feature is set using Fewest Set Features THEN
                        SET learning_change to true
                    ENDIF
                ENDIF
            ENDIF //features set by contrast pair
            IF learning_change is true THEN
                IF learning is complete in lang_hyp THEN
                    Store lang_hyp with completed language hypotheses
                ELSE
                    Perform single-form learning
                    BREAK out of WHILE loop
                ENDIF
            ELSE
                IF lang_hyp is consistent THEN
                    Store lang_hyp with incomplete, consistent language hypotheses
                ELSE
                    Discard lang_hyp
                ENDIF
            ENDIF
        ENDWHILE
    ENDIF
END
```

A complete outline of the Commitment-Based Learner, beginning with phonotactic learning, appears in (288). The ECM and Fewest Set Features procedures begin at step 7.b.

(288) Outline of the Commitment-Based Learner

Within each language hypothesis, beginning with Hyp0, and for each observed form, repeat until learning stops in all language hypotheses:

Phonotactic Learning

1. Check for errors
    a. If the form lacks a committed structural interpretation and yields an error, apply the IDL to extend branches. Repeat step 1 for each branch.
    b. If a form has a committed structural interpretation and produces an error, perform error-driven learning. Repeat step 1.
    c. If the form does not produce an error, process the next overt form.
2. Phonotactic learning ends when no errors are detected on any observed forms.

Non-phonotactic Learning – first pass through data

3. Perform error-driven learning over all known words.
    a. Reject hypothesis if it is inconsistent
4. Does the form have a committed interpretation?
    a. Yes – apply the ODL to set features from the single form.
        i. If features are set, seek non-phonotactic ranking information from unfaithful mappings using the low-markedness ranking bias.
        ii. If no features are set, observe the next form.
    b. No – perform error detection on the overt form.
        i. If the overt form passes error detection, observe the next form. Go to step 3.
        ii. If it does not pass error detection, apply the IDL to assign interpretations and extend branches.
        iii. Continue learning in the resulting branches, beginning with the first observed form in the data set. Go to step 3.
5. Perform error detection on the list of known words.
    a. If all words pass error detection, this language hypothesis is complete.
        i. Are all consistent language hypotheses complete?
            1. Yes – stop. Learning is complete.
            2. No – continue learning in the incomplete language hypotheses.
    b. If some words fail error detection, go to step 6.

Non-phonotactic Learning – after the first pass through the data

6. Were any features set by single-form learning in the last pass through the data?
    a. Yes – repeat steps 3-5 for each word that currently fails error detection.
    b. No – apply the ODL to set features from contrast pairs in the list of known words. Go to step 7.
7. Were any features set by contrast pairs in this pass?
    a. Yes – repeat steps 3-5 for each word that currently fails error detection.
    b. No – employ ECM procedure. Go to step 8.

8. Was any ranking information learned by ECM?
    a. Yes – repeat steps 3-5 for each word that currently fails error detection.
    b. No – employ Fewest Set Features. Go to step 9.
9. Was a feature set by Fewest Set Features?
    a. Yes – repeat steps 3-5 for each word that currently fails error detection.
    b. No – store this language hypothesis with the incomplete, consistent language hypotheses. Stop.

Having shown where ECM and Fewest Set Features fall with respect to single-form and contrast pair learning, the next section will demonstrate how both of these alternative procedures are implemented to learn L39.

4.3.3 LEARNING L39

The map of L39 and the stratified hierarchy derived from its skeletal basis are repeated in (289) and (290).

(289) L39

r1 =/ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[s(Ys)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s1 = /-s/
[s(Ys)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	s2 = /-Y/

(290) {FNF, FT-BIN} >> {IAMB, PARSE-σ} >> MAXSTRESS >> {RMOST, *LAPSE} >> {AFL, LMOST}

Suffixes in L39 are not contrastive and can remain unset in the learned lexicon. The two root behaviors evinced in the language require setting r2 to /Ys/ and, for the remaining roots, some combination of setting the first syllable to –stress or the second syllable to +stress. These requirements are summarized in the idealized lexicon for LgHyp39 below.

(291) LgHyp39 idealized lexicon

r1	r2	r3	r4	s1	s2
/s?/ or /?Y/	/Ys/	/s?/ or /?Y/	/s?/ or /?Y/	/-?/	/-?/

The support for LgHyp39 created from phonotactic learning appears in (292), with its ranking in (293).

(292) LgHyp39 support from phonotactics

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	FNF	PARSE-σ	IAMB	MAXSTR	LMOST	RMOST	AFL	*LAPSE
a. 2P	r1s1	/sY-s/	[s(Ys)]	[(Y)(Xs)]	W	W	L		W	L	W		
b. 1P	r1s1	/sY-s/	[s(Ys)]	[s(sY)]		W		L	W				W
c. 3P	r1s1	/sY-s/	[s(Ys)]	[(Ys)s]					W	L	W	L	W
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(Ys)]					W	W	L	W	L

(293) {FNF, FT-BIN} >> {IAMB, PARSE-σ} >> MAXSTRESS >> {LMOST, RMOST, AFL, *LAPSE}

The unresolved conflict involves the constraints in the lowest stratum. In order to set all the features required to distinguish the root behaviors, the learner needs to determine that RMOST and *LAPSE dominate AFL and LMOST. Doing so in turn requires a committed mapping with an input that makes those constraints decisive in determining the winner, such as /ss-s/, for which no candidate will incur a MAXSTRESS violation, or /YY-s/, for which every candidate will incur one MAXSTRESS violation. But L39 is a paradigmatic equal, and as expected, the unresolved conflict between the constraints in the lowest stratum means the learner will not be able to commit to these mappings using inconsistency detection.

First, for single-form learning the test candidates for the root in r1s1 are /**Y**Y-s/[s(Ys)] and /s**s**-s/[s(Ys)], which are both actual mappings in the target and therefore would be consistent even if all ranking conditions were already known.[28] These candidates cannot produce the lexically-informative inconsistency, but neither can the test candidates for r2, which map the same inputs to different outputs -- /Y**Y**-s/[(Ys)s] and /**s**s-s/[(Ys)s]. This time, the test candidates are consistent with the paradigmatic equal, L51, whose map is repeated in (294). Having the correct ranking information for L39 would allow the learner to detect inconsistencies and set features with these test candidates, but here the familiar problem of paradigmatic equals arises: the correct ranking is needed to set these features, and these feature values are needed to determine the correct ranking.

(294) L51

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(Ys)s]	[(Ys)s]	[s(Ys)]	[(Ys)s]	s1 = /-s/
[(Ys)s]	[(Ys)s]	[s(Ys)]	[(Ys)s]	s2 = /-Y/

Contrast pairs face the same problems for setting features: the pairs are always consistent with one of the paradigmatic equals. The test candidates used to attempt to set root features in the contrast pair r1s1 and r2s1 are shown in (295). All other contrast pairs will use the same mappings and achieve the same uninformative outcome.

[28] Because the roots r3 and r4 behave like r1, their test candidates and outcomes will be identical to those of r1.

(295) Test candidates for contrast pair r1s1, r2s1

a. r1s1 /**Y**Y-s/ → [s(Ys)] *r1 σ1: consistent with L39*
r2s1 /Ys-s/ → [(Ys)s]

b. r1s1 /s**s**-s/ → [s(Ys)] *r1 σ2: consistent with L39*
r2s1 /Ys-s/ → [(Ys)s]

c. r1s1 /sY-s/ → [s(Ys)] *r2 σ1: consistent with L51*
r2s1 /**s**s-s/ → [(Ys)s]

d. r1s1 /sY-s/ → [s(Ys)] *r2 σ2: consistent with L51*
r2s1 /Y**Y**-s/ → [(Ys)s]

4.3.3.1 Committing to a minimal mismatch candidate

Single-form and contrast pair learning have both failed; this is the point at which the discussion of LgHyp39 in section 4.1.3 begins. Any further ranking information must come from committing to a consistent mismatch candidate. As 4.1.3 explains, the maximal mismatch candidates for r1s1 and r2s1 are inconsistent, but each of the minimal mismatch candidates is consistent and informative. After committing to one of these, r1s1 /ss-s/[s(Ys)], the learner updates the support as below, repeated from (256). The minimal mismatch candidate is the winner in W-L pair 5.

(296) LgHyp39 support updated with minimal mismatch candidate

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	FNF	PARSE-σ	IAMB	MAXSTR	RMOST	*LAPSE	LMOST	AFL
a. 2P	r1s1	/sY-s/	[s(Ys)]	[(Y)(Xs)]	W	W	L		W	W		L	
b. 1P	r1s1	/sY-s/	[s(Ys)]	[s(sY)]		W		L	W		W		
c. 3P	r1s1	/sY-s/	[s(Ys)]	[(Ys)s]					W	W	W	L	L
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(Ys)]					W	L	L	W	W
e. 5	r1s1	/ss-s/	[s(Ys)]	[(Ys)s]						W	W	L	L

(297) {FNF, FT-BIN} >> {IAMB, PARSE-σ} >> MAXSTRESS >> {RMOST, *LAPSE} >> {AFL, LMOST}

The ranking in (297) is now correct and complete, as it matches the ranking for the target L39 given in (290). Moreover, the ERC added by committing to the minimal mismatch candidate /ss-s/[s(Ys)] enables the learner to set both features of r2 by inconsistency detection, using the candidates below.

(298) Test candidates for r2s1

a. r2s1 /**s**s-s/ → [(Ys)s]
b. r2s1 /Y**Y**-s/ → [(Ys)s]

These test candidates have the same ranking restrictions: both require that LMOST and AFL dominate RMOST and *LAPSE. The tableau in (299) exposes a contradiction between these ranking conditions and those of the minimal mismatch candidate /ss-s/[s(Ys)]. With these lexically-informative inconsistencies the learner updates the lexicon, (300).

(299) R2 test candidates inconsistent with r1s1 maximal mismatch

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	FNF	PARSE-σ	IAMB	MAXSTR	RMOST	*LAPSE	LMOST	AFL
a. 5	r1s1	/ss-s/	[s(Ys)]	[(Ys)s]						W	W	L	L
b. test	r2s1	/ss-s/	[(Ys)s]	[s(Ys)]						L	L	W	W
c. test	r2s1	/YY-s/	[(Ys)s]	[s(Ys)]						L	L	W	W

(300) LgHy39 lexicon updated for r2

r1	r2	r3	r4	s1	s2
/??/	/Ys/	/??/	/??/	/-?/	/-?/

All that remains now is to distinguish the remaining roots from r2 by setting at least one feature in each. Yet, just as in L83, even with the complete ranking information, inconsistency detection cannot set the required features. The ERC from the consistent maximal mismatch candidate, represented by the W-L pair in (296)e, has only eliminated the interference of the paradigmatic equal, L51, but not the subset problem. That is, the single-disparity candidates for r1, r3 and r4 are a subset of the mappings corresponding to the non-r2 behavior in this language, and therefore they are necessarily consistent. The only recourse is the true last resort, Fewest Set Features. One feature in each of the roots must be set to match its surface form. In the lexicon below, the first syllable in each root is set to -stress. The language hypothesis is now complete.

(301) LgHy39 lexicon – final

r1	r2	r3	r4	s1	s2
/s?/	/Ys/	/s?/	/s?/	/-?/	/-?/

4.3.3.2 Committing to a maximal mismatch candidate

Section 4.1.3 ends the discussion of the consequences of maximal versus minimal mismatch candidates by mentioning that under the right circumstances, r1s1 in L39 can contribute a consistent maximal mismatch candidate. To do so, a feature of r1 must have already been set in the lexicon. Further, section 4.3.3.1 above has shown that because r1, r3, and r4 can never be set by inconsistency detection, a procedure like Fewest Set Features will always be necessary to set features in these roots. The failure of

inconsistency detection to set these features is independent of how complete the support is, making the question not if, but when, Fewest Set Features must apply. This section briefly shows that because Fewest Set Features must occur regardless, here it is possible to maintain a strategy of committing only to consistent maximal mismatches in spite of an initial failure to find one.

To begin, section 4.1.3 described why the phonotactic support, repeated in (302), originally fails to produce a consistent maximal mismatch. A strategy that would only allow commitments to maximal mismatches would next have to seek an alternative means of adding to the language hypothesis and would employ Fewest Set Features. A feature in r1 could be set now, as in (303).

(302) LgHyp39 support from phonotactics

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	FNF	PARSE-σ	IAMB	MAXSTR	LMOST	RMOST	AFL	*LAPSE
a. 2P	r1s1	/sY-s/	[s(Ys)]	[(Y)(Xs)]	W	W	L		W	L	W		
b. 1P	r1s1	/sY-s/	[s(Ys)]	[s(sY)]		W		L	W				W
c. 3P	r1s1	/sY-s/	[s(Ys)]	[(Ys)s]					W	L	W	L	W
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(Ys)]					W	W	L	W	L

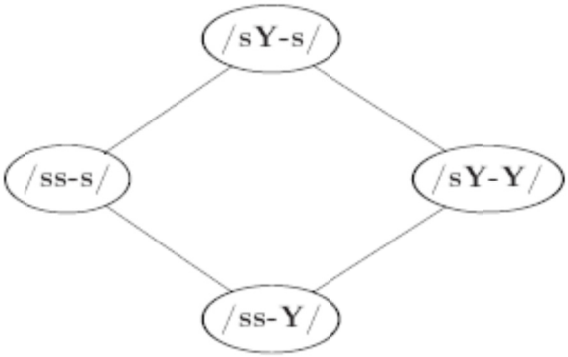
(303) LgHy39 lexicon – updated for r1

r1	r2	r3	r4	s1	s2
/s?/	/??/	/??/	/??/	/-?/	/-?/

This feature cannot elicit new ranking information because it never surfaces unfaithfully; however, the language hypothesis is far from complete: r2 must still have both of its features set, and r3 and r4 must each set one. Single-form and contrast pair

learning repeat, but with no changes to the ranking, all test candidates will continue to be consistent. ECM applies next, and now setting r1 has changed the maximal mismatch candidate for r1s1 into a consistent mapping that can contribute ranking information. The viable lexical subspace for r1s1 [s(Ys)] includes only the underlying forms in the lattice in (304). With the newly set feature in r1, the maximal mismatch candidate is /ss-Y/[s(Ys)], which is a mapping in the target language. This candidate is necessarily consistent with the support.

(304) Viable lexical subspace for r1s1



Committing to this maximal mismatch candidate yields the expected error brought on by the conflict between constraints that favor the right-aligned output [s(Ys)] and those that favor the left-aligned output [(Ys)s]. The conflict is resolved as in the preceding section, with RMOST and *LAPSE dominating AFL and LMOST. The updated support appears in (305); again, W-L pair 5 in this support is produced from the error on the consistent mismatch candidate.

(305) LgHyp39 support updated by maximal mismatch /ss-Y/[s(Ys)]

ERC#	Morph. word	Input	Winner	Loser	FT-BIN	FNF	PARSE-σ	IAMB	MAXSTR	RMOST	*LAPSE	LMOST	AFL
a. 2P	r1s1	/sY-s/	[s(Ys)]	[(Y)(Xs)]	W	W	L		W	W		L	
b. 1P	r1s1	/sY-s/	[s(Ys)]	[s(sY)]		W		L	W		W		
c. 3P	r1s1	/sY-s/	[s(Ys)]	[(Ys)s]					W	W	W	L	L
d. 4P	r2s1	/Ys-s/	[(Ys)s]	[s(Ys)]					W	L	L	W	W
e. 5	r1s1	/ss-Y/	[s(Ys)]	[(Ys)s]						W	W	L	L

The commitment to the maximal mismatch /ss-Y/[s(Ys)], with two disparities, also commits the learner to the mappings containing the single-disparity inputs from (304): /ss-s/[s(Ys)] and /sY-Y/[s(Ys)]. The learner will only commit to a maximal mismatch candidate if it is consistent with the current support; thus, both of these entailed mappings must also be consistent with the current support. Moreover, because all three are mappings in the target L39, they will remain consistent with all the current commitments, regardless of whether the full ranking conditions of those commitments are already represented in the support. In this instance, the learner's commitment to the maximal mismatch candidate works, and LgHyp39 will remain consistent so long as all further commitments are also consistent with L39. The commitment to the mismatch candidate will enable the learner to set both features of r2 by inconsistency detection, as in the preceding section, and Fewest Set Features will apply to set one feature each in r3 and r4.

4.3.4 CONCLUSION

Learning a language that is both a paradigmatic equal and a paradigmatic subset requires separately addressing the complications of each relationship. Paradigmatic equality poses a ranking-based problem: an unresolved conflict in the ranking makes it

impossible to set some needed features by inconsistency detection. ECM offers a ranking-based solution: resolve the conflict by committing to a consistent mismatch candidate for one of the words that currently fails error detection. For paradigmatic subsets and languages like L39 and L83, however, the problem is that even the complete ranking may be insufficient to set features by inconsistency detection. Fewest Set Features offers a lexical solution to this problem: if all else fails, set features to match their surface values.

For languages that participate in both relationships, both solutions can be employed. For L39, using ECM resolves a conflict that ultimately enables features in r2 to be set by inconsistency detection. However, because the remaining three roots all behave alike, inconsistency detection fails to set their features, and Fewest Set Features can be applied to complete the lexicon.[29]

This section has also revisited the question introduced in 4.1.3: should ECM commit to maximal or minimal mismatch candidates? Again, LgHyp39 does not offer persuasive evidence for either choice. A strategy seeking only maximal mismatch candidates will fail to find a consistent candidate initially, but once Fewest Set Features steps in and assigns a value to a feature – as it must do at some point anyway – the change to the lexical subspace results in a new, and consistent, maximal mismatch candidate. It is conceivable that this process could repeat on a larger scale if necessary, with Fewest Set

[29] In fact, it is worthwhile to recall that Fewest Set Features can be used exclusively, setting all features to match their surface forms as described for LgHyp75 in 4.2. The disadvantage to this method is that it does not resolve the ranking conflict and therefore the ranking will generate errors when applied to the rich base.

Features slowly causing the maximal mismatch candidate to have fewer and fewer disparities until at last the maximal mismatch is consistent.

In general, whatever advantages or disadvantages the maximal mismatch choice offers are tied to the fact that in an output-driven map, a commitment to one mapping entails a commitment to all mappings with fewer disparities. The disadvantages are clear: if a later round of error-driven learning exposes ranking conditions that make the mismatch candidate inconsistent with prior commitments, the language hypothesis is inconsistent, and the target may not be learned. The advantages are less clear, however. In LgHyp39, the commitment to the maximal mismatch candidate does not make it possible to set any more features by inconsistency detection than committing to the minimal mismatch candidate does.

If all paradigmatic equals are like L75 or L39, then the circumstances of learning a paradigmatic equal may ensure that either choice produces the same outcome, whether because the maximal mismatch candidate has exactly one disparity, as in LgHyp75, or because the maximal mismatch entails only the mappings of morphemes that behave alike, as in L39. The phenomenon of paradigmatic equality and the outcome of this particular solution could appear very different in a system with more features and more constraints. Determining exactly what benefit, if any, the maximal mismatch candidate offers will require more investigation of paradigmatic equals beyond those in the Stress system.

4.4 GLOBAL SURFACE AMBIGUITY AND THE PARADIGMATIC EQUAL

The preceding section illustrates how one language can simultaneously participate in both kinds of paradigmatic relationships with other languages in the typology. L39 is a paradigmatic equal with L51 and a paradigmatic subset of L37, among others. Both relationships preclude setting features by inconsistency detection given the observed data, but the learner's response to each relationship differs; most importantly, for paradigmatic equals the learner can add ranking information that makes inconsistency detection viable for setting features. This section similarly examines the learner's response to languages that relate in multiple ways to others in the typology, but now the focus shifts from paradigmatic relationships to global ambiguities. What is interesting about this combination of relationships is not what the learner has to do to process the data, but what the learner ultimately learns from the data.

4.4.1 GLOBALLY AMBIGUOUS LANGUAGES AND THE LEARNING DATA

First, as explained in 4.1, paradigmatic equals exhibit global lexical ambiguity: the languages have identical morpheme behaviors, including structural interpretations, but those behaviors arise from different underlying forms. Because the languages have different rankings, their maps are different, too, yet the shared morpheme behaviors mean that, from the learner's perspective, the data of the languages are the same. To illustrate, compare L75 and L76, the paradigmatic equals introduced in 4.1 and whose maps and rankings appear below.

(306) L75

r1 =/ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(X)(sY)]	[(Y)(sX)]	[(X)(Ys)]	[(Y)(sX)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

(307) {PARSE-σ, *LAPSE} >> {AFL, FT-BIN} >> MAXSTRESS >> IAMB >> {RMOST, FNF} >> LMOST

(308) L76

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(X)(sY)]	[(Y)(sX)]	[(X)(Ys)]	[(X)(Ys)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

(309) { PARSE-σ, *LAPSE} >> {AFL, FT-BIN} >> MAXSTRESS >> RMOST >> {IAMB, LMOST} >> FNF

Both languages have two suffix behaviors and three root behaviors, but whereas r4 /YY/ behaves like r2 /Ys/ in L75, it behaves like r3 /sY/ in L76. This difference is caused by the differing dominance relations between IAMB and RMOST. When IAMB dominates RMOST, the second syllable of r4 surfaces unfaithfully, producing a binary, iambic secondary foot at the right edge. As a result, the output looks like a faithful mapping from r2 /Ys/. For the opposite ranking, the second syllable of r4 surfaces faithfully instead, as the head of the right-aligned trochaic head-foot. This output looks like a faithful mapping from r3 /sY/.

The chart in (310) compresses and overlays the morpheme behaviors of both languages to show that their morpheme behaviors are identical, with the only difference being the underlying forms producing the behaviors. L75 and L76 are therefore globally

lexically-ambiguous: they produce the same morpheme behaviors from different underlying forms. Consequently, the learning data for these languages appear identical to the learner.

(310) L75 and L76 overlaid

/ss/	/Ys/ (/YY/ L75)	/sY/ (/YY/ L76)	
[(X)(sY)]	[(Y)(sX)]	[(X)(Ys)]	/-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	/-Y/

In addition to being globally lexically-ambiguous with L75, L76 is also globally surface-ambiguous with L69. The maps for these languages appear in (311) and (312). These languages contain the same overt forms for each morphological word, but they assign different interpretations to r2s1, shaded. The map-based definition of global surface ambiguity given in (119) of chapter 3 is repeated in (313).

(311) L76

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(X)(sY)]	[(Y)(sX)]	[(X)(Ys)]	[(X)(Ys)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

(312) L69

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(X)(sY)]	[(Ys)(X)]	[(X)(Ys)]	[(X)(Ys)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

(313) Global surface ambiguity (map-based definition)

> Languages L*A* and L*B* are globally surface-ambiguous if their maps are identical with respect to overt forms.

The stratified hierarchies derived by BCD for L76 and L69, below, indicate that the relative rankings of AFL and RMOST are responsible for the languages' differences. In L76, AFL dominates RMOST, and r2s1 is parsed with a unary head-foot, reducing the AFL violations incurred by the secondary foot at the cost of making the head-foot farther from the right edge. When RMOST dominates AFL, as in L69, the secondary stressed syllable in r2s1 is parsed into a unary foot so that the head-foot is only one syllable from the right edge.

(314) L76 ranking

{PARSE-σ, *LAPSE} >> {AFL, FT-BIN} >> MAXSTRESS >> RMOST >> {IAMB, LMOST} >> FNF

(315) L69 ranking

{PARSE-σ, *LAPSE} >> FT-BIN >> MAXSTRESS >> RMOST >> {IAMB, AFL, LMOST} >> FNF

As globally surface-ambiguous counterparts, L69 and L76 produce the same learning data, shown in (316) grouped by morpheme behavior. Yet, because L76 is the paradigmatic equal of L75, L76 and L75 also produce data that appear the same to the learner. These data are given in (317), again grouped by morpheme behavior.

(316) Learning data for L69 & L76

/ss/	/Ys	/sY/ /YY/	
XsY	*YsX*	*XYs*	/-s/
XsY	*XsY*	*XsY*	/-Y/

(317) Learning data for L75 & L76

/ss/	/Ys/ (/YY/ L75)	/sY/ (/YY/ L76)	
XsY	*YsX*	*Xys*	/-s/
XsY	*XsY*	*XsY*	/-Y/

Because L76 has the same learning data as both L69 and L75, it will appear to the learner that L69 and L75 have the same learning data, too. In fact, this transitive outcome simply demonstrates that L69 and L75 are globally surface-ambiguous, as defined by reference to their morpheme behaviors. This revised definition, originally given in (127) of chapter 3, is repeated in (318). For comparison, the definition of global lexical ambiguity given in section 4.1 is also repeated, in (319).

(318) Global surface ambiguity (morpheme behavior definition)

> Language L*A* and L*B* are globally surface-ambiguous if they have the same morpheme behaviors, excluding structural interpretations.

(319) Global lexical ambiguity

> Languages L*A* and L*B* are globally lexically-ambiguous if they have the same morpheme behaviors, including structural interpretations, but differ only by which underlying forms of the rich base produce which behaviors.

Thus, global ambiguity can be identified through the morpheme behaviors of two languages. The variety of global ambiguity is then distinguished by comparing the structural interpretations associated with the morpheme behaviors. If two languages share all the same morpheme behaviors, and those behaviors yield the same structural interpretations, then the languages are paradigmatic equals and are globally lexically

ambiguous. If the languages share all the same morpheme behaviors, but those behaviors yield different structural interpretations, the languages are globally surface-ambiguous. The definition for general global ambiguity appears in (320).

(320) Global ambiguity

> Languages L*A* and L*B* are globally ambiguous if they have the same morpheme behaviors.

Because the CBL has the ability to construct and retain separate language hypotheses, it is possible for this learner to derive language hypotheses corresponding to a variety of globally ambiguous languages from the same learning data. Chapter 3 and the preceding sections of this chapter have already illustrated this ability, but only with regard to one kind of global ambiguity at a time. The remainder of this section will show that the CBL effectively manages the learning data of targets that are globally ambiguous in both ways, as L75 and L76 are globally lexically ambiguous with each other and globally surface-ambiguous with L69.

4.4.2 LEARNING GLOBALLY AMBIGUOUS LANGUAGES

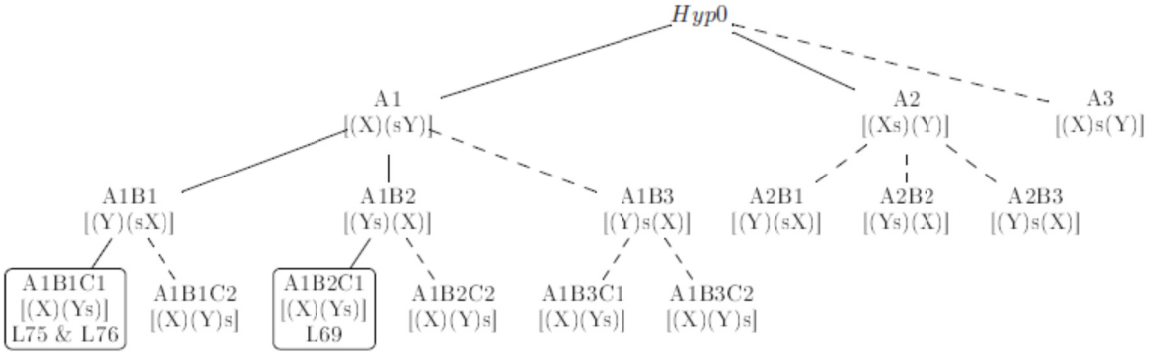
Suppose the learner observes the data in (321).

(321) Learning data

XsY r1s1	*XsY* r1s2	*YsX* r2s1	*XsY* r2s2	*XYs* r3s1	*XsY* r3s2	*YsX* r4s1	*XsY* r4s2

When phonotactic learning ends, two consistent language hypotheses remain. In the tree in (322), language hypothesis A1B1C1 corresponds to both L75 and L76, while A1B2C1 corresponds to L69.

(322) Language hypothesis tree



The fact that these three languages split into two branches reflects the difference in how the CBL handles global surface ambiguity versus global lexical ambiguity. Chapter 3 has shown that the CBL will learn all globally surface-ambiguous languages from a data set as a consequence of the branching that occurs when an error is detected on a form without a committed interpretation. Errors provide evidence for the grammar; when an error occurs on a form lacking a committed interpretation, branching for each possible interpretation enables the learner to explore different grammars simultaneously. In this case, when the overt form *YsX* is processed in the A1 hypothesis, it yields an error that causes A1 to extend three branches. L69 is globally surface-ambiguous with L76, and because it assigns a different interpretation to *YsX*, its corresponding language hypothesis occupies a different branch in the tree.

In contrast, although globally lexically-ambiguous languages like L75 and L76 have different grammars, the differences are not expressed in a way that a learner can distinguish on the basis of observed forms. L75 and L76 assign the same structural interpretations, and therefore they occupy the same branch of the tree. The difference between the languages instead lies in which underlying forms map to these

interpretations, and learning that requires knowing additional mappings beyond those observed.

The remainder of the learner's progress through the data is unexceptional. Repeated rounds of single-form and contrast pair learning will complete L69, which is merely globally surface-ambiguous with a paradigmatic equal and not a paradigmatic equal itself. Learning L75 and L76 will follow the procedure established in 4.1, committing to a consistent mismatch candidate to derive ranking information that the observed data cannot themselves produce. By committing to one consistent mismatch candidate the learner will derive the ranking for the corresponding paradigmatic equal, while implementing a branching procedure for each consistent mismatch candidate that produces a unique ERC, as discussed at the end of 4.1, would enable the learner to derive both paradigmatic equals.

When learning terminates, the language hypothesis corresponding to L69 will have the support in (323), with ranking (324) and lexicon (325).

(323) A1B2C1 (L69) – final support

ERC#	Morph. word	Input	Winner	Loser	PARSE-σ	*LAPSE	FT-BIN	MAXSTR	RMOST	LMOST	AFL	IAMB	FNF
a. 1P	r1s1	/ss-Y/	[(X)(sY)]	[s(Ys)]	W		L	W				W	L
b. 3P	r1s1	/ss-Y/	[(X)(sY)]	[(sY)s]	W		L	W	W	L	L		L
c. 10	r3s2	/sY-Y/	[(X)(sY)]	[s(sY)]	W	W	L						L
d. 2P	r1s1	/ss-Y/	[(X)(sY)]	[(Y)(sX)]				W	W	L			
e. 4P	r2s1	/Ys-s/	[(Ys)(X)]	[(X)(sY)]				W	L	W	L	L	W
f. 5P	r2s1	/Ys-s/	[(Ys)(X)]	[(X)(Ys)]				W	L	W	L		
g. 6P	r2s1	/Ys-s/	[(Ys)(X)]	[(Y)(Xs)]					W		L		
h. 8P	r2s1	/Ys-s/	[(Ys)(X)]	[(Y)(sX)]					W		L	L	W
i. 9P	r3s1	/sY-s/	[(X)(Ys)]	[(sY)(X)]					W	L	W	L	W
j. 7P	r1s1	/ss-Y/	[(X)(sY)]	[(Xs)(Y)]						W	W	W	L

(324) {PARSE-σ, *LAPSE} >> FT-BIN >> MAXSTRESS >> RMOST >> {IAMB, AFL, LMOST} >> FNF

(325) A1B2C1 (L69) - final lexicon

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/?Y/	/Ys/	/-s/	/-Y/

In the learned lexicon, r2 and r4 are both set to /Ys/, whereas r3 includes an unset feature in its first syllable as the ranking neutralizes /YY/ to /sY/. This lexicon and the ranking in (324) generate the map in (326) below. The shaded cells highlight the fact that r2s1 and r4s1 are identical, but recall that in the original map of L69, repeated in (327), r2 and r4 behave differently from each other.

(326) A1B2C1 map

r1 = /ss/	r2 = /Ys/	r3 = /?Y/	r4 = /Ys/	
[(X)(sY)]	[(Ys)(X)]	[(X)(Ys)]	[(Ys)(X)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

(327) L69 – original map

r1 = /ss/	r2 = /Ys/	r3 = /sY/	r4 = /YY/	
[(X)(sY)]	[(Ys)(X)]	[(X)(Ys)]	[(X)(Ys)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

What should be made of this result? It must be noted first that morpheme labels merely denote identity and do not affect how the learner interprets the data. The learner has no prior belief that different morphemes have different underlying forms, or that every possible underlying form will be attested in the observed data.

Now observe that L69 has three root behaviors and two suffix behaviors, summarized in the chart below. This chart says nothing about the particular labels of the root morphemes, only what their underlying forms must be based on their behaviors in relation to the suffixes.

(328) L69 morpheme behaviors – compressed

/ss/	/Ys/	/sY/ /YY/	
[(X)(sY)]	[(Ys)(X)]	[(X)(Ys)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

Following the chart, a root which surfaces as [(Ys)(X)] in the context of the suffix s1 but surfaces as [(X)(sY)] in the context of s2 must have the underlying form /Ys/. The map in (326) includes two roots which behave this way. Observe that in the learning data, (321), r2 and r4 do in fact behave alike. Thus, this map simply reflects the fact that the learner observed this particular behavior for two different roots, whose labels happened to be r2 and r4 in the data set. In the original map of L69 in (327), this behavior appears

only once, for the root labeled r2. On the other hand, the map of L69 in (327) shows that two other root morphemes, labeled r3 and r4, shared a different behavior, surfacing as [(X)(Ys)] in the context of s1 and [(X)(sY)] in the context of s2. The map of A1B2C1 in (326) includes just one root with this behavior, labeled r3. Again, the map is reflecting the fact that the learner observed this behavior only one time in the data set. Finally, there is a third root behavior that appears exactly once in both L69 and A1B2C1, in which the output is [(X)(sY)] regardless of suffix environment; the root with this behavior is labeled r1 in both maps. In short, there is no substantive difference between (326) and (327), only a difference in how many root morphemes evince each behavior and what label these morphemes are given.

This kind of result has been seen before, in 4.1.2, where the learner commits to a consistent mismatch candidate to learn L75 or L76. Depending on which candidate is chosen, the learner derives a different ranking and winds up with one of the lexica below. Both reflect data in which morphemes labeled r2 and r4 behave the same, but the different rankings will allow for these morphemes to have different underlying forms.

(329) A1B1C1 lexicon corresponding to L75 ranking

r1	r2	r3	r4	s1	s2
/ss/	/Y?/	/sY/	/Y?/	/-s/	/-Y/

(330) A1B1C1 lexicon corresponding to L76 ranking

r1	r2	r3	r4	s1	s2
/ss/	/Ys/	/?Y/	/Ys/	/-s/	/-Y/

To complete the discussion of learning from the data in (321), by committing to one consistent mismatch candidate in language hypothesis A1B1C1, the learner will derive a ranking corresponding to L75 or L76 – see (307) and (309) – and consequently derive the appropriate lexicon from (329) and (330) above. Regardless, the map produced is the same, (331); observe that it is globally surface-ambiguous with the map produced by A1B2C1 (corresponding to L69) in (326).

(331) A1B1C1 map

r1 = /ss/	r2 = /Ys/	r3 = /?Y/	r4 = /Ys/	
[(X)(sY)]	[(Y)(sX)]	[(X)(Ys)]	[(Y)(sX)]	s1 = /-s/
[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	[(X)(sY)]	s2 = /-Y/

The data in (321) are derived from the overt forms of the map of L75 in (306). In L75, r2 and r4 behave alike, and therefore the overt form *YsX* occurs twice in the data. If the data instead consisted of the overt forms from the map of L76 in (308), *XYs* would occur twice in the data set, for r3s1 and r4s1. The outcomes of learning would be only superficially different, in that the lexica would be adjusted to account for a different pair of words matching and reflecting the fact that a different pair of morphemes behave alike.

4.4.3 CONCLUSION

The learning data in (321) evince a dual ambiguity: they derive from languages which are globally lexically-ambiguous – L75 and L76 – and they derive from languages which are globally surface-ambiguous – L69 and L76. In total, the data can derive language hypotheses with three different rankings. The CBL does not have any further difficulty in learning languages which are both paradigmatic equals and globally surface-ambiguous, like L76, as it does learning languages which are globally ambiguous in only one way;

whatever challenges one kind of ambiguity creates are neither increased nor lessened by the addition of a second kind.

The interesting consequence of this dual ambiguity is the relationship it creates between languages like L75 and L69, which do not at first appear to be globally ambiguous with each other, and yet which can be learned by the same data that yields L76. As the paradigmatic equal of L75, L76 acts as a kind of gateway to learning L69. Because the data cannot distinguish L75 and L76, they cannot distinguish L75 and L69, either.

These ambiguities demonstrate the value of looking at data in terms of its morpheme behaviors, as in (316) and (317), rather than as catalog of observed forms, as in (321). Morpheme behaviors matter, in that they can reveal distinctions and similarities between data that are obscured in a simple list of overt forms, while the frequency of a particular form's occurrence in that list may not reveal anything more substantive about the grammar than the presence of homophones. In this example, language hypothesis A1B2C1 can derive the target L69 entirely from forms containing r1, r2 and r3; forms containing r4 contribute nothing about the grammar beyond what those containing r2 contribute.

Arranging the learning data for these languages according to morpheme behavior, as in (316) and (317), emphasize the value of characterizing global ambiguity with reference to morpheme behavior. Globally lexically-ambiguous languages like L75 and L76 have the same morpheme behaviors, including structural interpretations, whereas globally surface-ambiguous languages like L76 and L69 have the same morpheme behaviors, not

including structural interpretations. By this definition, L69 and L75 are clearly globally surface-ambiguous.

From this perspective, it seems quite empirically plausible that there are languages like L69, L76, and L75. They instantiate a set of morpheme behaviors which could derive from different rankings and include different structural interpretations, but at heart, the data is the same.

4.5 CHOOSING BETWEEN BRANCHES

Chapter 3 and the preceding sections of this chapter have described examples in which the learner derives two or three consistent language hypotheses from one set of learning data. These examples illustrate that each language in the Stress typology can be learned from overt forms only, and therefore they offer a clear demonstration of the CBL's success for learning; however, the question remains whether or not it benefits the learner to continue maintaining multiple consistent language hypotheses once learning ends. If all the learning data have been processed to completion, as this simulation assumes, then the learner could safely select just one language hypothesis to keep without fear that new information will render it inconsistent. In this case, the learner should select the language hypothesis least likely to overgenerate for a rich base of inputs. This section considers culling language hypotheses for this reason, using their r-measures (Tesar 2002, Prince and Tesar 2004).

The r-measure provides a means of evaluating the restrictiveness of a ranking. The r-measure is calculated by counting, for each faithfulness constraint, the number of markedness constraints that dominate it, and then adding these separate values. For the

Stress system, which includes a single faithfulness constraint, the r-measure of a language hypothesis equals the number of markedness constraints dominating MAXSTRESS. The more markedness constraints dominate MAXSTRESS, the fewer the surface forms likely to be found in the language.

The r-measure is especially useful for sifting out language hypotheses whose learned lexica do not reflect a rich base and therefore whose rankings may have unresolved conflicts that lead to overgeneration. In these language hypotheses, the learner derives a lexicon that contains only a subset of the inputs in the rich base, so that multiple morphemes are assigned the same underlying form.

There are two ways for the learner to derive a lexicon that includes only a subset of the rich base. The first is by application of the Fewest Set Features procedure, described in 4.2 for learning LgHyp83. In this language hypothesis, features that cannot be set using inconsistency detection methods are instead set to match their surface values. As a result, r1, r2, and r4, which behave alike in the target L83, are set alike in the final lexicon of LgHyp83, repeated in (332).

(332) LgHyp83 final lexicon

r1	r2	r3	r4	s1	s2
/Y?/	/Y?/	/sY/	/Y?/	/-?/	/-?/

The underlying form /ss/ is missing from the learned lexicon, and consequently the learner has not derived the ranking conditions to explicitly ensure that any input containing /ss/ as the root will map to an output allowed in the target language. It happens that this language hypothesis will not overgenerate, however, as the ranking derived by

BCD ensures that these inputs – /ss-s/ and /ss-Y/ – map to the same output as in the target. This will not always be the case, especially for language hypotheses which assign a subset of the rich base to their lexica in the second way: by standard application of inconsistency detection methods for setting features.

The learning data for target L78 produce just such a language hypothesis. L78 has predictable stress: each word is [(X)(sY)]. The learning data for this language are given in (333).

(333) Learning data for L78

XsY r1s1	*XsY* r1s2	*XsY* r2s1	*XsY* r2s2	*XsY* r3s1	*XsY* r3s2	*XsY* r4s1	*XsY* r4s2

L78 is the only language in the typology that produces this learning data – it has no globally ambiguous counterparts – yet the learner will pursue language hypotheses for each of the three interpretations of the overt form *XsY* in accordance with the standard response to error detection on an uncommitted overt form. LgHyp78, corresponding to the target, commits to the interpretation [(X)(sY)]. The learner derives the support in (334) from phonotactic information, and because stress is predictable, the lexicon, in (335), leaves all features unset.

(334) LgHyp78 support

ERC#	Morph. word	Input	Winner	Loser	RMOST	PARSE-σ	IAMB	*LAPSE	LMOST	AFL	FT-BIN	FNF	MAXSTR
2P	r1s1	/ss-Y/	[(X)(sY)]	[(Y)(sX)]	W				L				W
3P	r1s1	/ss-Y/	[(X)(sY)]	[(sY)s]	W	W			L	L	L	L	W
1P	r1s1	/ss-Y/	[(X)(sY)]	[s(Ys)]		W	W				L	L	W

(335) LgHyp78 final lexicon

r1	r2	r3	r4	s1	s2
/??/	/??/	/??/	/??/	/-?/	/-?/

In a second language hypothesis, LgHyp78-B, the learner commits to the interpretation [(Xs)(Y)] for the overt form and derives the support in (336).

(336) LgHyp78-B support

ERC#	Morph. word	Input	Winner	Loser	RMOST	PARSE-σ	*LAPSE	FT-BIN	FNF	IAMB	MAXSTR	LMOST	AFL
1P	r1s1	/ss-Y/	[(Xs)(Y)]	[s(Ys)]		W		L	L		W	L	L
2P	r1s1	/ss-Y/	[(Xs)(Y)]	[(X)(sY)]					W	L		L	L
3P	r1s1	/ss-Y/	[(Xs)(Y)]	[(X)(Ys)]							W	L	L

Whereas [(X)(sY)] is a potential optimum for any input, [(Xs)(Y)] is not. It is harmonically bounded by [(X)(Ys)] for all candidates except two: those containing the inputs /ss-Y/ and /Ys-Y/. Because it is impossible for every input in the rich base of the typology to map to [(Xs)(Y)], LgHyp78-B does not have a corresponding target language containing just this single output in its inventory, yet the learner can only rule out this language hypothesis given proof of its inconsistency. As long as it is possible to set the features of the morphemes to values that can map to [(Xs)(Y)], the hypothesis will remain consistent and it will survive. The ODL forces this outcome.

Each root is set to /?s/ using the test candidate /s**Y**-Y/→[(Xs)(Y)], which is harmonically bounded by /sY-Y/[(X)(Ys)]. Then, both suffixes can be set to /-Y/ by test

candidate /ss-**s**/→ [(Xs)(Y)], which is harmonically bounded by /ss-s/[(X)(Ys)]. The final lexicon appears in (337).

(337) LgHyp78-B final lexicon

r1	r2	r3	r4	s1	s2
/?s/	/?s/	/?s/	/?s/	/-Y/	/-Y/

LgHyp78-B survives because its lexicon allows for just the two inputs that can map to [(Xs)(Y)]. It explains all the observed learning data, and therefore is successful from the perspective of the learner despite the fact that it would overgenerate for a rich base. The table in (338) shows that there are five languages in the Stress typology that include the mappings /ss-Y/[(Xs)(Y)] and /Ys-Y/[(Xs)(Y)]. The ranking learned for LgHyp78-B corresponds to L95.

(338) Stress system languages containing /ss-Y/[(Xs)(Y)] and /Ys-Y/[(Xs)(Y)]

	/ss-s/	/Ys-s/	/sY-s/	/YY-s/	/ss-Y/	/Ys-Y/	/sY-Y/	/YY-Y/
L35	[s(Ys)]	[s(Ys)]	[s(Ys)]	[s(Ys)]	[(Xs)(Y)]	[(Xs)(Y)]	[s(Ys)]	[s(Ys)]
L41	[s(Ys)]	[(Ys)s]	[s(Ys)]	[s(Ys)]	[(Xs)(Y)]	[(Xs)(Y)]	[s(Ys)]	[s(Ys)]
L44	[s(Ys)]	[(Ys)(X)]	[s(Ys)]	[s(Ys)]	[(Xs)(Y)]	[(Xs)(Y)]	[s(Ys)]	[s(Ys)]
L89	[(X)(Ys)]	[(Ys)(X)]	[(X)(Ys)]	[(X)(Ys)]	[(Xs)(Y)]	[(Xs)(Y)]	[(X)(Ys)]	[(X)(Ys)]
L95	[(X)(Ys)]	[(X)(Ys)]	[(X)(Ys)]	[(X)(Ys)]	[(Xs)(Y)]	[(Xs)(Y)]	[(X)(Ys)]	[(X)(Ys)]

Following Tesar (2002), the r-measure can prove useful for deciding which of the two language hypotheses learned from the data in (333) to keep. Both language hypotheses find a way to produce the observed forms, but whereas LgHyp78 uses the ranking to derive the observed inventory, LgHyp78-B derives the observed inventory only by severely limiting the underlying forms in the lexicon and learns a less-restrictive ranking as a result. These different explanatory strategies are reflected in the r-measures of these

language hypotheses. For the ranking learned in LgHyp78, in (339), the r-measure is 8, the highest r-measure possible for this constraint set. The less-restrictive ranking learned for LgHyp78-B, in (340), has an r-measure of 6. By keeping the language hypothesis with the greatest r-measure, the learner keeps the one that corresponds to the intended target of the learning data.

(339) LgHyp78 ranking: r-measure = 8

{RMOST, PARSE-σ, IAMB, *LAPSE} >> {LMOST, AFL, FT-BIN, FNF} >> MAXSTRESS

(340) LgHyp78-B ranking: r-measure = 6

{RMOST, PARSE-σ, *LAPSE} >> {FT-BIN, FNF} >> IAMB >> MAXSTRESS >> {LMOST, AFL}

The learning data of the Stress system produce 14 language hypotheses like LgHyp78-B from 12 different datasets. Each language hypothesis corresponds to a superset language in the typology. Each of these language hypotheses has a lower r-measure than at least one sibling branch that does correspond to a true target of the learning data. Eliminating all branches but the one with the highest r-measure as a final learning step would eliminate all 14 of these language hypotheses.

The r-measure is introduced here as a potential criterion for choosing among language hypotheses; however, it is not ideal, as in many cases it fails to provide a meaningful justification for the choice. This selection procedure would apply to language hypothesis branches, which account for the same data and are explicitly not related by restrictiveness

to each other: no branch can be a subset of another, as their grammars are responsible for generating the same set of overt forms.. Two language hypothesis branches may differ in their r-measures, therefore, yet both be as restrictive as necessary to generate all and only the observed forms of their corresponding target languages.

To illustrate, compare the two language hypotheses corresponding to L76 and L69. The stratified hierarchy learned for LgHyp76 appears with the structural commitments in (341). The ranking of LgHyp69 and its structural commitments, which differ from LgHyp76's, follow in (342). As these are branch hypotheses derived from the same learning data, the inventories of both language hypotheses are the same and include only *XsY, YsX,* and *XYs*.

(341) LgHyp76: r-measure = 4

a. {PARSE-σ, *LAPSE} >> {AFL, FT-BIN} >> MAXSTRESS >> RMOST >> IAMB >> {FNF, LMOST}
b. [(X)(sY)], [(Y)(sX)], [(X)(Ys)]

(342) LgHyp69: r-measure = 3

a. {PARSE-σ, *LAPSE} >> FT-BIN >> MAXSTRESS >> RMOST >>{LMOST, AFL, IAMB} >> FNF
b. [(X)(sY)], [(Ys)(X)], [(X)(Ys)]

LgHyp76's ranking has an r-measure of 4, compared to the r-measure of 3 for LgHyp69's ranking, but the values do not reflect a meaningful difference in restrictiveness. Given the rich base of the Stress system, both rankings yield all and only the committed structural interpretations of the target languages corresponding to their

respective language hypotheses. Random selection of a language hypothesis to keep would be just as valid in this case as selection based on r-measure. The same is true for the language hypotheses learned in chapter 3, corresponding to L4, L5, and L6. The rankings for these language hypotheses, repeated below, all have an r-measure of 2. There would be no harm in randomly keeping just one of these, as each corresponds to a language in the typology and none will overgenerate, but it is also not clear that there is anything to be gained.

(343) LgHyp4 (15A1BC2): r-measure = 2

FT-BIN >> PARSE-σ >> MAXSTRESS >> IAMB >> {RMOST, FNF}>> {LMOST, AFL, *LAPSE}

(344) LgHyp5 (15A1BC1): r-measure = 2

FT-BIN >> PARSE-σ >> MAXSTRESS >> RMOST >> {LMOST, AFL, IAMB} >> {FNF, *LAPSE}

(345) LgHyp6 (15A1BC1): r-measure = 2

IAMB >> FNF>> MAXSTRESS >> {RMOST, PARSE-σ, FT-BIN} >> {LMOST, AFL, *LAPSE}

To the extent that it is worthwhile to maintain a single language hypothesis after learning ends, the r-measure provides a criterion – albeit an imperfect one – to distinguish between options. At its most useful, it will cull out explanations like LgHyp78-B, which fit the data by limiting the learned lexicon to a subset of the underlying forms in the rich

base. Otherwise, the r-measure provides no better justification than random selection for keeping one language hypothesis over another.

4.6 Conclusion

Evaluating the success of a learner depends in great part on understanding the languages for which the learner is responsible. Most languages in the Stress system are like L4, L5, and L6, from Chapter 3, and the Commitment-Based Learner (CBL) readily derives their corresponding language hypotheses using the standard procedures outlined in that chapter. However, a learner must be able to derive language hypotheses for all languages, including those whose complex relationships to one another complicate and interfere with the standard learning procedures. This chapter thus completes the illustration of the CBL that was begun in Chapter 3, by introducing the data for these challenging relationships and demonstrating how the CBL handles them.

Most importantly, this chapter introduces the previously unrecognized paradigmatic equality relationship. Paradigmatic equals are globally lexically-ambiguous with each other, with the consequence that the learning data for one can yield a language hypothesis consistent with the other. This similarity is problematic in particular for a learner that sets features by inconsistency detection. The language hypotheses for paradigmatic equals ultimately reach a point of persistent uncertainty that can only be overcome by adding new information in the absence of certainty.

The solution proposed here is to add a committed mapping, just as the learner does routinely during phonotactic learning, with the ERC by Consistent Mismatch (ECM) procedure. The mapping contributes new ranking information and disambiguates the

languages, enabling the learner to continue setting features. This method works successfully in all nine pairs of paradigmatic equals in the Stress typology; however, much further investigation remains about both the phenomenon of paradigmatic equality and the options for resolving global lexical ambiguity.

The Stress system also contains paradigmatic subsets, which are likewise problematic for setting features by inconsistency detection. For the languages discussed in this chapter, the problem is not simply that there is a superset language which interferes with setting features in the subset language, but that the mappings permitted by the subset language itself are too broad to permit the necessary feature-setting by inconsistency detection. Some test candidates in the subset language are actual mappings in the language, and therefore they can never be inconsistent with the corresponding language hypothesis. The solution advocated here is Tesar's Fewest Set Features procedure, which sets features one at a time to match their surface correspondents. Furthermore, the chapter has demonstrated that ECM and Fewest Set Features can work in combination to learn languages that participate in both kinds of paradigmatic relationships.

The illustration of the CBL is not complete without understanding the products of its learning: namely, the consistent language hypotheses derived from a single dataset. In particular, this chapter exposes the sensitivity of the CBL to the morpheme behaviors expressed in the learning data. From one set of data, the learner will derive all languages that have exactly the same morpheme behaviors, down to the same structural interpretations – paradigmatic equals, the globally lexically ambiguous languages – and

all the languages that have the same morpheme behaviors, considering only overt forms – the globally surface ambiguous languages.

Finally, this chapter has shown how the CBL allows for some final branches to accommodate learning data by limiting the range of underlying forms in the lexicon, resulting in a less-restrictive ranking that overgenerates when predicted across the rich base. The r-measure is considered as a criterion for making a final selection between branches when learning concludes.

5 CONCLUSION

This dissertation has introduced the Commitment-Based Learner, or CBL, which simultaneously learns a ranking and lexicon for a target language from its overt forms by making incremental commitments to structural information. The CBL commits to structural interpretations for overt forms, feature values in the lexicon, and input-output mappings as W-L pairs in a stored support. A commitment allows the learner to advance a language hypothesis from a position of certainty with respect to the committed structure, and all information entailed by that commitment becomes incorporated into the hypothesis for further learning. With these commitments the learner can exploit the mutual dependency of structural interpretations and underlying forms, using inconsistency detection to progressively narrow the space of possible languages consistent with the observed data.

Language hypotheses for the CBL store these committed structures, but they also form a larger structure among themselves as branches from the initial language hypothesis Hyp0. This branching structure, a component of the CBL's incorporation of the Inconsistency Detection Learner (IDL) is the key to managing structural ambiguity. Whereas the learner can make piecewise commitments to individual ranking requirements and single feature values within a morpheme, commitments to structural interpretations are made in full, not by the foot, because there is no clear way to determine where a foot in isolation should be placed to ultimately form the most harmonic interpretation.

Contrast this with setting features in an output-driven map by inconsistency detection. The relative similarity lattice for an observed form defines the range of possible underlying forms. Most importantly, there is a "most similar" underlying form and a "least similar" underlying form defined in the lattice, and while these forms are independent of the current ranking, they provide a structure that can be used with the ranking to determine which particular features must be set. There is no counterpart to the relative similarity lattice for learning structural interpretations, because what is optimal, or most harmonic, depends entirely on the ranking.

Since the piecewise commitment approach is unavailable for structural interpretations, the learner must commit to an entire interpretation, but which one? Extending separate language hypothesis branches for each interpretation settles this problem. Thus, as a general strategy, the CBL makes piecewise commitments wherever possible; however, if a hidden structure cannot be decomposed into its component parts, the CBL branches to evaluate the different grammars resulting from different commitments for a complete structure.

5.1 Success and efficiency of Commitment-Based Learning

In computer simulations performed over a typology of the 97 languages in the Stress system, the Commitment-Based Learner successfully learns each language from its overt forms, including all globally ambiguous languages. A target is successfully learned if the learner derives a lexicon and restrictive ranking that generate the target's outputs, including correct structural interpretations. More precisely, the learner must set in the

lexicon the underlying values of all contrastive features in the target and derive a restrictive ranking consistent with the stratified hierarchy of the target's skeletal basis.

It is plain that the CBL performs more efficiently than an exhaustive search. The Stress system contains nine constraints, giving rise to 362,880 different possible rankings. It contains four disyllabic roots and two monosyllabic suffixes, with each syllable bearing a binary stress feature, so that 1024 different lexica are possible. In all, there are 371,589,102 different systems possible from these grammars and lexica. By comparison, for all the simulations performed, the maximum number of language hypothesis branches created for any one dataset is 13.

Just how efficient the CBL is in relation to other learners remains an open question, but even without comparing figures, the CBL's reliance on inconsistency detection with Multi-Recursive Constraint Demotion (MRCD) suggests that it will fare favorably against its competitors. Use of inconsistency detection enables the learner to eliminate large spaces of grammar hypotheses at once and permanently, so that new information is evaluated against only the grammar hypotheses that include the previously committed structures. The cost of the CBL using inconsistency detection and MRCD is the required storage structures: a support for W-L pairs and a separate lexicon; however, as the number of stored commitments grows, the learner's ability to detect inconsistencies generally grows as well. Storing these relatively few commitments is justified by the work they do to pinpoint the possible grammar hypotheses for the data.

In the learning simulations for the Stress system, the CBL stores an average of 43 W-L pairs for all the language hypothesis branches created from a single dataset; the

maximum number of W-L pairs stored for any one dataset is 101. For lexical commitments, the CBL sets 15 features on average across all the branches created for a dataset; the maximum number of set features is 34. These are already low numbers compared to the roughly 370 million systems possible from the constraints and morphemes used in the Stress system, but recall that the CBL does more than learn a language consistent with the data: it learns all of the languages consistent with the data, including the globally ambiguous ones. If the CBL stopped after deriving one restrictive language hypothesis consistent with the data, these numbers would be reduced somewhat further.

To compare the CBL with other error-driven learners, such as those based on the Gradual Learning Algorithm (GLA), one could ask, in the manner of Tesar (1997, 2000), how many iterations of Recursive Constraint Demotion (RCD) must the learner make to derive the ranking of the target? Apoussidou's GLA learner is the most relevant error-driven example because it shares the CBL's goal of learning both a ranking and underlying forms, but without the CBL's separate support and lexicon structures. In simulations, this learner is given repeated exposures to data to learn the ranking and the underlying forms of a handful of morphemes. Unfortunately, the number of updates required before converging on a ranking is not stated, but extrapolating from the fact that 1 million forms were processed in similar simulations suggests that there were many thousands of updates. Although this example cannot provide a direct comparison in efficiency between these learners, it seems unlikely that the maximum 101 applications of RCD (one for each W-L pair stored) required by the CBL to learn a language in the Stress system could be matched by the updates required by this GLA learner.

Another basis for comparison is on the number of forms processed. Jarosz (to appear) takes this approach in order to create a baseline for efficiency using a random search learner. The baseline is offered for learning a ranking from structurally ambiguous data where underlying forms are not at issue. As explained in chapter 1, the random search learner is surprisingly efficient, especially as compared to exhaustive search, but simulations of the Inconsistency Detection Learner (IDL) over the same kind of learning data show that the IDL is extremely efficient, as judged by RCD applications. Again, extrapolating from the RCD applications given the number of unique words in the data suggests that the IDL requires far fewer than the 10,000 forms needed for the random search learner to succeed, as explained in section 1.3.4. While the simulations of the CBL did not keep track of how many forms were processed, the CBL's use of inconsistency detection suggests that it will compare favorably to a random search learner as well, especially one that must learn a restrictive ranking in order to succeed.

Finally, the CBL's inconsistency detection strategy compares quite favorably to a learner that evaluates every grammar and lexicon possibility, as in the early implementation of the Maximal Likelihood Learning of Lexicons and Grammars, or MLG (Jarosz 2006). Eliminating many possibilities at the cost of stored commitments is surely more efficient than updating every possibility after every observed form. The later sampling versions of the MLG appear to be more efficient, but do not yet achieve 100% success in simulations (Jarosz, to appear).

In sum, while it is not possible at this time to make a direct comparison between the CBL and these other learners, the simulations executed over the Stress system typology

suggest that the CBL is an efficient learner. However, it is true that the small size of the Stress system has helped to limit the number of hypotheses constructed and the numbers of stored ERCs and set features. Increasing the size of the system will certainly increase these numbers, and it will be enlightening to see in larger systems how much inconsistency detection serves to reduce the number of consistent, simultaneous hypotheses the learner must process at any given time. The answer is likely to be very much, as suggested by the performance of the CBL here and by the performance of the original IDL.

5.2 AREAS FOR FURTHER WORK

5.2.1 GLOBAL AMBIGUITIES AND PARADIGMATIC RELATIONSHIPS

This dissertation has also expanded the understanding of global ambiguity among languages by introducing paradigmatic equality, a previously unrecognized relationship exhibited by nine pairs of languages in the Stress system typology. Paradigmatic equality is a property of global lexical ambiguity, evinced by languages with different rankings and different input-output mappings, yet with all the same morpheme behaviors, including structural interpretations. In contrast, globally surface-ambiguous languages have the same morpheme behaviors as realized by overt forms only. Global ambiguity can thus be defined as the sharing of identical morpheme behaviors between two languages, with surface and lexical ambiguity distinguished by whether the structural interpretations assigned to the overt realizations of the morpheme behaviors are also identical in the languages. The dissertation additionally has shown that a language that is globally lexically ambiguous with one language may also be globally surface-ambiguous with another.

Scaling up the system for evaluation should also provide more insight into the paradigmatic relationships discussed in chapter 4. Do paradigmatic equals multiply or simply disappear as the Stress system grows? If they disappear, what is the property of the Stress system as it is defined in this dissertation that enables them to exist within it? If they increase, and if they appear in other systems as well, then how does the ERC by Consistent Mismatch procedure advocated in 4.1.2 perform in learning them?

Additionally, within the Stress system seven of the nine paradigmatic equals were also paradigmatic subsets of other languages. If the system were larger, could a paradigmatic superset itself be a paradigmatic subset of yet another language? And naturally, what are the consequences for the learner if paradigmatic subsets could be nested in this way? It should be noted that although paradigmatic equality could be unique to the Stress system, paradigmatic subsets were first discovered within the Stress/Length system by Tesar (to appear), and therefore it would not be unexpected to find that they exist in other systems and in larger versions of the Stress system as well.

Finally, if paradigmatic equality turns out to exist in other systems, does the relationship manifest empirically, and if so, how? One possibility is that paradigmatic equality is the source of intra-speaker variation[30], potentially identifiable through a Wug test (Gleason 1958). Chapter 4 discusses an implementation of ERC by Consistent Mismatch that branches when there is more than one informative consistent mismatch candidate. As a result of branching, the learner derives the two language hypotheses that correspond to the paradigmatic equals. Speakers who vary between two responses on a given Wug test could be demonstrating alternating choices between these hypotheses.

[30] I thank Shigeto Kawahara for suggesting this possibility.

5.2.2 OTHER LEARNING ISSUES

There are several major issues for learning that this implementation of the CBL does not address. First, there is the question of how to model learning stages beyond the rough phonotactic/ non-phonotactic divide presented in this dissertation. The CBL is capable of adopting learning strategies that can model stages of acquisition as long as they allow for piecewise, incremental commitments. For example, the Error-Selective Learner (ESL: Tessier 2007) models intermediate stages of phonological development by waiting for a critical mass of errors before triggering error-driven learning. The ESL makes use of an additional structure – the Error Cache, for temporary error storage – but otherwise, its use of a support and error-driven learning make it compatible for incorporation into the Commitment-Based Learner.

Similarly, this implementation of the CBL has deferred the work of learning the morphological decompositions of the observed forms. In these simulations, the learner has access to morphological information as soon as phonotactic learning ends, but this information must itself be learned. It is likely, however, that the commitment-based approach can be extended to manage hidden morphological structure in addition to the hidden prosodic and lexical structures handled in this dissertation. For example, knowledge of morphemic alternations and contrasts is not essential for setting features, as the CBL can use the techniques of the Output-Driven Learner to identify what underlying feature values must be in some single forms. It is conceivable that features set from single forms could be used to identify like or unlike morphemes in other forms.

5.2.3 OUTPUT-DRIVENNESS AND THE CBL

This dissertation has demonstrated the success of the CBL in learning the languages of a reasonably small system that nonetheless exhibits properties that learners must be sensitive to, such as restrictiveness relations. Applying the CBL to a scaled-up version of the Stress system, as well as to other systems with a wider, if not more interesting, array of constraints will help to support through concrete simulations the assertion here that this is a successful learner. Lurking behind this assertion, however, is the big question raised by the CBL's exploitation of the property of output-drivenness: what does the CBL do when it confronts the data of a language whose map is not output-driven, such as one that includes chain shifts?

What is essential to the CBL is its ability to make incremental commitments to ranking requirements and feature values. Incorporating the methods of the Output-Driven Learner (ODL) offers the CBL an efficient means of making these lexical commitments, but the CBL could adopt an alternative approach that achieves the same effect. The alternative must be able to function using only the information available within the CBL's language hypotheses: the structural information provided by overt forms and the ranking information provided by W-L pairs in the support. For the ODL, an overt form is used to define the complete space of possible lexical hypotheses for a word and stored ranking information narrows that space through inconsistency detection, enabling features to be set independently. Non-output-driven maps will change the space of lexical hypotheses, but any alternative feature-setting component incorporated into the CBL also will need to be able to search this space using only partial ranking information. An efficient alternative to the ODL is likely to find a way to structure the space, or organize

the search within it, in a way that continues to allow inconsistency detection a significant role in identifying underlying feature values.

5.3 FINAL SUMMATION

The Commitment-Based Learner introduced in this dissertation has been shown in simulations to successfully manage structural ambiguity while learning a lexicon and grammar, an accomplishment that requires the learner to simultaneously address the hidden structures of surface representations and the lexicon. The computer simulations provide a rigorous test of the CBL: the linguistic system used in the simulations generates languages with significant interaction between the hidden structures, and the CBL is tested for its ability to learn each language in the constructed Stress system typology from its overt forms alone. The CBL succeeds in every case.

The key feature of the CBL is its commitment to partial information, in particular to the structural interpretations of overt forms, lexical feature values, and ranking conditions. The growth of the stores of these commitments is both a sign that a language hypothesis is developing and the primary means of that development: the CBL can begin learning without any prior commitments, but as commitments are made, they provide a foundation for efficiently inferring other information. At any given moment in the learning process, a language hypothesis can be committed to the structural interpretations for some overt forms but not others, the underlying values of some features but not others, and the ranking information that resolves some conflicts, but not others. Successful language hypotheses need not have commitments associated with all structures, as long as the stored commitments can generate all of the observed data and

the structural "gaps" do not allow for errors. Thus, a feature need not have a committed underlying value if its value always neutralizes, and a hierarchy need not be a total ranking as long as no conflicts remain and all observed forms can be generated without error. All successful language hypotheses created in learning simulations for the Stress system include committed interpretations for each overt form, but even these need not be necessary as long as the other commitments ensure that only one surface representation can be generated for a given form.

Finally, the typology used to simulate the CBL has not only offered a test of the learner, it has exposed the phenomenon of paradigmatic equality. This dissertation has related paradigmatic equality to the previously-recognized paradigmatic subset relationship and established differences between the two. Both relationships are identified within the Stress system typology, in some cases within the same language, and the CBL's strategies for learning these languages are analyzed and applied with successful outcomes. Restrictiveness is traditionally characterized by a subset relation between sets of outputs, but paradigmatic equals and subsets involve substantially more complex relationships of restrictiveness between language paradigms which any learner of paradigmatically structured phonological systems must address.

.

APPENDIX A SUPPORTS FOR SKELETAL BASES

The following are the supports for the skeletal bases of consistent mappings for *tírasim* and *mèvugár* from section 2.1. The combination labels given refer to chart (42) in section 2.1.

A-1 COMBINATION (42)F: /TIRAS+IM/[(TÍRA)SIM] AND /MEVUGÁR/[(MÈVU)(GÁR)]

Input	W~L	MAXSTR	FNF	PARSE-σ	IAMB	FT-BIN	AFL	LMOST	RMOST	*LAPSE
/mevugár/	[(mèvu)(gár)] ~ [me(vúgar)]	W	L	W		L	L	L		
/tiras+im/	[(tíra)sim] ~ [(tirá)(sìm)]		W	L	L	W	W			L
/mevugár/	[(mèvu)(gár)] ~ [me(vugár)]			W	L	L	L	L		W
/tiras+im/	[(tíra)sim] ~ [ti(rásim)]						W	W	L	L

A-2 COMBINATION (42)I: /TÍRAS+IM/[(TÍRA)SIM] AND /MEVUGAR/[(MÈ)(VUGÁR)]

Input	W~L	MAXSTR	RMOST	IAMB	AFL	LMOST	PARSE-σ	*LAPSE	FNF	FT-BIN
/tíras+im/	[(tíra)sim] ~ [ti(rasím)]	W	L	L	W	W			W	
/tíras+im/	[(tíra)sim] ~ [(tí)rasim]		W	L			W		W	W
/mevugar/	[(mè)(vugár)] ~ [(mevú)gar]		W		L	L	W		L	L
/mevugar/	[(mè)(vugár)] ~ [(mè)(vúgar)]			W					L	
/mevugar/	[(mè)(vugár)] ~ [me(vúgar)]			W			W		L	L
/tíras+im/	[(tíra)sim] ~ [(tirá)(sìm)]				W		L	L	W	W
/mevugar/	[(mè)(vugár)] ~ [me(vugár)]						W	W	L	L

A-3 COMBINATION (42)K: /TÍRAS+IM/[(TÍ)RASIM] AND /MEVUGAR/[(MÈ)(VUGÁR)]

Input	W~L	MAXSTR	IAMB	RMOST	AFL	LMOST	PARSE-σ	*LAPSE	FNF	FT-BIN
/tíras+im/	[(tí)rasim] ~ [ti(rasím)]	W		L	W	W	L			L
/tíras+im/	[(tí)rasim] ~ [(tíra)sim]		W	L			L		L	L
/mevugar/	[(mè)(vugár)] ~ [(mevú)gar]			W	L	L	W		L	L
/tíras+im/	[(tí)rasim] ~ [(tí)(rasìm)]				W		L	L	W	
/mevugar/	[(mè)(vugár)] ~ [me(vugár)]						W	W	L	L

A-4 COMBINATION (42)M: /TÍRAS+IM/[(TÍRA)SIM] AND /MEVUGÁR/[(MÈ)(VUGÁR)]

Input	W~L	MAXSTR	AFL	LMOST	RMOST	IAMB	PARSE-σ	*LAPSE	FNF	FT-BIN
/tíras+im/	[(tíra)sim] ~ [ti(rasím)]	W	W	W	L	L			W	
/mevugár/	[(mè)(vugár)] ~ [(mevú)gar]	W	L	L	W		W		L	L
/tíras+im/	[(tíra)sim] ~ [(tíra)(sìm)]		W				L	L	W	W
/tíras+im/	[(tíra)sim] ~ [(tí)rasim]				W	L	W		W	W
/mevugár/	[(mè)(vugár)] ~ [me(vugár)]						W	W	L	L

A-5 COMBINATION (42)N: /TÍRAS+IM/[(TÍRA)SIM] AND /MEVUGÁR/[(MÈVU)(GÁR)]

Input	W~L	MAXSTR	FNF	RMOST	PARSE-σ	*LAPSE	IAMB	FT-BIN	AFL	LMOST
/tíras+im/	[(tíra)sim] ~ [ti(rasím)]	W	W	L			L		W	W
/mevugár/	[(mèvu)(gár)] ~ [me(vúgar)]	W	L		W			L	L	L
/tíras+im/	[(tíra)sim] ~ [(tíra)(sìm)]		W		L	L		W	W	
/mevugár/	[(mèvu)(gár)] ~ [me(vugár)]				W	W	L	L	L	L

A-6 COMBINATION (42)O: /TÍRAS+IM/[(TÍ)RASIM] AND /MEVUGÁR/[(MÈ)(VUGÁR)]

Input	W~L	MAXSTR	IAMB	AFL	LMOST	RMOST	PARSE-σ	*LAPSE	FNF	FT-BIN
/tíras+im/	[(tí)rasim] ~ [ti(rasím)]	W		W	W	L	L			L
/mevugár/	[(mè)(vugár)] ~ [(mevú)gar]	W		L	L	W	W		L	L
/tíras+im/	[(tí)rasim] ~ [(tíra)sim]		W			L	L		L	L
/tíras+im/	[(tí)rasim] ~ [(tí)(rasìm)]			W			L	L	W	
/mevugár/	[(mè)(vugár)] ~ [me(vugár)]						W	W	L	L

APPENDIX B RUBY CODE FOR THE CBL

B-1 STRESS_FEAT.RB

```
# Author: Crystal Akers, based on Bruce Tesar's sl/stress_feat

require 'feature'

module SF

  # A stress feature is a Feature of type STRESS.
  # It has two possible feature values, represented
  # by the constants UNSTRESSED, MAIN_STRESS.
  class Stress_feat < Feature
    #-- Symbols are used as lightweight, readable constants ++

    # Feature type stress
    STRESS = :stress
    # Feature value unstressed syllable
    UNSTRESSED = :unstressed
    # Feature value main stress syllable
    MAIN_STRESS = :main_stress
```

```
# Returns a new stress feature, with the feature value unset.
def initialize
  super(STRESS) # Pass the feature type to Feature#initialize.
end

# Returns true if the feature instance is unstressed; false otherwise.
def unstressed?
  self.value == UNSTRESSED
end

# Returns true if the feature instance is main_stress; false otherwise.
def main_stress?
  self.value == MAIN_STRESS
end

# Sets the feature to the value UNSTRESSED.
def set_unstressed
  self.value = UNSTRESSED
  self
end
```

```
# Sets the feature to the value MAIN_STRESS.
 def set_main_stress
  self.value = MAIN_STRESS
  self
 end


 # Returns a string representation of the feature:
 # "stress=<value>"
 def to_s
  return "stress=unset" if unset?
  return "stress=unstressed" if unstressed?
  return "stress=main_stress" if main_stress?
 end
```

```
  #-- Generic interface ++

    # Passes each possible value for this feature to the given code block,
    # one at a time (iterator style). This generic interface should be
    # used by all feature types.
    def each_value
      yield UNSTRESSED
      yield MAIN_STRESS
    end

  end # class Stress_feat

end # module SF
```

B-2 SYLLABLE.RB

```
 # encoding: UTF-8
#
# Author: Crystal Akers, based on Bruce Tesar's sl/syllable
#

require 'sf/stress_feat'

module SF

  # A syllable for the SF system has one feature, stress. It also can have an
  # affiliated morpheme.
  #
  # Learning algorithms are expected to use the "generic" interface, consisting
  # of the methods #each_feature() and #get_feature(). The method #each_feature()
  # is an iterator that yields each feature of the syllable in turn,
  # allowing other routines to work with syllables without knowing in advance
  # how many or what types of features they have.
  class Syllable
```

```
# Returns a syllable, initialized to the parameters if provided. Otherwise,
# returns a syllable with unset features, and an empty string for the
# morpheme.
def initialize(stress=Stress_feat.new, morph="")
  @stress = stress
  @morpheme = morph # label of the morpheme this syllable is affiliated with.
end

# A duplicate makes copies of the features, so that they may be altered
# independently of the original's features.
def dup
  self.class.new(@stress.dup, @morpheme)
end
```

```
# Protected accessors, only used for #==()
attr_reader :stress # :nodoc:
protected :stress # :nodoc:

# Returns true if this syllable matches _other_, a syllable, in the value of
# the stress feature and morpheme identity.
def ==(other)
  return false unless other.class == self.class
  return false unless @stress==other.stress
  return false unless @morpheme==other.morpheme
  return true
end

# The same as ==(other).
def eql?(other)
  self==other
end

# Returns true if the syllable's stress feature has the value main_stress.
def main_stress?
  @stress.main_stress?
end
```

```
# Returns true if the syllable's stress feature has the value unstressed.
def unstressed?
  @stress.unstressed?
end

# Returns true is the stress feature is unset.
def stress_unset?
  @stress.unset?
end

# Returns the morpheme that this syllable is affiliated with.
def morpheme
  @morpheme
end

# Sets the syllable's stress feature to the value main_stress.
def set_main_stress
  @stress.set_main_stress
  self
end
```

```
# Sets the syllable's stress feature to the value unstressed.
  def set_unstressed
    @stress.set_unstressed
    self
  end

  # Set the morpheme that this syllable is affiliated with to _m_.
  def set_morpheme(m)
    @morpheme = m
    self
  end

  # Returns the number of syllables in this "word element".
  # A syllable always contains 1 syllable. This allows us to
  # easily add up the number of syllables in a word, without
  # having to worry about whether each element of the word is
  # an unfooted syllable or a foot: each element knows how
  # to answer the question "how many syllables do you have?"
  def syllable_count
    return 1
  end
```

```
# Iterates over the single syllable.
  def each_syllable

    yield self

  end


  # Returns a string representation of the syllable, consisting of one

  # character, denoting the stress feature:

  #

  # unstressed:: [s]

  # main stress:: [Y]

  # unset:: [?]

  def to_s

    stress_s = case

    when main_stress? then "Y"

    when unstressed? then "s"

    when stress_unset? then "?"

    end

    return stress_s

  end
```

```
def to_gv
    base = "morpheme_type_not_defined"
    if morpheme.root? then
      base = "p"
    elsif morpheme.suffix? then
      base = "k"
    elsif morpheme.prefix? then
      base = "t"
    end
    stress_s = case
    when main_stress? then "á"
    when unstressed? then "a"
    when stress_unset? then "?"
    end
    return base + stress_s
  end

  # Iterator over the features of the syllable.
  def each_feature() # :yields: feature
    yield @stress
  end
```

```
# Returns the syllable's _type_ feature. Raises an exception if the
    # syllable does not have a feature of type _type_.
    def get_feature(type)
      each_feature{|f| return f if f.type==type}
      raise "SF::Syllable#get_feature(): parameter #{type.to_s} is not a valid feature type."
    end

 # Sets the syllable's _feat_type_ to _value_.
    def set_feature(feat_type,val)
      f = get_feature(feat_type)
      f.value = val
    end


  end # class Syllable


end # module SF
```

B-3 OUTPUT_SYLLABLE.RB

```
# encoding: UTF-8
#
# Author: Crystal Akers, based on Bruce Tesar's sl/syllable
#

require 'sf/syllable'

module SF

  # An output syllable for the SF system has one feature, stress. The stress feature
  # can have the value primary stress or unstressed. Secondary stress is assigned
  # with a T/F parameter. The combinations of stress feature and secondary stress
  # create the following output syllables:
  #   Primary stress and false for sec stress = primary stress syllable
  #   Unstressed and false for sec stress = unstressed syllable
  #   Unstressed and true for sec stress = secondary stress syllable
  #   (Primary stress and true for sec stress is not allowed)
  # Output syllable also can have an affiliated morpheme.

  class Output_Syllable < Syllable
```

```
# Returns a syllable, initialized to the parameters if provided. Otherwise,
# returns a syllable with unset features, an empty string for the
# morpheme, and does not have secondary stress.
def initialize(stress=Stress_feat.new,sec_stress =false, morph="" )
  @stress = stress
  @sec_stress = sec_stress
  @morpheme = morph # label of the morpheme this syllable is affiliated with
  if @stress.main_stress? and @sec_stress==true then
    raise "Cannot have both primary and secondary stress"
  end
end

# A duplicate makes copies of the features, so that they may be altered
# independently of the original's features.
def dup
  self.class.new(@stress.dup, @sec_stress, @morpheme)
end
```

```
# Protected accessors, only used for #==()
attr_reader :stress # :nodoc:
protected :stress # :nodoc:

# Returns true if this syllable matches _other_, a syllable, in the values
# the stress feature, and morpheme identity.
def ==(other)
  return false unless other.class == self.class
  return false unless @stress==other.stress
  return false unless @sec_stress == other.sec_stress?
  return false unless @morpheme==other.morpheme
  return true
end

# The same as ==(other).
def eql?(other)
  self==other
end

# Returns true if the syllable's stress feature has the value
# main_stress.
def main_stress?
  @stress.main_stress?
end
```

```
# Returns true if the syllable's stress feature has the value
 # unstressed and  the sec_stress parameter is true.
 def sec_stress?
   !@stress.main_stress? and @sec_stress == true
 end

 # Returns true if the syllable's stress feature has the value
 # main_stress or the sec_stress is true.
 def stressed?
   @stress.main_stress? or @sec_stress == true
 end

 # Returns true if the syllable's stress feature has the value
 # unstressed.
 def unstressed?
   @sec_stress == false and super
 end

 # Returns true is the stress feature is unset.
 def stress_unset?
   @stress.unset? and @sec_stress == false
 end
```

```
# Returns the morpheme that this syllable is affiliated with.
def morpheme
  @morpheme
end

# Sets the syllable's stress feature to the value main_stress.
def set_main_stress
  @stress.set_main_stress
  self
end

# Sets the syllable's sec_stress parameter to true and the stress feature value
# to unstressed.
def set_sec_stress
  @stress.set_unstressed
  @sec_stress = true
  self
end
```

```
# Sets the syllable's stress feature to the value unstressed and sets
  # the sec_stress parameter to false.
  def set_unstressed
    @stress.set_unstressed
    @sec_stress = false
    self
  end

  # Set the morpheme that this syllable is affiliated with to _m_.
  def set_morpheme(m)
    @morpheme = m
    self
  end

  # Returns the number of syllables in this "word element".
  # A syllable always contains 1 syllable. This allows us to
  # easily add up the number of syllables in a word, without
  # having to worry about whether each element of the word is
  # an unfooted syllable or a foot: each element knows how
  # to answer the question "how many syllables do you have?"
  def syllable_count
    return 1
  end
```

```
# Iterates over the single syllable.
def each_syllable
  yield self
end


# Returns a string representation of the syllable, consisting of one
# character, denoting the stress feature:
#
# unstressed:: [s]
# main stress:: [Y]
# sec_stress:: [X]
# unset:: [?]
def to_s
  stress_s = case
  when main_stress? then "Y"
  when sec_stress? then "X"
  when unstressed? then "s"
  when stress_unset? then "?"
  end
  return stress_s
end
```

```
def to_gv
  base = "morpheme_type_not_defined"
  if morpheme.root? then
    base = "p"
  elsif morpheme.suffix? then
    base = "k"
  elsif morpheme.prefix? then
    base = "t"
  end
  stress_s = case
  when main_stress? then "á"
  when sec_stress? then "à"
  when unstressed? then "a"
  when stress_unset? then "?"
  end
  return base + stress_s
end


# Iterator over the features of the syllable.
def each_feature()
  yield @stress
end
```

```
    # Returns the syllable's _type_ feature. Raises an exception if the
    # syllable does not have a feature of type _type_.
    def get_feature(type)
      each_feature{|f| return f if f.type==type}
      raise "SF::Syllable#get_feature(): parameter #{type.to_s} is not a valid feature type."
    end

  end # class Output_Syllable


end # module SF
```

B-4 FOOT.RB

```
# Author: Crystal Akers, based on Bruce Tesar's Ruby_on_RORG foot.rb

require 'sf/syllable'

module SF

# A foot consists of one or two syllables.
class Foot
```

```
 # A complete foot must be created at once, with the
  # syllables being provided as arguments to #new().
  # The '*' operator in front of the parameter args
  # stores the parameters passed into the method in
  # an array, referenced by args. This allows the method
  # to accommodate a variable number of passed
  # parameters; in this case, one or two syllables.
  def initialize(*args)
    raise "No empty feet!" if args.empty?
    raise "No suprabinary feet!" if args.size > 2
    raise "One syllable must be stressed" if args.all? {|syl| syl.unstressed?}
    raise "Only one syllable may be stressed" if args.size == 2 and !args.any? {|syl|
syl.unstressed?}
    @syllables = args
  end



  # Returns the number of syllables in the foot.
  def syllable_count
    return @syllables.size
  end
```

```
# Returns the first syllable in the foot.
 def first_syl
   return @syllables[0]
 end

 # Returns the second syllable in the foot. Returns nil if the
 # foot only has one syllable.
 def second_syl
   return @syllables[1]
 end

 # Returns the last syllable in the foot, whether it is the first syllable
 # or the second.
 def last_syl
   if @syllables.size == 1
     return @syllables[0]
   else return @syllables[1]
   end
 end
```

```
# Iterator over each syllable in a foot
 def each_syllable
   yield self.first_syl
   yield self.second_syl if @syllables.size == 2
 end

 # Returns true if the stress feature of any syllable in the foot has the value
 # main_stress.
 def main_stress?
   @syllables.any? { |syl| syl.main_stress?  }
 end
```

```
# Returns true if this foot and _other_ foot have the same number of syllables and the
# syllables themselves are equivalent.
def ==(other)
  val = false
  if self.class == other.class then
    if self.syllable_count == other.syllable_count then
      val = true if self.first_syl == other.first_syl && self.second_syl == other.second_syl
    end
  end
  return val
end

# Equivalent to ==().
def eql?(other)
  self==other
end
```

```
  #  A duplicate makes copies of the syllables, so that their features may be altered
  #  independently of the original syllables' features.
    def dup
      if self.syllable_count == 2 then
        return Foot.new(self.first_syl.dup, self.second_syl.dup)
      else return Foot.new(self.first_syl.dup)
      end
    end

  # Represents a foot as a pair of parentheses containing
  # the to_s representation of each syllable in the foot,
  # without separators.
  def to_s
    outstr = "("
    @syllables.each {|syl| outstr += syl.to_s}
    return outstr += ")"
  end

end # class Foot

end # module SF
```

B-5 SF_OUTPUT.RB

```
# Author: Crystal Akers
#

require 'output'
require 'sf/foot'
require 'sf/output_syllable'

module SF

  class SF::Sf_output < Output

    # A newly created output is empty, with no morphological word, so that
    # it can be built up piece by piece.
    def initialize
      @morphword = nil
    end
```

```
# Returns the number of syllables in the output, by adding up the number of
 # syllables in each element of the word.
 def syllable_count
  return inject(0){|total,element| total + element.syllable_count}
 end

 # Iterator over the elements of the output (unparsed syllables or feet)
 def each_element()
  self.each { |el| yield el }
 end

 # Iterator over each syllable in an output element
 # (unparsed syllable or foot)
 def each_syllable()
  self.each do |el|
   el.each_syllable { |syl| yield syl }
  end
 end
```

```
# Creates an array containing each syllable in the output in order.
  def syl_list
    list = Sf_output.new
    self.each_syllable { |syl| list << syl }
    return list
  end


 # Returns a copy of the output as an overt form, containing a duplicate
 # of each syllable and a duplicate of the morphological word.
 def overt
   overt_copy = Sf_output.new
   self.each_syllable {|syl| overt_copy << syl }
   overt_copy.morphword = @morphword.dup unless @morphword.nil?
   return overt_copy
 end
```

```
# Returns a copy of the output, containing a duplicate of each
  # correspondence element and a duplicate of the morphological word.
  def dup
    # Call Array#map to get an array of dups of the elements, and add
    # them to a new Output.
    copy = Sf_output.new.concat(super.map { |el| el.dup })
    copy.morphword = @morphword.dup unless @morphword.nil?
    return copy
  end

  end # class SF::Sf_output

end # module SF
```

B-6 SF_WORD.RB

```
# Author: Crystal Akers
#

require 'rubot'
require 'candidate'
require 'input'
require 'output'
require 'io_correspondence'
require 'word'
require 'sf/sf_output'

module SF

  # An Sf_word is a Word with an Sf_output.
  class SF::Sf_word < Word
```

```
# A word starts out with an empty input and empty sf_output by default,
# but input and output can be optionally passed as parameters.
# The linguistic system is a mandatory parameter, and
# the correspondence relation is initially empty;
# correspondences must be added after the word is created.
def initialize(system, input=Input.new, output=Sf_output.new)
  super(system, input, output)
end
```

```
# Returns a deep copy of the word, with distinct input syllables and features,
# distinct output elements and features, and appropriately revises UI and
# IO correspondences.
def dup
  copy = Sf_word.new(@system)
  copy.label = self.label
  copy.opt=self.opt?
  # Make local references to reduce number of method calls
  c_input = copy.input
  c_output = copy.output
  c_io_corr = copy.io_corr
  # dup the morphological word for the copy's input and output
  unless input.morphword.nil?
    c_morphword = input.morphword.dup
    c_input.morphword = c_morphword
    c_output.morphword = c_morphword
  end
  # Make a copy of the input, constructing updated versions of the UI
  # and IO correspondences using the new copies of the input syllables.
  input.each do |old_in_syl|
    new_in_syl = old_in_syl.dup # duplicate the old input syllable
    c_input << new_in_syl # add the dup to the copy
    # Get the corresponding underlying syllable in the original's UI correspondence.
```

```
      # If it exists, add a correspondence to the copy between this underlying
      # syllable and the duplicated input syllable in the copy.
      under_syl = input.ui_corr.under_corr(old_in_syl)
      c_input.ui_corr << [under_syl,new_in_syl] unless under_syl.nil?
      # get the corresponding output syllable in the original word's IO corresp.
      out_syl = @io_corr.out_corr(old_in_syl)
      c_io_corr << [new_in_syl,out_syl] unless out_syl.nil?
    end
    # Make a copy of the output, adjusting the O part of IO correspondence.
    output.each do |old_out_el|
      new_out_el = old_out_el.dup # duplicate the old output element (foot or unparsed
syllable)
      c_output << new_out_el # add the dup to the copy
      # find the IO pair.
      new_out = []; old_out = []
      new_out_el.each_syllable {|syl| new_out << syl}
      old_out_el.each_syllable {|syl| old_out << syl}
      gen = SyncEnumerator.new(new_out,old_out)
      gen.each do |new_out_syl,old_out_syl|
        corr_pair = c_io_corr.find{|p| p[1].equal?(old_out_syl)}
        corr_pair[1] = new_out_syl unless corr_pair.nil? # replace old with new output syl.
      end
    end
```

```
    copy.eval # set the constraint violations
    return copy
  end

  # Returns the overt form of the word.
  def overt()
    return self.output.overt
  end

end # class Sf_word

end # module SF
```

B-7 DATA.RB

```
# Author: Crystal Akers, based on Bruce Tesar's sl/data
#
# This adds, to the module SF, routines for generating data of various types
# within the SF (stress-feet) linguistic system.

require 'sf/system'
require 'sf/grammar'
require 'sf/syllable'
require 'morpheme'
require 'morph_word'
require 'underlying'
require 'lexical_entry'
require 'most_harmonic'
require 'rubot'
require 'competition'
require 'competition_list'
require 'hypothesis'
require 'otlearn/data_manip'
require 'facets/array/product' # Adds cartesian product to class Array.

module SF
```

```
# Returns a list of lexical entries for the possible morphemes
# of morphological type _type_ with underlying form length
# _uf_length_ (measured in syllables). Each morpheme is assigned
# a label with a distinct number, with _id_number_ providing
# the base (the first generated morpheme gets number _id_number_ + 1,
# the next generated gets number _id_number_ + 2, etc.).
# If a code block is given, each generated lexical entry is passed to it.
def SF.generate_morphemes(uf_length, type, id_number)
  if type==Morpheme::ROOT then label_pref = "r"
  elsif type==Morpheme::PREFIX then label_pref = "p"
  elsif type==Morpheme::SUFFIX then label_pref = "s"
  else raise "Unrecognized morpheme type."
  end
  lexical_entry_list = []
  SF.generate_underlying_forms(uf_length) do |uf|
    id_number += 1
    morph = Morpheme.new("#{label_pref}#{id_number.to_s}", type)
    uf.each {|s| s.set_morpheme(morph)}
    lexical_entry_list << Lexical_Entry.new(morph,uf)
  end
  # If a code block was given, run it on each lexical entry.
  lexical_entry_list.each {|le| yield le} if block_given?
```

```
  return lexical_entry_list
end
```

```
# Generates all possible underlying forms with _uf_length_ syllables.
# If _uf_length_ == 0, a list with a single empty UF is returned.
# If a code block is given, each UF is passed to it. Note: Syllables in
# underlying forms can only be unstressed or have main stress.
def SF.generate_underlying_forms(uf_length)
  raise "UF length cannot be <0!" if uf_length<0
  uf_list = [Underlying.new]
  uf_length.times do
    new_uf_list = []
    SF.generate_syllables do |s|
      uf_list.each do |uf|
        new_uf = (uf.dup << s.dup)
        new_uf_list << new_uf
      end
    end
    uf_list = new_uf_list
  end
  # If a code block was given, run it on each underlying form.
  uf_list.each {|uf| yield uf} if block_given?
  return uf_list
end
```

```
# Generate all possible syllables (possible combinations of feature values).
 # Note that the sf system include secondary stress as a property of output syllables,
 # not as a feature; therefore, none of the syllables generated will have secondary stress.
 # If a code block is given, each syllable is passed to the code block.
 # Returns a list of the possible syllables.
 def SF.generate_syllables
   syl_list = [] << Syllable.new
   base_syl = Syllable.new
   base_syl.each_feature do |f|
     fresh_syl_list = []
     f.each_value do |v|
       syl_list.each do |s|
         syl = s.dup
         syl.get_feature(f.type).value = v
         fresh_syl_list << syl
       end
     end
     syl_list = fresh_syl_list
   end
   # If a code block was given, run it on each syllable.
   syl_list.each {|s| yield s} if block_given?
   return syl_list
 end
```

```
# Generates the optimal candidates with respect to constraint
# hierarchy _hier_ for each input in _inputs_, using the lexicon
# in grammar _gram_. The hierarchy in _gram_ is set to _hier_.
# _gram_ needs to already contain a lexicon with entries for all
# of the morphemes appearing in the inputs.
# Returns a list of the optimal candidates of the language.
def SF.generate_language(hier, inputs, gram)
  competitions = inputs.map{|i| SYSTEM.gen(i)}
  comp_list = Competition_list.new.concat(competitions)
  gram.hierarchy = hier
  comp_mh = comp_list.map{|comp| MostHarmonic.new(comp,gram.hierarchy)}
  # each competition returns a list of winners; collapse to one-level list.
  lang = comp_mh.inject([]){|winners, mh_list| winners.concat(mh_list) }
  lang.each{|winner| winner.opt=true}
  return lang
end
```

```
def SF.competitions_from_morphwords(words, gram)
  # Generate the corresponding input for each morphological word
  inputs = words.map{|mw| SYSTEM.input_from_morphword(mw,gram)}
  # Generate the corresponding competition for each input
  competitions = inputs.map{|i| SYSTEM.gen(i)}
  # Convert the array of competitions into a proper Competition_list.
  comp_list = Competition_list.new.concat(competitions)
  comp_list.label = "SF"
  return comp_list
end
```

def SF.generate_competitions_2r1s

```
  # Generate the morphemes
  roots = SF.generate_morphemes(2, Morpheme::ROOT, 0)
  suffixes = SF.generate_morphemes(1, Morpheme::SUFFIX, 0)
  # Create a new grammar, and add all of the morphemes to the lexicon.
  gram = Grammar.new
  roots.each{|root_le| gram.lexicon.add(root_le)}
  suffixes.each{|suf_le| gram.lexicon.add(suf_le)}
  # Morphology: create all combinations of one root and one suffix
  word_parts = roots.product(suffixes)
  words = word_parts.map do |parts|
    # Add the morphemes of the combination to a new morphological word.
    parts.inject(MorphWord.new){|w,le| w.add(le.morpheme); w}
  end
  # Generate the competition for each morphword
  comp_list = competitions_from_morphwords(words, gram)
  return comp_list, gram
end
```

```
def SF.generate_default_inputs
  gram = Grammar.new
  # Generate the possible monosyllabic roots and suffixes
  roots = SF.generate_morphemes(1, Morpheme::ROOT, 0)
  suffixes = SF.generate_morphemes(1, Morpheme::SUFFIX, 0)
  # Create all combinations of root-suffix
  # Make sure the roots are first in the cartesian product, because they
  # must be added first when constructing MorphWords.
  word_parts = roots.product(suffixes)
  #
  # Next line: how to include free roots as (monomorphemic) words
  # word_parts += roots.product()
  #
  # Convert each morpheme-tuple into a MorphWord
  words = word_parts.map{|t| t.inject(MorphWord.new){|w,le| w.add(le.morpheme);
w}}
  # Add the morphemes to the lexicon
  roots.each{|root_le| gram.lexicon.add(root_le)}
  suffixes.each{|suf_le| gram.lexicon.add(suf_le)}
  # Generate the input for each morph_word.
  inputs = words.map{|mw| SYSTEM.input_from_morphword(mw,gram)}
  return inputs, gram
end
```

```
  #--
  # Data for testing purposes.
  #++

  def SF.generate_words_lang_a
    inputs, gram = SF.generate_default_inputs
    competitions = inputs.map{|i| SYSTEM.gen(i)}
    comp_list = Competition_list.new.concat(competitions)
    winner_list, hyp = OTLearn::generate_learning_data_from_competitions(comp_list,
SF.hier_3,Grammar)
    return winner_list, hyp
  end

  def SF.generate_outputs_lang_a
    inputs, gram = SF.generate_default_inputs
    lang = SF.generate_language(SF.hier_2a, inputs, gram)
    outputs = lang.map{|w| w.output}
    return outputs
  end
```

```
#--
# Hierarchies
#++

 # Constraints: Lmost Rmost AFL ParSyl FtBin FNF Iamb Lapse MaxStress

   # This is the hierarchy for language 2 in the 2r1s typology
  def SF.hier_2
   hier = Hierarchy.new
   hier << [SYSTEM.iamb] << [SYSTEM.ftbin] << [SYSTEM.fnf] <<
[SYSTEM.parsyl] << [SYSTEM.maxstress] << [SYSTEM.rmost] << [SYSTEM.afl] <<
[SYSTEM.lmost] << [SYSTEM.lapse]
   return hier
  end

    # This is the hierarchy for language 3 in the 2r1s typology
  def SF.hier_3
   hier = Hierarchy.new
   hier << [SYSTEM.rmost] << [SYSTEM.ftbin] << [SYSTEM.afl] << [SYSTEM.lmost]
<<   [SYSTEM.parsyl] << [SYSTEM.maxstress] << [SYSTEM.iamb] << [SYSTEM.fnf]
<< [SYSTEM.lapse]
   return hier
  end
```

```
  # This is the hierarchy for language 6 in the 2r1s typology
  def SF.hier_6
    hier = Hierarchy.new
    hier << [SYSTEM.iamb] << [SYSTEM.fnf] << [SYSTEM.maxstress] <<
[SYSTEM.ftbin] <<  [SYSTEM.rmost] << [SYSTEM.parsyl] << [SYSTEM.afl] <<
[SYSTEM.lmost] << [SYSTEM.lapse]
    return hier
  end

  # This is the hierarchy for language 13 in the 2r1s typology
  def SF.hier_13
    hier = Hierarchy.new
    hier  << [SYSTEM.maxstress] << [SYSTEM.lmost] << [SYSTEM.afl] <<
[SYSTEM.ftbin] << [SYSTEM.rmost] << [SYSTEM.parsyl] << [SYSTEM.lapse] <<
[SYSTEM.fnf] << [SYSTEM.iamb]
    return hier
  end
```

```
# This is the hierarchy for language 37 in the 2r1s typology
 def SF.hier_37
   hier = Hierarchy.new
   hier  << [SYSTEM.maxstress] << [SYSTEM.ftbin] << [SYSTEM.parsyl] <<
[SYSTEM.lapse] <<  [SYSTEM.rmost] << [SYSTEM.fnf] << [SYSTEM.iamb] <<
[SYSTEM.afl] << [SYSTEM.lmost]
   return hier
 end

# This is the hierarchy for language 39 in the 2r1s typology (2A1B1)
 def SF.hier_39
   hier = Hierarchy.new
   hier  << [SYSTEM.fnf] << [SYSTEM.ftbin] << [SYSTEM.maxstress] <<
     [SYSTEM.iamb] << [SYSTEM.parsyl] << [SYSTEM.rmost] << [SYSTEM.lapse] <<
     [SYSTEM.afl] << [SYSTEM.lmost]
   return hier
 end
```

```
 # This is the hierarchy for language 55 in the 2r1s typology
 def SF.hier_55
  hier = Hierarchy.new
  hier << [SYSTEM.lapse] << [SYSTEM.maxstress] << [SYSTEM.lmost] <<
[SYSTEM.fnf] <<  [SYSTEM.rmost] << [SYSTEM.parsyl] << [SYSTEM.iamb] <<
[SYSTEM.ftbin] << [SYSTEM.afl]
  return hier
 end

 # This is the hierarchy for language 58 in the 2r1s typology
 def SF.hier_58
  hier = Hierarchy.new
  hier << [SYSTEM.lapse] << [SYSTEM.parsyl] << [SYSTEM.lmost] <<
[SYSTEM.ftbin] << [SYSTEM.fnf] << [SYSTEM.rmost] << [SYSTEM.maxstress] <<
[SYSTEM.iamb] << [SYSTEM.afl]
  return hier
 end
```

```
 # This is the hierarchy for language 66 in the 2r1s typology
 def SF.hier_66
   hier = Hierarchy.new
   hier << [SYSTEM.maxstress] << [SYSTEM.iamb] << [SYSTEM.rmost] <<
[SYSTEM.lmost] <<  [SYSTEM.afl] << [SYSTEM.parsyl]  << [SYSTEM.lapse] <<
[SYSTEM.fnf] << [SYSTEM.ftbin]
   return hier
 end

 # This is the hierarchy for language 69 in the 2r1s typology (37A1B2C1)
 def SF.hier_69
   hier = Hierarchy.new
   hier  << [SYSTEM.maxstress] << [SYSTEM.parsyl] << [SYSTEM.lapse] <<
     [SYSTEM.ftbin] << [SYSTEM.rmost] << [SYSTEM.iamb] << [SYSTEM.afl] <<
     [SYSTEM.lmost] << [SYSTEM.fnf]
   return hier
 end
```

```
# This is the hierarchy for language 75 in the 2r1s typology
  def SF.hier_75
    hier = Hierarchy.new
    hier << [SYSTEM.maxstress] << [SYSTEM.parsyl] << [SYSTEM.lapse] <<
[SYSTEM.ftbin] <<  [SYSTEM.afl] << [SYSTEM.iamb] << [SYSTEM.fnf] << [
SYSTEM.rmost] << [SYSTEM.lmost]
    return hier
  end

  # This is the hierarchy for language 80 in the 2r1s typology
  def SF.hier_80
    hier = Hierarchy.new
    hier  << [SYSTEM.maxstress] << [SYSTEM.lapse] << [SYSTEM.lmost] <<
[SYSTEM.fnf] <<  [SYSTEM.parsyl] << [SYSTEM.iamb] << [SYSTEM.ftbin] <<
[SYSTEM.afl] << [SYSTEM.rmost]
    return hier
  end
```

```
# This is the hierarchy for language 87 in the 2r1s typology. Corresponds to
  # Set 17, Lang A1B1.
  def SF.hier_87
    hier = Hierarchy.new
    hier  << [SYSTEM.parsyl] << [SYSTEM.lapse] << [SYSTEM.fnf] <<
      [SYSTEM.ftbin] << [SYSTEM.afl] << [SYSTEM.maxstress] << [SYSTEM.iamb]
<<  [SYSTEM.lmost] << [SYSTEM.rmost]
    return hier
  end

  # This is the hierarchy for language 88 in the 2r1s typology
  def SF.hier_88
    hier = Hierarchy.new
    hier  << [SYSTEM.maxstress] << [SYSTEM.rmost] << [SYSTEM.lmost] <<
[SYSTEM.afl] << [SYSTEM.lapse] << [SYSTEM.parsyl] << [SYSTEM.ftbin] <<
[SYSTEM.fnf] << [SYSTEM.iamb]
    return hier
  end
```

```
    # This is the hierarchy for language 94 in the 2r1s typology
  def SF.hier_94
    hier = Hierarchy.new
    hier << [SYSTEM.parsyl] << [SYSTEM.lapse] << [SYSTEM.fnf] <<
[SYSTEM.ftbin] << [SYSTEM.afl] << [SYSTEM.maxstress] <<  [SYSTEM.iamb] <<
[SYSTEM.rmost] << [SYSTEM.lmost]
    return hier
  end



end # module SF
```

B-8 GRAMMAR.RB

```
# Author: Crystal Akers, based on Bruce Tesar's sl/grammar
#

require 'rubot'
require 'rcd'
require 'comparative_tableau'
require 'sf/system'
require 'lexicon'

module SF

  # A grammar for the SF linguistic system consists of a reference to
  # the SF::System linguistic system, a constraint hierarchy, and a lexicon.
  class Grammar
    attr_accessor :hierarchy, :lexicon

    # Stores the linguistic system associated with this grammar.
    # In this case, the SF (stress-feet) linguistic system.
    @@system = System.instance
```

```
# Returns a new grammar. If a hierarchy or a lexicon are not provided
 # as parameters, default initial values are used:
 # * the default initial lexicon is empty
 # * the default initial hierarchy results from applying RCD to an empty
 #   comparative tableau.
 def initialize(hier=default_initial_hierarchy, lex=default_initial_lexicon)
   @hierarchy = hier
   @lexicon = lex
 end


 # Returns a reference to the linguistic system associated with this grammar.
 def system
   @@system
 end
```

```
# Returns a copy of the grammar, with duplicates of the hierarchy and
 # the lexicon.
 # The duplicate of the hierarchy contains references to the same constraint
 # objects, but duplicated strata.
 # The duplicate of the lexicon contains duplicates of the lexical entries,
 # and the duplicate lexical entries contain duplicates of the underlying
 # forms but references to the very same morpheme objects.
 def dup
   return self.class.new(@hierarchy.dup, @lexicon.dup)
 end

 # Returns a copy of the grammar, with a duplicate of the hierarchy, but
 # a reference to the very same lexicon object.
 # The duplicate of the hierarchy contains references to the same constraint
 # objects, but duplicated strata.
 def dup_hier_only
   return self.class.new(@hierarchy.dup, @lexicon)
 end
```

```
# Returns the underlying form for the given morpheme, as stored in
  # the grammar's lexicon. Returns nil if the morpheme does not appear
  # in the lexicon.
  def get_uf(morph)
    lex_entry = @lexicon.find{|entry| entry.morpheme==morph} # get the lexical entry
    return nil if lex_entry.nil?
    return lex_entry.uf  # return the underlying form
  end

  private

  # The default initial hierarchy is the one resulting from applying RCD
  # to an empty comparative tableau.
  def default_initial_hierarchy
    Rcd.new(Comparative_tableau.new('empty',system.constraints)).hierarchy
  end
```

```
# The default lexicon is simply a new (empty) lexicon.
    def default_initial_lexicon
      Lexicon.new
    end

  end # class Grammar

end # module SF
```

B-9 SYSTEM.RB

```
# Author: Crystal Akers, from Bruce Tesar's sl/system
#

require 'singleton'
require 'constraint_eval'
require 'ui_correspondence'
require 'rubot'
require 'competition'
require 'sf/sf_word'
require 'sf/foot'
require 'sf/output_syllable'
require 'sf/sf_output'

# For SyncEnumerator
if RUBY_VERSION =~ /^1\.9/ then
  require 'generator19'  # Tesar's ruby 1.9 version
else
  require 'generator'    # the ruby 1.8 version
end

module SF
```

```
# Contains the core elements of the SF (stress-feet) linguistic system.
# It defines the constraints of the system, provides the #gen(_input_) method
# generating the candidates for _input_, provides a method for
# constructing the phonological input corresponding to a morphological
# word with respect to a given grammar, and provides a method for parsing
# a phonological output for a morphological word into a full structural
# description with respect to a given grammar.
#
# This is a singleton class.
class System
  include Singleton

  # Create local references to the constraint type constants.
  # This is strictly for convenience, so that the "Constraint_eval::"
  # prefix doesn't have to appear in the constraint definitions below.
  # Note: done this way because constants cannot be aliased.

  # Indicates that a constraint is a markedness constraint.
  MARK = Constraint_eval::MARK
  # Indicates that a constraint is a faithfulness constraint.
  FAITH = Constraint_eval::FAITH
```

```
# Creates the constraint list and freezes it, as well as freezing
    # each of the constraints. Creation of <em>constraint_list</em>
    # also initializes the constraint attributes (nolong(), etc.).
    def initialize
      initialize_eval_procs
      @constraints = constraint_list # private method creating the list
      @constraints.each {|con| con.freeze} # freeze the constraints
      @constraints.freeze # freeze the constraint list
    end

    # Returns the list of constraints (each constraint is a Constraint object).
    # Note that the returned list is frozen, as are the constraints that
    # it contains.
    def constraints() return @constraints end

    # Returns the markedness constraint lmost.
    def lmost() return @lmost end

    # Returns the markedness constraint rmost.
    def rmost() return @rmost end

    # Returns the markedness constraint afl.
    def afl() return @afl end
```

Returns the markedness constraint parsyl.

def parsyl() return @parsyl end

Returns the markedness constraint ftbin.

def ftbin() return @ftbin end

Returns the markedness constraint fnf.

def fnf() return @fnf end

Returns the markedness constraint iamb.

def iamb() return @iamb end

Returns the markedness constraint lapse.

def lapse() return @lapse end

Returns the faithfulness constraint maxstress.

def maxstress() return @maxstress end

```
# Accepts parameters of a morph_word and a grammar. It builds an input form
   # by concatenating the syllables of the underlying forms of each of the
   # morphemes in the morph_word, in order. It also constructs the correspondence
   # relation for the input, with an entry for each corresponding pair of
   # underlying/input syllables.
   def input_from_morphword(mw, gram)
     input = Input.new
     input.morphword = mw
     mw.each do |m| # for each morpheme in the morph_word, in order
       uf = gram.get_uf(m)
       raise "Morpheme #{m.label} has no entry in the lexicon." if uf.nil?
       uf.each do |syl| # for each syllable of the underlying form
         in_syl = syl.dup
         input.push(in_syl) # add a duplicate of the underlying syllable to input.
         input.ui_corr << [syl,in_syl] # create a correspondence between underlying and
input syllables.
       end
     end
     return input
   end
```

```
  # gen takes an input, generates all candidate words for that input, and returns
    # them in the form of a Competition. All candidates are marked
    # as not optimal.
    # All candidates in the competition share the same input object. The outputs
    # for candidates may also share some of their syllable objects.
    def gen(input)
      #Creates all of the outputs equal in length to the input, then creates new
      #words with each of these outputs.
      start_rep = SF::Sf_output.new
      # create a list of partial outputs
      not_long_enough = [start_rep]
      final_output_list = [] #List of complete outputs, each containing main stress


      # Create the list of all syllables and feet that can appear in a word.
      element_list = []
      element_list << Output_Syllable.new.set_unstressed #unstressed element
      # degenerate feet
      element_list << Foot.new(Output_Syllable.new.set_main_stress)
      element_list << Foot.new(Output_Syllable.new.set_sec_stress)
      # trochaic and iambic primary feet
      element_list <<
Foot.new(Output_Syllable.new.set_main_stress,Output_Syllable.new.set_unstressed)
```

```
    element_list <<
Foot.new(Output_Syllable.new.set_unstressed,Output_Syllable.new.set_main_stress)
    # trochaic and iambic secondary feet
    element_list <<
Foot.new(Output_Syllable.new.set_sec_stress,Output_Syllable.new.set_unstressed)
    element_list <<
Foot.new(Output_Syllable.new.set_unstressed,Output_Syllable.new.set_sec_stress)

    # Set the word length in syllables
    word_length = input.length

    # Keep processing not_long_enough until no structures remain that need adding
    # on to (that is, have fewer than the required number of syllables).
    until not_long_enough.empty?
      base = not_long_enough.shift # take the first output in the queue
      # Separately extend copies of the base output with each possible word element.
      element_list.each do |el|
        next_output = base # Copy the partial output
        # Extends next_output with the word element el unless both already include
        # main stress.
        # If the newly extended output is long enough and contains main stress,
        # it is moved to the final_output_list. Otherwise, if it is not yet long enough,
        # then it is added to the back of not_long_enough to be extended further.
```

```
        unless (next_output.any? {|element| element.main_stress?}) and el.main_stress?
then
          extended_out = extend_output(next_output, el)
          if extended_out.syllable_count == word_length and (extended_out.any?
{|element| element.main_stress?}) then
            final_output_list << extended_out
          elsif extended_out.syllable_count < word_length then
            not_long_enough << extended_out
          end
        end
      end
    end

    # Create a new word for each of the completed outputs
    final_word_list = []
    final_output_list.each do |output|
      #Creates a new word with full input, but empty output, io_corr
      new_word = Sf_word.new(SYSTEM,input,output)
      new_word.output.morphword = input.morphword
      # Sets the morpheme of the output syllable equal to the morpheme of the input
syllable
      # Also sets the io_corr pairs for the input and output syllables
      g = SyncEnumerator.new(input,output.syl_list)
```

```
    g.each do |in_syl,out_syl|
      out_syl.set_morpheme(in_syl.morpheme)
      new_word.io_corr << [in_syl,out_syl]
    end
    final_word_list << new_word
  end

  # Put actual candidates into competition, calling eval on each to set
  # the constraint violations.
  competition = Competition.new
  final_word_list.each{|c| c.eval; competition.push(c)}
  return competition
end
```

```
# Constructs a full structural description for the given output using the
 # lexicon of the given grammar. The constructed input will stand in
 # 1-to-1 IO correspondence with the output; an exception is thrown if
 # the number of syllables in the lexical entry of each morpheme doesn't
 # match the number of syllables for that morpheme in the output.
 def parse_output(output, gram)
  mw = output.morphword
  # If any morphemes aren't currently in the lexicon, create new entries, with
  # the same number of syllables as in the output, and all features unset.
  mw.each do |m|
   unless gram.lexicon.any?{|entry| entry.morpheme==m} then
    under = Underlying.new
    output.each_syllable do |syl|
     if syl.morpheme == m then
      under << SF::Syllable.new.set_morpheme(m)
     end
    end
    gram.lexicon << Lexical_Entry.new(m,under)
   end
  end
  # Construct the input form
  input = input_from_morphword(mw, gram)
  word = Sf_word.new(SYSTEM,input,output)
```

```
  # create 1-to-1 IO correspondence
  if input.size != output.syllable_count then
    raise "Input size #{input.size} not equal to output size #{output.syllable_count}."
  end
  gen = SyncEnumerator.new(input, output.syl_list)
  gen.each do |in_syl,out_syl|
    word.io_corr << [in_syl,out_syl]
    if in_syl.morpheme != out_syl.morpheme then
      raise "Input syllable morph #{in_syl.morpheme.label} != " +
        "output syllable morph #{out_syl.morpheme.label}"
    end
  end
  word.eval
  return word
end
```

```
# Constructs a list containing all the outputs that correspond to the given overt form.
 def parse_overt(overt, gram)
  # Construct the structural interpretations by creating two lists of partial
  # interpretations. Interpretations in openft are incomplete: each ends with
  #  a syllable that could be parsed into a foot. Interpretations in closedft
  # are complete: they end with either a foot or an unparsed syllable that will remain
  # unparsed.
  closedft =[]; openft =[]
  overt.each do |syl|
   # copy the partial interpretation lists to old_*, and reset the lists to empty.
   old_clft = closedft; old_opft = openft
   closedft =[]; openft =[]
   # If old_opft is empty and syl is unstressed, add an interpretation with
   # syl as the first syllable of an incomplete foot
   if old_opft.empty? and syl.unstressed? then
    interp = Sf_output.new
    interp << syl
    openft << interp.dup
   else
    # Otherwise, for each interpretation, check that the last syllable of the
    # interpretation and syl are not both stressed or unstressed. If they have different
    # stress feature values, extended the interpretation with a binary foot
    # using last_syllable and syl. If they have the same stress values, do nothing
```

```
  # (eliminate this partial interpretation, because it will be a duplicate
  # of a partial interpretation in the closedft list).
  until old_opft.empty?
    interp = Sf_output.new
    i = old_opft.shift
    last_syllable = i.pop
    unless syl.stressed? == last_syllable.stressed? then
      # Copy each remaining element in i and add to interp. Also complete the binary
    # foot by extending the last syllable with syl.
      unless i.empty?
        i.each { |el| interp << el }
      end
      interp << Foot.new(last_syllable, syl)
      # With the completed foot, the interpretation is added to the closedft list
      closedft << interp.dup
    end
  end
end
# If old_clft is empty, add an interpretation with syl as the first closed element --
# either an unparsed syllable or a unary foot.
if old_clft.empty? then
  interp = Sf_output.new
  if syl.stressed? then
```

```
    open_interp = Sf_output.new
    open_interp << syl
    openft << open_interp.dup
    interp << Foot.new(syl)
    closedft << interp.dup
   else
    interp << syl
    closedft <<  interp.dup
   end
  else
   # Extend each old closed interpretation with another closed element (either an
   # unparsed syllable or a binary foot. Also extend with an open element (either
   # the beginning of an iamb or a trochee, depending on syl.
   until old_clft.empty?
    i = old_clft.shift
    interp = Sf_output.new
    # copy each element in i and add to interp. Also copy and add syl to begin the
    # potential binary foot.
    i.each do |el|
     interp << el
    end
    open_interp = Sf_output.new
    open_interp = interp.dup << syl
```

```
      openft << open_interp.dup
      if syl.stressed? then
        interp << Foot.new(syl)
      else
        interp << syl
      end
      closedft << interp.dup
    end
  end
end
# Closedft will contain all completely parsed interpretations. Each of these
# must have the same morphword as the overt form.
closedft.each do |i|
  i.morphword = overt.morphword
end
# If a code block was given, run it on each interpretation given.
closedft.each {|interp| yield interp} if block_given?
return closedft
end
```

```
# Returns a list of the words whose outputs are structural interpretations
   # of the given overt form.
   def get_interpretations(overt,gram)
     output_list = parse_overt(overt,gram)
     list = []
     output_list.each do |output|
       list << parse_output(output,gram)
     end
     return list
   end


   # The constraint evaluation procedure declarations.
   #
   def lmost_eval() return @lmost_eval end
   def rmost_eval() return @rmost_eval end
   def afl_eval() return @afl_eval end
   def parsyl_eval() return @parsyl_eval end
   def ftbin_eval() return @ftbin_eval end
   def fnf_eval() return @fnf_eval end
   def iamb_eval() return @iamb_eval end
   def lapse_eval() return @lapse_eval end
   def maxstress_eval() return @maxstress_eval end
```

```
private



def initialize_eval_procs
 # Lmost
 @lmost_eval = lambda do |cand|
  viol_count = 0
  cand.output.each_element do |el|
   break if el.main_stress?
   viol_count += el.syllable_count
  end
  viol_count
 end
 # RMost
 @rmost_eval = lambda do |cand|
  viol_count = 0
  stress_found = false
  cand.output.each_element do |el|
   viol_count += el.syllable_count if stress_found
   stress_found = true if el.main_stress?
  end
  viol_count
 end
```

```
# AFL
@afl_eval = lambda do |cand|
  viol_count = 0
  cand.output.each_index do |ind|
    # For each foot, add up the number of syllables in the slice of output
    # to the left of the foot
    if ind > 0 and cand.output[ind].class == Foot then
      output_slice = cand.output.slice(0..ind-1)
      output_slice.each { |el| viol_count += el.syllable_count }
    end
  end
  viol_count
end
# ParSyl
@parsyl_eval = lambda do |cand|
  viol_count = 0
  cand.output.each_element do |el|
    viol_count += 1 if el.class != Foot
  end
  viol_count
end
# FtBin
@ftbin_eval = lambda do |cand|
```

```
  viol_count = 0
  cand.output.each_element do |el|
    viol_count += 1 if el.class == Foot and el.syllable_count ==1
  end
  viol_count
end
# FNF
@fnf_eval = lambda do |cand|
  viol_count = 0
  cand.output.each_element do |el|
    viol_count +=1 if el.class == Foot and el.last_syl.stressed?
  end
  viol_count
end
# Iamb
@iamb_eval = lambda do |cand|
  viol_count = 0
  cand.output.each_element do |el|
    viol_count += 1 if el.class == Foot and !el.last_syl.stressed?
  end
  viol_count
end
# Lapse
```

```
    @lapse_eval = lambda do |cand|
      viol_count = 0
      cand.output.syl_list.each_index do |ind|
        unless cand.output.syl_list[ind].equal?cand.output.syl_list[-1]
          viol_count += 1 if cand.output.syl_list[ind].unstressed? and
!cand.output.syl_list[ind+1].stressed?
        end
      end
      viol_count
    end
    # MaxStress
    @maxstress_eval = lambda do |cand|
      cand.io_corr.inject(0) do |sum, pair|
        if pair[0].stress_unset? then sum
        elsif pair[0].main_stress? & !pair[1].main_stress? then sum+1
        else sum
        end
      end
    end
  end
```

```
# Takes a partial output, along with
 # a reference to the next word element (syllable or foot) to be added
 # to the output. A copy of the new output, containing the new output syllable(s)
 #is returned.
 def extend_output(output, el)
   new_output = output.dup
   new_output << el.dup
   return new_output
 end
```

```
# Define the constraint list.
  # Each constraint has a label, a number, and a string defining the
  # violation evaluation procedure. Passing the eval string as an argument
  # to #eval will return a reference to a Proc, itself the actual violation
  # evaluation procedure. Calling that Proc with a candidate will
  # return the number of violations of that constraint in the candidate.
  def constraint_list
    list = []
    list << @lmost = Constraint_eval.new("Lmost", 1, MARK,
"SF::System.instance.lmost_eval")
    list << @rmost = Constraint_eval.new("RMost", 2, MARK,
"SF::System.instance.rmost_eval")
    list << @afl = Constraint_eval.new("AFL", 3, MARK,
"SF::System.instance.afl_eval")
    list << @parsyl = Constraint_eval.new("ParSyl", 4, MARK,
"SF::System.instance.parsyl_eval")
    list << @ftbin = Constraint_eval.new("FtBin", 5, MARK,
"SF::System.instance.ftbin_eval")
    list << @fnf = Constraint_eval.new("FNF", 6, MARK,
"SF::System.instance.fnf_eval")
    list << @iamb = Constraint_eval.new("Iamb", 7, MARK,
"SF::System.instance.iamb_eval")
```

```
      list << @lapse = Constraint_eval.new("Lapse", 8, MARK,
"SF::System.instance.lapse_eval")
      list << @maxstress = Constraint_eval.new("MaxStress", 9, FAITH,
"SF::System.instance.maxstress_eval")
      return list
    end

  end # class SF::System

  # The system object for the linguistic system SF (stress-feet).
  SYSTEM = System.instance

end # module SF
```

B-10 OVERT_LANGUAGE_LEARNING.RB

```
# Author: Crystal Akers
#

# For Generator
if RUBY_VERSION =~ /^1\.9/ then
  require 'generator19'  # Tesar's home-cooked ruby 1.9 version
else
  require 'generator'    # the ruby 1.8 version
end

require 'otlearn/contrast_pair'
require 'otlearn/ranking_learning'
require 'otlearn/grammar_test'
require 'otlearn/rcd_bias_low'
require 'otlearn/uf_learning'
require 'overt_otlearn/overt_grammar_test'
require 'overt_otlearn/label_set'
require 'overt_otlearn/language_hypothesis'
require 'facets/array/product' # Adds cartesian product to class Array.
require 'set'
require 'morph_word'
require 'otlearn/mrcd'
```

```
module Overt_OTLearn

 # A  OvertLanguageLearning object instantiates a particular instance of
 # language learning. An instance is created with a set of overt forms
 # (the data to be learned from), and a starting language hypothesis (which
 # will likely be altered and branched during the course of learning).
 #
 # The learning proceeds in the following stages, in order:
 # * Phonotactic learning.
 # * Single form learning (one word at a time until no more can be learned).
 # * Repeat until the language is learned or no more progress is made.
 #   * Try a contrast pair.
 #   * If no contrast pair succeeds, look for an error on a consistent,
 #     mismatch candidate.
 #   * If no consistent mismatch candidate is informative, t
 #     try minimum uf setting.
 #   * If any of these is successful, and the language is not yet learned,
 #     run another round of single form learning.
 # After each stage in which hypothesis change occurs, the state of
 # the learner is stored and evaluated in an OvertGrammarTest object. These
 # objects are stored in a list, obtainable via #results_list().
 #
 # Learning is initiated upon construction of the object.
 class OvertLanguageLearning

   attr_reader :labels, :letter, :lang_hyp_list, :discards, :results_list
```

```
# Executes learning on _overt_forms_ with respect to _language_hypothesis_, and
# stores the results in the returned OvertLanguageLearning object.
def initialize(overt_forms, language_hypothesis)
  # List of overt forms provided as data to the learner
  @overt_forms = overt_forms
  # List of consistent language hypotheses
  @lang_hyp_list = []
  @lang_hyp_list << language_hypothesis
  # List of discarded, inconsistent language hypotheses
  @discards = []
  # Stores the results for learning across all language hypotheses.
  @results_list = []
  # Stores the label hashes associated with the overt forms.
  @labels = Label_set.new
  # Stores the letter to be associated with the next new label hash
  @letter = "A"
  @learning_successful = execute_learning
end

# Returns the overt forms that were the data for learning.
def data_overt_forms() return @overt_forms end

# Returns the list of language hypotheses that are the result of learning.
def lang_hyp_list() return @lang_hyp_list end
```

```
    # Returns the list of grammar_test objects generated at various stages
    # of learning.
    def results_list()  @results_list end

    # Returns a boolean indicating if learning was successful.
    def learning_successful?() return @learning_successful end

    #Returns the list of discarded language hypotheses.
    def discards() @discards end

    # The main, top-level method for executing learning. This method is
    # protected, and called by the constructor #initialize, so learning
    # is automatically executed whenever an  OvertLanguageLearning object is
    # created.
    # Returns true if learning was successful, false otherwise.
    def execute_learning
      # Phonotactic learning
      puts "Phonotactic learning"
      phonotactic_learning(@overt_forms, @lang_hyp_list)
      @results_list << ["Phonotactic Learning - #{lang_sim_results(@lang_hyp_list)}",
learning_completed?]
      return true if learning_completed? == true
      # Single form UF learning
      puts "single form learning"
      run_single_forms_until_no_change(@lang_hyp_list)
```

```
    @results_list << ["Single Form Learning - #{lang_sim_results(@lang_hyp_list)}",
learning_completed?]
    return true if learning_completed? == true
    # Pursue further learning until the language is learned, or no
    # further improvement is made.
    puts "Further learning"
    further_learning()
    @results_list << ["Further Learning - #{lang_sim_results(@lang_hyp_list)}",
learning_completed?]
    return true if learning_completed? == true
    # Consider learning successful if at least one consistent hypothesis
    # has learned the language. This partial success will allow for
    # comparing the expected simulation outcome with the actual outcome, in
    # case more than one language hypothesis was expected to succeed.
    return true if @lang_hyp_list.any? {|lang_hyp| lang_hyp.results_list.last.all_correct?}
    # Return boolean indicating if learning was successful.
    # This should be false, because a "true" would have triggered an earlier
    # return from this method.
    fail if learning_completed? == true
    return learning_completed?
  end
```

```
# This method returns true if learning is complete; that is, if the
 # last grammar test for each language hypothesis is all correct.
 def learning_completed?()
   @lang_hyp_list.each do |lang_hyp|
     # Return false unless the last grammar test for the language
     # hypothesis is all correct.
     return false unless lang_hyp.results_list.last.all_correct?
   end
   return true
 end



 # Reports on the number of successful and discarded hypotheses, along with
 # their labels. Used in the Typ Summary Excel worksheet.
 def lang_sim_results(successful_hyps)
   results = String.new
   results += "#{successful_hyps.size} consistent hyps: "
   successful_hyps.each {|hyp| results += hyp.lang_hyp_label.to_s + " - " }
   results += " #{@discards.size} inconsistent hyps: "
   @discards.each {|hyp| results += hyp.lang_hyp_label.to_s + " - "}
   return results
 end
```

```
# Tests each overt form against each language hypothesis until all
 # language hypotheses cycle through all overt forms without making
 # any learning changes.
 def phonotactic_learning(overt_forms, lang_hyp_list)
  ranking_bias = nil   # FaithLow ranking bias
  l_hyp_list = lang_hyp_list

  while l_hyp_list.any? {|l_hyp| l_hyp.learning_change == true}
   # _changed_hyps_: hyps that changed in the last pass are re-tested
   # _@lang_hyp_list_: hyps that did not change in the last pass are stored
   changed_hyps, @lang_hyp_list = l_hyp_list.partition do
    |l_hyp| l_hyp.learning_change
   end
   overt_forms.each do |overt|
    l_hyp_list = changed_hyps
    changed_hyps = []
    until l_hyp_list.empty? do
     lang_hyp = l_hyp_list.shift
     # Add the complete list of overt forms to _lang_hyp_. The overt
     # forms are used by OvertGrammarTest to determine whether anything
     # can be learned from overt forms without committed outputs.
     @overt_forms.each { |o| lang_hyp.overt_forms << o } if lang_hyp.overt_forms.empty?
     # Reset _lang_hyp_ to unchanged during learning
     lang_hyp.hyp_change(false)
     input = OTLearn::input_from_overt(overt)
     competition = lang_hyp.system.gen(input)
```

```
        # Find the most harmonic candidates
        mh = MostHarmonic.new(competition, lang_hyp.grammar.hierarchy)
        commitment = lang_hyp.commitments.existing_commitment_pair(overt)

        if commitment then
          # If any optimum has an *output* distinct from the
          # committed output, perform ranking learning.
          if mh.any? {|cand| !lang_hyp.commitments.forms_match?(cand.output, commitment)}
then
            OTLearn::ranking_learning(lang_hyp.winner_list, lang_hyp, ranking_bias)
            lang_hyp.hyp_change(true)
          end
          # Add _lang_hyp_ to _changed_hyps_ list to be tested against
          # the next overt form if it's consistent; otherwise, discard.
          if lang_hyp.consistent? then
            changed_hyps << lang_hyp
          else
            @discards << lang_hyp
          end
        else # No existing commitment
          # If there is only one optimum and its overt form matches
          # _overt_form,add the language hypothesis to _changed_hyps_
          # to be tested against the next overt form.
          # Otherwise, extend new language hypothesis branches and
          # add the returned, consistent branches to _changed_hyps_
          # (inconsistent branches go directly to @discards).
```

```
            # Otherwise,
            # to be tested against the next overt form.
            if mh.size ==1 && mh[0].overt.to_s==overt.to_s then
              changed_hyps << lang_hyp
            else
              branch_list = extend_branches(lang_hyp, overt)
              branch_list.each {|branch| changed_hyps << branch}
              add_branch_info_to_sim_results(changed_hyps, l_hyp_list, lang_hyp, overt)
            end
          end

        end #until
      end #
      l_hyp_list << changed_hyps.shift until changed_hyps.empty?
    end #while
    # Phonotactic learning has ended.
    # For each lang hyp, store the number of ERCs created during phonotactic
    # learning. For the consistent lang hyps only, populate the lexicon with
    # all the morphemes before beginning single form learning.
    l_hyp_list.each do |l_hyp|
      l_hyp.store_phonotactic_erc_size(l_hyp.erc_list.size)
      overt_forms.each { |overt| add_morphemes_to_lexicon(l_hyp, overt) }
      l_hyp.results_list << Overt_OTLearn::OvertGrammarTest.new(l_hyp, "Phonotactic
Learning")
      @lang_hyp_list << l_hyp
    end
```

```
  @discards.each do |discard_hyp|
    # Store the number of ERCs created during phonotactic learning
    discard_hyp.store_phonotactic_erc_size(discard_hyp.erc_list.size)
  end
  return
end
```

```
# Calls the branch method for language hypotheses to extend
 # branches from _lang_hyp_ for _overt_ form. Updates branch labels
 # to indicate the overt form (by letter) and structural
 # interpretation (by number).
 # Example:
 #   A1B1 = This hypothesis has branched from A1 and committed to
 #          interpretation 1 for overt form B.
 def extend_branches(lang_hyp, overt)
   br_list, discards = lang_hyp.branch(overt, ranking_bias=nil)
   hyp_list = []
   br_list.each {|hyp| hyp_list << hyp}
   discards.each {|hyp| hyp_list << hyp}
   # Update label for each hyp
   hyp_list.each do |hyp|
     @letter = @labels.update_lang_hyp_label(overt, hyp, @letter)
   end
   # Add all inconsistent branches to _@discards_
   discards.each {|hyp| @discards << hyp}
   # Return the list of consistent branches
   return br_list
 end
```

```
# This method processes all of the overt forms in _overt_forms_, one
 # at a time in order, with respect to each language hypothesis
 # in _untested_hyps_. Each overt form is processed as follows:
 #   * Attempt to find new ranking info
 #     - If there's an existing commitment but no winner,
 #       create a new winner.
 #     - Check if the winner is optimal when unset input features
 #       are matched to the output, and if not, find more ranking info.
 #     - If there's no existing commitment, check if any optimum differs
 #       from _overt_form_ in string representation. Branch as required.
 #   * If a winner exists, attempt to set any of its unset underlying features.
 #   * For each newly set feature, check for new ranking information.
 # The method passes repeatedly through the list of overt forms until
 # a pass is made with no learning changes to the language hypothesis.
 # The language hypothesis's constraint hierarchy is updated with the
 # Faith-Low version of RCD. If the language hypothesis is consistent,
 # it is ultimately added back to @lang_hyp_list; otherwise, it is added
 # to @discards.
 def run_single_forms_until_no_change(untested_hyps)
   tested_hyps = []

   until untested_hyps.empty? do
     lang_hyp = untested_hyps.shift
     lang_hyp.hyp_change(true)
     skip_lang_hyp = false
```

```
while lang_hyp.learning_change == true do
 # Skip single-form learning if _lang_hyp_ is already complete
 if lang_hyp.results_list.last.all_correct? then
  tested_hyps << lang_hyp
  skip_lang_hyp = true
 end
 break if skip_lang_hyp
 lang_hyp.hyp_change(false)
 set_feature_list = []
 # Tests lang_hyp_ with each overt form. First checks for
 # errors, then tries to set unset features in the form.
 @overt_forms.each do |overt|
  break if skip_lang_hyp
  commitment = lang_hyp.commitments.existing_commitment_pair(overt)

  if commitment then
   # Check for a winner (full structural description, though the
   # input may include unset features) associated with the overt
   # form, and create a new winner if one does not already exist.
   winner = lang_hyp.existing_winner(overt)
   if winner.nil? then
    # Add a new winner to lang_hyp
    winner = lang_hyp.add_winner(overt, commitment)
   end
   # Check for new ranking information
```

```
            lang_hyp.hyp_change(true) if
OTLearn::ranking_learning_faith_low(lang_hyp.winner_list, lang_hyp)
            # If lang_hyp is consistent, attempt to set the unset features
            # _winner_. Otherwise, add lang_hyp to @discards.
            if lang_hyp.consistent? then
              new_set_features = OTLearn.set_uf_values([winner], lang_hyp)
              set_feature_list.concat(new_set_features)
              unless new_set_features.empty? then
                lang_hyp.hyp_change(true)
                set_f_string = String.new
                new_set_features.each do |f|
                  set_f_string << " - " <<  f.morpheme.to_s << " " << f.element.to_s
                end
                lang_hyp.results_list << Overt_OTLearn::OvertGrammarTest.new(lang_hyp, "Single
Form Learning: #{overt.morphword.to_s}. Set #{set_f_string}")
                # Add the number features just set to the lang hyp's
                # current number of set Fs
                lang_hyp.store_num_set_features(new_set_features.size)
              end
            else
              lang_hyp.results_list << Overt_OTLearn::OvertGrammarTest.new(lang_hyp, "Single
Form Learning")
              @discards << lang_hyp
              skip_lang_hyp = true
              break
            end
```

```
        else # No existing commitment
          input = OTLearn::input_from_lexicon_and_overt(overt, lang_hyp.grammar)
          competition = lang_hyp.system.gen(input)
          # Check for an error on _overt_.
          skip_lang_hyp = error_test_overt_form(overt, competition, lang_hyp, tested_hyps,
untested_hyps)
          break if skip_lang_hyp
          # Check for an error on the maximimal mismatch of _overt_.
          # An error indicates that the features have the potential
          # to be set; lang_hyp branches for interpretations of _overt_
          OTLearn::mismatches_input_to_overt(lang_hyp.grammar, overt) do |mismatched_input|
            competition = lang_hyp.system.gen(mismatched_input)
            skip_lang_hyp = error_test_overt_form(overt, competition, lang_hyp, tested_hyps,
untested_hyps)
            break if skip_lang_hyp
          end
        end # if commitment then

        # For each newly set feature, check winners in _lang_hyp_ that unfaithfully
        # map that feature for new ranking information.
        set_feature_list.each do |set_f|
          # Skips features set to -stress, as they do not violate MaxStress
          unless set_f.feature.unstressed? then
            if OTLearn::new_rank_info_from_feature(lang_hyp, lang_hyp.winner_list, set_f) then
```

```
            lang_hyp.results_list << Overt_OTLearn::OvertGrammarTest.new(lang_hyp, "New
ranking info from set feature (Single Form #{set_f.morpheme})")
            lang_hyp.hyp_change(true)
          end
        end
      end
      set_feature_list = []
    end # overt_forms.each
    break if skip_lang_hyp
  end # while

  # If there are no learning changes in _lang_hyp_, add it to _tested_hyps_
  # if it is still consistent; otherwise, discard it.
  unless skip_lang_hyp
    lang_hyp.update_grammar {|ercs| OTLearn::RcdFaithLow.new(ercs)}
    lang_hyp.results_list << Overt_OTLearn::OvertGrammarTest.new(lang_hyp, "End of
Single Form Learning")
    tested_hyps << lang_hyp if lang_hyp.consistent?
    @discards << lang_hyp unless lang_hyp.consistent?
  end
 end #until
 @lang_hyp_list << tested_hyps.shift until tested_hyps.empty?
end
```

```
# If there is a tie or any optimum has an *overt* form distinct
  # from _overt_, extend new language hypothesis branches.
  # Returns true if _lang_hyp_ branches; false otherwise.
  def error_test_overt_form(overt, competition, lang_hyp, tested_hyps, untested_hyps)
    mh = MostHarmonic.new(competition, lang_hyp.grammar.hierarchy)
    if mh.any? {|cand| cand.overt.to_s != overt.to_s} then
      branch_list = extend_branches(lang_hyp, overt)
      branch_list.each {|branch| @lang_hyp_list << branch}
      add_branch_info_to_sim_results(tested_hyps, untested_hyps, lang_hyp, overt)
      return true
    end
    return false
  end
```

```
# For any morphemes not currently in the lexicon, create new entries,
   # with the same number of syllables as in the output,
   # and all features unset.
   def add_morphemes_to_lexicon(lang_hyp, overt)
    mw = overt.morphword
    mw.each do |m|
     unless lang_hyp.grammar.lexicon.any?{|entry| entry.morpheme==m} then
      under = Underlying.new
      overt.each_syllable do |syl|
       under << SF::Syllable.new.set_morpheme(m)if syl.morpheme == m
      end
      lang_hyp.grammar.lexicon << Lexical_Entry.new(m,under)
     end
    end
   end

   # Adds to the simulation's results list an entry recording the
   # creation of a new branch.
   def add_branch_info_to_sim_results(hyp_list1, hyp_list2, lang_hyp, overt)
    simultaneous_hyps = []
    hyp_list1.each {|h| simultaneous_hyps << h}
    hyp_list2.each {|h| simultaneous_hyps << h}
    label = lang_hyp.lang_hyp_label
    @results_list << ["#{label.to_s} branches for #{overt.to_s}:
#{lang_sim_results(simultaneous_hyps)}"]
   end
```

```
    # Pursues further learning until each language hypothesis
    # has learned the language or until no further improvements
    # can be made in any language hypothesis.
    def further_learning()
      hyp_list = []
      until @lang_hyp_list.empty? do
        lang_hyp = @lang_hyp_list.shift
        # Only pursue further learning on incomplete language hypotheses
        if lang_hyp.results_list.last.all_correct? then
          hyp_list << lang_hyp if lang_hyp.consistent?
        else
          fw_list = []
          lang_hyp.results_list.last.failed_winners.each { |fw| fw_list << fw.morphword.to_s }
          learning_change = true
          while learning_change==true
            learning_change=false
            # First, try to learn from a contrast pair
            contrast_pair = run_contrast_pair(lang_hyp.winner_list, lang_hyp,
lang_hyp.results_list.last, strict=true)
            # These lines will enable a wider range of contrast pairs to
            # be considered. Useful for some languages in the 2r1s Stress system
            # typology.
            if contrast_pair.nil? then
              contrast_pair = run_contrast_pair(lang_hyp.winner_list, lang_hyp,
lang_hyp.results_list.last, strict=false)
            end
```

```
          unless contrast_pair.nil?
            cp0 = contrast_pair[0].morphword.to_s
            cp1 = contrast_pair[1].morphword.to_s
            lang_hyp.results_list << Overt_OTLearn::OvertGrammarTest.new(lang_hyp, "Contrast
Pair Learning - #{cp0}-#{cp1}")
            learning_change = true
          else
            # No suitable contrast pair, so pursure learning by minimal mismatch
            mismatch_winners = failed_winner_mismatch_consistency_check(lang_hyp)
            if mismatch_winners then
              until mismatch_winners.empty? do
                mismatch_winner = mismatch_winners.shift
                # Check for new ranking info from mismatch_winner.
                # Break for first error found.
                mrcd_result = OTLearn::MrcdFaithLow.new([mismatch_winner], lang_hyp)
                if mrcd_result.any_change? == true then
                  learning_change = true
                  lang_hyp.results_list << Overt_OTLearn::OvertGrammarTest.new(lang_hyp, "New
ranking info from failed winner mismatch")
                  break
                end
              end
            end
            # No ranking information from consistent mismatches, so pursue
            # minimal UF learning
```

```
            if learning_change == false
              set_feature = run_minimal_uf_for_failed_winner(lang_hyp.winner_list, lang_hyp,
lang_hyp.results_list.last)
              unless set_feature.nil?
                # Increment num_set_features to add this newly
                # set feature - used in Excel performance summary
                lang_hyp.store_num_set_features(1)
                lang_hyp.results_list << Overt_OTLearn::OvertGrammarTest.new(lang_hyp,
"Minimal UF Learning: #{set_feature.to_s}")
                learning_change = true
              end
            end
          end  #unless
          # If a learning change occured, check to see if the change
          # completed learning. If not,follow up with another round of
          # single form learning. (Any consistent language hypotheses
          # remaining after single form learning will be added to
          # _@lang_hyp_list_.
          # If no change resulted, no further learning is currently possible;
          # cease learning attempts on this language hypothesis.
            if learning_change == true then
              if lang_hyp.results_list.last.all_correct?
                hyp_list << lang_hyp
              else
               run_single_forms_until_no_change([lang_hyp])
              end
```

```
            break
          else
            hyp_list << lang_hyp if lang_hyp.consistent?
            @discards << lang_hyp unless lang_hyp.consistent?
          end
        end #while
      end #if
    end #until
    @lang_hyp_list << hyp_list.shift until hyp_list.empty?
    return
  end
```

```
# Returns the list of consistent minimal mismatch words
  def failed_winner_mismatch_consistency_check(main_lang_hyp)
    consistent_mismatch_list = []
    prior_results = main_lang_hyp.results_list.last
    return nil if prior_results.failed_winners.empty?
    prior_results.failed_winners.each do |fw_orig|
      # Check if there is a failed winner whose minimal mismatch is consistent
      # Dup hypothesis and words, so originals aren't modified.
      hyp = main_lang_hyp.dup
      fw = fw_orig.dup.sync_with_hypothesis!(main_lang_hyp)
      # Set fw's input so that features unset in the hypothesis lexicon
      # mismatch their output correspondents.
      mismatch_list = OTLearn::minimal_mismatches_input_to_output(fw)
      mismatch_list.each do |mismatch|
       # Run MRCD to see if the mismatched FW is consistent.
        mrcd_result = OTLearn::Mrcd.new([mismatch], hyp)
       # Add each consistent mismatched failed winner
        consistent_mismatch_list << mismatch if mrcd_result.hypothesis.consistent?
      end
    end
    return consistent_mismatch_list
  end
```

```
# Select a contrast pair, and process it, attempting to set underlying
 # features. If any features are set, check for any newly available
 # ranking information.
 #
 # This method returns the first contrast pair that was able to set
 # at least one underlying feature. If none of the constructed
 # contrast pairs is able to set any features, nil is returned.
 #
 # *Note* This method is adapted with few changes from Tesar's method
 # of the same name. The key difference is the "strict" parameter, which
 # determines whether to search for strict contrast pairs -- in which
 # the pair members have alternating values for unset features
 # (using Tesar's generate_contrast_pair method) -- or to search for all contrast
#  pairs, including those with non-alternating values of unset features.
 def run_contrast_pair(winner_list, hyp, prior_result, strict)
   # Create an external iterator which calls generate_contrast_pair()
   # to generate contrast pairs.
   cp_gen = Generator.new do |result|
     if strict then
       OTLearn::generate_contrast_pair(result, winner_list, hyp, prior_result)
     else
       generate_all_contrast_pairs(result, winner_list, hyp, prior_result)
     end
   end
   # Process contrast pairs until one is found that sets an underlying
   # feature, or until all contrast pairs have been processed.
```

```
    while cp_gen.next? do
      contrast_pair = cp_gen.next
      # Process the contrast pair, and return a list of any features
      # that were newly set during the processing.
      set_feature_list = OTLearn::set_uf_values(contrast_pair, hyp)
      # Increment num_set_features to add these newly set features
      hyp.store_num_set_features(set_feature_list.size)
      # For each newly set feature, see if any new ranking information
      # is now available.
      set_feature_list.each do |set_f|
        # Does not check for new rank info from features set to -stress,
        # as they do not violate MaxStress.
        unless set_f.feature.unstressed? then
          if OTLearn::new_rank_info_from_feature(hyp, winner_list, set_f) then
            hyp.results_list << Overt_OTLearn::OvertGrammarTest.new(hyp, "New rank info from
set feature (CP #{set_f.morpheme})")
          end
        end
      end
      # If an underlying feature was set, return the contrast pair.
      # Otherwise, keep processing contrast pairs.
      return contrast_pair unless set_feature_list.empty?
    end
    # No contrast pairs were able to set any features; return nil.
    return nil
  end
```

```
  # *Note* This method is adapted with few changes from Tesar's method
  # of the same name. The key difference is that this method generates all
  # contrast pairs, and not just those that have a conflicting value
  # for an unset feature in the lexicon.
def generate_all_contrast_pairs(cp_return, winners, hyp, test_result=nil)
  test_result ||= GrammarTest.new(winners, hyp)
  # The failed winners of the test are connected to a different
  # lexicon. Make duplicates of the failed winners, and synchronize
  # them with _hyp_.
  f_winners = test_result.failed_winners.map do |winner|
    winner.dup.sync_with_hypothesis!(hyp)
  end
  # For each failed winner, look for all contrast pairs
  f_winners.each do |failed_winner|
    OTLearn::match_input_to_uf!(failed_winner)
    failed_winner.morphword.each do |morph|
      if OTLearn::find_unset_features([morph], hyp)then
        all_containing_words = []
        c_words = OTLearn::find_morphemes_in_words(winners)[morph]
        c_words = c_words.delete_if do |cword|
        cword.morphword == failed_winner.morphword
        cword.output == failed_winner.output
        end
        c_words.each {|cw| all_containing_words << cw}
        all_containing_words.each do |word|
```

```
          cp = OTLearn::ContrastSet.new([failed_winner,word])
          cp_return.yield cp
        end
      end
    end
  end
end
```

```
# *Note* This method is adapted with few changes from Tesar's method
# of the same name.
#
# Given the result of error-testing, find a previously unset feature
# for one of the failed winners such that setting it to match its
# surface correspondent in the failed winner results in the winner
# succeeding (consistent with all of the winners that passed
# error-testing). This method is expected to be invoked only when
# single-word and contrast-pair inconsistency detection has failed
# to completely learn the language, suggesting that a paradigmatic
# subset relation is present. The goal is to find the smallest set
# of feature values that will allow learning to continue (fewer set
# features corresponds to greater restrictiveness).
# Each failed winner is checked in turn until one is found that can
# succeed on the basis of one newly set feature, returning that instance
# without checking to see if there are other possibilities.
#
# Returns the feature instance of the newly set feature, or nil if no feature was set.
#
# At present, #select_most_restrictive_uf checks each unset feature of
# a failed winner in isolation, and returns a feature value allowing
# that winner to succeed if there is exactly one.
# In principle, if there is no single feature leading to success for
# a previously failed winner, this method should try combinations
# of two unset features (and larger, if necessary) to find the minimum
# set of additional feature value commitments resulting in the success
```

```
    # of a failed winner. Future work will be needed to determine if
    # the learner should evaluate each failed winner, and then select
    # the failed winner requiring the minimal number of set features.
    def run_minimal_uf_for_failed_winner(winner_list, hyp, prior_result)
      fw_list = prior_result.failed_winners
      set_feature = nil
      fw_list.each do |failed_winner|
        # Get the FeatureValuePair of the feature and its succeeding value.
        fv_pair = select_most_restrictive_uf(failed_winner, hyp, prior_result)
        unless fv_pair.nil?
          fv_pair.set_to_alt_value  # Set the feature permanently in the lexicon.
          set_feature = fv_pair.feature_instance
          # Check for any new ranking information based on the newly set feature.
          # Does not check for new rank info from features set to -stress,
          # as they will not incur MaxStress violations
          unless set_feature.feature.unstressed? then
            if OTLearn::new_rank_info_from_feature(hyp, winner_list, set_feature) then
              hyp.results_list << Overt_OTLearn::OvertGrammarTest.new(hyp, "New ranking info
from set feature (Min UF #{set_feature.morpheme})")
            end
          end
          break # Stop looking once the first successful feature is found.
        end
      end
      return set_feature
    end
```

```
    # *Note* This method is adapted with few changes from Tesar's method
    # of the same name.
    #
    #
    # Finds the unset underlying form feature of _failed_winner_ that,
    # when assigned a value matching its output correspondent,
    # makes _failed_winner_ consistent with the success winners. Consistency
    # is evaluated with respect to the parameter _main_hypothesis_ with its
    # lexicon augmented to include the tested underlying feature value, and with
    # the other unset features given input values opposite of their output values).
    #
    # Returns nil if none of the features succeeds.
    # If more than one underlying feature succeeds, returns the first found.
    # Returns the successful underlying feature (and value) if exactly one of them succeeds.
    # The return value is a _FeatureValuePair_: the underlying feature instance and
    # its successful value (the one matching its output correspondent in the
    # previously failed winner).
    def select_most_restrictive_uf(failed_winner_orig, main_hypothesis, prior_result)
      failed_winner = failed_winner_orig.dup.sync_with_hypothesis!(main_hypothesis)
      # Find the unset underlying feature instances
      unset_uf_features =
OTLearn::find_unset_features_in_words([failed_winner],main_hypothesis)
      # Set, in turn, each unset feature to match its output correspondent.
      # For each case, test the success winners and the current failed winner
      # for collective consistency with the hypothesis.
```

```
  consistent_feature_val_list = []
  unset_uf_features.each do |ufeat|
    # set the tested underlying feature to the output value
    out_feat_inst = failed_winner.out_feat_corr_of_uf(ufeat)
    ufeat.value = out_feat_inst.value
    # Add the failed winner to (a dup of) the list of success winners.
    word_list = prior_result.success_winners.dup
    word_list << failed_winner
    # Check the list of words for consistency, using the main hypothesis,
    # with each word's unset features mismatching their output correspondents.
    mrcd_result = mismatch_consistency_check(main_hypothesis, word_list)
    # If result is consistent, add the UF value to the list.
    if mrcd_result.hypothesis.consistent? then
      ufeat_val_pair = FeatureValuePair.new(ufeat, ufeat.value)
      consistent_feature_val_list << ufeat_val_pair
    end
      # Unset the tested feature in any event.
    ufeat.value = nil
  end
  # Return the first consistent tested feature found.
  return nil if consistent_feature_val_list.empty?
  return consistent_feature_val_list[0]
end
```

```
# Given a list of words and a hypothesis, check the word list for
    # consistency with the hypothesis using MRCD. Any features unset
    # in the lexicon of the hypothesis are set in the input of a word
    # to the value opposite its output correspondent in the word.
    # The mismatching is done separately for each word (the same unset feature
    # for a morpheme might be assigned different values in the inputs of
    # different words containing that morpheme, depending on what the outputs
    # of those words are).
    # Returns the Mrcd object containing the results.
    # To find out if the word list is consistent with the hypothesis, call
    # result.hypothesis.consistent? (where result is the Mrcd object returned
    # by #mismatch_consistency_check).
    def mismatch_consistency_check(hypothesis, word_list)
      # Dup hypothesis and words, so originals aren't modified.
      hyp = hypothesis.dup
      w_list = word_list.map { |winner| winner.dup.sync_with_hypothesis!(hyp) }
      # Set each word's input so that features unset in the hypothesis lexicon
      # mismatch their output correspondents. A given output could appear
      # more than once in the mismatch list ONLY if there are suprabinary
      # features (a suprabinary feature can mismatch in more than one way).
      mismatch_list = []
      w_list.map do |word|
        OTLearn::mismatches_input_to_output(word) { |mismatched_word| mismatch_list <<
mismatched_word }
      end
      # Run MRCD to see if the mismatched candidates are consistent.
```

```
      return OTLearn::Mrcd.new(mismatch_list, hyp)
    end

    protected :execute_learning, :run_single_forms_until_no_change,
      :run_contrast_pair, :run_minimal_uf_for_failed_winner,
      :select_most_restrictive_uf

  end # class OvertLanguageLearning

end # module Overt_OTLearn
```

B-11 OVERT_GRAMMAR_TEST.RB

```
# Author: Crystal  Akers
#


require 'otlearn/grammar_test'
require 'otlearn/uf_learning'
require 'otlearn/data_manip'
require 'overt_otlearn/commitment_list'
require 'sf/sf_output'
require 'morph_word'


module Overt_OTLearn


  # An OvertGrammarTest object holds the results of the evaluation of a set
  # of winners with respect to a hypothesis. The tests are initiated by
  # creating an OvertGrammarTest; the constructor takes a list of winners and
  # a hypothesis as parameters.
  #
  # Each winner is a Word, possibly with unset features in the input.
  class OvertGrammarTest < OTLearn::GrammarTest
```

```
 # Returns a new OvertGrammarTest, for the provided _winners_, and with
   # respect to the provided _lang_hyp_.
   def initialize(lang_hyp, label="NoLabel")
    super(lang_hyp.winner_list, lang_hyp, label)
    # Dup the commitments
    @commitments = lang_hyp.commitments.dup
#     # Dup the set of overt forms
    @overt_forms = lang_hyp.overt_forms.dup
    # Freeze the test results, so they cannot be accidentally altered later.
#     @l_hyp.freeze
    @commitments.each {|c_pair| c_pair.freeze}
    @commitments.freeze
    @overt_forms.each {|overt| overt.freeze}
    @overt_forms.freeze
   end
```

```
# Returns true if
   #   - all winners in the winner list are the sole optima for inputs with all
   #     unset features set to mismatch the surface of the winner.
   #   - no new ranking or lexical information can be learned from the remaining
   #     overt forms without committed outputs
   def all_correct?
     return false unless @failed_winner_info_list.empty?
     return false unless check_overt_forms_for_ranking_and_lexical_info(uncommitted_overts)
     return true
   end

   # Returns a list of overt forms which do not have committed output interpretations
   def uncommitted_overts
     uncommitted_forms = Array.new
     @overt_forms.each do |overt|
       uncommitted_forms << overt unless @commitments.any? {|c_pair|
@commitments.forms_match?(overt, c_pair)}
     end
     return uncommitted_forms
   end
```

```
def check_overt_forms_for_ranking_and_lexical_info(overt_forms)
    overt_forms.each do |overt_form|
      input = OTLearn::input_from_lexicon_and_overt(overt_form, @hypothesis.grammar)
      competition = @hypothesis.system.gen(input)
      # Find the most harmonic candidates
      mh = MostHarmonic.new(competition, @hypothesis.grammar.hierarchy)
      # Return if new ranking info is available (if there is a CTie or if
      # the optimum does not have the same overt form)
      return false if mh.size > 1
      return false if mh.any? {|cand| cand.overt.to_s != overt_form.to_s}
      ranking_info_test_optimum = mh[0].to_s
       # Check that no new lexical information is available
      OTLearn::mismatches_input_to_overt(@hypothesis.grammar, overt_form) do
|mismatched_input|
        competition2 = @hypothesis.system.gen(mismatched_input)
        mh2 = MostHarmonic.new(competition2, @hypothesis.grammar.hierarchy)
        # Return false if there is more than one optimum, if the optimum has
        # an *overt* form distinct from _overt_, or if the optimum differs
        # from the ranking test optimum.
        return false if mh2.size > 1
        return false if mh2.any? {|cand| cand.overt.to_s != overt_form.to_s}
        return false unless mh2[0].to_s == ranking_info_test_optimum
      end
    end
    return true
  end
```

```
  end # class OvertGrammarTest


end # module Overt_OTLearn
```

B-12 DATA_MANIP.RB

```
# Author: Crystal Akers
#
# This file contains a collection of methods for generating and
# manipulating data.

require 'hypothesis'
require 'otlearn'
require 'morph_word'
require 'input'
require 'sf/syllable'
require 'io_correspondence'


module OTLearn
```

```
# Creates and returns an input given an overt form, useful in
# identity maps. This input has an empty UI correspondence.
def OTLearn::input_from_overt(overt_form)
  input =  Input.new
  overt_form.each do |syl|
    in_syl = SF::Syllable.new
    in_syl.set_morpheme(syl.morpheme)
    syl.each_feature do |f|
      val = f.value
      in_syl.set_feature(f.type,val)
    end
    input.push(in_syl)
    input.morphword = overt_form.morphword
  end
  return input
end


# Performs ranking learning using the given ranking bias.
def OTLearn::ranking_learning(winner_list, lang_hyp, ranking_bias_flag)
  if ranking_bias_flag == nil then
    OTLearn::ranking_learning_faith_low(winner_list,lang_hyp)
  else
    OTLearn::ranking_learning_mark_low(winner_list,lang_hyp)
  end
end
```

```
# Creates and returns an input. The input contains all features set
# in _gram_; any features unset in the lexicon are set in the input
# to match those in the _overt_ form. This input has an empty
# UI correspondence.
def OTLearn::input_from_lexicon_and_overt(overt, gram)
 input = Input.new
 input.morphword = overt.morphword
 mw = input.morphword
 mw.each do |m| # for each morpheme in the morph_word, in order
  uf = gram.get_uf(m)
  # If the morpheme is in the lexicon, add a duplicate of each
  # underlying syllable to input. Otherwise, for each syllable
  # of the morpheme, add a new syllable to the input.
  if uf then
   uf.each { |syl| input.push(syl.dup) }
  else
   m_syls = overt.find_all {|syl| syl.morpheme == m}
   m_syls.each { |syl| input.push(SF::Syllable.new.set_morpheme(m)) }
  end
 end
 # Match any unset features of the input syllables to the values
 # of the corresponding _overt_ syllables.
 gen_syl = SyncEnumerator.new(input, overt)
 gen_syl.each do |in_syl, o_syl|
  in_syl.each_feature do |f|
   if f.unset? then
```

```
        o_feat = o_syl.get_feature(f.type)
        in_syl.set_feature(f.type, o_feat.value)
      end
    end
  end
  return input
end
```

```
# For the given overt form, test the *unset* features by examining
# each combination of values such that each unset feature does *not*
# match its output correspondent.
# For each combination, the code block is run.
#
# If all features are strictly binary, then there is only one input that
# maximally mismatches the output with respect to the unset features.
# If one or more features is suprabinary, then the
# different possible combinations of non-surface-matching values are
# all tried.
def OTLearn::mismatches_input_to_overt(gram, overt_form, &block)
  input = Input.new
  input.morphword = overt_form.morphword
  mw = input.morphword
  mw.each do |m| # for each morpheme in the morph_word, in order
    uf = gram.get_uf(m)
    # If the morpheme is in the lexicon, add a duplicate of each underlying
    # syllable to input. Otherwise, for each syllable of the morpheme, add a
    # new syllable to the input.
    if uf then
      uf.each { |syl| input.push(syl.dup) }
    else
      m_syls = overt_form.find_all {|syl| syl.morpheme == m}
      m_syls.each { |syl| input.push(SF::Syllable.new.set_morpheme(m)) }
    end
  end
```

```
  # Create an IO correspondence between the input and the overt form.
  io_corr = IOCorrespondence.new
  gen = SyncEnumerator.new(input, overt_form)
  gen.each do |in_syl,overt_syl|
    io_corr << [in_syl,overt_syl]
    if in_syl.morpheme != overt_syl.morpheme then
      raise "Input syllable morph #{in_syl.morpheme.label} != " +
        "overt syllable morph #{overt_syl.morpheme.label}"
    end
  end
  # Construct a list of the unset features in _input_
  unset_features = []
  input.each do |in_el|
    in_el.each_feature do |f|
      unset_features << FeatureInstance.new(in_el,f) if f.unset?
    end
  end
  # Invoke the block on each combination of mismatched values, by
  # passing the block as a procedure object.
  OTLearn::test_each_overt_mismatch_value(input, io_corr, unset_features, block)
end
```

```
# Used for creating minimal mismatch candidates for failed winners.
# For the given word, create each minimal mismatch candidate by setting one
#  *unset* feature at a time to a value that does *not* match its output
# correspondent.
#
# If all features are strictly binary, then there is only one input that
# maximally mismatches the output with respect to the unset features.
# If one or more features is suprabinary, then the
# different possible combinations of non-surface-matching values are
# all tried.
def OTLearn::minimal_mismatches_input_to_output(word_param)
  word = word_param.dup
  OTLearn::match_input_to_uf!(word)
  # Construct a list of the unset features in the word
  unset_features = []
  input = word.input
  input.each do |in_el|
    in_el.each_feature do |f|
      unset_features << FeatureInstance.new(in_el,f) if f.unset?
    end
  end
  mismatch_words = []
  unset_features.each do |unset_f_inst|
    # Obtain the value of the unset feature's corresponding instance
    # in the output.
    out_f_inst = word.out_feat_corr_of_in(unset_f_inst)
```

```
    out_f_val = out_f_inst.value
    # Obtain the unset feature itself.
    unset_f = unset_f_inst.feature
    # For each value of the unset feature type that does not match the
    # value in the output correspondent, assign that value to the
    # unset feature *in the input* (i.e., not in the lexicon).
    unset_f.each_value do |val|
      if val!=out_f_val then
        OTLearn::match_input_to_output!(word)
        # Set the value of _unset_f_ to the value that mismatches the output
        unset_f.value = val
        mismatch_words << word.dup
        #reset the value of this unset feature for the next test
        unset_f.value = out_f_inst.value
      end
    end
  end
  return mismatch_words
end
```

```
# Run the provided procedure object _block_proc_ on variations of _input_.
  # The variations are all possible combinations of values for the input
  # features in _unset_features_ such that all of those input features do
  # not match their output correspondent values in a previously given overt form.
  #
  # _block_proc_ is a procedure object version of the code block to be
  # called on each combination of output-mismatched feature values.
  def OTLearn::test_each_overt_mismatch_value(input, io_corr, unset_features,
block_proc)
    # Base case: if no unset features remain, call the block on a duplicate
    # of the input, and return.
    if unset_features.empty? then
      block_proc.call(input.dup)
      return
    end
    # Get the first unset feature instance on the list, and make a copy list of
    # the rest of the unset features (that way, the original list is unchanged
    # when referenced by other recursive calls).
    unset_f_inst = unset_features[0]
    rest_unset_features = unset_features.slice(1..-1) # list with first element removed
    # Obtain the value of the unset feature's corresponding instance in the overt form.
    overt_f_inst = OTLearn::overt_feat_corr_of_in(unset_f_inst, io_corr)
    overt_f_val = overt_f_inst.value
    # Obtain the unset feature itself.
    unset_f = unset_f_inst.feature
    # For each value of the unset feature type that does not match the
```

```
    # value in the overt correspondent, assign that value to the
    # unset feature *in the input* (i.e., not in the lexicon).
    unset_f.each_value do |val|
      if val!=overt_f_val then
        unset_f.value = val
        OTLearn::test_each_overt_mismatch_value(input,io_corr, rest_unset_features, block_proc)
      end
    end
end
```

```
# Returns the corresponding overt feature instance for the given _in_feat_inst_.
# This method assumes that the corresponding overt feature instance of
# _in_feat_inst_ simply the corresponding output feature instance.
def OTLearn::overt_feat_corr_of_in(in_feat_inst, io_corr)
   # Get the corresponding overt element and feature for the input element.
   overt_corr_element = io_corr.out_corr(in_feat_inst.element)
   return nil if overt_corr_element.nil?
   overt_corr_feat = overt_corr_element.get_feature(in_feat_inst.feature.type)
   return FeatureInstance.new(overt_corr_element, overt_corr_feat)
end



end # module OTLearn
```

B-13 EXCEL_FOR_OVERT_OTLEARN.RB

```
# Author: Crystal Akers, based on Tesar's Excel_for_OTLearn.rb
#
# This file contains revisions to the Excel interface for RUBOT for use in
# learning multiple simulataneous language hypotheses.

require 'otlearn'
require 'overt_otlearn/language_hypothesis'
require 'overt_otlearn/overt_language_learning'
require 'excel'

module Overt_OTLearn

# A session for interacting with Excel, specifically for the
# overt_otlearn simulations.
```

```
 class Excel_session_for_overt_otlearn < Excel_session
    # Excel Constants
    # Load built-in Excel constants (Xl*); for a list, see
    # http://msdn.microsoft.com/en-us/library/aa221100(office.11).aspx
    # The constants of the variety Xl* can be obtained from WIN32OLE.
    # const_load(<excel object>, Excel_session) obtains the
    # defined Excel constaints and makes them constants of the class Excel_session.
    # That way, they can be used in the methods of this class, which makes it
    # much easier to use the MSDN Reference Library for the Excel VBA calls.
    # Reference: http://msdn.microsoft.com/en-us/library/aa220733(office.11).aspx
    begin
      WIN32OLE.const_load(WIN32OLE.connect("excel.application"),
Excel_session_for_overt_otlearn) # Load excel constants
    rescue WIN32OLERuntimeError # error exception thrown if Excel isn't running.
      # Create a temporary excel app, so that constants can be loaded from it.
      excel_temp = WIN32OLE.new("excel.application")
      WIN32OLE.const_load(excel_temp, Excel_session_for_overt_otlearn) # Load excel
constants
      excel_temp.ole_free # Terminate the temporary excel app
    end
```

```
# The color index constants aren't defined as colors in Excel, because they
    # actually point to positions in the default color palette, which can be
    # dynamically changed by the user. The color assignments given below
    # as constants reflect the Excel defaults: the colors that by default are
    # in the numbered positions in the color palette.
    # Color index constant usage: obj.colorindex = CONST
    RED = 3 #:nodoc:
    BRIGHTGREEN = 4 #:nodoc:
    BLUE = 5 #:nodoc:
    PALEYELLOW = 19 #:nodoc:
    LIGHTGREEN = 35 #:nodoc:
    MIDYELLOW = 36 #:nodoc:
    MAGENTA = 38 #:nodoc:

    def initialize
      super
    end
```

```
def put_learning_results(hyp, success)
  @excel.screenUpdating=false # turn off updating while writing to a worksheet
  ws = @excel.sheets.add('After'=>@excel.activesheet)
  if success == true then
   if hyp.results_list.last.all_correct? then
     ws.name = name_sheet(hyp.lang_hyp_label)
   else
     ws.name = name_sheet("(" + "*" + hyp.lang_hyp_label + ")")
   end
  else
   ws.name = name_sheet("(" + hyp.lang_hyp_label+ ")")
  end
  range(ws,1,1,1,1).value = hyp.label
  row = 1; col = 1
  #
  row +=2
  range(ws,2,1,2,3).merge
  range(ws,2,1,2,1).value = "Commitments"
  hyp.commitments.each  do |c_pair|
   row += 1
   range(ws, row, 1, row, 1). value = "#{c_pair[0].to_s}"
   range(ws, row, 2, row, 2).value = "#{c_pair[1].to_s}"
  end
  row+= 2
  range(ws,row, 1, row, 10).merge
  hyp_dup = hyp.dup
```

```
  rcd_result = hyp_dup.update_grammar{|ercs| OTLearn::RcdFaithLow.new(ercs)}
  range(ws, row, 1, row, 1). value = "#{hyp_dup.grammar.hierarchy.to_s}"
  hyp.results_list.each do |entry|
    row += 3
    range(ws,row,1,row,14).merge
    range(ws,row,1,row,1).value = "#{entry.label}:"
    row, col = learning_result_to_ws(ws, entry, row+1, 1)
  end
  #
ensure # make sure screen updating is turned back on, even if an exception is raised.
  @excel.screenUpdating=true
end
```

```
 # Modified from the method in the superclass Excel.rb file to include
    # a column in the comparative tableau output that includes, for each ERC,
    # the order in which that ERC was added to the CT.
    def learning_result_to_ws(ws, gram_test_result, row_first, col_first)
      # Test result components are frozen, so dup before updating.
      hyp = gram_test_result.hypothesis.dup
      rcd_result = hyp.update_grammar{|ercs| OTLearn::RcdFaithLow.new(ercs)}
      # Add the unranked constraints as a "final stratum" to the hierarchy.
      hier_with_unranked = Hierarchy.new
      hier_with_unranked.concat(rcd_result.hierarchy)
      hier_with_unranked << rcd_result.unranked unless rcd_result.unranked.empty?
      sorted_cons = hier_with_unranked.flatten
      # sort the ercs with respect to the RCD constraint hierarchy
      sorted_ercs, ercs_by_stratum, explained_ercs = sort_rcd_results(rcd_result)
      # write the main CT to the new worksheet
      wl_pairs_to_ws(ws,gram_test_result.hypothesis.erc_list,sorted_cons,
sorted_ercs,row_first,col_first)
      # Set some table index values
      pre_con_columns = 5
      col_count = pre_con_columns + sorted_cons.size
      pre_erc_rows = 1
      last_ex_row = pre_erc_rows + explained_ercs.size # row of last explained erc
      row_count = last_ex_row + rcd_result.unex_ercs.size

      # put vertical lines between the strata
      vl_col_count = pre_con_columns
```

```
    hier_with_unranked.each do |stratum|
      vl_col_count += stratum.size
      vl_row_last = row_first-1+row_count
      vl_col_last = col_first-1+vl_col_count
      range(ws,row_first,vl_col_last,vl_row_last,vl_col_last).borders(XlEdgeRight).weight =
        XlMedium # vertical line between strata
    end
    # put horizontal lines between the "stratified" clusters of ercs
    hl_row_count = pre_erc_rows
    ercs_by_stratum.each do |ercs|
      hl_row_count += ercs.size
      hl_row_last = row_first-1+hl_row_count
      hl_col_last = col_first-1+vl_col_count
      range(ws,hl_row_last,col_first,hl_row_last,hl_col_last).borders(XlEdgeBottom).weight =
        XlMedium
    end
    # Flag any unexplained ercs with color
    range(ws,row_first+last_ex_row,1,row_first-1+row_count,1).interior.colorindex = RED
unless row_count == last_ex_row
    # Extra formatting if the data were inconsistent
    range(ws,row_first+row_count,1,row_first+row_count,1).value = "FAIL!" unless
rcd_result.consistent?

    # Put the lexicon to the worksheet
    first_lex_row = row_first-1 + row_count + 2
    lex = hyp.grammar.lexicon
```

```
      r_row = first_lex_row-1
      r_row, r_col = morphs_to_ws(ws, lex.get_prefixes, r_row+1, col_first) unless
lex.get_prefixes.empty?
      r_row, r_col = morphs_to_ws(ws, lex.get_roots, r_row+1, col_first)
      r_row, r_col = morphs_to_ws(ws, lex.get_suffixes, r_row+1, col_first) unless
lex.get_suffixes.empty?


      test_row = r_row + 2
      range(ws,test_row,col_first,test_row,col_first+2).merge
      range(ws,test_row,col_first,test_row,col_first).value = "Learned:
#{gram_test_result.all_correct?}"
      return test_row, col_first-1+col_count
    end
```

```
# Format a comparative tableau, and write it to the specified section of the worksheet.

  def wl_pairs_to_ws(ws, unsorted_ct, sorted_cons, sorted_ercs, row_first, col_first)
    # first row contains the column headers
    sheet_image = [] << (row_image = [])
    row_image.concat(["#", "ERC\#", "Input", "Winner", "Loser"])
    pre_con_columns = 5
    row_image.concat(sorted_cons.map{|con| con.to_s})
    # add the erc rows to the sheet image
    sorted_ercs.each do |erc|
      sheet_image << (row_image = []) # elements of the eventual output row
      row_image << unsorted_ct.get_erc_index(erc).to_s
      row_image << erc.label
      if erc.respond_to?(:winner) then # pair contains a winner and a loser
        row_image << erc.winner.input.to_s << erc.winner.merged_outputs_to_s
        row_image <<  erc.loser.output.to_s
      else
        3.times{row_image << nil} # base ercs don't have winner or loser
      end
      add_prefs_to_row(erc, sorted_cons, row_image)
    end
    # write the sheet_image array to the worksheet
    row_last = row_first + sheet_image.size - 1
    col_last = col_first + pre_con_columns + sorted_cons.size - 1
    range(ws,row_first,col_first,row_last,col_last).value = sheet_image
    ct_borders(ws,row_first,col_first,row_last,col_last,pre_con_columns)
```

```
  return row_last, col_last
end
```

```
def put_learning_results_of_sim(lang_sim)
    @excel.screenUpdating=false # turn off updating while writing to a worksheet
    ws = @excel.activesheet
    ws.name = name_sheet("Sim Results")
    row = 1; col = 1
    range(ws,1,1,1,1).value = "Results of Language Learning Simulation"
    lang_sim.results_list.each do |entry|
      row += 2
      range(ws,row,1,row,8).merge
      range(ws, row, 1, row, 1).value = entry
    end
    #
  ensure # make sure screen updating is turned back on, even if an exception is raised.
    @excel.screenUpdating=true
  end
```

```
 def put_typ_results_to_ws(overt_forms_list, overt_forms_set_label, learned_lgs,
failed_consis_lgs, discards, row_first)
    ws = @excel.activesheet
    # first row contains a header for overt forms set
    sheet_image = [] << (row_image = [])
    row_image << overt_forms_set_label
    until overt_forms_list.empty? do
      sheet_image << (row_image = [])
      4.times do
        row_image << overt_forms_list.shift
      end
    end
    # second row contains the column headers
    sheet_image << (row_image = []) # elements of the eventual output row
    row_image.concat(["Learned Lgs", "Fail - Consis.", "Fail - Inconsis."])
    # Create the entries for the Learned Lgs column
    learned_col = []
    learned_lgs.each do |hyp|
      summary = String.new
      summary << hyp.lang_hyp_label << "; " << hyp.erc_list.size.to_s
      hyp.commitments.each  do |c_pair|
        summary << " " << c_pair[1].to_s
      end
      learned_col << summary
    end
    # Create entries for the Fail-Consist. lgs column
```

```
failed_consis_col = []
failed_consis_lgs.each do |hyp|
  summary = String.new
  summary << hyp.lang_hyp_label << "; " << hyp.erc_list.size.to_s
  hyp.commitments.each  do |c_pair|
    summary << " " << c_pair[1].to_s
  end
  failed_consis_col << summary
end
# Create entries for the Discard column
discard_col = []
discards.each do |hyp|
  summary = String.new
  summary << hyp.lang_hyp_label << "; " << hyp.erc_list.size.to_s
  hyp.commitments.each  do |c_pair|
    summary << " " << c_pair[1].to_s
  end
  discard_col << summary
end
# Create row images.
max_rows = learned_lgs.size
if failed_consis_lgs.size > max_rows then
  max_rows = failed_consis_lgs.size
elsif discards.size > max_rows then
  max_rows = discards.size
end
```

```
  curr_row = 0
  until curr_row == max_rows do
    sheet_image << (row_image = []) # elements of the eventual output row
    col1 = learned_col[curr_row]
    col2 = failed_consis_col[curr_row]
    col3 = discard_col[curr_row]
    row_image << col1 << col2 << col3
    curr_row +=1
  end
  # write the sheet_image array to the worksheet
  row_last = row_first + sheet_image.size - 1
  range(ws,row_first,1,row_last,4).value = sheet_image
  # autosize the columns
  range(ws,row_first,1,row_last,4).entirecolumn.autofit
  #
  return row_last +=2
  #
end
```

```
def put_perf_results_to_ws(overt_forms_set_label, learned_lgs, discards, row_first)
    ws = @excel.activesheet
    # first row contains a header for overt forms set
    sheet_image = [] << (row_image = [])
    row_image.concat(["Set #", "LgHyp", "Consistent?", "Phonotactic ERCS", "Total ERCs",
"Set Fs" ])
    # Create the entries for the Learned Lgs
    learned_lgs.each do |hyp|
      sheet_image << (row_image = [])
      row_image.concat([overt_forms_set_label, hyp.lang_hyp_label, "Yes" ,
hyp.num_phonotactic_ercs, hyp.erc_list.size, hyp.num_set_features])
    end
    # Create the entries for the inconsistent languages
    discards.each do |hyp|
      sheet_image << (row_image = [])
      row_image.concat([overt_forms_set_label, hyp.lang_hyp_label, "No" ,
hyp.num_phonotactic_ercs, hyp.erc_list.size, hyp.num_set_features])
    end
    # write the sheet_image array to the worksheet
    row_last = row_first + sheet_image.size - 1
    range(ws,row_first,1,row_last,6).value = sheet_image
    # autosize the columns
    range(ws,row_first,1,row_last,6).entirecolumn.autofit
    #
    return row_last +=2
    end
```

```
    # Saves the active workbook as the given filename, then closes that workbook. Leaves
    # a new output workbook open in the current Excel session.
    def close(filename)
      wb = @excel.activeworkbook
      wb.saveas(filename)
      wb.close
      add_output_workbook('New')
    end


  end #class Excel_for_overt_otlearn
end #module Overt_OTLearn
```

B-14 LABEL_SET.RB

```
# Author: Crystal Akers

require 'sf/sf_word'
require 'sf/system'
require 'set'

module Overt_OTLearn
  class Label_set < Set

    # Create a new, blank label set. The label set is composed of individual label hashes.
    def initialize
      super
    end
```

```
# Creates a new label and adds it to the label set. Each label consists of a hash.
 # The first hash key is the string representation of _overt_form_ and is given the
 # value of _letter_. The following keys are for structural interpretations,
 # with one key for each string representation of a structural interpretation of
 #  _overt_form_, and number values for each key. Returns the label.
 # Label:  [overt_form => letter, output1 => 1, output2 => 2, ...]
 def create_new_label_hash(overt_form, lang_hyp, letter)
  label= Hash.new
  overt = overt_form.dup
  # Add the string rep. of the overt form and letter value to the hash
  label[overt.to_s] = letter.dup
  # Create keys and values for string reps. of structural interpretations
  num = "0"
  interpretations = lang_hyp.system.get_interpretations(overt_form, lang_hyp.grammar)
  interpretations.each do |word|
   output = word.output.dup
   num = num.succ
   label[output.to_s] = num
  end
  # Add the new label to the label set
  self << label
  return label
 end
```

```
# Updates the label of _lang_hyp_ with the label associated with _overt_form_.
# Unless some label hash already contains _overt_form_, a new one is created.
def update_lang_hyp_label(overt, lang_hyp, letter)
  overt_hash = find_label_hash(overt)
  if overt_hash ==  nil then
    overt_hash = create_new_label_hash(overt, lang_hyp, letter)
    letter = letter.succ
  end
  # Break if the _lang_hyp_ label already includes the label for this overt form
  unless lang_hyp.lang_hyp_label.empty? then
    return letter if lang_hyp.lang_hyp_label.include?(overt_hash[overt.to_s])
  end
  # Get the output commitment for this overt form in _lang_hyp_
  output = lang_hyp.commitments.existing_commitment_pair(overt)[1]
  raise "Cannot create new label without a committed output" if output == nil
  # Append the values of the keys matching _overt_ and _output_ in string representation.
  lang_hyp.lang_hyp_label << overt_hash[overt.to_s] << overt_hash[output.to_s]
  # Return the letter to be used for the next new label hash
  return letter
end
```

```
 # Searches through the label set to find the label containing the given form _f_.
    # Returns that label.
    def find_label_hash(f)
      form = f.to_s
      matching_hashes = self.find_all {|label| label.any? {|el| el.include?(form)}}
      if matching_hashes.size >1 then
        raise "Error - more than one label hash exists for the form #{f.to_s}"
      else
        return matching_hashes[0]
      end
      return nil
    end


  end # class Label_set
end #module Overt_OTLearn
```

B-15 LANGUAGE_HYPOTHESIS.RB

```
# Author: Crystal Akers
#

require 'hypothesis'
require 'otlearn'
require 'overt_otlearn/data_manip'
require 'overt_otlearn/commitment_list'
require 'overt_otlearn/label_set'
require 'overt_otlearn/overt_grammar_test'
require 'sf/sf_word'
require 'sf/system'

module Overt_OTLearn
```

```
# A language hypothesis is a grammar hypothesis plus a list of paired
# overt forms and outputs comprising the committed structural
# interpretations for the language.
# Each hypothesis also contains a list of winners, a list of
# OvertGrammarTest results, and a boolean record of whether the
# hypothesis has been changed during learning.

class Language_Hypothesis < Hypothesis

  attr_reader :commitments, :overt_forms, :results_list, :winner_list, :learning_change
```

```
# Creates a new language hypothesis. If they are not provided as
 # parameters, the lists of commitment pairs, winners, and results
 # are created as empty initial lists.
 #   @commitments stores commitment pairs: [overt form, output].
 #   @winner_list stores full structural descriptions whose outputs
 #    are included in _@commitments_
 #   @results_list stores a history of the language hypothesis'
 #   results on Grammar Tests. The results list is duplicated and
 #   extended if the hypothesis branches.
 #   @learning_change stores a boolean for indicating whether the
 #   hypothesis has changed during (some period of) learning.
 #
 # The following store measures for evaluating efficiency.
 #   num_phonotactic_ercs stores the number of ERCs created during
 #   phonotactic learning
 #   num_set_features stores the number of features set
 def initialize(gram, erc_list=nil, commitments=nil, overt_forms = nil,
    winner_list=nil, results_list = nil, learning_change = nil,
    lang_hyp_label = nil, num_phonotactic_ercs = nil, num_set_features=nil)
  super(gram, erc_list)
  if commitments.nil? then
    @commitments = Commitment_List.new
  else
    @commitments = commitments
  end
  if overt_forms.nil? then
```

```
  @overt_forms = Array.new
else
  @overt_forms = overt_forms
end
if winner_list.nil? then
  @winner_list = []
else @winner_list = winner_list
end
if results_list.nil? then
  @results_list = []
else
  @results_list = results_list
end
if learning_change.nil? then
  @learning_change = true
end
if lang_hyp_label.nil? then
  @lang_hyp_label = String.new
end
if num_phonotactic_ercs.nil? then
  @num_phonotactic_ercs = 0
else
  @num_phonotactic_ercs = num_phonotactic_ercs
end
if num_set_features.nil? then
  @num_set_features = 0
```

```
  else
    @num_set_features = num_set_features
  end
end

# Returns the commitments for the language hypothesis.
def commitments() @commitments end

# Returns the list of winners for the language hypothesis.
def winner_list() @winner_list end

# Returns the results list for the language hypothesis.
def results_list() @results_list end

# Returns a boolean representing the learning change for the language hypothesis.
def learning_change() @learning_change end

# These values are output in the Excel learning summary spreadsheets
# Returns the number of ERCs created during phonotactic learning
def num_phonotactic_ercs() @num_phonotactic_ercs end

# Returns the number of features set in the language hypothesis
def num_set_features() @num_set_features end
```

```
# Sets the value of @learning_change to a boolean
def hyp_change(boolean)
  @learning_change = boolean
end

# Returns the set of overt forms used in commitment pairs
# in the language hypothesis.
def overt_forms() @overt_forms end

# Return the label of the language hypothesis
def lang_hyp_label() @lang_hyp_label end
```

```
# Returns a copy of the language hypothesis, with a duplicated
 # grammar, erc_list,commitment_pair list, overt_forms list,
 # winner list, and results list, num phonotactic ercs,
 # num set features, and label.
def dup
  hyp_dup = super
  winners = []
  @winner_list.each do |win|
    w =  win.dup
    w.sync_with_hypothesis!(hyp_dup)
    winners << w
  end
  label = String.new
  label = @lang_hyp_label.dup
  lang_hyp = Language_Hypothesis.new(hyp_dup.grammar, hyp_dup.erc_list,
    @commitments.dup,@overt_forms.dup, winners, @results_list.dup)
  lang_hyp.lang_hyp_label << label
  lang_hyp.store_phonotactic_erc_size(@num_phonotactic_ercs)
  lang_hyp.store_num_set_features(@num_set_features)
  return lang_hyp
end
```

```
  # Given a language hypothesis and an overt form, returns a list of
   # all consistent branches from that hypothesis. Each branch begins
   # as a copy of the given language hypothesis, to which a new
   # commitment_pair and winner are added if necessary.
   # The ranking bias flag determines which ranking bias to use:
   # if the flag is nil, the FaithLow bias is used, otherwise the MarkLow
   # bias is used. The flag should be nil except when the Branch method is
   # called after setting a feature.
   def branch(overt_form, ranking_bias_flag)
     branch_list = []
     discards = []
     interpretations = []
     interpretations = system.get_interpretations(overt_form,grammar)
     interpretations.each do |word|
       lang_hyp = self.dup
       # Add a new commitment pair and winner to the branch
       commit_pair = lang_hyp.commitments.add_commitment_pair(word.output)
       lang_hyp.add_winner(word.overt, commit_pair)
       OTLearn::ranking_learning(lang_hyp.winner_list, lang_hyp, ranking_bias_flag)
       lang_hyp.results_list <<
          Overt_OTLearn::OvertGrammarTest.new(lang_hyp, "Branch committed to
#{word.output.to_s} ")
       if lang_hyp.consistent? then
         lang_hyp.hyp_change(true)
         branch_list << lang_hyp
       else
```

```
      discards << lang_hyp
    end
  end
  return branch_list, discards
end
```

```
# Updates the constraint hierarchy in the grammar, regardless of whether the
# hypothesis is consistent. This update ensures that inconsistent language
# hypotheses will show which ERCs lead to the inconsistency.
# An optional block provides the code for generating the updated
# grammar (some variation of Rcd). If no block is provided, then
# regular RCD is used (all constraints has high as possible).
def update_grammar
  if block_given?
    rcd_result = yield(@erc_list)
  else
    rcd_result = Rcd.new(@erc_list)
  end
  @consistent = rcd_result.consistent?
  @grammar.hierarchy = rcd_result.hierarchy
  return rcd_result
end
```

```
# This method creates a new winner having the same morphword
   # as _overt_form_ and the output from the given
   # _commitment_pair_, then adds the winner to the winner list.
   # The method returns the winner.
   def add_winner(overt_form, commitment_pair)
      prior_winner = existing_winner(overt_form)
      raise "Prior winner exists: #{prior_winner.to_s}" unless prior_winner.nil?
      output = commitment_pair[1].dup
      # Syllables in _output_ are set to the same morpheme as the corresponding
      # syllables in _overt_form_
      gen = SyncEnumerator.new(overt_form, output.syl_list)
      gen.each do |overt_syl,output_syl|
        output_syl.set_morpheme(overt_syl.morpheme)
      end
      output.morphword = overt_form.morphword
      winner = self.system.parse_output(output, grammar)
      self.winner_list << winner
      return winner
   end
```

```
# This method returns the winner (full structural description,
 # though the input may include unset features)whose morphword
 # matches _overt_form_, if such a winner exists in the language
 # hypothesis; it returns nil otherwise.
def existing_winner(overt_form)
  match = self.winner_list.find {|winner| winner.morphword == overt_form.morphword}
  if match then
    if match.overt != overt_form then
      raise "Overt form #{overt_form.to_s} doesn't match existing winner #{match.to_s}"
    end
  end
  return match
end

def store_phonotactic_erc_size(num)
  @num_phonotactic_ercs=num
end

def store_num_set_features(num)
  @num_set_features +=num
end
```

```
    # Returns a string containing string representations of the hierarchy, lexicon,
    # ERC list and structural commitment_pair_pairs and results list of this language hypothesis.
    def to_s
      out_str = "HIERARCHY" + "\n"
      out_str += @grammar.hierarchy.to_s + "\n"
          out_str += "LEXICON"+ "\n"
      out_str += @grammar.lexicon.to_s + "\n"
          out_str += "ERC LIST"+ "\n"
      out_str += @erc_list.join("\n")
          out_str += "\n"+ "COMMITMENT PAIRS"+ "\n"
      out_str += @commitments.to_s
          out_str += "\n"+"WINNER LIST"+ "\n"
      out_str += @winner_list.join("\n")
          out_str += "\n"+"RESULTS LIST"+ "\n"
      out_str += @results_list.join("\n")
      out_str
    end

  end # class Language_Hypothesis

end # module Overt_OTLearn
```

B-16 COMMITMENT_LIST.RB

```
# Author: Crystal Akers
#

require 'hypothesis'
require 'otlearn'
require 'overt_otlearn/data_manip'
require 'sf/sf_output'

module Overt_OTLearn

# Commitments are arrays containing pairs of overt form and committed output,
# each pair a size 2 array with the first element an overt form and the second
# element a structural interpretation of the overt form.

  class Commitment_List < Array

    # Returns an empty Commitment list
    def initialize
    end
```

```
# Returns true if the string representation of _form_ matches that of either
  # member of the commitment (overt form or structural interpretation);
  # it returns false otherwise.
  def forms_match?(form, commit_pair)
    return true if commit_pair.any? {|committed_form| committed_form.to_s == form.to_s}
    return false
  end

  # Returns the commitment pair whose overt form or committed output interpretation
  # matches _form_; it returns nil if there's no such pair.
  def existing_commitment_pair(form)
    self.each { |commit_pair| return commit_pair if self.forms_match?(form, commit_pair) }
    return nil
  end
```

```
# Adds to a new commitment pair with the structural interpretation provided
# by _output_. Returns the new pair.
def add_commitment_pair(output)
  # _output_ must not match an existing commitment pair
  raise "Existing commitment for output: #{output.overt.to_s}, #{output.to_s}" if
    self.existing_commitment_pair(output)
  # The overt form of _output_ must not match an existing commitment pair
  overt = output.overt
  raise "Existing commitment for overt form: #{output.overt.to_s}, #{output.to_s}" if
    self.existing_commitment_pair(overt)
  commitment_pair = [overt, output.dup]
  self << commitment_pair
  return commitment_pair
end

#Returns a copy of the commitment pair
def dup
  super
end
```

```
  def to_s
    out_str = ""
    self.each do |commit_pair|
      out_str << "["
      out_str << commit_pair[0].join
      out_str << ", "
      out_str << commit_pair[1].join
      out_str << "]"
      out_str << "\n"
    end
    return out_str
  end

 end # class Commitment
end # module Overt_OTLearn
```

BIBLIOGRAPHY

Alber, Birgit. 2005. Clash, Lapse, and Directionality. *Natural Language and Linguistic Theory* 23: 485 – 542.

Alderete, John, Adrian Brasoveanu, Nazarré Merchant, Alan Prince, and Bruce Tesar. 2005. Contrast analysis aids the learning of phonological underlying forms. In *Proceedings of the Twenty-Fourth West Coast Conference on Formal Linguistics,* ed. by John Alderete, Chung-hye Han, and Alexei Kochetov, 34-42. Cascadilla Press.

Angluin, Dana. 1980. Inductive inference of formal languages from positive data. *Information and Control* 45, 117-135.

Apoussidou, Diana. 2007. The Learnability of Metrical Phonology, Linguistics, University of Amsterdam: PhD.

Apoussidou, Diana and Paul Boersma. 2003. The learnability of Latin Stress. In *Proceedings of the Institute of Phonetic Sciences, Amsterdam* 25: 101-148. ROA 643.

Baker, C.L. 1979. Syntactic Theory and the projection problem. *LI* 10:4, 533-581.

Bat-El, Outi. 1993. Parasitic metrification in the Modern Hebrew stress system. *The Linguistic Review* 10: 189-210.

Becker, Michael. 2003a. Lexical stratification of Hebrew: The disyllabic maximum. In *Proceedings of the Israel Association for Theoretical Linguistics 19*, ed. Yehuda Falk.

Becker, Michael. 2003b. Hebrew Stress: Can't you hear those trochees? In *Proceedings of Penn Linguistics Colloquium 26,* eds. Kaiser and Arunachalam, 9.1, 45–58.

Boersma, Paul. 1997. How we learn variation, optionality, and probability. In *Proceedings of the Institute of Phonetic Sciences* 21: 43-58. University of Amsterdam. ROA-221.

Boersma, Paul. 2001. Phonology-semantics interaction in OT, and its acquisition. In *Papers in Experimental and Theoretical Linguistics*, eds. Robert Kirchner, Wolf Wikeley and Joe Pater, 24-35. Edmonton: University of Alberta. ROA-369.

Boersma, Paul and Bruce Hayes. 2001. Empirical tests of the Gradual Learning Algorithm. *Linguistic Inquiry* 32: 45–86.

Bolozky, Shmuel. 1982. Remarks on Rhythmic Stress in Modern Hebrew. *Journal of Linguistics* 18:2, 275-289. Cambridge University Press.

Brasoveanu, Adrian. 2003. Minimal Fusion Normal Form. Ms. Rutgers University, NB. http://people.ucsc.edu/~abrsvn/MFNF.pdf

Brasoveanu, Adrian and Alan Prince. 2011. Ranking and necessity: the Fusional Reduction Algorithm.In *Natural Language and Linguistic Theory* 29: 3-70.

Gleason, Jean Berko. 1958. The Child's Learning of English Morphology. *Word* 14: 150-77.

Gold, E. M. 1967. Language Identification in the Limit. *Information and Control*, 16:447-474.

Graf, Dafna and Adam Ussishkin. 2003. Emergent iambs: Stress in Modern Hebrew. Lingua 113: 239-270.

Graf, Dafna. 2000. Stress Assignment in the Nominal System of Modern Hebrew (MH). In *The Proceedings of the 15th Annual Conference of IATL Vol 7,* ed. Adam Z Wyner. Israel.

Gordon, Matthew. 2002. A factorial typology of quantity-insensitive stress. *Natural Language and Linguistic Theory* 20, 491–552.

Hayes, Bruce. 1995. Metrical stress theory: principles and case studies. University of Chicago Press, Chicago.

Hayes, Bruce. 2004. Phonological acquisition in Optimality Theory: the Early Stages. In *Constraints on Phonological Acquisition,* ed. by René Kager, Joe Pater, and Wim Zonneveld, 158-203. Cambridge: Cambridge University Press. ROA-327.

Iversen, J.R., Patel, A.D., & Ohgushi, K. (2008). Perception of rhythmic grouping depends on auditory experience. *Journal of the Acoustical Society of America*, 124: 2263-2271.

Jarosz, Gaja. 2006. *Rich Lexicons and Restrictive Grammars - Maximum Likelihood Learning in Optimality Theory*. Ph.D. dissertation, Johns Hopkins University.

Jarosz, Gaja. 2007. Restrictiveness in Phonological Grammar and Lexicon Learning. In *Proceedings of the 43rd Annual Meeting of the Chicago Linguistics Society* 43: 125-139.

Jarosz, Gaja. To appear. Naïve Parameter Learning for Optimality Theory – the Hidden Structure Problem. In *Proceedings of the 40th Annual Meeting of the North East Linguistic Society.*

Kager, René. 2001. Rhythmic Directionality by Positional Licensing. Handout of talk at the Fifth HILP Phonology Conference, Potsdam. ROA-514.

Kager, René. 2006. Rhythmic licensing theory: An extended typology. In *Proceedings of the Third International Conference on Phonology,* 5-31. Seoul: The Phonology-Morphology Circle of Korea.

Kusumoto, Kiyomi, and Elliott Moreton. 1997. Native language determines parsing of nonlinguistic rhythmic stimuli. Abstract in *Journal of the Acoustical Society of America* 102(5, Part 2):3204. Poster: http://www.unc.edu/~moreton/Papers/ASA1997DecKM.pdf

McCarthy, John and Alan Prince. 1993. Generalized Alignment. In G. Booij and J. van Marle (eds), *Yearbook of Morphology 1993*, 79-153. Kluwer: Boston. ROA-7.

Merchant, Nazarré. 2008. *Discovering Underlying Forms: Contrast Pairs and Ranking.* PhD dissertation, Rutgers University. ROA-964.

Merchant, Nazarré and Bruce Tesar. 2008. Learning underlying forms by searching restricted lexical subsets. In *The proceedings of CLS 41* (2005). ROA-811.

Ota, M., Ladd, and D. R., Tsuchiya, M. 2003. Effects of foot structure on mora duration in Japanese? *Proceedings of the 15th International Conference on Phonetic Sciences*, 459-462. Barcelona: Universitat Autonoma de Barcelona.

Prince, Alan. 2000. Comparative Tableaux. ROA-964.

Prince, Alan. 2002a. *Entailed Ranking Arguments*. ROA-500.

Prince, Alan. 2002b. Arguing Optimality. ROA-562.

Prince, Alan and Paul Smolensky. 1993. *Optimality Theory: Constraint Interaction in Generative Grammar.* RuCCS-TR-2, Rutgers Center for Cognitive Science, Rutgers University. ROA – 537.

Prince, Alan and Bruce Tesar. 2004. Learning Phonotactic Distributions. In *Constraints on Phonological Acquisition,* ed. by René Kager, Joe Pater, and Wim Zonneveld, 245-291. Cambridge: Cambridge University Press. ROA-353.

Samek-Lodovici, Vieri, and Alan Prince. 1999. Optima. RuCCS-TR-57. ROA-363.

Samek-Lodovici, Vieri, and Alan Prince. 2005. Fundamental Properties of Harmonic Bounding. RUCCSTR-71: http://ruccs.rutgers.edu/tech_rpt/harmonicbounding.pdf. Corrected 2005 as ROA-785.

Selkirk, Elisabeth O. 1984. *Phonology and Syntax: The Relation between Sound and Structure.* MIT Press, Cambridge, MA.

Tesar, Bruce. 1995. *Computational Optimality Theory.* Unpublished PhD dissertation. University of Colorado, Boulder, CO.

Tesar, Bruce. 1997. Multi-Recursive Constraint Demotion. Ms, Rutgers University. ROA-197.

Tesar, Bruce. 1998a. Error-driven learning in Optimality Theory via the efficient computation of optimal forms. In: P. Barbosa, D. Fox, P. Hagstrom, M.McGinnis, & D. Pesetsky (Eds.), *Is the Best Good Enough? Optimality and Competition in Syntax* (pp. 421-435).Cambridge, Mass.: MIT Press and MIT Working Papers in Linguistics.

Tesar, Bruce. 1998b. An iterative strategy for language learning. *Lingua* 104: 131-145.

Tesar, Bruce. 2000. Using inconsistency detection to overcome structural ambiguity in language learning. Technical Report RuCCS-TR-58, Rutgers Center for Cognitive Science, Rutgers University. ROA-426.

Tesar, Bruce. 2002. Enforcing grammatical restrictiveness can help resolve structural ambiguity. In Proceedings of the Twenty-First West Coast Conference on Formal Linguistics, eds. Line Mikkelsen and Christopher Potts, 443-456. Somerville, MA: Cascadilla Press. ROA-618.

Tesar, Bruce. 2004a. Using inconsistency detection to overcome structural ambiguity in language learning. *Linguistic Inquiry* 35: 219-253. ROA-426.

Tesar, Bruce. 2004b. Contrast analysis in phonological learning. Ms., Linguistics Dept., Rutgers University. ROA-695.

Tesar, Bruce. 2006. Learning from paradigmatic information. In *The Proceedings of NELS 36*. ROA-795.

Tesar, Bruce. 2008. Output-Driven Maps. Manuscript, Linguistics Dept., Rutgers University. ROA-956.

Tesar, Bruce. 2009. Learning Phonological Grammars for Output-Driven Maps. To appear in *The Proceedings of NELS 39*. ROA-1013.

Tesar, Bruce, & Alan Prince. 2003. Using phonotactics to learn phonological alternations. In *Proceedings of the thirty-ninth conference of the Chicago Linguistics Society (2003), Vol. II: The Panels*, 209-237. ROA-620.

Tesar, Bruce and Paul Smolensky. 1994. The learnability of Optimality Theory. In *Proceedings of the Thirteenth West Coast Conference on Formal Linguistics,* ed. by Raul Aranovich, William Byrne, Susanne Preuss, and Martha Senturia, 122-137. Stanford, Calif.: CSLI Publications. ROA-2.

Tesar, Bruce and Paul Smolensky. 1998. Learnability in Optimality Theory. Linguistic Inquiry, 29, 229-68.

Tesar, Bruce and Paul Smolensky. 2000. *Learnability in Optimality Theory.* Cambridge, Mass.: MIT Press.

Tessier, Anne-Michelle. 2007. *Biases and Stages in Phonological Acquisition*. Ph.D. dissertation, University of Massachusetts Amherst. http://scholarworks.umass.edu/dissertations/AAI3254906

Wexler, Kenneth and Peter Culicover. 1980. *Formal principles of language acquisition.* Cambridge, Mass.: MIT Press.

Yang, Charles. 2002. *Knowledge and Learning in Natural Language.* Oxford: Oxford University Press.

CURRICULUM VITAE

CRYSTAL G. AKERS

EDUCATION

2005- 2012	Ph.D in Linguistics Rutgers University, New Brunswick, NJ.
1997-2001	BA in Linguistics, with Honors Swarthmore College, Swarthmore, PA.

POSITIONS

2010-2011	Mellon Dissertation Fellow, Rutgers University.
2008-2009	Teaching Assistant, Writing Program, Rutgers University.
2007-2008	Teaching Assistant, Department of Linguistics, Rutgers University.
2005-2007	Graduate Fellow, Rutgers University.
2004-2005	English teacher, Western Reserve Academy, Hudson, OH.
2003-2004	Math teacher, Laurel School, Shaker Heights, OH.
2002-2003	Math teacher, Dutchess Day School, Millbrook, NY.
2001-2002	Intern, Christchurch School, Christchurch,VA.