Prologue

Optimality Theory (OT) was first described in depth by its creators, Alan Prince and Paul Smolensky, in a course presented at the University of California, Santa Cruz, in 1991 (Prince and Smolensky 1991). The first detailed exposition of the theory appears in Prince and Smolensky (1993). Since 1993, there has been a great deal of interest in this emerging theory; it has been the subject of a large and growing literature, an extensive electronic archive (http://roa.rutgers.edu), many courses and conference papers, and several textbooks. Although it was originally applied to phonology, the relevance of OT to topics in morphology, syntax, sociolinguistics, psycholinguistics, and semantics has become increasingly apparent.

One of the most compelling features of OT, in my view, is the way that it unites description of individual languages with explanation of language typology. As a phonologist, I have always been impressed and sometimes overwhelmed by how the complexity and idiosyncrasy of each language's phonology is juxtaposed with the clarity and abundance of solid typological generalizations. Even though this is arguably the central research problem of phonology and of linguistic theory in general, progress in consolidating description and explanation has at best been halting and occasionally retrograde.

OT, though, is inherently typological: the grammar of one language inevitably incorporates claims about the grammars of all languages. This joining of the individual and the universal, which OT accomplishes through ranking permutation, is probably the most important insight of the theory. It comes up again and again throughout this book, as a core premise of the theory, as a discipline for practitioners, and as the source of many empirical results in phonology, syntax, and allied fields.

1

The Core of Optimality Theory

This chapter introduces the central premises of Optimality Theory. The chapter begins (§1.1) with the overall structure of OT, as proposed by Prince and Smolensky (1993).🕮§1.5¶1 It continues with some general remarks about the nature of constraints (§1.2) and their modes of interaction through ranking (§1.3). These threads are joined to some practical suggestions for doing OT in §1.4. Readers encountering OT for the first time are advised not to read this chapter straight through; see "How to Use This Book" for a better plan of attack.

1.1 Basic Architecture

1.1.1 Candidate Comparison

Many theories of language can best be described as operational, rule based, or transformational: they take an input and apply some procedure that changes it into an output. But the primary action in OT is *comparative*: the actual output is the optimal member of a set of *candidate* output forms. Interesting analytic and theoretical results in OT come from understanding the details of how candidates are compared.

Candidates are compared by applying a hierarchy of violable constraints. The constraints assess the form of a candidate and its relationship to the input. Candidates inevitably differ in performance on various constraints. Of two candidates, the more *harmonic* is the one that performs better on the highest-ranking constraint that distinguishes between them. The actual output – the most harmonic or *optimal* candidate – is the one that is more harmonic in all its pairwise competitions with other candidates.[1]

Because constraints are violable, the output typically disobeys at least some of the lower-ranking constraints. To draw an analogy from ethics, optimality is more like moral relativism or the Three Laws of Robotics[2] than the Ten Commandments; it is about being the best among a choice of options, not about

being objectively perfect. In the simplest situation, two candidates are under evaluation by a single constraint C. The optimal candidate is the one that incurs fewer violations of C. When there is more than one constraint, the ranking is strictly respected in comparing candidates; there is no global assessment of candidates based on their performance on the whole constraint gestalt. In fact, the optimal candidate may actually perform worse than its competitor on some constraint(s) ranked below the decisive one. So, if constraint C1 is ranked above C2 and C3 (that is, C1 *dominates* C2 and C3), then the output may perform worse than its competitor on both C2 and C3, as long as it performs better on C1. To cite an example from Prince and Smolensky (1993), "azzzzz" is alphabetized before "baaaaa" because alphabetical order is based on the leftmost distinguishing letter, regardless of how much the letters farther to the right might seem to encourage a different order.

This property, which Prince and Smolensky dub the *strictness of strict domination*, is somewhat counterintuitive, since it is quite unlike the more flexible system of priorities we apply in our everyday lives. For example, given a primary career goal of making lots of money and a secondary goal of living in an exciting city, few among us would stubbornly persist with these priorities when faced with offers of a job paying $61,000 in Paris, Texas, and a job paying $60,000 in Paris, France. Yet constraint ranking in OT has exactly that stubborn persistence. (Strict domination is the main difference between OT and connectionist models. See §2.4.)

Candidate comparison is often shown in a *tableau*, where an optimal candidate is compared with one or more of its competitors with respect to their performance on two or more constraints. A tableau therefore gives a perspicuous view of some of the constraints and rankings that are crucial in selecting a candidate as optimal. As in (1), constraints are given in domination order from left to right, and the rows contain the different candidates, one of which is optimal. The individual cells show the violation-marks (*) incurred by each candidate relative to each constraint. The optimal candidate is called out by the pointing hand.[3]

(1) A ranking argument

	C1	C2
a. ☞ $Cand_{Opt}$		*
b. $Cand_{Comp}$	*	

Readers wanting to see real tableaux now can take a look at §1.3, and in §1.4.1 I discuss the practical aspects of ranking constraints, introducing another tableau format that is particularly useful for discovering rankings.

In (1), C1 and C2 conflict in their evaluation of two candidates. C1 prefers $Cand_{Opt}$, but C2 prefers the competitor $Cand_{Comp}$. Since $Cand_{Opt}$ is the observed

output form, the conflict is resolved by ranking C1 above C2. A situation like this is a necessary condition for a valid *ranking argument*, a kind of proof that C1 dominates C2 in the hierarchy (written ⟦C1 ≫ C2⟧). To ensure sufficient conditions for the validity of a ranking argument, it is also necessary to check that there is no constraint C3 with both of the following properties: C3 is ranked above C2, and C3 concurs with C1 by preferring Cand_{Opt}. In that situation, C3 invalidates the argument for ⟦C1 ≫ C2⟧ because C3 can also produce the effect of the ranking being argued for.[4]

Conflict is not the only possible relation between two constraints, but it is the only relation that can serve as the basis for a valid ranking argument. In the situations shown in (2a–c), there is no conflict between the constraints and so there is no basis for ranking them.

(2) a. C1 and C2 agree

	C1	C2
i. ☞ Cand_{Opt}		
ii. $\text{Cand}_{\text{Comp}}$	*	*

b. C1 does not distinguish the candidates (both obey it)

	C1	C2
i. ☞ Cand_{Opt}		
ii. $\text{Cand}_{\text{Comp}}$		*

c. C1 does not distinguish the candidates (both violate it)

	C1	C2
i. ☞ Cand_{Opt}	*	
ii. $\text{Cand}_{\text{Comp}}$	*	*

These tableaux separate the constraint columns with a dotted line to show that neither constraint provably dominates the other. These tableaux will not support a ranking argument because C1 and C2 concur in eliminating $\text{Cand}_{\text{Comp}}$ (2a) or one of them assesses both candidates as equally good or bad (2b–c).

A constraint may assign more than one violation-mark to a candidate in one of two situations: either the constraint is violated at several different spots in the candidate under evaluation (e.g., the constraint assesses some aspect of syllable form, and a polysyllabic word contains several offenders, as in (14d)), or the constraint is violated gradiently, distinguishing noncompliant candidates by extent of violation (as is the case with edge Alignment constraints (§1.2.3)). As

OT is presently understood, multiple violations from either source are usually treated the same; they are just lumped together in the pile of violation-marks assigned to a candidate.

Candidate comparison is no different when there are multiple violations, and there is no need to count violation-marks, since better or worse performance is all that matters. Driving this point home, Prince and Smolensky introduce the *method of mark cancellation*.📖§1.5¶2 If and only if a tableau compares exactly two candidates, violation-marks that the two candidates share can be ignored or *canceled*, since those violation-marks contribute nothing to that particular comparison. For example, both candidates in (2c) share a violation-mark in the C1 column. These shared marks can be canceled, reducing (2c) to (2b). By reducing (2c) in this way, we can readily see that C1 contributes nothing to selecting the optimal candidate. Though this example involves single violations, mark cancellation is also useful when candidates incur multiple violations: if one candidate has three violation-marks from some constraint and another candidate has five, mark cancellation reduces this to zero and two, respectively. Comparison, rather than counting, is what matters.

Mark cancellation cannot be meaningfully applied to tableaux with more than two candidates since its purpose is to bring out the better and the worse in a pairwise comparison. With several candidates in play, it is better to use the comparative tableau format described in §1.4.1.

1.1.2 Ranked Constraints and EVAL

Winning isn't everything. It's the only thing.
– Attributed to Vince Lombardi

The grammar of a language is a specific constraint ranking. Language-particular ranking is the most important and perhaps only method in OT for explaining how and why languages differ from one another (§3.1.5). The ranking in a particular language is, in theory, a total ordering of a set of universal constraints.

In practice, though, it is not usually possible to discover a total ordering, and so the analyst must be satisfied with a partial ordering. There are just two legitimate ways of showing that C1 dominates C2: by a valid direct ranking argument like (1) or by a legitimate inference from valid direct ranking arguments. An example of the latter is a ranking argument based on transitivity of constraint domination, such as showing that C1 dominates C3 by establishing that C1 dominates C2 and that C2 dominates C3.[5] When direct and inferred arguments for ranking are both present, they have to agree. Otherwise the analysis or the theory is just plain wrong. But when there is no evidence or inference available for ranking certain constraints, it is good analytic practice to report a partial order, as in (11) in §1.3.2. Partial ordering in the absence of constraint conflict is not the same thing as deliberate *ties* between conflicting constraints.

Tied rankings are a proposed extension of standard OT to account for within-language variation (§4.5).

Under the assumption that all constraints are universal (§1.2.1), the ranking is all that the learner must discover, and learning some workable ranking turns to be a surprisingly easy task (§4.2.1). The analyst's job is much harder than the learner's: ranking arguments need to be discovered and their validity checked in a context where all hypotheses about universal constraints are necessarily tentative and mutable. Still, there are some useful heuristics to follow when positing or assessing proposed constraints (§1.4.4).

Suppose H is the constraint hierarchy for some language. To use H to select the most harmonic member of some candidate set, OT calls on the function EVAL, which gives meaning to the domination relation "≫," generalizing pairwise comparison to larger (possibly infinite) sets of candidates. The function EVAL returns the candidate set as a partial order, with its most harmonic member, the actual output form, standing at the top.📖§1.5¶1

In theory, there is no guarantee that EVAL will always return a single most harmonic member of the candidate set. Suppose two candidates incur identical violation-marks from all constraints. EVAL will be unable to decide between them, and if no other candidate is more harmonic, both will be optimal. In this case, within-language variation ought to be observed. In practice, though, this possibility might not be easy to realize; the universal constraint set is rich enough that EVAL usually returns a unique winner for any real-life H applied to any real-life candidate set. For this reason, within-language variation has usually been analyzed in other ways (§4.1.3, §4.5).[6]

Although EVAL imposes a harmonic ordering on all the candidates, the standard approach assigns no interpretation to the details of the ordering below the topmost candidate. Suppose EVAL returns the harmonic ordering $[\![\text{Cand}_{\text{Opt}} \succ \text{Cand}_{\text{Comp1}} \succ \text{Cand}_{\text{Comp2}}]\!]$, where $\succ$ denotes the relation "is more harmonic than." From this, we know that Cand_{Opt} is the actual output form, but nothing can be concluded from the relative harmony of $\text{Cand}_{\text{Comp1}}$ and $\text{Cand}_{\text{Comp2}}$ – only the optimum is given a linguistic interpretation. This is an important methodological point: valid ranking arguments like (1) must always involve an actual output form as one of the candidates being compared.

Samek-Lodovici and Prince (1999: 18) have a particularly clear and insightful way of describing EVAL.[7] Think of a constraint as a function from sets of candidates to sets of candidates. Each constraint takes a set of candidates and returns the subset consisting of those candidates that perform best on that constraint. EVAL can then be understood in terms of *function composition*: a lower-ranking constraint takes as input the set of best performers on the higher-ranking constraint. For instance, if the set of candidates {Cands} and the hierarchy $[\![\text{C1} \gg \text{C2}]\!]$ are handed to EVAL, then the set of winners will be given by $(\text{C2} \circ \text{C1})(\{\text{Cands}\})$ or equivalently $\text{C2}(\text{C1}(\{\text{Cands}\}))$. Since a constraint can never return less than one best performer, this formalization of EVAL correctly

guarantees at least one winner. It also allows for the theoretical possibility of more than one winner when the outermost constraint returns a set containing two or more candidates. This formalization conforms rather well to the usual intuitive sense of how EVAL works: first it applies the highest-ranking (or innermost) constraint, then the next highest, and then the next, downward through the hierarchy (or outward through the composed functions) until there are no constraints left.

1.1.3 GEN

Thus far I have described two of the main components of OT, the language-particular constraint hierarchy H and the universal function EVAL, which applies H to a set of candidates. There are two others: a putatively universal set of constraints CON, discussed in §1.2, and the universal candidate generator GEN. The latter has two closely related functions: it constructs candidate output forms, such as words or sentences, and it specifies a relation between the candidate output forms and the input. Though details of the internal structure of GEN are still under development, the general principles underlying the theory of GEN are clear.§1.5¶1

GEN is universal, meaning that the candidate forms emitted by GEN for a given input are the same in every language. These candidates are also very diverse. This property of GEN has been called *inclusivity* or *freedom of analysis*. Precisely because GEN is universal, it must at a minimum supply candidates varied enough to fit all of the ways in which languages can differ. For example, languages disagree in how they syllabify a consonant cluster like *br* (cf. English *alge.bra* vs. Arabic *jab.rī* 'algebraic'), so GEN will offer competing candidates that differ along this dimension, leaving the choice of the right one to the language-particular rankings in H. This freedom is limited only by primitive structural principles essential in every language, perhaps restricting GEN to a specific alphabet of distinctive features (in phonology) or to some version of X-bar theory (in syntax). Beyond this, the details of GEN are a matter for empirical investigation in the context of specific hypotheses about the nature of the input and the constraints. In phonology, there is a rough consensus about the properties of GEN (§1.1.3), but in syntax it is still more of an open question (§4.1).

Since GEN is the same in every language, it initially seems like a good place to deposit a wide variety of "hard" universals, beyond the bare structural principles just mentioned. For example, no known language syllabifies intervocalic *br* as **algebr.a*, so why not incorporate this observation into the statement of GEN? This strategy is a natural continuation of several decades of linguistic theorizing that has sought to document various universal constraints and refine the statement of them. There is a flaw here, though. Hardwiring universals into GEN is inevitably a matter of brute-force stipulation, with no hope of explanation or connection to other matters – it is the end of discussion rather than the begin-

ning. The right way to look at most universals in OT is in terms of the core idea of the theory: constraint interaction through ranking. By deriving universals and typology from constraint interaction, we ensure that there are connections between the universal properties of language and between-language variation, since both follow from the properties of constraint ranking. Interesting universals and successful explanations for them can indeed be obtained from constraint interaction (§3.1.5).[8]

GEN is also *input dependent*. The candidates emitted by GEN bear a determinate relation to some sort of input form, which might be a phonological underlying representation, a syntactic D-structure, or a morphosyntactic feature specification. The candidates record, by some means, how they differ from the input. This record is used by constraints that evaluate candidates for their faithfulness to the input (§1.2.2). Various implementations of this basic idea can be imagined and have been explored: candidates distinguish derived properties structurally, as in trace theory (Chomsky 1973); each candidate brings with it a function describing how it differs from the input; or each candidate brings with it a description of the operations that produced it from the input. Except for the need to maintain this record in some form, the theory of GEN, and of OT generally, has no special representational or operational commitments.

If GEN incorporates any recursive or iterative operations, as it surely must, then there is no bound on the size of a candidate and every candidate set, from every input, is infinite. This is perhaps not too surprising in syntax, where the infinity of sentences has long been accepted, but it is also true in phonology. Epenthesis is an iterative procedure of candidate-generation, so the set of candidates derived from input /ba/ must include *bati*, *batiti*, *batititi*, . . . No GEN-imposed bound on the number of epenthesis operations is appropriate. Rather, the economy of epenthesis should and does follow from constraint interaction (§3.2.3).

In this context, it is appropriate to point out that OT shares with the rest of generative grammar a commitment to *well-definition* but not to *efficient computation*. Here is how Chomsky has characterized that distinction:

> To avoid what has been a continuing misunderstanding, it is perhaps worth while to reiterate that a generative grammar is not a model for a speaker or a hearer. . . . When we say that a sentence has a certain derivation with respect to a particular generative grammar, we say nothing about how the speaker or hearer might proceed, in some practical or efficient way, to construct such a derivation. (Chomsky 1965: 9)
>
> [A]lthough we may describe the grammar *G* as a system of processes and rules that apply in a certain order to relate sound and meaning, we are not entitled to take this as a description of the successive acts of a performance model such as *PM* – in fact, it would be quite absurd to do so. (Chomsky 1968: 117)

> Recall that the ordering of operations is abstract, expressing postulated properties of the language faculty of the brain, with no temporal interpretation implied. (Chomsky 1995: 380 n. 3)

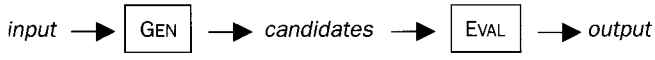
In short, a grammar is a function from some kind of input to some kind of output. A grammar is not an algorithm for computing that function nor is it a description of how speakers actually go about computing that function. Chomsky (1968: 117) sums up with "If these simple distinctions are overlooked, great confusion must result."

That confusion has sometimes led to skepticism about OT: how can EVAL sort an infinite set of candidates in finite time (cf. Bromberger and Halle 1997)? The error lies in asking how long EVAL takes to execute. It is entirely appropriate to ask whether EVAL, like Chomsky's *G*, is well defined, captures linguistically significant generalizations, and so on. But questions about execution time or other aspects of (neural) computation are properly part of the performance model *PM* and must be addressed as such. And, not too surprisingly, there are computational models for OT that do not require infinite time to execute (see §4.3 and the references in §4.6 ¶11).

1.1.4 Summary, with Possible Variations

The core universal elements of the OT architecture are summarized in (3).

(3) Basic OT architecture



GEN receives an input and emits a set of candidates that, in some precise way, depend upon the input. (There are also important things to say about the input itself – see §3.1.2.4.) EVAL applies the language-particular constraint hierarchy H to this candidate set, locating its most harmonic member. The most harmonic candidate is the output; it may be a phonological surface form, a syntactic S-Structure, or some other linguistic object.

The model in (3) is the simplest architecture compatible with OT's basic assumptions. It maximally exploits OT's capacity for *global*, *parallel* evaluation (§3.3). The output of an entire linguistic component, such as the phonology, is obtained from the input in a single pass through GEN and EVAL, which means that the candidates offered by GEN may show the effects of several notionally distinct processes simultaneously. The constraints applied by EVAL then rank these candidates for their global fitness, evaluating the effects of all of those processes in parallel. To see why it is described as global and parallel, compare this model to a theory like standard generative phonology (§2.1), where each rule applies in serial order and in isolation from all other rules coexisting in the grammar.

Some variations on this basic architecture reduce or eliminate the effects of global, parallel evaluation. Suppose that the output in (3) becomes the input for

another pass through GEN, yielding a new set of candidates for evaluation. The most familiar version of this approach imposes a kind of modular or componential structure, treating the whole grammar of a language as a composite entity, as in Lexical Phonology or various instantiations of the Principles and Parameters (P&P) approach. Each module has its own distinct constraint hierarchy H_i and perhaps even its own set of universal constraints CON. The output of the final module in the series is the observed surface form of the language (§3.3.3.4).

Another version of this approach is called *harmonic serialism*. It applies the same constraint hierarchy at each pass through EVAL, continuing until there is convergence, when the output of one pass is identical to the output of the immediately preceding pass. Harmonic serialism unpacks some of the effects of globality and parallelism by imposing restrictions on GEN's freedom of analysis. See §3.3.2.8 and §3.3.3.2 for further discussion.

Refinements or extensions like these still have the essential elements of OT: EVAL-mediated comparison of candidates by a hierarchy of violable constraints. No matter how the details are executed or in what overall context it is embedded, any model with these indispensable characteristics will express the central claim and insight of OT.

1.2 The Theory of Constraints

1.2.1 The Universality of Constraints

Apart from the bare structural primitives embedded in GEN, all constraints in OT are in principle and in fact violable. This statement follows from the basic architecture of the theory: constraints have nowhere else to reside except in the language-particular hierarchy H, which means that any constraint could, in some language, be ranked below another constraint that compels it to be violated.

The null hypothesis is that all constraints are universal and universally present in the grammars of all languages (Prince and Smolensky 1993), and so UG incorporates a constraint component CON. What makes this the null hypothesis is a kind of Occamite reasoning: since language-particular ranking is in general able to account for languages where a putatively universal constraint does not hold true, it does not seem necessary to recognize a special class of language-particular constraints. (See §1.2.3 for some possible qualifications.) Differences between languages are no barrier to constraint universality when constraints are violable.

Constraint violability is a very different thing from parametrization. A parameter describes a requirement that is either reliably enforced or completely ignored: syllables must have onsets (yes/no); heads must precede/follow their complements. A constraint, no matter where it is ranked, always asserts its preference: ONSET is violated by any syllable that lacks an onset in any language,