24 Learnability

Bruce Tesar
Rutgers University

24.1 Learnability in Phonology

A fundamental tenet of cognitive science is that human mental processes, including those involved in language, are computational processes. On this view, for a child to learn their native language, there must exist a learning *algorithm* capable of determining the grammar of that language from a reasonable amount of data, and with a reasonable amount of computational effort. Language learnability is the study of the computational dimensions of language learning. Learnability works in tandem with the study of child language data, commonly known as language acquisition, to constitute the overall study of how children learn their native language.

While it is difficult to quantify how much data a child *needs* in order to reliably learn a grammar, it is not too difficult to impose a generous upper bound on the amount of data a child could possibly have by estimating the number of utterances a child could hear during the waking hours of the first few years of their life. Even such generous overestimates prove to have real consequences for learnability; it is remarkably easy to define learning algorithms which demonstrably require decades worth of data to work for even rather simple classes of grammars.[1]

Quantifying a 'reasonable' amount of computational effort is a far more murky matter, due in no small part to science's current vast ignorance of the computational properties of the human brain. But some fairly rudimentary assumptions turn out to have non-trivial implications for language learning. The very nature of computation requires that the learning algorithm use only a *finite* amount of computational effort. That requirement turns out to be sufficient to rule out certain simple learning proposals. In practice, researchers use gross measures of evaluation, based upon basic plausibility. If a proposed learning algorithm, implemented on the world's fastest supercomputer, would require several centuries to complete, then it is generally regarded as requiring an implausible amount of computation.

Learnability is a class phenomenon: it is a property of classes of grammars. A class is learnable if an algorithm exists which can determine, based on example data, the target grammar (the one generating that data), and do so for data from every grammar in the class. What makes learning easy or difficult is the amount of data and effort necessary to reliably identify the target grammar out of the class of allowable grammars. The challenge arises from the need to distinguish between different possible grammars, rather than from the individual grammars themselves. Learning Japanese is trivial if you need not consider any alternatives; you just build Japanese into the system. Human

[1] For an example involving parametric grammars, see the discussion by Sakas and Fodor (2001) of the Triggering Learning Algorithm (Gibson & Wexler 1994).

language learning is challenging because the class of allowable languages must include all possible human languages.

Linguistic theory is central to language learnability. A linguistic theory proposes a class of possible human languages, which is effectively the class of allowable languages for the learning problem. It is a reasonable and uncontroversial assumption that the formal relationships among the allowable grammars that make language learnable are intimately related to the formal properties of the linguistic theory defining the class of allowable grammars.

In this chapter, I will assume that the class of allowable languages is defined by an Optimality Theoretic system (Prince & Smolensky 1993/2004). Each language is generated by a grammar consisting of a total ranking of the universal constraints and a lexicon of phonological underlying forms for morphemes. The learner's job is to identify, for a given set of data, the *target* grammar that gave rise to that data. The learner attempts to do this by searching for the grammar that best captures the patterns in the data. This involves finding a grammar which generates all of the observed data (it is consistent with the data), but as little as possible of the unattested data (the most restrictive such grammar). If the observed data include no examples of voiced obstruents, a grammar that does not generate any voiced obstruents is more restrictive and more likely to be correct than a grammar that does.

(1) *Goals of the learner*
(a) to find a ranking and a set of underlying forms that successfully reproduce the observed data.
(b) to find the most restrictive ranking consistent with the data.

A complete and satisfactory solution to this problem will not be found in this chapter, or anywhere else. Simple exhaustive search of the possible grammars isn't an option for the simple reason that there are an infinite number of possible lexica (=plural of 'lexicon'), given that there is in principle no bound on the size of an underlying form. One could derive some formal limits on how long an underlying form could need to be, given some maximal observed morpheme length, but the resulting space of possible grammars would be finite but too large to plausibly search exhaustively. This is due to combinatorial explosion: the number of total rankings of a set of constraints of size C is C! (i.e. the factorial C!: the number of permutations of the constraints). If segments are described with F binary features, there are 2^F possible underlying forms for a segment, and a word with at most S segments in its underlying form has $2^F + (2^F)^2 + (2^F)^3 + \ldots + (2^F)^S$ possible underlying forms, which is on the order of 2^{FS}. Modest assumptions still yield a huge number of grammars. 50 constraints yields 3×10^{64} possible rankings. 10 binary features and an average limit of 4 segments for an underlying form yields $2^{40} \approx 1\times10^{12}$ possible underlying forms for a morpheme; for a lexicon of 1000 morphemes, that yields $2^{40\times1000} \approx 10^{12041}$ possible lexica. The number of grammars itself is of course the product of the number of possible rankings and the number of possible lexica.

Most of the results in this chapter focus on the learning of constraint rankings, reflecting a majority of the work done to date. These results support algorithms that exploit the formal structure of Optimality Theory to efficiently learn the correct constraint ranking. Relatively little work has been done on learning phonological lexica, but see section 7 for discussion of current work on this topic.

24.2 Learning with soft constraints: Constraint Demotion

The fundamental unit of information about constraint rankings is the elementary ranking condition. In learning (as in linguistic argumentation), elementary ranking conditions are normally constructed from pairwise comparisons between candidates, with one candidate presumed to be grammatical in the target language, labeled the *winner*, and the other candidate a competitor for the same input, labeled the *loser*. Such comparisons are commonly called winner-loser pairs.

I will illustrate several properties of learning algorithms with a very simple linguistic system allowing grammars with different consonant inventories. The constraints are given in (2).

(2) The Consonant Inventory Constraints

*+VOICE (*VOI)	‘Consonants should not be voiced.’ (Lombardi 1999)
*+NASAL (*NAS)	‘Consonants should not be nasal’
{–VOICE&+NASAL} (–VOI/NAS):	‘Consonants should not be both nasal and voiceless’
IDENT[voice] (ID[voi]):	‘Correspondents should have the same value for voicing’ (McCarthy & Prince 1995)
IDENT[nasal] (ID[nas]):	‘Correspondents should have the same value for nasality’ (McCarthy & Prince 1995)

(3) *An example winner-loser pair*

input	win ~ lose	*VOI	*NAS	*–VOI/NAS	ID[voi]	ID[nas]
/ni/	ni ~ ti	L	L	e	W	W

A winner-loser pair has a particular logical structure: at least one of the constraints preferring the winner must dominate all of the constraints preferring the loser. In the example in (3), at least one of ID[voi] and ID[nas] must dominate both *VOI and *NAS.

The key to learning a ranking from a set of winner-loser pairs is to find a way of combining the information in each of the pairs. An algorithm which does this is Recursive Constraint Demotion (Tesar 1995, Tesar & Smolensky 1998), referred to here as RCD. RCD constructs a constraint ranking consistent with a set of winner-loser pairs. It does so top-down, determining the highest-ranked constraints first, then the next-highest, and so forth. The key observation is that a constraint can be ranked at the top if it does not prefer the loser for any of the pairs. The discussion will be illustrated using Comparative tableaux (Prince 2002).

(4) *The winner-loser pairs before the first pass of RCD*

input	win ~ lose	*Voi	*Nas	*–Voi/Nas	ID[voi]	ID[nas]
/ni/	ni ~ ti	L	L	e	W	W
/di/	ti ~ di	W	e	e	L	e

In the set of pairs in (4), *–Voi/Nas and ID[nas] do not prefer any losers. RCD places them together at the top of the ranking, as it lacks any basis for determining a ranking relation between them. The top stratum is {*–Voi/Nas, ID[nas]}. RCD then checks for pairs in which one of the constraints just ranked prefers the winner. Such pairs are now fully accounted for. In the example, the first pair is such a case, as ID[nas] prefers the winner. By definition of the algorithm, none of the constraints preferring the loser can have been ranked yet, so the ranking of ID[nas] insures that a constraint preferring the winner dominates all of the constraints preferring the loser. RCD removes all such pairs from the list, as well as the constraints just ranked. In the example, this results in the table in (5).

(5) *The winner-loser pairs after the first pass of RCD*

input	win ~ lose	*Voi	*Nas	ID[voi]
/di/	ti ~ di	W	e	L

RCD now performs a second pass through the list, using identical logic. For the constraints not yet ranked, the ones that can be ranked highest are those that do not prefer any (remaining) losers. In the example, *Voi and *Nas qualify, and are placed by RCD into the ranking. The top two strata are now {*–Voi/Nas, ID[nas]} » {*Voi, *Nas}. RCD now observes that in the remaining winner-loser pair, *Voi prefers the winner, so the pair may be removed. This leaves an empty list; there are no losers remaining, so all remaining constraints are now free to be placed in the ranking. In the example, the only remaining constraint is ID[voi], which is ranked at the bottom. RCD is now complete. The constructed ranking is {*–Voi/Nas, ID[nas]} » {*Voi, *Nas} » ID[voi].

Given a consistent set of winner-loser pairs, RCD is guaranteed to construct a constraint hierarchy consistent with all of them. In fact, it always constructs a specific hierarchy relative to the winner-loser pairs: it constructs the hierarchy in which each constraint is ranked as high as possible, consistent with the data. This follows from the fact that the algorithm places constraints into the hierarchy as soon as they are 'free' (they do not prefer the loser for any remaining winner-loser pair). Choosing the hierarchy with each constraint ranked as high as possible is very convenient computationally. However, that is not quite the right bias for learning, and that motivates the modifications to the RCD algorithm discussed in the section on phonotactic learning below.

RCD is computationally efficient. Given consistent data, it is guaranteed to place at least one constraint into the hierarchy on each pass through the list of winner-loser pairs, so the number of passes is at most the number of constraints.

One important property of RCD is that it automatically detects inconsistencies in the list of winner-loser pairs, as a side effect of trying to find a ranking. An example of a list of inconsistent winner-loser pairs is given in (6).

(6) *An inconsistent set of winner-loser pairs ('n̥' denotes a voiceless nasal)*

input	win ~ lose	*Voi	*Nas	*–Voi/Nas	ID[voi]	ID[nas]
/n̥i/	n̥i ~ ti	e	L	L	e	W
/ni/	di ~ ni	e	W	e	e	L

The first pass of RCD places *Voi and ID[voi] into the top stratum of the hierarchy. However, neither constraint prefers a winner, so no pairs are removed from the list. Once those two constraints are removed, the remaining winner-loser pair list is as shown in (7).

(7) *After the first pass, all remaining constraints prefer a loser*

input	win ~ lose	*Nas	*–Voi/Nas	ID[nas]
/n̥i/	n̥i ~ ti	L	L	W
/ni/	di ~ ni	W	e	L

RCD is unable to continue constructing the hierarchy at this point, because all of the remaining constraints prefer a loser. No matter which constraint is placed into the hierarchy next, at least one of the designated optimal candidates (a winner) will lose to a competitor. This is the key indicator that the list is inconsistent. RCD can halt immediately and return an indication to the learner that inconsistency has been detected.

Inconsistency detection is powerful because it allows the learner to test the compatibility of different hypotheses about the grammar. The use of inconsistency detection in learning will be discussed further in the section on structural ambiguity.

24.3 Selecting competitors and the role of parsing

RCD will find a ranking compatible with a list of winner-loser pairs, but does not construct the list itself. One issue in the construction of such a list is the selection of appropriate competitors (the losers). Assuming (for the moment) that a winner can be determined from observed data, there are typically many losing candidates that the winner must beat. Constructing a separate winner-loser pair for each competitor is very computationally expensive and, in cases where the number of competitors is infinite, completely untenable. It is also unnecessary; normally many winner-loser comparisons are redundant in the information they supply about the ranking. What is needed is a way to efficiently select a set of informative competitors.

Informative competitors can be selected using error-driven learning (Tesar 1998). Given a grammatical form, the learner tests the input (i.e. phonological underlying form) for that form with their current hypothesized constraint ranking, to see what candidate is assigned as optimal. If the candidate chosen by the learner's current ranking is identical to the grammatical form, then the grammatical form will not motivate any changes to the ranking; all possible winner-loser pairs for that form are already satisfied. If, on the other hand, a candidate other than the grammatical form is chosen by the learner's current ranking, then the ranking needs to be changed; an *error* has occurred in the sense that the learner's current grammar does not generate the independently observed grammatical form.

In the most general sense of error-driven learning, the error indicates that the learner's grammar must be changed somehow, but doesn't necessarily indicate how. In Optimality Theory, the error gives a concrete indication of how to change the ranking. This is because the learner's parsing mechanism has produced a different candidate as optimal for the learner's current ranking. This means that the comparison between the grammatical candidate and the one generated by parsing is guaranteed to provide information not reflected in the learner's current ranking. In other words, the candidate generated by parsing is informative. Thus the learner can use the parsing mechanism, which is independently necessary in any event, to select informative competitors.

One consequence of using error-driven learning to select competitors is that the learner must have a hypothesis ranking in order to select a competitor. RCD constructs rankings based upon winner-loser pairs. The two can be productively combined in the form of an algorithm known as Multi-Recursive Constraint Demotion, or MRCD (Tesar 1997). MRCD maintains a list of winner-loser pairs at all times. Whenever presented with data in the form of a grammatical candidate, the learner can use RCD to generate a ranking from the list of winner-loser pairs. The learner can then parse the input for the grammatical candidate. If an error occurs, the learner constructs a new winner-loser pair, using the data-presented grammatical candidate as the winner, and the candidate generated by the parser as the loser. This new winner-loser pair is then added to the learner's list. The learner thus accumulates winner-loser pairs over time, with each new pair guaranteed to provide information not found in the previous pairs.

This can be illustrated with data generated by the target grammar in (8).

(8) *–VOI/NAS » ID[nas] » *VOI » *NAS » ID[voi]

The possible CV-form outputs for the ranking are [ti] and [ni], restricting consonants to coronals and vowels to [i].

The learner starts out with an empty list of winner-loser pairs. When presented with data, the learner needs a hierarchy to parse with. RCD, when applied to an empty list, ranks all of the constraints in the top stratum of the hierarchy (there are no losers to be preferred by any of the constraints).

(9) *Initial Hierarchy:* {*–VOI/NAS, *NAS, *VOI, ID[nas], ID[voi]}

Suppose the learner first receives (via observed data) the grammatical candidate /ni/↔[ni], with both input and output as [ni]. The evaluation of the four relevant candidates is shown in

(10) *Evaluation of the four candidates for /ni/*

input: /ni/	*VOI	*NAS	*–VOI/NAS	ID[voi]	ID[nas]
ti				*	*
di	*				*
n̥i		*	*	*	
ni	*	*			

Using the initial hierarchy, the parser will return the outputs [ti], [di], and [ni] as tied for optimal (assuming that evaluation is performed based upon ‘pooling the marks’ – i.e. taking all the violations of all the constraints of a stratum together). Not all of these candidates match the grammatical output [ni]. The learner (arbitrarily) chooses [ti] as a loser, forms a new winner-loser pair, and adds it to the list, resulting in the list in (11).

(11) *The list with the first winner-loser pair*

input	win ~ lose	*VOI	*NAS	*–VOI/NAS	ID[voi]	ID[nas]
/ni/	ni ~ ti	L	L	e	W	W

The learner then applies RCD to this list, resulting in the hierarchy in (12).

(12) {*–VOI/NAS, ID[nas], ID[voi]} » {*NAS, *VOI }

Using that hierarchy, the learner then reparses the input /ni/, and observes that [ni] is now generated. The learner no longer gets a learning error on the form.

Next the learner considers the grammatical candidate /ti/↔[ti]. Using the hierarchy in (12), the learner parses the input /ti/ and generates [ti]. No learning error occurs, so no winner-loser pairs are created.

Next, the learner considers the grammatical candidate /di/↔[ti]. Using the hierarchy in (12), the learner parses the input /di/ and generates [di]. This is a learning error, so the learner constructs a new winner-loser pair for input /di/, with output [ti] as the winner and output [di] as the loser. This results in the list in (13).

(13) *The list after the second learning error*

input	win ~ lose	*VOI	*NAS	*–VOI/NAS	ID[voi]	ID[nas]
/ni/	ni ~ ti	L	L	e	W	W
/di/	ti ~ di	W	e	e	L	e

The learner then applies RCD to that list, constructing the hierarchy in (14).

(14) {*–VOI/NAS, ID[nas]} » {*VOI, *NAS} » {ID[voi]}

The learner then parses /di/ with the new hierarchy, and generates [ti]. No more learning errors occur, and in fact the learner now has a constraint hierarchy that is compatible with the entire language, so no more learning errors will occur. Learning has succeeded.

MRCD has efficient data complexity. There is a formally proven upper bound on the number of learning errors that can occur prior to the construction of a compatible constraint hierarchy: N(N-1)/2, where N is the number of constraints. This is less than N^2. Each learning error adds a winner-loser pair to the list, so this is also a bound on the number of pairs accumulated during learning with MRCD. In the most extreme case, each pair comes from a distinct data form, so the number of mutually informative pieces of data required by the algorithm is less than the square of the number of constraints. In practice, this upper bound is a gross overestimate, but it makes the point that the data demands of MRCD are not wildly large relative to the number of constraints.

24.4 Robustness to Data Errors

MRCD is vulnerable to data errors. For present purposes, a data error is a form which is ungrammatical in the target language, but appears in the learner's data. An ungrammatical form will commonly have one of two effects on MRCD: it will detect inconsistency if there is no ranking consistent with all of the data, or it will incorrectly select a less restrictive ranking permitting the ungrammatical form as well as all of the grammatical data.

A learning algorithm that demonstrates some robustness to data errors is the Gradual Learning Algorithm, or GLA (Boersma 1998, Boersma & Hayes 2001). The GLA uses a familiar strategy for error robustness: sensitivity to frequency. It is designed so that patterns which occur more frequently in the data are likely to have a greater impact on the choice of grammar; if errors occur infrequently enough, they probably will not have enough impact to affect the learner's choice of grammar.

The GLA realizes frequency sensitivity in part by adopting a different formal theory of phonology than MRCD does. The GLA presumes a stochastic variation of Optimality Theory (StOT). In this theory, a grammar assigns a *ranking value* to each constraint, where a ranking value is a real number. The absolute values of the ranking values are irrelevant; what matters for the model is the distances between the ranking values assigned different constraints. Whenever the grammar is used to evaluate a form, the ranking values of the constraints are used to determine an 'evaluation ranking' of the constraints. That ranking is then treated exactly as a normal Optimality Theoretic ranking for that evaluation.

The stochastic element of StOT lies in the method for constructing an evaluation ranking from the ranking values. For each constraint, a number called a *selection point* is generated randomly using a normal distribution with a mean equal to the constraint's ranking value. The standard deviation of the distribution is an independent and arbitrary value, called the *evaluation noise*. The selection point will be within one standard deviation of the ranking value approximately 66% of the time (this follows from the definitions of normal distribution and standard deviation). At evaluation time, a selection point is generated for each constraint, and then the constraints are ranked by their selection points, with the highest selection point deciding the highest-ranked constraint, and so forth. New selection points are generated for each evaluation, so it is possible for the same set of ranking values to give rise to different rankings on different evaluations. If constraint C1 has a higher ranking value than C2, we expect C1 to dominate C2 at evaluation time more than half of the time. The probability that C2 dominates C1 on an evaluation is determined by the distance between their ranking values, measured in terms of the evaluation noise. Again, the absolute magnitudes of the differences between ranking values is not key in and of itself, what matters is the distance between ranking values relative to the size of the evaluation noise.

A StOT grammar really defines a probability distribution over the possible total orderings of the constraints. It is not difficult to define a StOT grammar that is equivalent for practical purposes to a traditional OT ranking: simply assign the constraints ranking values that are several standard deviations apart, ordered according to the desired ranking, and the probability of any ranking other than the ranking value order being used at evaluation time will be extremely small.

In StOT, then, learning a grammar means learning a set of ranking values for the constraints that will generate the target language. That is the task of the GLA. The GLA shares several components with MRCD. It uses error-driven learning, and when a learning error occurs it constructs a winner-loser pair just as MRCD does. Instead of storing the winner-loser pair, however, the GLA uses it to make small changes to the ranking values of the constraints. The algorithm learns 'gradually', through the accumulated effects of many errors upon the ranking values.

The GLA responds to an error-induced winner-loser pair by increasing the ranking values of the constraints preferring the winner, and decreasing the ranking values of the constraints preferring the loser. The amount by which a ranking value is increased or decreased, a value called *plasticity*, is an independent parameter of the learning algorithm. One might normally imagine the size of plasticity to be small relative to the size of evaluation noise. However, versions of the GLA often employ schedules that change the size of plasticity during learning.

The GLA will eventually learn a consistent ranking for consistent data by accumulation of adjustments to the ranking values. The highest-ranked constraint will never prefer any losers. It can have its ranking value increased, but never decreased, and its ranking value will grow until it is enough higher than the ranking values of the relevant other constraints that it never is ranked below them at evaluation time. Given a relatively small plasticity value, constraint ranking values cannot be pushed far apart on the basis of one or a few learning errors; many over time will be needed. This is where the robustness comes from: a data error will only cause a minor change to the learner's grammar; the ranking values will change by at most the plasticity value. To have a long-term impact on learning, data errors would have to occur with some substantive frequency in the data.

The GLA has a couple of parameters that must be set to define a specific algorithm. A set of initial ranking values must be specified, so that error-driven learning can get started. A schedule of values for plasticity must also be provided. Needless to say, the performance of the algorithm can vary depending upon the values chosen for these parameters.

A fundamental difference between the GLA and MRCD lies in the response to inconsistent data. Recall the data in (6), repeated here as (15).

(15) *An inconsistent set of winner-loser pairs*

input	win ~ lose	*Voi	*Nas	*–Voi/Nas	ID[voi]	ID[nas]
/n̥i/	n̥i ~ ti	e	L	L	e	W
/ni/	di ~ ni	e	W	e	e	L

The GLA treats the winner-loser pairs in succession. The first pair causes the ranking value of ID[nas] to be increased, and the ranking values of *Nas and *–Voi/Nas to be decreased. The second pair then causes the ranking value of *Nas to be increased, and the ranking value of ID[nas] to be decreased. The GLA learner is completely unaware of the fact that these two pairs are contradictory; it just alters ranking values in response to each error. MRCD, on the other hand, is acutely aware of the contradiction, and is so preoccupied that it stops constructing a grammar. If one of the pairs is the result of a data error, then the obliviousness of the GLA pays off, giving it an intrinsic robustness to data errors that MRCD lacks. On the other hand, section 6 below, on structural ambiguity, will argue that the ability to detect inconsistency can be quite valuable in dealing with certain challenges in learning. Constructing a learner that can do both in a satisfying way remains a significant challenge.

24.5 The subset problem and phonotactic learning

Language learning is done largely on the basis of positive evidence; the learner is provided with grammatical data, but not with explicitly labeled ungrammatical data. This can pose a challenge to purely error-driven learners when the linguistic theory defines languages with subset relations. If the grammatical forms of one language are a subset of the grammatical forms of another, then the data from the subset language will be fully consistent with the grammars for both languages. To correctly capture the generalizations of the language, the learner must be able to recognize that the grammar generating the more restrictive (subset) language better captures the learning data than the grammar generating the less restrictive (superset) language. This problem, known as the *subset problem*, requires the learner to attend to more than the ability of a grammar to generate the observed forms. The learner must (directly or indirectly) compare the relationships between multiple grammars and the observed data.

The subset problem relates to the Richness of the Base in Optimality Theory. All languages have the same possible inputs, so differences in restrictiveness between languages must be consequences of differences in constraint rankings. If language A is a proper subset of language B, then each input mapped by the ranking of B to an output not contained in A must be mapped by the ranking of A to some other output, an output in A. If the learner had access to all possible input/output pairs, this would not be a problem. But the learner isn't actually provided with input-output pairs in the data (as was assumed above). The learner is provided with grammatical outputs; the inputs must be inferred. Phonological alternations can provide some information about some non-faithful mappings, but not all, and properly analyzing alternations requires interaction with the determination of the ranking.

This problem of determining inputs can be finessed early in learning by assuming that the phonological input for each word matches the surface form. This is only tenable if the learner either ignores or is not yet aware of morphological relationships between the words. Such a stage has been labeled the phonotactic learning stage (Prince & Tesar 2004). The learner can gain partial information about the constraint ranking by looking for the most restrictive ranking that maps each observed form 'to itself'. The most restrictive ranking will map the most inputs corresponding to unobserved forms not to themselves but to other forms.

One proposal for dealing with restrictiveness relations between grammars builds on the observation that active markedness constraints tend to increase restrictiveness, while active faithfulness constraints tend to decrease restrictiveness. This leads to an intuitive expectation that rankings with lots of markedness constraints dominating lots of faithfulness constraints will be more restrictive than rankings showing the opposite pattern. This suggests an approach to the subset problem: given several rankings capable of producing the observed data, choose the one having the largest degree of markedness dominating faithfulness.

The restrictiveness effects of markedness dominating faithfulness are illustrated in (16) and (17). The rankings differ in the location of the faithfulness constraint ID[nas]: it is dominated by all markedness constraints in (16), but dominates two markedness constraints in (17).

(16) Ranking: *–VOI/NAS » ID[voi] » {*VOI, *NAS} » ID[nas]
Mapping: /ta/→[ta] /da/→[da] /n̥a/→[ta] /na/→[da]
Inventory: {[ta], [da]}

(17) Ranking: *–VOI/NAS » {ID[voi], ID[nas]} » {*VOI, *NAS}
Mapping: /ta/→[ta] /da/→[da] /n̥a/→[ta] /na/→[na]
Inventory: {[ta], [da], [na]}

The language in (16) is a subset of the language in (17); both contain [ta] and [da], but (17) also contains [na]. The more restrictive subset language has ID[nas] dominated by more markedness constraints, and maps /na/→[da].

Of course, it may not be entirely obvious just precisely what a 'degree of markedness dominating faithfulness' is, let alone how to compute the ranking with the greatest amount of it. A formalization of this notion is the *r-measure* (Prince & Tesar 2004). The r-measure assesses a constraint ranking by adding up the number of faithfulness constraints dominated by each markedness constraint, thus expressing 'degree of markedness dominating faithfulness' as a number. Rankings can be compared with respect to their r-measures, with higher r-measures suggesting (but not guaranteeing) greater restrictiveness. The ranking in (16) has r-measure 4, while the ranking in (17) has r-measure 2.

Recall that RCD ranks each constraint as high as possible, consistent with the data provided it. With respect to restrictiveness, this is the right thing to do for markedness constraints, but the wrong thing to do for faithfulness constraints. This is rectified by the Biased Constraint Demotion algorithm (BCD). BCD differs from RCD in that it introduces a bias against placing faithfulness constraints into the ranking. When constructing a stratum of a hierarchy, if both markedness and faithfulness constraints are available to be placed into the ranking, BCD will place the markedness constraints into the ranking but not the faithfulness constraints, with the effect that the faithfulness constraints will end up lower in the hierarchy (and thus be dominated by more markedness constraints).

BCD can be combined with MRCD in the same way the RCD can. Such a learner would construct its initial constraint hierarchy by applying BCD to an empty list of winner-loser pairs, like the one in (18).

(18) *An empty list of winner-loser pairs*

input	win ~ lose	*VOI	*NAS	*–VOI/NAS	ID[voi]	ID[nas]

BCD observes that none of the constraints prefer any losers, so all are available. Some of them, however, are markedness constraints. BCD places the markedness constraints into the top stratum, but not the faithfulness constraints. Then, on the next pass, it observes that all remaining constraints are available, but all of them are faithfulness constraints, so they are placed into the next stratum, giving the constraint ranking in (19).

(19) {*–VOI/NAS, *VOI, *NAS} » {ID[voi], ID[nas]}

This hierarchy has all markedness constraints dominating all faithfulness constraints. In this way, BCD derives as a consequence the initial ranking that has been proposed elsewhere (Demuth 1995, Gnanadesikan 1995, Smolensky 1996).

Suppose that the target language is the one given in (16), with inventory {ti, ni}. The learner will observe [ni], assign it the input /ni/, parse /ni/ with the ranking in (19), and generate [ti]. This provokes the construction of the first winner-loser pair, shown in (20).

(20) *First winner-loser pair for /ni/*

input	win ~ lose	*VOI	*NAS	*–VOI/NAS	ID[voi]	ID[nas]
/ni/	ni ~ ti	L	L	e	W	W

BCD now applies, and a complication arises. After *–VOI/NAS is ranked, a faithfulness constraint must be placed into the hierarchy. But which one? Either one will account for the winner-loser pair. If the learner chooses ID[nas], the correct one, they will end up with the correct ranking. But what if the learner mistakenly chooses ID[voi]? This would result in the hierarchy in (21).

(21) *–VOI/NAS » ID[voi] » {*VOI, *NAS} » ID[nas]

The learner at this point will parse /ni/ with the new hierarchy, and generate [di], triggering construction of another winner-loser pair, which is then added to the list as shown in (22).

(22) *The complete winner-loser pair list*

input	win ~ lose	*VOI	*NAS	*–VOI/NAS	ID[voi]	ID[nas]
/ni/	ni ~ ti	L	L	e	W	W
/ni/	ni ~ di	e	L	e	e	W

BCD now applies to this list, and the complication arises again: which faithfulness constraint to rank? This time, there is a basis for choosing. If the learner places ID[voi] into the ranking first, it will eliminate the first pair only, freeing up *VOI. The learner can then place *VOI into the hierarchy next, but must then rank ID[nas] before ranking *NAS. The resulting hierarchy would be (23), with an r-measure of 3.

(23) *–VOI/NAS » ID[voi] » *VOI » ID[nas] » *NAS
Mapping: /ta/→[ta] /da/→[da] /n̥a/→[ta] /na/→[na]
Inventory: {[ta], [da], [na]}

BCD chooses, instead, to maximize the r-measure by choosing the faithfulness constraint that frees up the most markedness constraints. Here that is ID[nas], which accounts for both winner-loser pairs and thus frees up two markedness constraints. The resulting hierarchy, (24), has an r-measure of 4.

(24) *–VOI/NAS » ID[nas] » {*VOI, *NAS} » ID[voi]
Mapping: /ta/→[ta] /da/→[ta] /n̥a/→[na] /na/→[na]
Inventory: {[ta], [na]}

By seeking to maximize the r-measure, the learner succeeds in finding the most restrictive grammar consistent with the observed data, solving the subset problem.

Neither BCD, nor the r-measure itself, are perfect. It is possible to construct cases where the hierarchy with the highest r-measure is not the most restrictive grammar, as well as cases where BCD fails to construct the hierarchy with the highest r-measure. For discussion and examples of such cases, see Prince & Tesar (2004).

24.6 Structural ambiguity

An *overt form* is the phonetically audible portion of an utterance. An *interpretation* of an overt form is a full structural description that is phonetically realized as the overt form. The problem of structural ambiguity arises when more than one interpretation may be assigned to the same overt form. Constraints evaluate entire structural descriptions, so the choice of interpretation for an overt form affects the relationship of that form to the grammar.

I will illustrate this problem with stress data from Polish (Rubach & Booij 1985). The word *spokójny* has three syllables and medial main stress. Assume for present purposes that GEN permits two interpretations of this overt form: a right-aligned trochaic foot, *spo(kójny)*, and a left-aligned iambic foot, *(spokój)ny*. The only way for the learner to choose between these interpretations is to appeal to other data from the language, data which may also be structurally ambiguous. Structural ambiguity is not idiosyncratic to foot structure; the problem arises with other levels of prosodic structure, such as syllable structure, as well as in many aspects of language outside of phonology.

This problem can be approached using an algorithm called the *Inconsistency Detection Learner*, or IDL (Tesar 2004). IDL works by considering different combinations of interpretations of overt forms, and determining which combinations are consistent, that is, which combinations of interpretations can be simultaneously optimal under some ranking. Given sufficient data, the correct combination of interpretations will be the only consistent one.

The illustration continues with a highly idealized linguistic system in which all feet are bisyllabic, and syllables are parsed into feet to the extent allowed by the bisyllabicity restriction. The constraints, which follow the basic pattern of generalized alignment (McCarthy & Prince 1993), are given in (25).

(25) *The Stress System Constraints*
MAINL: the head foot should be aligned with the left edge of the word.
MAINR: the head foot should be aligned with the right edge of the word.
TROCH: a head syllable should be aligned with the left edge of a foot.
IAMB: a head syllable should be aligned with the right edge of a foot.
FEETL: a foot should be aligned with the left edge of the word.
FEETR: a foot should be aligned with the right edge of the word.

Consider the overt forms given in (26) and (27). Each is structurally ambiguous, and the possible interpretations are indicated for each.

(26) OvertA: *spokójny*
Interpretation A1: *spo(kójny)*
Interpretation A2: *(spokój)ny*

(27) OvertB: *sàksofonísta*
Interpretation B1: *(sàkso)fo(nísta)*
Interpretation B2: *(sàkso)(foní)sta*

Suppose IDL processes OvertA first. IDL will observe that OvertA is structurally ambiguous, and separately pursue each interpretation by applying MRCD, building up a separate list of winner-loser pairs for each. The winner-loser pairs for interpretation A1 are shown in the first two rows of (28), while the winner-loser pairs for interpretation A2 are shown in the first two rows of (29).

The learner then processes OvertB, which also has two interpretations. The learner responds to this by creating and testing the four possible combinations of interpretations: A1+B1, A1+B2, A2+B1, and A2+B2. The learner tests a combination like A1+B1 by starting with the winner-loser pairs and the associated ranking for A1, and applying MRCD to B1, adding any new winner-loser pairs to the list for A1. This testing process results in either a hierarchy which makes both A1 and B1 optimal, or an inconsistency, indicating that at least one of the interpretations is wrong.

The correct combination of interpretations, A1+B1, results in a consistent set of winner-loser pairs, defining a ranking, as shown in (28). The incorrect combinations all result in inconsistency. For instance, the winner-loser pairs for A2+B1, shown in (29), are inconsistent, as is apparent from the fact that every constraint prefers at least one loser.

(28) *Winner-loser pairs for interpretations A1+B1*

win ~ lose	MAIN-L	MAIN-R	TROCH	IAMB	FEET-L	FEET-R
spo(kójny) ~ (spókoj)ny	L	W			L	W
spo(kójny) ~ spo(kojný)			W	L		
(sàkso)fo(nísta) ~ sak(sòfo)(nísta)					W	L

(29) *Winner-loser pairs for interpretations A2+B1; they are inconsistent*

win ~ lose	MAIN-L	MAIN-R	TROCH	IAMB	FEET-L	FEET-R
(spokój)ny ~ spo(kojný)	W	L			W	L
(spokój)ny ~ (spókoj)ny			L	W		
(sàkso)fo(nísta) ~ (saksó)(fonì)sta	L	W	W	L	L	W

OvertA and OvertB are *mutually constraining*. OvertA constrains the interpretations of OvertB: interpretation B2 is not consistent with any interpretation of OvertA, so OvertA constrains the interpretation of OvertB to B1. Likewise, OvertB constrains the interpretation of OvertA to A1.

Once IDL has tested the combinations, it discards any inconsistent combinations. The consistent combinations are retained as active hypotheses, just as the hypotheses for interpretations A1 and A2 were retained after processing OvertA above. It then proceeds on to process further data, combining their interpretations with each of the active hypotheses. Once enough data has been processed, IDL will have determined both the correct interpretations of the data and the correct grammar. In this fashion, IDL uses MRCD to simultaneously eliminate incorrect interpretations via inconsistency detection and construct a ranking based on the correct interpretations.

Given data that are all consistent with a single grammar, IDL is guaranteed to find the correct combination of interpretations and a grammar consistent with them. The efficiency of IDL depends upon the linguistic system. One danger is forms with a very high degree of structural ambiguity: if the learner needs to learn from the form, it will have to separately consider each of the possible interpretations of the form. A second danger is growth in the combinations of interpretations, which can happen if forms are only weakly mutually constraining. If the learner processes three forms, each two-ways ambiguous, and the forms are not by themselves constraining enough to eliminate any of the combinations of interpretations, the learner will have 8 active hypotheses as a result. The number of combinations of interpretations, unchecked by mutual constraint, grows exponentially in the number of ambiguous overt forms.

IDL will require less work when overt forms have low degrees of ambiguity, and when overt forms are strongly mutually constraining. IDL can benefit from a bias towards processing forms with a low degree of ambiguity earlier. This is because the ranking information obtained from low ambiguity forms can collectively constrain a form with greater ambiguity, eliminating most or all of the latter form's incorrect interpretations.

24.7 Learning underlying forms

Learning the content of underlying forms is less well understood than the other issues addressed in this chapter, and it is the subject of current research. The problem is challenging in part because of the strong mutual dependence between the lexicon and the constraint ranking. In general, an underlying form for a morpheme cannot be confidently chosen independently of knowledge about the ranking. At the same time, all of the algorithms for constructing rankings presented in this chapter require underlying forms in order to function. The two need to be learned in tandem.

BCD, for phonotactic learning, can make progress by assuming underlying forms identical to surface forms, but that is not sufficient to determine the entire ranking. In fact, it is possible to have multiple languages with identical surface forms but different mappings: the underlying forms not identical to grammatical surface forms are mapped to different surface forms in different languages, and the different mappings are the result of different constraint rankings. Only information from morphemic alternations can distinguish from among such languages.

Two ideas for learning underlying forms have been proposed that build on the work already presented in this chapter. One proposal is to use the ranking that results from phonotactic learning to test different hypothesized underlying forms (Pater 2000, Tesar & Prince to appear). This could involve using BCD during the phonotactic learning stage to construct a phonotactic ranking. Then, once the learner is able to segment the words into constituent morphemes, the learner could test different possible underlying forms for the morphemes using the phonotactic ranking, keeping those underlying forms that surface correctly in all attested contexts. When none of the possible underlying forms surfaces correctly in all contexts, it indicates to the learner a shortcoming of the phonotactic ranking, and the learner can explore further changes to ranking, testing them on that morpheme.

Another proposal is to use inconsistency detection to help choose underlying forms (Tesar et al. 2003).[2] The surgery learning algorithm uses MRCD combined with BCD as described in section 5. The algorithm starts with an initial lexicon constructed by comparing the surface realizations of each morpheme to determine which features alternate (have different values in different contexts) and which features do not. The non-alternating features are set underlyingly to match their (unchanging) surface value, while the alternating features are initially set to a default unmarked value. The algorithm accumulates winner-loser pairs until an inconsistency is reached. This is the signal to the learner that something must change in their lexical hypothesis, because no change to the ranking alone can solve the problem. The learner then tries different changes to the lexicon, altering 'surgically' the winner-loser pairs that include the morpheme with the underlying form being altered. If a lexical change is found that resolves the inconsistency, the learner keeps the change to the lexicon.

[2] An idea of a similar spirit was proposed by Kager (1999). His proposal used a learning algorithm that could not definitively detect inconsistencies in the way RCD can, and instead looked for patterns in the behavior of the learning algorithm to suggest inconsistency.

24.8 Discussion

The learning challenges discussed here arise from the role of non-overt elements (both structural and lexical) in linguistic analyses, the need to relate the different and possibly non-identical surface realizations of a morpheme, and the reliance on positive evidence in learning. Non-overt elements pose a challenge when linguistic theory allows ambiguity, so that different languages assign different interpretations to the same overt forms. This is because of mutual dependence. The constraint ranking and the non-overt elements are mutually dependent, and neither is known in advance by the learner, so they must be learned together. In order to overcome that mutual dependence, the learner must relate different forms of the language to each other, via the grammar. Structural ambiguities must be addressed by relating the full structural descriptions of different forms to each other; underlying forms must be learned by relating the surface realizations of morphemes in different contexts to each other.

The proposed approaches to these challenges draw on the structure of linguistic theory in several ways. Structuring the space of possible grammars via possible rankings of violable constraints leads to an efficient algorithm for constructing rankings from structural descriptions, MRCD. The fact that MRCD detects inconsistency with equal efficiency makes it plausible to approach the problems of structural ambiguity and underlying forms with an inconsistency detection strategy. The fundamental organization of constraints into markedness constraints and faithfulness constraints provides the basis for BCD's approach to dealing with the lack of reliable negative evidence. The learner can be biased towards grammars with more markedness constraints dominating more faithfulness constraints.

Many learning challenges remain. The GLA is robust in the face of errors, but lacks the capacity for inconsistency detection, while BCD is capable of detecting inconsistency but lacks robustness. Clearly, it would be an accomplishment to capture both in a single approach. There is a great deal that is not yet understood about how underlying forms can be efficiently learned. And there are challenges beyond those discussed here, such as the matter of morpheme discovery itself, which has mutual dependencies with phonological learning: you need to know the morphological structure of words in order to identify morphemes that alternate across contexts and from that learn the phonology, but you need some grasp on the phonology in order to identify non-identical surface strings across words that plausibly represent the same morpheme. It remains to be seen if the strategies already developed for learning can be extended to these larger problems.

Further Reading

1.1 Constraint Demotion and Error-Driven Learning

(Riggle, 2004, Tesar and Smolensky, 1998)

1.2 Gradual Learning Algorithm

(Boersma, 1998, Boersma and Hayes, 2001)

1.3 Learning Restrictive Grammars

(Hayes, 2004, Prince and Tesar, 2004, Smolensky, 1996)

1.4 Structural Ambiguity

(Dresher, 1999, Eisner, 2000, Tesar and Smolensky, 2000, Tesar, 2002, Tesar, 2004)

1.5 Learning Underlying Forms

(Kager, 1999, Pulleyblank and Turkel, 2000, Tesar et al., 2003)

References

Boersma, Paul. 1998. *Functional Phonology*. The Hague: Holland Academic Graphics.

Boersma, Paul, and Hayes, Bruce. 2001. Empirical tests of the Gradual Learning Algorithm. *Linguistic Inquiry* 32:45-86.

Demuth, Katherine. 1995. Markedness and the development of prosodic structure. In *Proceedings of the North East Linguistics Society 25*, ed. by Jill Beckman, 13-25. Amherst, MA: GLSA, University of Massachusetts. ROA-50.

Dresher, B. Elan. 1999. Charting the learning path: Cues to parameter setting. *Linguistic Inquiry* 30:27-67.

Eisner, Jason. 2000. Easy and hard constraint ranking in Optimality Theory: Algorithms and complexity. In *Finite-State Phonology: Proceedings of the 5th Workshop of the ACL Special Interest Group in Computational Phonology*, ed. by Jason Eisner, Lauri Karttunen, and Alain Theriault, 22-33.

Gibson, Edward, and Wexler, Ken. 1994. Triggers. *Linguistic Inquiry* 25:407-454.

Gnanadesikan, Amalia. 1995. Markedness and faithfulness constraints in child phonology. Ms., University of Massachusetts, Amherst, MA. ROA-67.

Hayes, Bruce. 2004. Phonological acquisition in Optimality Theory: The early stages. In *Constraints in Phonological Acquisition*, ed. by René Kager, Joe Pater, and Wim Zonneveld, 158-203. Cambridge: Cambridge University Press.

Kager, Rene. 1999. *Optimality Theory*. Cambridge: Cambridge University Press.

Lombardi, Linda. 1999. Positional Faithfulness and Voicing Assimilation in Optimality Theory. *Natural Language & Linguistic Theory* 17:267-302.

McCarthy, John, and Prince, Alan. 1993. Generalized alignment. In *Yearbook of Morphology*, ed. by Geert Booij and Jaap Van Marle, 79-154. Dordrecht: Kluwer.

McCarthy, John, and Prince, Alan. 1995. Faithfulness and Reduplicative Identity. In *University of Massachusetts Occasional Papers 18: Papers in Optimality Theory*, ed. by Jill Beckman, Laura Walsh Dickey, and Suzanne Urbancyzk, 249-384. Amherst, MA: GLSA, University of Massachusetts.

Pater, Joe. 2000. Unpublished course handout from Ling 751, University of Massachusetts, Amherst.

Prince, Alan, and Smolensky, Paul. 1993. Optimality Theory: Constraint interaction in generative grammar. Ms., Rutgers University, New Brunswick, and the University of Colorado, Boulder. ROA-537.

Prince, Alan. 2002. Arguing Optimality. Ms., Linguistics Dept., Rutgers University. ROA-562.

Prince, Alan, and Tesar, Bruce. 2004. Learning phonotactic distributions. In *Constraints in Phonological Acquisition*, ed. by René Kager, Joe Pater, and Wim Zonneveld, 245-291. Cambridge: Cambridge University Press.

Pulleyblank, Douglas, and Turkel, William J. 2000. Learning phonology: Genetic algorithms and Yoruba tongue-root harmony. In *Optimality Theory: Phonology, Syntax, and Acquisition*, ed. by Joost Dekkers, Frank van der Leeuw, and Jeroen van de Weijer, 554-591. Oxford: Oxford University Press.

Riggle, Jason. 2004. Contenders and learning. In *The Proceedings of the 23rd West Coast Conference on Formal Linguistics*, ed. by Benjamin Schmeiser, Vineeta Chand, Ann Kelleher, and Angelo Rodriguez, 101-114. Somerville, MA: Cascadilla Press.

Rubach, Jerzy, and Booij, Geert E. 1985. A grid theory of stress in Polish. *Lingua* 66:281-319.

Sakas, William, and Fodor, Janet Dean. 2001. The Structural Triggers Learner. In *Language Acquisition and Learnability*, ed. by Stefano Bertolo, 172-233. Cambridge: Cambridge University Press.

Smolensky, Paul. 1996. The initial state and "richness of the base" in Optimality Theory. Technical Report JHU-CogSci-96-4, The Johns Hopkins University, Baltimore, MD. ROA-154.

Tesar, Bruce. 1995. Computational Optimality Theory. PhD. dissertation, University of Colorado, Boulder, CO.

Tesar, Bruce. 1997. Using the mutual inconsistency of structural descriptions to overcome ambiguity in language learning. In *Proceedings of the North East Linguistic Society 28*, ed. by Pius N. Tamanji and Kiyomi Kusumoto, 469-483. Amherst, MA: GLSA, University of Massachusetts.

Tesar, Bruce. 1998. Error-driven learning in Optimality Theory via the efficient computation of optimal forms. In *Is the Best Good Enough? Optimality and Competition in Syntax*, ed. by Pilar Barbosa, Danny Fox, Paul Hagstrom, Martha Jo McGinnis, and David Pesetsky, 421-435. Cambridge, MA: MIT Press.

Tesar, Bruce, and Smolensky, Paul. 1998. Learnability in Optimality Theory. *Linguistic Inquiry* 29:229-268.

Tesar, Bruce, and Smolensky, Paul. 2000. *Learnability in Optimality Theory*. Cambridge, MA: MIT Press.

Tesar, Bruce. 2002. Enforcing grammatical restrictiveness can help resolve structural ambiguity. In *Proceedings of the Twenty-First West Coast Conference on Formal Linguistics*, ed. by Line Mikkelsen and Christopher Potts, 443-456. Somerville, MA: Cascadilla Press. ROA-618.

Tesar, Bruce, Alderete, John, Horwood, Graham, Merchant, Nazarré, Nishitani, Koichi, and Prince, Alan. 2003. Surgery in language learning. In *Proceedings of the*

Twenty-Second West Coast Conference on Formal Linguistics, ed. by G. Garding and M. Tsujimura, 477-490. Somerville, MA: Cascadilla Press. ROA-619.

Tesar, Bruce. 2004. Using inconsistency detection to overcome structural ambiguity [Spring]. *Linguistic Inquiry* 35:219-253.

Tesar, Bruce, and Prince, Alan. to appear. Using phonotactics to learn phonological alternations. In *Proceedings of the Thirty-Ninth Conference of the Chicago Linguistics Society, vol. II: The Panels*, ed. by Johnathon E. Cihlar, Amy L. Franklin, David W. Kaiser, and Irene Kimbara. ROA-620.