Generating Contenders

Jason Riggle, University of Chicago

August 24, 2009

Abstract

In Optimality Theory, a *contender* is a candidate that is optimal under some ranking of the constraints. When the candidate generating function *Gen* and all of the constraints are *rational* (i.e., representable with (weighted) finite state automata) it is possible to generate the entire set of contenders for a given input form in much the same way that optima for a single ranking are generated. This paper gives a brief introduction to rational constraints and provides an algorithm for generating contenders whose complexity, modulo the number of contenders generated, is linear in the length of the underlying form with a multiplicative constant representing the size of the finite-state representation of the constraint set.

1 Introduction

In Optimality Theory (OT; Prince and Smolensky 1993/2004), a candidate that cannot be optimal under any ranking of the constraints is said to be **harmonically bounded**.[1] Riggle (2004) dubs the candidates that *can* win under some constraint ranking **contenders** and provides an algorithm for generating the entire set of contenders for a given input form that is applicable to OT-models in which the candidate generating function *Gen* and all of the constraints are **rational** (i.e., representable with (weighted) finite state automata).

This paper provides a brief introduction to rational constraints in §2 and then, in §3, presents algorithms for generating contenders and constructing OT typologies in cases where constraints are rational. These algorithms are more efficient than those presented in Riggle (2004) and are accompanied by explicit complexity analysis in contrast to the algorithm in Riggle (2004) which was only shown to be guaranteed of termination.

1.1 Optimization and Complexity

In his seminal finite-state characterization of the generation problem in OT, Ellison (1994a) showed that a relatively standard dynamic programming approach to optimization could be used to efficiently compute optimal input-output mappings. In Ellison's characterization of the problem, the constraints and ranking are held out as fixed parameters and the input to the problem consists of the underlying form. Ellison showed that, in this characterization, optimization requires on the order of $n|E|\log n|Q|$ computational steps where n is the length of the underlying form and $(|E|, |Q|)$ is a constant denoting the number of arcs and states in the finite state representation of the constraints in Con. Optimization is considered to be efficient in this case because the complexity is log-linear in n.

[1] See also, Samek-Lodovici (1992), Samek-Lodovici & Prince (1999), and Samek-Lodovici & Prince (2002).

Responding to Ellison's results, Eisner (1997a) argues that there are instances such as learning where the ranking is not known in advance and thus cannot be a fixed parameter of the problem. Eisner goes on to demonstrate that, when the constraint set and ranking are *not* fixed parameters of the problem, optimization can be NP-hard because the size of $|E|$ and $|Q|$ can grow exponentially with the number of constraints. The distinction between Ellison's and Eisner's characterization of the generation problem in OT corresponds to what Barton et al. (1987) define as the distinction between the generation problem and the *universal* generation problem.

The distinction between whether $(|E|, |Q|)$ is a fixed parameter of the problem or part of the input to the problem is critical from the perspective of computational complexity theory because constants do not figure into the assessment of the complexity of a problem. Though it might seem counterintuitive to ignore potentially large constants in assessing complexity, they are considered irrelevant from the limiting-case perspective of complexity theory because they will be dwarfed by the value of non-fixed parameters like input length n in all but finitely many cases. (See Papadimitriou (1994) for a review of the fundamentals of computational complexity theory.) In addition to taking issue with the usefulness of the assumption that the constraint set is fixed and the ranking known in advance, Eisner considers the universal generation problem to be more practically relevant for linguists on the grounds that large constants can matter in real-world applications outside the limiting-case perspective of complexity theory.[2]

Heinz et al. (2009) respond to Eisner's complexity results by proposing a slightly different characterization of the generation problem in OT as what they call a 'quasi-universal' problem. In the quasi-universal characterization of the generation problem, the constraint set is held out as a fixed parameter and the input to the generation task consists of a ranking $\mathcal{R}$ and an underlying form. Heinz et al. (2009) argue that this characterization of the problem more accurately reflects the usual definition of OT with a fixed universal constraint set and that this characterization is appropriate for the problem of learning an unknown ranking for a *known* set of constraints.

Adopting the quasi-universal characterization of optimization proposed by Heinz et al. (2009), Riggle (2009) provides an optimization algorithm that requires on the order of $n(|E| \log |Q|)$ computation steps for underlying forms of length n. This tightens Ellison's (1994a) complexity result by a logarithmic factor establishing that complexity of quasi-universal optimization is *linear* in the length of the underlying form rather than log-linear. Riggle argues that this result makes the quasi-universal characterization especially appealing because it isolates the complexity of the grammar in the multiplicative constant $(|Q| \log |E|)$, which teases apart the contribution to complexity that comes from a specific constraint set and the contribution to complexity that comes from the computation of optimality.

[2] This is why Barton et al. (1987) formulate the universal generation problem for grammars. This factor also motivates Idsardi (2006) to adopt Eisner's complexity characterization of OT and is cited by Wareham (1998) who provides an independent proof that optimization in OT is NP hard in the universal case.

In this work, I adopt the algebraic characterization of violation profiles from Riggle (2009), but instead of doing generation with a modified Dijkstra-(1959)-style shortest path algorithm as in Riggle (2009), I provide a contender generation scheme that has more in common with a Kay-(1980)-style chart-parser. This latter approach is more appropriate to the task of generating contenders because the goal is the generation of all candidates that correspond to non-harmonically-bounded paths rather than the generation of a single optimal candidate. I show here that the contender generation algorithm is efficient modulo the number of contenders that it produces (i.e., there can be as many as $k!$ contenders for k constraints, but the generation of n contenders has complexity that is polynomial in n).

1.2 Relative bounding

When generating contenders for a given constraints set Con, I assume that the candidate generating function Gen takes an underlying form i and yields all and only the candidates that can be derived from i by changes (unfaithful mappings) that are penalized by constraints in Con. If Con is a subset of the universal set of (possible) constraints Con—as it is in most OT analyses—this is equivalent to assuming that all other unfaithful mappings are ruled out by undominated faithfulness constraints against changes other than those penalized by constraints in Con. Furthermore, I assume that only the markedness constraints in Con can motivate unfaithful mappings. This is equivalent to assuming that any markedness constraint in Con but not in Con is ranked below the constraints in Con. I will refer to the constraints in Con as **active** constraints.

Following these assumptions an OT analysis can be characterized as a micro-typology in which the rankings of the inactive constraints are held fixed in order to focus attention on just those candidates whose optimality is crucially determined by the ranking of the active constraints. The selection of these candidates can be achieved by an extension of the concept of **harmonic bounding** (Samek-Lodovici (1992), Prince & Smolensky (1993), Samek-Lodovici & Prince (1999, 2002)).

(1) **Definition: Harmonic Bounding** (Samek-Lodovici & Prince 1999:2)
A candidate is harmonically bounded if there is another candidate that is (a) at least as good on all constraints, and (b) better on at least one.

This definition seems to presuppose something like the notion of active constraints above. The critical point is the meaning of 'all constraints.' Consider the four candidates in (2).

(2)

/VC/	Onset	NoCoda	Dep	Max
a. VC**.**	*	*		
b. CV**.**			*	*
c. CV**.**CV**.**			**	
d. ϵ				**

Candidate *b* shows what Samek-Lodovici and Prince call **collective harmonic bounding**. There is no ranking of these four constraints that will allow candidate *b* to simultaneously beat candidates *c* and *d*. But what about other constraints? If DEP is divided into a specific version for vowels and a general version, then candidate *b* can triumph as in (3).

(3)

/VC/	ONSET	NOCODA	DEPV	MAX	DEP
a. VC.	*!	*			
b.☞ CV.				*	*
c. CV.CV.			*!		**
d. ϵ				**!	

Clearly the notion of harmonic bounding is not intended to refer to the entire universal constraint set. Even if we had a model of *all* the constraints and a way to feasibly work with them, the utility of harmonic bounding lies in its simplification of the candidate space.

Prince & Smolensky (1993:194-195) construe harmonic bounding slightly more narrowly as a condition that arises under a given (partial) ranking in which a particular structure can never surface because it is always ill-formed relative to another structure. The intended meaning of 'all constraints' in (1) seems to be somewhere in the middle, referring to any ranking of the constraints explicitly mentioned in an analysis but not to any ranking of any imaginable constraints. It is precisely the intuition behind this construal of the scope of harmonic bounding that the notion of active constraint above is meant to codify.

(4) **Definition: Relative Bounding**
A candidate is **relatively bounded** for a set of active constraints $\mathcal{A}$ if there is no ranking of $\mathcal{A}$ under which that candidate is optimal.

This definition synthesizes Prince and Smolensky's (1993) notion of harmonic bounding as a condition that holds relative to a partial ranking of the constraints with a default assumption about what that partial ranking is for all the constraints that are not mentioned in a given analysis. From this assumption, it immediately follows that any candidate that violates an inactive faithfulness constraint that is ranked above the active constraints will be relatively bounded by any candidate that violates no inactive faithfulness constraints.

(5) **Identity Candidate Corollary**
All candidates that violate faithfulness constraints ranked above active constraints $\mathcal{A}$ are relatively bounded for constraint set $\mathcal{A}$ by the fully-faithful identity candidate.

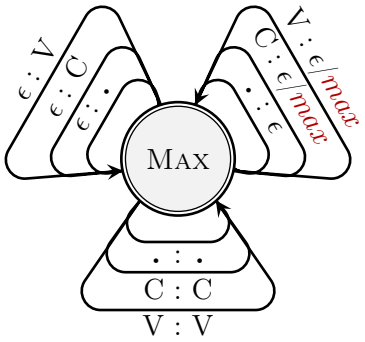
In generating contenders, our goal is to pair down the infinite candidate set to just those that are not relatively bounded for the active constraints.

2 OT with Rational Constraints: OT_R

Constraints in OT are relations from candidates to numbers of violations. Because relations that can be represented with finite state machines are often called 'rational' relations I will refer to OT analyses whose active constraints are drawn from the rational fragment of the universal constraint set as analyses within OT_R. Many constraints proposed in the literature lie outside the scope of OT_R, but in cases where all the active constraints are rational, we can generate contenders without regard for the other constraints.

2.1 Rational faithfulness constraints

Faithfulness constraints in OT_R can be represented as weighted finite-state transducers. For a concrete illustration of how these work, I will present a simple syllable structure grammar over the symbols {C, V, .} (where . marks syllable boundaries). The constraint Max can be instantiated as the finite state transducer ⟦Max⟧ in Fig. 1.



$\Sigma = \{\mathrm{C}, \mathrm{V}, .\}$ $\Delta = \{\mathrm{C}, \mathrm{V}, .\}$ $Q = \{\textsc{Max}\}$
$q0 = \textsc{Max}$ $F = \{\textsc{Max}\}$
$E = \{ (\textsc{Max}, \epsilon, \mathrm{V}, \varnothing, \textsc{Max}),\ (\textsc{Max}, \mathrm{V}, \mathrm{V}, \varnothing, \textsc{Max}),$
$(\textsc{Max}, \epsilon, \mathrm{C}, \varnothing, \textsc{Max}),\ (\textsc{Max}, \mathrm{V}, \epsilon, max, \textsc{Max}),$
$(\textsc{Max}, \epsilon, ., \varnothing, \textsc{Max}),\ (\textsc{Max}, \mathrm{C}, \epsilon, max, \textsc{Max}),$
$(\textsc{Max}, ., . , \varnothing, \textsc{Max}),\ (\textsc{Max}, . , \epsilon, \varnothing, \textsc{Max})\}$
$(\textsc{Max}, \mathrm{C}, \mathrm{C}, \varnothing, \textsc{Max}),$

Figure 1: A finite-state representations of the constraint Max

The *arcs* (arrows) in ⟦Max⟧ are labeled with (*input* : *output* / *weight*) triples that assign the weight *max* (one violation) each time C or V is mapped to ϵ the empty string. Candidates are evaluated by 'walking' along the arcs and adding up violations. For instance, each path in ⟦Max⟧ whose input labels spell out 'VC' is a candidate for the underlying form /VC/.

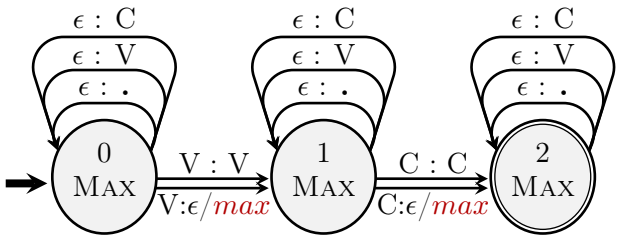


Figure 2: ⟦Max⟧(VC)

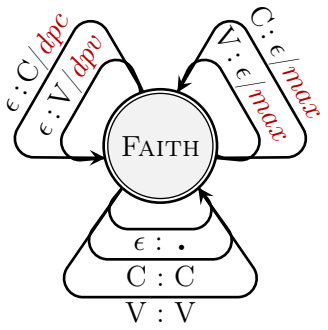
Formally, a weighted transducer $[\![M]\!]$ with weights from the set $\mathbb{X}$ is defined by a six-tuple $(\Sigma, \Delta, Q, q_0, F, E)$ where Σ and Δ are finite alphabets of 'input' and 'output' symbols, Q is

a finite set of states, $q_0 \in Q$ is the 'start' state, $F \subseteq Q$ are the 'final' states, and E is a finite set of arcs from $(Q \times \Sigma \cup \{\epsilon\} \times \Delta \cup \{\epsilon\} \times \mathbb{X} \times Q)$.[3] The notation in (6) will be helpful.

(6)
a. Given an arc $e \in E$: $s[e]$ denotes the source of the arc, $t[e]$ denotes the arc's terminus, $i[e]$ is the input label, $o[e]$ is the output label, and $w[e]$ is the weight.
b. A path $\langle e_1 ... e_k \rangle \in E^*$ is sequence of connected arcs: $t[e_{i-1}] = s[e_i]$ for $i = 2...k$.
c. The notation for arcs extends to paths in the obvious way. For $\pi = \langle e_1 ... e_k \rangle$: $s[\pi] = s[e_1]$, $t[\pi] = t[e_k]$, $i[\pi] = (i[e_1]...i[e_k])$, and $o[\pi] = (o[e_1]...o[e_k])$.
d. A path is **complete** if its source is the start state and its terminus a final state.
e. $[\![M]\!](x)$ is all complete paths that *accept* x: $\{\pi : s[\pi] = q_0,\ i[\pi] = x,\ t[\pi] \in F\}$.

There are infinitely many complete paths through [[MAX]](VC) in Fig. 2 and each one of them encodes a different candidate for the input /VC/ with its own surface form.

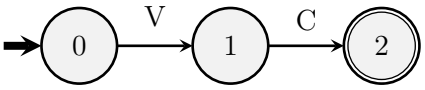
If the transducers for a set of faithfulness constraints are isomorphic (i.e., have exactly the same structure), they can be *intersected* to create a single weighted transducer by simply merging the weights on corresponding arcs.



$\Sigma = \{\text{C, V}\}$ $\Delta = \{\text{C, V, •}\}$ $Q = \{\text{FAITH}\}$
$q_0 = \text{FAITH}$ $F = \{\text{FAITH}\}$
$E = \{(\text{FAITH}, \epsilon, \text{C}, dpc, \text{FAITH}),\ (\text{FAITH}, \epsilon, \text{V}, dpv, \text{FAITH}),$
$(\text{FAITH, C}, \epsilon, max, \text{FAITH}), (\text{FAITH, V}, \epsilon, max, \text{FAITH})$
$(\text{FAITH}, \epsilon, \bullet, \varnothing, \text{FAITH}),\ (\text{FAITH, C, C}, \varnothing, \text{FAITH}),$
$(\text{FAITH, V, V}, \varnothing, \text{FAITH}),\}$

Figure 3: [[FAITH]] = [[DEPV]] ∩ [[DEPC]] ∩ [[MAX]]

The domain of a faithfulness constraint can be restricted to a particular underlying form with a finite state acceptor for that form. Fig. 4 illustrates the acceptor for /VC/.



$\Sigma = \{C, V, \bullet\}$ $Q = \{0, 1, 2\}$
$q0 = 0$ $F = \{2\}$ $E = \{(0,\text{V},1),\ (1,\text{C},2)\}$

Figure 4: *Accept*(VC), an acceptor for the string VC

Finite state acceptors are defined with five-tuples (Σ, Q, q_0, F, E) similarly to transducers, but they have only one alphabet Σ and the arcs in E are drawn from $Q \times \Sigma \times Q$. To generate candidates for underlying form x, the domain of [[FAITH]] is restricted with *Accept*(x). This

[3] Weighted automata are also typically defined with a weight function on the final states. I omit this detail for brevity because machines here do not impose additional penalties for stopping at a final state. For a thorough introduction to automata see Hopcroft and Ullman (1979) or Roche and Schabes (1997).

corresponds to the machine in Fig. 5, which can be thought of intuitively as representing all ways of walking along the arcs of the acceptor in Fig. 4 and the constraints in Fig. 3 at the same time.

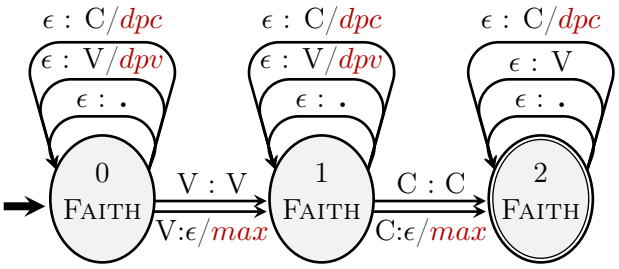


Figure 5: *Accept*(VC) $\cap_L$ [[FAITH]] = [[FAITH]](VC)

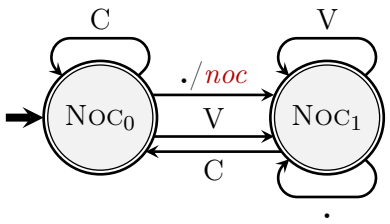
Formally, this machine is created by intersecting the acceptor with the domain of the transducer. This operation, which I will call left-intersection (denoted $\cap_L$) is defined in (7). For the sake of generality, I assume in the definition of left-intersection that the acceptor is weighted. This allows the possibility of underlying forms to be marked with violations and is harmless for unweighted acceptors whose arcs can be thought of as having $\varnothing$ weights.

(7) $(\Sigma_A, Q_A, q_{0A}, F_A, E_A) \cap_L (\Sigma_B, \Delta_B, Q_B, q_{0B}, F_B, E_B) =$
$(\Sigma_A \cap \Sigma_B, \Delta, Q_A \times Q_B, \langle q_{0A}, q_{0B}\rangle, F_A \cap F_B, E)$ where:
for each $(p, i, v, q), (p', i', o', v', q')$ in $E_A \times E_B$
if $i = i'$ then $(\langle p, p'\rangle, i, o, v \uplus v', \langle q, q'\rangle)$ is in E
and for each $q, (p', i', o', v', q')$ in $Q \times E_B$
if $i' = \epsilon$ then $(\langle q, p'\rangle, i', o', v \uplus v', \langle q, q'\rangle)$ is in E.

Right-intersection (denoted $\cap_R$) is defined analogously and is used to combine an acceptor with the range of a transducer. The terms 'right' or 'left' intersection accord with whether the right or left symbol on the arcs of the transducer is used in combining the machines. Right-intersection can be used to combine markedness and faithfulness constraints. This will make it possible to assemble a single weighted automaton that represents the whole set of active constraints.

2.2 Rational markedness constraints

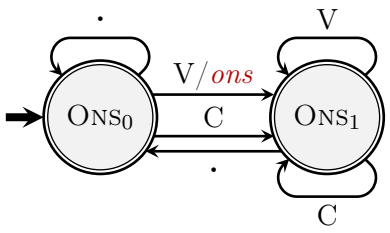
In OT_R, markedness constraints are weighted finite-state acceptors that are **complete** in the sense that they accept and assign a weight to every possible surface form in Δ^*. The constraint NOCODA, represented in Fig. 6, assigns violations to 'C**.**' sequences.
Weighted finite state acceptors (WFSA) are defined with five-tuples (Σ, Q, q_0, F, E) just like their unweighted counterparts; the only difference is that the arcs in E are drawn from the set $Q \times \Sigma \times \mathbb{X} \times Q$ where $\mathbb{X}$ is the set of possible weights.



$\Sigma = \{\text{C, V, .}\}$ $Q = \{\text{NOC}_0, \text{NOC}_1\}$
$q0 = \text{NOC}_0$ $F = \{\text{NOC}_0, \text{NOC}_1\}$
$E = \{(\text{NOC}_0, \text{C}, \varnothing, \text{NOC}_0),\ (\text{NOC}_1, \text{V}, \varnothing, \text{NOC}_1),$
$(\text{NOC}_0, \text{.}, \textit{noc}, \text{NOC}_1), (\text{NOC}_1, \text{C}, \varnothing, \text{NOC}_0),$
$(\text{NOC}_0, \text{V}, \varnothing, \text{NOC}_1),\ (\text{NOC}_1, \text{.}, \varnothing, \text{NOC}_1)\}$

Figure 6: NoCoda as a weighted finite state acceptor $[\![\text{NoCoda}]\!]$

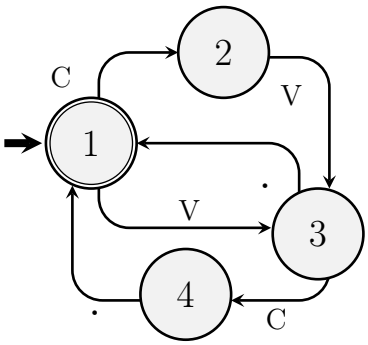
The constraint $[\![\text{Onset}]\!]$ presented in Fig. 7 is structurally similar to $[\![\text{NoCoda}]\!]$, but instead of assigning violations to the sequence 'C.' it penalizes vowel-initial syllables (which are characterized here as V occurring at the beginning of a form or immediately after a '.').



$\Sigma = \{\text{C, V, .}\}$ $Q = \{\text{ONS}_0, \text{ONS}_1\}$
$q0 = \text{ONS}_0$ $F = \{\text{ONS}_0, \text{ONS}_1\}$
$E = \{(\text{ONS}_0, \text{.}, \varnothing, \text{ONS}_0),\ (\text{ONS}_1, \text{V}, \varnothing, \text{ONS}_1),$
$(\text{ONS}_0, \text{V}, \textit{Ons}, \text{ONS}_1), (\text{ONS}_1, \text{.}, \varnothing, \text{ONS}_0),$
$(\text{ONS}_0, \text{C}, \varnothing, \text{ONS}_1),\ (\text{ONS}_1, \text{C}, \varnothing, \text{ONS}_1)\}$

Figure 7: Onset as a weighted finite state acceptor $[\![\text{Onset}]\!]$

'Hard' markedness constraints can be implemented as **incomplete** acceptors that simply reject some surface forms rather than assigning violations. Fig. 8 illustrates an inviolable constraint, which I will call $[\![\text{Syll}]\!]$, that requires all syllables to have the shape ((C)V(C)**.**).



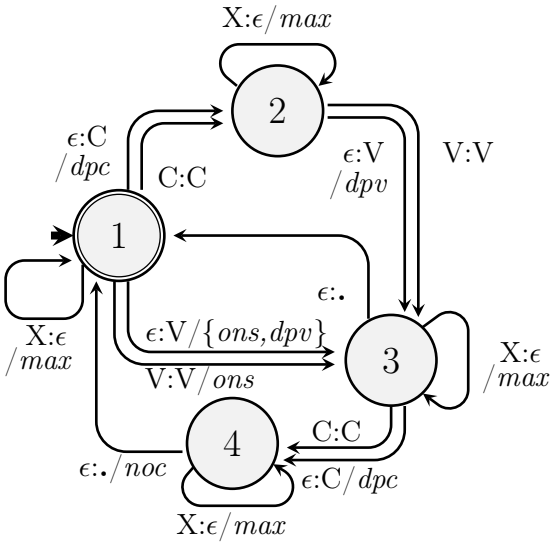
$\Sigma = \{\text{C, V, .}\}$ $Q = \{1, 2, 3, 4\}$
$q0 = 1$ $F = \{1\}$
$E = \{(1, \text{C}, 2),\ (3, \text{.}, 1),$
$(1, \text{V}, 3),\ (3, \text{C}, 4),$
$(2, \text{V}, 3),\ (4, \text{.}, 1)\}$

Figure 8: $[\![\text{Syll}]\!]$, a hard constraint on syllable structure

The restriction this constraint imposes could also be obtained using a ranked set of violable constraints. Hard constraints are convenient in that they simplify an analysis by restricting the typology to a particular set of languages.

2.3 Putting the pieces together

Intersecting all the constraints produces a single finite-state representation of the evaluation function. Because intersection is commutative, the order in which the pieces are put together is irrelevant; the machine ⟦Eval⟧, as presented in Fig. 9, is the same for every ranking of its constituent constraints. For the sake of parsimony, I use the symbol 'X' for 'C' *or* 'V' in order to collapse pairs of arcs labeled (C:ϵ/max) and (V:ϵ/max) in several places.[4]



$\Sigma = \{\text{C, V, .}\}$ $\Delta = \{\text{C, V, .}\}$
$Q = \{1, 2, 3, 4\}$ $q0 = 1$ $F = \{1\}$
$E = \{(1, \text{X}, \epsilon, max, 1), (2, \text{X}\epsilon, max, 2),$
$(1, \text{C}, \text{C}, \varnothing, 2), (2, \text{V}, \text{V}, \varnothing, 3),$
$(1, \epsilon, \text{C}, dpc, 2), (2, \epsilon, \text{V}, dpv, 3),$
$(1, \text{V}, \text{V}, \varnothing, 3), (3, \text{X}, \epsilon, max, 1),$
$(1, \epsilon, \text{V}, dpv, 3), (3, \epsilon, \text{.}, \varnothing, 1),$
$(4, \text{X}, \epsilon, max, 4), (3, \text{C}, \text{C}, \varnothing, 1),$
$(4, \epsilon, \text{.}, noc, 1), (3, \epsilon, \text{C}, dpv, 1)\}$

Figure 9: ⟦Eval⟧ = ⟦Syll⟧ $\cap_R$ ⟦Onset⟧ $\cap_R$ ⟦NoCoda⟧ $\cap_R$ ⟦Faith⟧

In order to evaluate the set of candidates for a specific input form, the acceptor for that form is left-intersected with ⟦Eval⟧. This is illustrated for the input /VC/ Fig. 10.

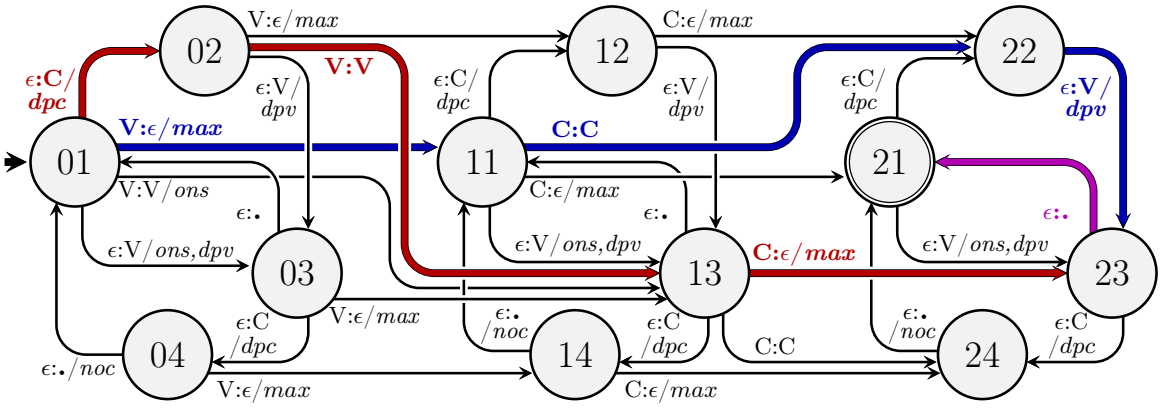


Figure 10: Two /VC/ → [CV.] candidates in *Accept*(VC) $\cap_L$ ⟦Eval⟧

[4]Features or sets of symbols can also be used in machines. For a discussion of the ramifications of using features vs. symbols in finite state rules/constraints see van Noord & Gerdemann (2000).

The intersection of the acceptor for /VC/ with ⟦Eval⟧ produces a machine that encodes *every* candidate that can be generated by the structure changing operations in ⟦Eval⟧. The highlighted paths in Fig. 10 describe two ways the input-output pairing (VC, CV.) can be generated. Because there are loops in the graph, there are infinitely many distinct paths and each one encodes a candidate $(input, output)$ mapping for the underlying form /VC/.

2.4 Working with infinite tableaux

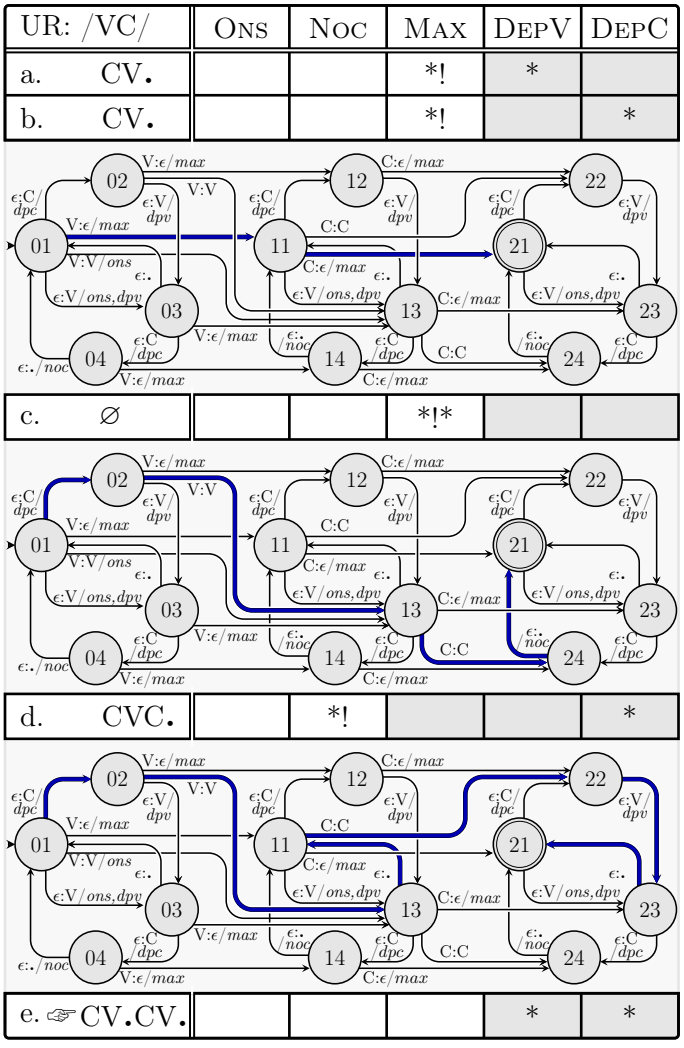


Figure 11: Five contenders for /VC/

In Fig. 11, I present a (relatively) traditional tableau that includes the two candidates from Fig. 10 along with finite state representations of three others. Because each of the paths in $Accept(\text{VC}) \cap_L$ ⟦Eval⟧ corresponds to a candidate for the input /VC/, one could think of this machine as an infinite tableau with each path representing a row.

It is fairly easy to verify that candidate e is optimal among the five candidates for the ranking given in Fig. 11, however *proving* that e is superior to all possible alternatives is another matter entirely. Such a proof requires the representation of the entire candidate space provided by ⟦Eval⟧(VC). In §3.7 I will show how this can be done.

In §3.4 I will show that working with the 'infinite' tableaux encoded by weighted automata makes it possible to generate all of the non-harmonically-bounded candidates in a single derivation.

The critical insight behind all computational approaches to OT is that optimization can be done over a finite representation of the infinite space of possible candidates and does *not* require searching this infinite space by generating and testing candidates one at a time.[5]

[5] E.g., Ellison (1994b), Tesar (1995), Eisner (1997a,b), Gerdemann & van Noord (2000), Riggle (2004).

3 Generating Contenders

In §2, I showed how sets of rational constraints could be combined with each other and with the representation of an underlying form to produce a single finite state representation of the space of possible candidates for that form. In this section, I will show how this representation can be used to efficiently generate optimal forms.

3.1 Violations multisets and the violation semiring

In §2 the merge operation $\uplus$ was used to combine violations across machines. This implies that violation profiles are represented as **multisets**. Formally, a multiset X is a pair (C, m) where C, the **basis**, is a standard Cantorian set and $m_X(c)$ is a **multiplicity** function that maps each $c \in C$ to the number of times it occurs in X. The merge operation is basically addition: $A \uplus B = C$ where $m_C(x) = m_A(x) + m_B(x)$ and the basis of C is the union of the basis sets of A and B. When writing out multisets, I will denote multiplicities greater than one with numeric prefixes and, in cases where confusion with ordinary sets might arise, I will surround the multisets with 'bag-braces', e.g. $\wr ons, max \wr \uplus \wr max \wr = \wr ons, 2max \wr$.

For any given constraint set *Con*, the range of possible violation profiles is precisely the set of all multisets that share *Con* as their basis, which I will denote as V_{Con}. Violation profiles could be represented in a variety of ways—a list would suffice—but multisets are an elegant choice because their structure does not imply any ordering among the constraints and because merger $\uplus$ corresponds to the natural sense of addition for violation profiles. Likewise, multiset-difference corresponds to subtraction and the subset relation $\Subset$ corresponds to (simple) harmonic bounding. Put a bit more formally, $\uplus$ is a closed binary operator that is both commutative and associative. This means that the triple $(V_{Con}, \uplus, \varnothing)$ is a commutative monoid that provides a basic system of arithmetic for violation profiles.

Given a constraint ranking $\mathcal{R}_{Con}$, the relation of **harmonic inequality** (denoted $\succ_{\mathcal{R}}$), provides a total ordering of the violation profiles in V_{Con}.

(8) **Harmonic Inequality**
For A, $B \in V_{Con}$, A is more harmonic than B according to $\mathcal{R}_{Con}$, written $A \succ_{\mathcal{R}} B$, iff $m_A(c) < m_B(c)$ for the highest ranked $c \in Con$ where $m_A(c) \neq m_B(c)$.

Optimization in OT can be seen as minimization according to harmonic inequality. For two violation profiles A and B, the function $\mathbf{min}_{\mathcal{R}}(A, B)$ returns A if $A \succ_{\mathcal{R}} B$ and B otherwise.

The 'empty' violation profile $\varnothing$ is the most harmonic element of V_{Con} regardless of the ranking of the constraints. Conversely, an infinite violation profile '∞' in which $m(c) = \infty$ for all $c \in Con$ can be created to serve as the antithesis of $\varnothing$. If the definition of V_{Con} is extended slightly to include the infinite violation profile, then ∞ will be the least harmonic element of V_{Con} regardless of the ranking of the constraints. I will assume henceforth that V_{Con} contains the infinite violation profile, this provides another commutative monoid $(V_{Con}, \min_{\mathcal{R}}, \infty)$ over violation profiles. Note that there are as many $\min_{\mathcal{R}}$ operators as there are rankings $\mathcal{R}_{Con}$ but that they all share these same properties.

Together, these monoids make up a semiring (V_{Con}, $\min_{\mathcal{R}}$, $\uplus$, ∞, $\varnothing$) over violation profiles. Semirings are abstract algebraic structures represented by five-tuples $(\mathbb{X}, \oplus, \otimes, \bar{0}, \bar{1})$ that allow a unified characterization of many seemingly different systems. For the set $\mathbb{X}$, semirings define two operations $\oplus$ and $\otimes$ that act intuitively like the $+$ and $\times$ operations on $\mathbb{N}$. Like $+$, $\oplus$ must be associative and commutative and, like $\times$, $\otimes$ must be associative and must distribute over the $\oplus$ operator. The $\otimes$ operator is not required to be commutative, but when it is the semiring is commutative. Finally, $\mathbb{X}$ must contain two elements $\bar{0}$ and $\bar{1}$ that act as identity elements for the $\oplus$ and $\otimes$ operators respectively and $\bar{0}$ must be an 'annihilator' for the $\otimes$ operation in the sense that $\bar{0} \otimes x = \bar{0}$ for any $x \in \mathbb{X}$. Two of the most familiar semirings are the 'counting' semiring $\mathcal{C}$ and the boolean semiring $\mathcal{B}$ given in (9).

(9)

The $\mathcal{C}$ and $\mathcal{B}$ semirings:	$\mathcal{C} = (\mathbb{N}, +, \times, 0, 1)$	$\mathcal{B} = (\{0,1\}, \vee, \wedge, 0, 1)$
1. $\oplus$ associativity	$(a+b)+c = a+(b+c)$	$(a \vee b) \vee c = a \vee (b \vee c)$
$\oplus$ commutativity	$(a+b) = (b+a)$	$(a \vee b) = (b \vee a)$
$\bar{0}$ identity for $\oplus$	$a+0 = 0+a = a$	$a \vee 0 = 0 \vee a = a$
2. $\otimes$ associativity	$(a \times b) \times c = a \times (b \times c)$	$(a \wedge b) \wedge c = a \wedge (b \wedge c)$
$\otimes$ commutativity	$(a \times b) = (b \times a)$	$(a \wedge b) = (b \wedge a)$
$\bar{1}$ identity for $\otimes$	$a \times 1 = 1 \times a = a$	$a \wedge 1 = 1 \wedge a = a$
3. $\otimes$ distributivity	$(a+b)\times c = (a\times c)+(b\times c)$	$(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c)$
	$c\times(a+b) = (c\times a)+(c\times b)$	$c \wedge (a \vee b) = (c \wedge a) \vee (c \wedge b)$
4. $\bar{0}$ annihilates for $\otimes$	$a \times 0 = 0 \times a = 0$	$a \wedge 0 = 0 \wedge a = 0$

For a thorough introduction to semirings, their relation to formal languages, their use in parsing, and their use in optimization problems see Fink (1992), Kuich (1997), Goodman (1998), Mohri (2002), Droste & Kuich (2007), and citations therein.

For the violation semiring, the $\min_{\mathcal{R}}$ operator takes the role of $\oplus$ and the $\uplus$ operator takes the role of $\otimes$. In (10), I present the violation semiring $\mathcal{V}$ alongside the tropical semiring $\mathcal{T}$, which is the semiring that is most commonly used for optimization problems.

(10)

$\mathcal{T} = (\{\mathbb{R}_+ \cup \infty\}, \min, +, \infty, 0)$	$\mathcal{V} = (V_{Con}, \min_{\mathcal{R}}, \uplus, \infty, \varnothing)$
1. $(a \min b) \min c = a \min (b \min c)$	$(a \ \min_{\mathcal{R}} \ b) \ \min_{\mathcal{R}} \ c = a \ \min_{\mathcal{R}} \ (b \ \min_{\mathcal{R}} \ c)$
$(a \min b) = (b \min a)$	$(a \ \min_{\mathcal{R}} \ b) = (b \ \min_{\mathcal{R}} \ a)$
$a \min \infty = \infty \min a = a$	$a \ \min_{\mathcal{R}} \ \infty = \infty \ \min_{\mathcal{R}} \ a = a$
2. $(a+b)+c = a+(b+c)$	$(a \uplus b) \uplus c = a \uplus (b \uplus c)$
$(a+b) = (b+a)$	$(a \uplus b) = (b \uplus a)$
$a+0 = 0+a = a$	$a \uplus \varnothing = \varnothing \uplus a = a$
3. $(a \min b) + c = (a+c) \min (b+c)$	$(a \ \min_{\mathcal{R}} \ b) \uplus c = (a \uplus c) \ \min_{\mathcal{R}} \ (b \uplus c)$
$c + (a \min b) = (c+a) \min (c+b)$	$c \uplus (a \ \min_{\mathcal{R}} \ b) = (c \uplus a) \ \min_{\mathcal{R}} \ (c \uplus b)$
4. $a + \infty = \infty + a = \infty$	$a \uplus \infty = \infty \uplus a = \infty$

The violation semiring is actually quite similar to the tropical semiring. In both cases the $\oplus$ operator is minimization and the $\otimes$ operator is addition.[6]

For a semiring-weighted finite state machine $[\![M]\!]$ with arcs E, the weight of each path $\pi = \langle e_1 ... e_k \rangle \in E^*$ is $(w[e_1] \otimes ... \otimes w[e_k])$. For $\mathcal{V}$, this is the merge of the weights in π.

$$(11) \quad w[\pi] = \bigotimes_{e \in \pi} w[e] \qquad \text{E.g.} \quad w[\pi] = \biguplus_{e \in \pi} w[e]$$

Each path is a candidate, and with a ranking $\mathcal{R}_{Con}$ the notion of an optimal path is defined.

$$(12) \quad [\![M]\!]_{\mathcal{R}}(x) = \bigoplus_{\pi \in [\![M]\!](x)} w[\pi] \qquad \text{E.g.} \quad [\![M]\!]_{\mathcal{R}}(x) = \min_{\mathcal{R}}{}_{\pi \in [\![M]\!](x)} w[\pi]$$

As illustrated in Fig. 10, the presence of epenthetic cycles (i.e., loops) can make the set of candidates (i.e., paths) in $[\![M]\!](x)$ infinite, so it is essential that 'infinite sums' be well defined (so called in reference to $\oplus$). In the case of the $\mathcal{V}$ semiring, what this means is that (12) must return well defined optima even when the set of candidates is infinite.

The $\min_{\mathcal{R}}$ operation defines a partial ordering over V_{Con} called the **natural order** and this order corresponds precisely to harmonic inequality (i.e., $A \succeq_{\mathcal{R}} B \Leftrightarrow A \min_{\mathcal{R}} B = A$). Because the $\bar{1}$ element acts as an annihilator for $\min_{\mathcal{R}}$ (i.e., $A \min_{\mathcal{R}} \varnothing = \varnothing$), the $\mathcal{V}$ semiring is **bounded** in the sense that the harmony of every violation profile X is between $\varnothing$ and ∞: $\varnothing \succeq_{\mathcal{R}} X \succeq_{\mathcal{R}} \infty$. This is sufficient to guarantee that optimality is well defined for infinite candidate sets because, for any loop π, $w[\pi]^0 \succeq_{\mathcal{R}} w[\pi]^n$ for any positive integer n. In other words, going around the loop zero times, which costs $\bar{1} = \varnothing$, is always better (or no worse than), going around the loop n times, which costs $(((w[\pi])_1 \uplus w[\pi])_2 \uplus ... w[\pi])_n$.

Bounded semirings are **idempotent** in the sense that $A \min_{\mathcal{R}} A = A$ for all $A \in V_{Con}$. This ensures that the semiring is **monotonic** (i.e., if $A \succeq_{\mathcal{R}} B$ then $(A \min_{\mathcal{R}} C) \succeq_{\mathcal{R}} (B \min_{\mathcal{R}} C)$ and $(A \uplus C) \succeq_{\mathcal{R}} (B \uplus C)$). The monotonicity of $\mathcal{V}$ makes it possible to use dynamic programming techniques whereby the optimization problem is recursively factored into sub-problems whose optimal solutions are combined to produce an optimal solution for the whole problem.

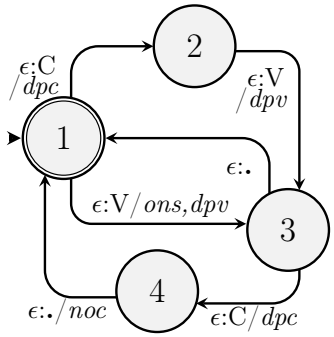
3.2 Optimization via dynamic programing

Given a graph-representation of the entire candidate space like the one in Fig. 10, the task of computing the optimal candidate under ranking $\mathcal{R}$ is simply a matter of finding the most harmonic path from the start state to a final state under $\mathcal{R}$. This is a relatively standard instance of a *shortest path* problem, where shortest is taken to mean most harmonic. For the purpose of illustration, I will find the most harmonic paths in $[\![\text{EVAL}]\!](\text{VC})$ under the ranking $\mathcal{R} = \text{ONSET} \gg \text{NOCODA} \gg \text{MAX} \gg \text{DEPV} \gg \text{DEPC}$.

[6] $\mathcal{T}$ is also called the (min, +) semiring. The 'tropical' moniker is an homage to Imre Simon's pioneering research on $\mathcal{T}$ (cf. Simon (1988)). In weighted-constraint-based models like Harmonic Grammar (Legendre et al. 1990, Goldsmith 1993, Smolensky & Legendre 2006, Pater et al. 2007a,b), grammars assigns a weight $w(c) \in \mathbb{R}_+$ to each $c \in Con$. This provides a morphism from $\mathcal{V}$ to $\mathcal{T}$ (i.e., summing $m(c) \times w(c)$, $c \in Con$ maps V_{Con} into $\mathbb{R}_+$). The analysis in this section is applicable to either system.

A key strategy of dynamic programing is the identification of 'overlapping sub-problems' that can be factored out of larger problems (or classes of problems). The idea is to solve a sub-problem just one time and save the results—a process called *memoization*—for plugging back into larger problems. Following this strategy, I will start by solving a sub-problem that recurs several times in the search for optimal paths in ⟦EVAL⟧(VC).

Suppose, one wanted to find the optimal path between each pair of states in ⟦EVAL⟧ with the caveat that this path should accept *zero* segments of the underlying form. Fig. 12 gives a graph of the subset of the arcs in ⟦EVAL⟧ that accept the empty string in the underlying form, along with an adjacency matrix M for the graph in which entry $M_{i,j}$ (i.e., the value in the i-th row and j-th column) gives the weight on the arc from state i to state j.



	1	2	3	4
1	∅	*dpc*	*ons*, *dpv*	∞
2	∞	∅	*dpv*	∞
3	∅	∞	∅	*dpc*
4	*noc*	∞	∞	∅

Figure 12: ⟦EVAL⟧ for the input /ϵ/ and the adjacency matrix for ϵ

The entries along the diagonal of M are ∅ because the zero-length path between each state and itself accepts the empty string and costs nothing. Pairs of distinct states that are not connected by a single arc are given infinite weight.

The matrix in Fig. 12 gives weights for paths that contain at most one arc. Finding optimal paths of arbitrary length between pairs of states is often called the 'all-paris shortest paths' problem. A standard dynamic programming approach to this problem is the Floyd-Warshall algorithm (see Cormen et al. 1990:ch25). For weighted a graph with states Q that are labeled $(1, ..., |Q|)$, this approach starts with a $|Q| \times |Q|$ adjacency matrix M as in Fig. 12 and then iteratively updates the matrix with the rule in (13).

(13) **The update rule**
For each $y \in Q$, for each $(x, z) \in Q \times Q$: set $M_{x,z} = M_{x,z} \ \min_{\mathcal{R}} (M_{x,y} \uplus M_{y,z})$.

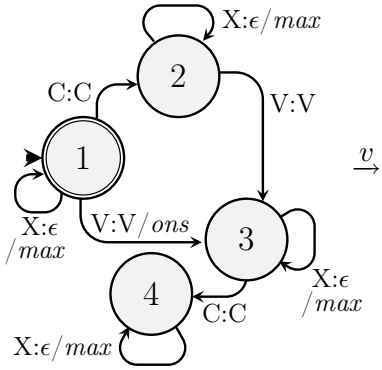
For each state y in the graph, the rule asks for each pair of states (x, z) whether it would it be cheaper to get from x to z via the path $x \rightarrow y \rightarrow z$. If the path through y is cheaper then the value of $M_{x,z}$ is updated accordingly.

Updating the matrix for a graph with $|Q|$ states using the rule in (13) requires $|Q|^3$ additions with the $\uplus$ operator and $|Q|^3$ comparisons of violation profiles with the $\min_{\mathcal{R}}$ operator. The resulting matrix, which I will refer to as λ, is presented in Fig. 13. The λ matrix gives the cost of the optimal path between each pair of states made up entirely of arcs whose input symbols are /ϵ/ (or made up of no arcs at all in the case of the entries along the diagonal).

$$\lambda = \begin{bmatrix} \varnothing & dpc & dpv, dpc & dpv, 2dpc \\ dpv & \varnothing & dpv & dpv, dpc \\ \varnothing & dpc & \varnothing & dpc \\ noc & noc & noc, dpv, dpc & \varnothing \end{bmatrix}$$

Figure 13: The λ matrix

Fig. 14 gives adjacency matrices for the arcs in ⟦Eval⟧ that accept /V/ and /C/. In these cases, the diagonal values are *max* because each arc must accept an underlying segment.



$$\underrightarrow{v} = \begin{bmatrix} max & \infty & ons & \infty \\ \infty & max & \varnothing & \infty \\ \infty & \infty & max & \infty \\ \infty & \infty & \infty & max \end{bmatrix} \quad \underrightarrow{c} = \begin{bmatrix} max & \varnothing & \infty & \infty \\ \infty & max & \infty & \infty \\ \infty & \infty & max & \varnothing \\ \infty & \infty & \infty & max \end{bmatrix}$$

Figure 14: Adjacency matrices for /V/ and /C/ in ⟦Eval⟧

Using these matrices, the search for optimal paths in ⟦Eval⟧(VC) can be broken into a sequence of five sub-problems $\langle \lambda \times \underrightarrow{v} \times \lambda \times \underrightarrow{c} \times \lambda \rangle$ whose solutions can be combined via standard matrix multiplication (using $\min_{\mathcal{R}}$ and $\uplus$ as if they were $+$ and $\times$ respectively).

The product of an $(m \times n)$ matrix A and an $(n \times p)$ matrix B is an $(m \times p)$ matrix C (the number of rows in A must be equal to the number of columns in B). In matrix C, the value of entry $C_{i,j}$ is the inner product of the i-th row in A and the j-th column in B. This is defined more formally in (14).

$$(14) \qquad \begin{bmatrix} A_{1,1} & A_{1,2} & ... & A_{1,n} \\ A_{2,1} & A_{1,2} & ... & A_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & ... & A_{m,n} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} & ... & B_{1,p} \\ B_{2,1} & B_{1,2} & ... & B_{1,p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{n,1} & B_{n,2} & ... & B_{n,p} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} & ... & C_{1,p} \\ C_{2,1} & C_{1,2} & ... & C_{1,p} \\ \vdots & \vdots & \ddots & \vdots \\ C_{m,1} & C_{m,2} & ... & C_{m,p} \end{bmatrix}$$

$$\text{where } C_{i,j} = (A_{i,1} \uplus B_{1,j}) \ \min_{\mathcal{R}} \ (A_{i,2} \uplus B_{2,j}) \ \min_{\mathcal{R}} \ ... \ \min_{\mathcal{R}} \ (A_{i,n} \uplus B_{n,j})$$

The matrices that result from taking the products $(\lambda \times \underrightarrow{v} \times \lambda) = V$ and $(\lambda \times \underrightarrow{c} \times \lambda) = C$ can serve as the building blocks of optimization. These matrices give the cost of the most harmonic path between each pair of states in ⟦Eval⟧ for exactly one underlying segment.

The product VC (denoted by juxtaposition of V and C) nicely illustrates the way matrix multiplication computes optimality.

$$V=\begin{bmatrix} dpc & 2dpc & dpc & 2dpc \\ \varnothing & dpc & \varnothing & dpc \\ dpc & 2dpc & dpc & 2dpc \\ \begin{matrix}noc,\\ dpc\end{matrix} & \begin{matrix}noc,\\ 2dpc\end{matrix} & \begin{matrix}noc,\\ dpc\end{matrix} & max \end{bmatrix} \quad C=\begin{bmatrix} dpv & \varnothing & dpv & \begin{matrix}dpv,\\ dpc\end{matrix} \\ 2dpv & dpv & 2dpv & dpv \\ dpv & \varnothing & dpv & \varnothing \\ \begin{matrix}noc,\\ dpv\end{matrix} & noc & \begin{matrix}noc,\\ dpv\end{matrix} & max \end{bmatrix} \quad VC=\begin{bmatrix} \begin{matrix}dpv,\\ dpc\end{matrix} & \boldsymbol{dpc} & \begin{matrix}dpv,\\ dpc\end{matrix} & dpc \\ dpv & \varnothing & dpv & \varnothing \\ \begin{matrix}dpv,\\ dpc\end{matrix} & dpc & \begin{matrix}dpv,\\ dpc\end{matrix} & dpc \\ \begin{matrix}noc,\\ dpv,\\ dpc\end{matrix} & \begin{matrix}noc,\\ dpc\end{matrix} & \begin{matrix}noc,\\ dpv,\\ dpc\end{matrix} & 2max \end{bmatrix}$$

Figure 15: The V matrix, the C matrix, and their product the VC matrix

Entry $VC_{i,j}$ is the inner product of the i-th row of V and the j-th column of C. The former encodes optimal paths $i \rightarrow x$ (i.e., paths from state i to state x) while the latter encodes optimal paths $x \rightarrow j$. The optimal path $i \rightarrow j$ will go through whichever x gives the most harmonic value for $i \rightarrow x \uplus x \rightarrow j$. This is simply the inner product. For example:

$$\begin{aligned} (15)\quad VC_{1,2} &= (\wr dpc\wr \uplus \varnothing) \min_{\mathcal{R}} (\wr 2dpc\wr \uplus \wr dpv\wr) \min_{\mathcal{R}} (\wr dpc\wr \uplus \varnothing) \min_{\mathcal{R}} (\wr 2dpc\wr \uplus \wr noc\wr) \\ &= \wr dpc\wr \min_{\mathcal{R}} \wr 2dpc, dpv\wr \min_{\mathcal{R}} \wr dpc\wr \min_{\mathcal{R}} \wr 2dpc, noc\wr \\ &= \wr dpc\wr \end{aligned}$$

Every underlying form in {C,V}* can be represented by a sequence of V and C matrices and the product of this sequence will be a matrix of optimal costs for that underlying form.

The complexity of matrix multiplication is dominated by the $\uplus$ operation. When an $(m \times n)$ matrix is multiplied by an $(n \times p)$ matrix there are $m \times n \times p$ applications of the merge operation. The complexity is generally said to be cubic because square matrices require n^3 merges. In the case of optimization, however, square matrices are only necessary if one wants to compute optimal paths for *every* pair of states. For optimal paths originating at a single state (e.g. the start state) a single row of the matrix is adequate. I will call the row of the λ matrix corresponding to the start state 'λ_S'; the product of $\lambda_S = \begin{bmatrix} \varnothing & dpc & \begin{matrix}dpv,\\ dpc\end{matrix} & \begin{matrix}dpv,\\ 2dpc\end{matrix} \end{bmatrix}$ and matrix V is given in Fig. 16.

$$\begin{bmatrix} \varnothing & dpc & \begin{matrix}dpv,\\ dpc\end{matrix} & \begin{matrix}dpv,\\ 2dpc\end{matrix} \end{bmatrix} \times \begin{bmatrix} dpc & 2dpc & dpc & 2dpc \\ \varnothing & dpc & \varnothing & dpc \\ dpc & 2dpc & dpc & 2dpc \\ \begin{matrix}noc,\\ dpc\end{matrix} & \begin{matrix}noc,\\ 2dpc\end{matrix} & \begin{matrix}noc,\\ dpc\end{matrix} & max \end{bmatrix} = \begin{bmatrix} dpc & 2dpc & 2dpc & dpc \end{bmatrix}$$

Figure 16: $\lambda_S V$ = optimal paths that originate at state 1 and accept /V/

Multiplying a $(1 \times n)$ matrix by an $(n \times n)$ matrix produces a $(1 \times n)$ matrix and requires n^2 applications of the $\uplus$ operator. The inclusion of the matrix λ_S as the base case in the sequence of multiplications will restrict the computation to paths that originate at the start state and thereby reduce the factor introduced by $|Q|$ from cubic to quadratic.

If matrices like V and C are computed ahead of time for each underlying segment and memoized, then the computation $\lambda_S \times M_1 \times ... \times M_n$ for an underlying form containing n segments involves n multiplications that each require $|Q|^2$ merges. This puts the complexity of optimization at $n|Q|^2$, which is linear in the length of the input with a constant factor that is quadratic in the size of $[\![\textsc{Eval}]\!]$. If larger chunks are memoized then the amount of computation will grow as a fraction of n. For instance, if the VC matrix in Fig. 15 had already been memoized then computing $VCVC$ would require just one multiplication.

Even if the matrices for individual segments are not computed in advance, doing some memoization on the fly will eliminate repeated computations. For an underlying form with n segments, after an initial investment of $|Q|^3$ calculations to produce λ, the product of the sequence $\langle \lambda_S \times \underrightarrow{x_1} \times \lambda \times \underrightarrow{x_2} \times \lambda \times ... \times \underrightarrow{x_n} \times \lambda \rangle$ can be computed in $2n|Q|^2$ steps (where $\underrightarrow{x_i}$ is the adjacency matrix for the i-th segment). This puts the complexity at $2n|Q|^2 + |Q|^3$, which is still linear in the length of the underlying form, albeit with larger constants.

3.3 Collecting the candidates

Thus far, the objects of optimization have been matrices of violation profiles for paths between pairs of states but have not contained any details about the surface forms that correspond to those paths. In order to generate actual candidates, the information in the matrices can be embellished to include fragments of surface forms along with the costs.

One way to do this is to represent candidates as (v, S) pairs where $v \in V_{Con}$ is a violation profile and $S \subset \Delta^*$ is a set of fragments of surface forms. Henceforth, the term **candidate** will refer to pairs like these.[7] The $\otimes$ operator for candidates will be a pair of operators $(\uplus, \cdot)$ merge and concatenate, the second of which concatenates the strings in the sets of surface form fragments (i.e., $A \cdot B = \{ab : a \in A \text{ and } b \in B\}$). Defining the behavior of $\min_{\mathcal{R}}$ for candidates can be done as in (16).

$$(16)\quad (v_1, K_1)\ \min_{\mathcal{R}}\ (v_2, K_2) = \begin{cases} (v_1, K_1 \cup K_2) & \text{if } v_1 = v_2 \\ (v_1, K_1) & \text{if } v_1 \succ_{\mathcal{R}} v_2 \\ (v_2, K_2) & \text{if } v_2 \succ_{\mathcal{R}} v_1 \end{cases}$$

If two candidates have identical violation profiles (i.e., there is a tie), they are unified into a single candidate with the union of their surface forms. Otherwise, the $\min_{\mathcal{R}}$ operator returns whichever candidate has a more harmonic violation profile under ranking $\mathcal{R}$.

The reason that the surface forms can be integrated seamlessly into the computation described in §3.2 is that the set of all sets of surface forms (i.e., the powerset, $\wp\Delta^*$) provides

[7] I will use the term candidate to describe the (v, S) pairs associated with single arcs, sequences of arcs, and complete paths and the term **complete candidate** when it is necessary to distinguish the latter.

another idempotent semiring $(\wp\Delta^*, \cup, \cdot, \emptyset, \{\epsilon\})$, the semiring of formal languages over Δ. Note that $\emptyset$ will be used for the empty string-set and $\varnothing$ for the empty violation-set.

(17) a. The violation semiring: $(V_{Con}, \min_{\mathcal{R}}, \uplus, \infty, \varnothing)$

b. The language semiring: $(\wp\Delta^*, \cup, \cdot, \emptyset, \{\epsilon\})$

c. The candidate semiring: $(V_{Con} \times \wp\Delta^*, \min_{\mathcal{R}}, (\uplus, \cdot), (\infty, \emptyset), (\varnothing, \{\epsilon\}))$

(17c) is similar to Goodman's (1998) *Viterbi-derivation* semiring for computing the most likely derivations in probabilistic context-free grammars. In Goodman's case the violation semiring is replaced by the *Viterbi* semiring $([0,1], \max, \times, 0, 1)$ over probabilities. While the Viterbi-derivation semiring collects the set of parses that are tied as the most probable, the candidate semiring collects the set of surface forms that are tied as most harmonic.

For a concrete illustration of how the new operators work, consider the computation of the matrix VC as the product of the V and C matrices. Figure 17 gives the V and C matrices for candidate fragments consisting of (violations, surface forms) pairs.

$$V = \begin{bmatrix} \langle dpc\rangle, \{\text{CV}\bullet\} & \langle 2dpc\rangle, \{\text{CV}\bullet\text{C}\} & \langle dpc\rangle, \{\text{CV}\} & \langle 2dpc\rangle, \{\text{CVC}\} \\ \varnothing, \{\text{V}\bullet\} & \langle dpc\rangle, \{\text{V}\bullet\text{C}\} & \varnothing, \{\text{V}\} & \langle dpc\rangle, \{\text{VC}\} \\ \langle dpc\rangle, \{\bullet\text{CV}\bullet\} & \langle 2dpc\rangle, \{\bullet\text{CV}\bullet\text{C}\} & \langle dpc\rangle, \{\bullet\text{CV}\} & \langle 2dpc\rangle, \{\bullet\text{CVC}\} \\ \langle noc, dpc\rangle, \{\bullet\text{CV}\bullet\} & \langle noc, 2dpc\rangle, \{\bullet\text{CV}\bullet\text{C}\} & \langle noc, dpc\rangle, \{\bullet\text{CV}\} & \langle max\rangle, \{\epsilon\} \end{bmatrix}$$

$$C = \begin{bmatrix} \langle dpv\rangle, \{\text{CV}\bullet\} & \varnothing, \{\text{C}\} & \langle dpv\rangle, \{\text{CV}\} & \langle dpv, dpc\rangle, \{\text{CVC}\} \\ \langle 2dpv\rangle, \{\text{V}\bullet\text{CV}\bullet\} & \langle dpv\rangle, \{\text{V}\bullet\text{C}\} & \langle 2dpv\rangle, \{\text{V}\bullet\text{CV}\} & \langle dpv\rangle, \{\text{VC}\} \\ \langle dpv\rangle, \{\bullet\text{CV}\bullet\} & \varnothing, \{\bullet\text{C}\} & \langle dpv\rangle, \{\bullet\text{CV}\} & \varnothing, \{\text{C}\} \\ \langle noc, dpv\rangle, \{\bullet\text{CV}\bullet\} & \langle noc\rangle, \{\bullet\text{C}\} & \langle noc, dpv\rangle, \{\bullet\text{CV}\} & \langle max\rangle, \{\epsilon\} \end{bmatrix}$$

Figure 17: The V and C matrices for candidate fragments

The value for $VC_{1,1}$ is the inner product of the first row in V and the first column in C.

(18)
$$\begin{aligned} \langle dpc\rangle, \{\text{CV}\bullet\} \uplus \langle dpv\rangle, \{\text{CV}\bullet\} &= \langle dpv, dpc\rangle, \{\text{CV}\bullet\text{CV}\bullet\} \\ \langle 2dpc\rangle, \{\text{CV}\bullet\text{C}\} \uplus \langle 2dpv\rangle, \{\text{V}\bullet\text{CV}\bullet\} &= \langle 2dpv, 2dpc\rangle, \{\text{CV}\bullet\text{CV}\bullet\text{CV}\} \\ \langle dpc\rangle, \{\text{CV}\} \uplus \langle dpv\rangle, \{\bullet\text{CV}\bullet\} &= \langle dpv, dpc\rangle, \{\text{CV}\bullet\text{CV}\bullet\} \\ \langle 2dpc\rangle, \{\text{CVC}\} \uplus \langle noc, dpv\rangle, \{\bullet\text{CV}\bullet\} &= \underline{\langle 2dpc, noc, dpv\rangle, \{\text{CVC}\bullet\text{CV}\bullet\}} \\ & \quad \langle dpv, dpc\rangle, \{\text{CV}\bullet\text{CV}\bullet\} \quad \leftarrow \text{optimal candidate} \end{aligned}$$

Because state 1 is the start state and the only final state in $[\![\text{Eval}]\!]$, the value at $VC_{1,1}$ represents the optimal candidate in $[\![\text{Eval}]\!](\text{VC})$. If there is more than one final state, the optimal candidate is selected by $\min_{\mathcal{R}}$ from the union of the candidates at the final states.

With candidates represented as (violation-profile, surface-form-set) pairs there can be no ties among candidates because each ranking provides a total ordering of violation profiles. Yet, it is reasonable to ask how big the surface-form-sets can be. If ⟦Eval⟧ contains a **free epenthetic loop** that incurs no violations, accepts the empty string, and outputs something other than the empty string (i.e., π such that $s[\pi] = t[\pi],\ w[\pi] = \varnothing,\ i[\pi] = \epsilon,$ and $o[\pi] \neq \epsilon$), then the set of surface forms that share any given violation profile may be infinite.[8]

If $(\varnothing, \{epsilon\})$ is the *only* candidate with $\varnothing$ violations then it follows that the candidate semiring is bounded and infinite surface-form-sets cannot occur. For instance, candidates with epenthetic loops will be harmonically bounded if a constraint like *Struc is present. Alternatively, if a specific machine like ⟦Eval⟧ in Fig. 9 contains no free epenthetic loops (i.e., due to the presence of Dep), then the sets of surface forms in candidates will always be finite even though the semiring admits the possibility of infinite surface-form-sets.

3.4 Generating Contenders

In this section, I present a generalization of the optimization strategy described in §3.2 that makes it possible to do optimization for multiple rankings simultaneously. If optimization is done for *all* rankings then the resulting forms are the entire set of contenders. Coupled with the assumptions in 1.2 about the active constraints, the use of complete sets of contenders in OT analyses guarantees that the analyses are stable in the sense that (i) only the constraints in *Con* determine the relative harmony of the candidates and (ii) none of the omitted candidates are relevant competitors. An additional practical benefit of the approach advocated here is that the algorithmic generation and evaluation of candidates prevents simple errors from corrupting OT analyses.[9]

3.5 Detecting (relative) harmonic bounding

As defined in (4), candidates that cannot be not optimal under any ranking of the active constraints are relatively bounded. Consider the candidates in (19):

(19)

/VC/	Onset	NoCoda	Dep	Max
a. VC.	*	*		
b. CV.			*	*
c. CV.CV.			**	
d. CV.CVC.		*	***	
e. ϵ				**

[8] Infinite summations and infinite sets of surface forms are well defined in the semiring of formal languages over Δ. These are concisely represented by regular expressions where the notation * corresponds to the loops.

[9] This latter point is the motivation behind software packages like OTSoft (Hayes et al. 2003) and OT-Help (Becker et al. 2007), which are designed to assist the researcher with reasoning about rankings. Algorithmically generating candidates and assigning violations takes the automation one step further to help prevent errors from creeping into analyses.

It is fairly easy to see that candidate d is superfluous in (19); it cannot win under any ranking of these four constraints because it has a strict superset of candidate c's violations.[10] The fact that candidate b is also bounded is far less obvious; it is doomed to perpetual sub-optimality by the combined competition from candidates c and e in what Samek-Lodovici and Prince (1999) call *collective harmonic bounding*. Fortunately, even subtle bounding can be readily detected via Recursive Constraint Demotion (RCD; Tesar 1995, Tesar and Smolensky 1996). This works roughly as follows: for each candidate k, the constraints that *favor* k are identified (i.e., those for which no competitor k' has fewer violations than k) and then all competitors that do worse than k on those constraints are thrown out. This process is iterated until no competitors remain, in which case k is a contender, or until k is not favored by any constraint, in which case k is harmonically bounded.

Prince (2002) recasts RCD as a strategy for checking the internal consistency of sets of Elementary Ranking Conditions (ERCs). ERCs are logical statements about how the constraints must be ranked for one candidate to be more harmonic than another. In (20) for example, Ons and Noc favor candidate a while Dep and Max favor b. Thus, the conditions under which b is optimal are described by the ERC "Dep or Max outranks Ons and Noc."

(20)

/VC/	Ons	Noc	Dep	Max
a. VC.	*	*		
b. CV.			*	*

Candidate fragments are denoted by (v, S) pairs where v is a violation profile and S is a set of strings. The function $erc(a, b)$ yields a pair (W, L) where W are the constraints for which candidate a has fewer violations and L are the constraints for which b has fewer violations.

(21) $erc((v_1, S_1), (v_2, S_2)) = (\{c \in Con : m_{v_1}(c) < m_{v_2}(c)\}, \{c \in Con : m_{v_1}(c) > m_{v_2}(c)\})$

The meaning of an ERC is that at least one of the constraints in the W set must outrank all of the constraints in the L set. Any ranking that meets this condition **satisfies** the ERC.

(22)

/VC/	Ons	Noc	Dep	Max
a. VC.	*	*		
b. CV.			*	*
c. CV.CV.			**	
d. ϵ				**

$erc(b, a) = (\{ons, noc\}, \{dep, max\})$
$erc(b, b) = (\{\}, \{\})$
$erc(b, c) = (\{dep\}, \{max\})$
$erc(b, d) = (\{max\}, \{dep\})$

Given a set of candidates K, like the ones in (22), the function $ercs(k, K)$ yields the set of ERCs describing the rankings under which candidate k more harmonic than each k' in K.

(23) $ercs(k, K) = \{erc(k, k') : k' \in K\}$

[10] The intuitively appropriate term 'superset' lines up neatly here with its formal definition for multisets (i.e., $A \supseteq B$ iff each element in B has equal or greater multiplicity in A: $\forall x \in B, m_B(x) \leq m_A(x)$).

For an ERC set E, the union of the left projections of the ERCs (the W sets) will be denoted $w(E)$ and the union of the right projections (the L sets) will be denoted $l(E)$.

E is **consistent** just in case there is at least one ranking that satisfies all ERCs in E. Consistency can be checked by recursively removing any $e \in E$ for which $w(\{e\}) \not\subseteq L(E)$ (i.e., any ERC that can be satisfied by ranking a constraint in $w(\{e\})$ above those in $l(E)$). This process is repeated until $l(E) = \varnothing$, which means that all the ERCs can be satisfied, or no removal is possible because $w(E) \subseteq l(E)$, which means that no ranking can satisfy E.

$$\text{(24)}\quad consistent(E) = \begin{cases} true \text{ if } l(E) = \varnothing, \text{ else} \\ false \text{ if } w(E) \subseteq l(E), \text{ else} \\ consistent(\{e \in E : w(\{e\}) \supset l(E)\}) \end{cases}$$

When an ERC set is generated for one candidate in a tableau, the process of checking consistency via the function in (24) is very similar to recursive constraint demotion. For example, if $E = ercs(b, K)$ for the candidates in (22), the computation goes as in (25).

(25)
1. $E = \{(\{ons, noc\}, \{dep, max\}), (\{\}, \{\}), (\{dep\}, \{max\}), (\{max\}, \{dep\})\}$
2. $l(E) = \{dep, max\}$
3. Is $l(E) = \varnothing$? No.
4. Is $w(E) \subseteq l(E)$? No, because $\{ons, noc\}$ from the first ERC are not in $l(E)$.
5. $E' = \{e \in E : w(\{e\}) \supset l(E)\} = \{(\{\}, \{\}), (\{dep\}, \{max\}), (\{max\}, \{dep\})\}$
6. $l(E') = \{dep, max\}$
7. Is $l(E') = \varnothing$? No.
8. Is $w(E') \subseteq l(E')$? Yes, thus E is not consistent.

After removing $(\{ons, noc\}, \{dep, max\})$ at step **5**, what's left is the empty ERC ({},{}) and a pair of ERCs that describe a circular ranking in which Dep outranks Max and Max outranks Dep. This circularity is exposed at step **8** by the fact that no ERC in E' has a constraint in its W set that is not also in the L set of another ERC.

An especially useful property of this set-up is that it is possible to add ranking conditions to the consistency check. Given a set of candidates K, if we want to know which candidates are both unbounded *and* consistent with a particular partial ranking $\mathcal{R}$, then we need only add ERCs describing $\mathcal{R}$ to the consistency check as in (26).

$$\text{(26)}\quad contenders(K, \mathcal{R}) = \{k \in K : consistent(ercs(k, K) \cup \mathcal{R})\}$$

Henceforth I assume that $\mathcal{R}$ is a set of ERCs. If $\mathcal{R}$ describes a total ranking of the constraints then $contenders(K, \mathcal{R})$ will return the candidate in K that is optimal under that ranking, if $\mathcal{R}$ is empty then $contenders(K, \mathcal{R})$ will return the candidates that are not relatively bounded, and in any other case $contenders(K, \mathcal{R})$ will return the unbounded candidates that meet the conditions in $\mathcal{R}$. This approach subsumes the optimization in §3.2 as a special

case and makes it possible to generate tableaux that represent the complete micro-typology that follows from any ranking conditions that can be specified with ERCs.

Finding contenders in candidate set K requires $|K|$ consistency checks of $|K|-1$ ERCs (setting aside the ERCs in $\mathcal{R}$). Each consistency check has at most $|Con|-1$ recursive steps, that involve computing $l(E)$, $w(E)$, and $\{e \in E : w(\{e\}) \supset l(E)\}$ which each require at most $|K| \times |Con|$ operations.[11] Thus the overall complexity is on the order of $|K|^2|Con|^2$.

Using the *contenders* function might seem like overkill in the case where $\mathcal{R}$ is a total ranking because, in this case, the optimal candidate can be found in $|K||Con|$ steps by simply listing the candidates $\langle k_1 ... k_n \rangle$ and then keeping the best in each comparison k_{i-1} vs. k_i for $i = 2 ... n$. However, when $\mathcal{R}$ is a total ranking the size of $|K|$ is always 2 when generating optimal candidates, so the difference is negligible in practice.

Finding contenders among a finite set of candidates is always possible. However, the candidate set for any underling form is generally taken to be infinite. This is where the dynamic programming approach is vital. By breaking OT optimization into a sequence of sub-problems we guarantee that the set of candidate fragments at each step is finite (and as small as possible because bounded candidates can be removed early and often).

3.6 The contender semiring

In this section I propose a generalization of the candidate semiring that makes it possible to generate sets of contenders in exactly the same way that optimal candidates were generated in §3.3. Recall that candidates are (v, S) pairs where v is drawn from V_{Con} (i.e., all violation profiles including ∞) and S is drawn from $\wp\Delta^*$ (i.e., the set of all sets of surface forms).

(27) a. The set of all sets of surface forms is $\mathbb{S} = \wp\Delta^*$

b. A **candidate set** is a function from V_{Con} to $\mathbb{S}$

c. The set of candidate sets is $\mathbb{K} = \mathbb{S}^{V_{Con}}$

d. The set of contender sets is $\mathbb{K}_{\mathcal{R}} = \{K \in \mathbb{K} : \mathit{contenders}(K, \mathcal{R}) = K\}$

In (27b), I assume that candidate sets are *functions* from violation profiles to surface forms. All that is meant by this is that there are no ties; surface forms that correspond to the same violation profile are grouped together as a single candidate. Thus, the set of candidate sets $\mathbb{K}$ is the set of all functions from violation profiles to surface-form-sets $\mathbb{S}^{V_{Con}}$. The $\min_{\mathcal{R}}$ operator can be generalized to cover sets of candidates with the definition in (28).

(28) $A \sqcup_{\mathcal{R}} B = \mathit{contenders}(A \sqcup B, \mathcal{R})$

The notation $A \sqcup B$ indicates that the candidate sets are *unified*; this is just union with the caveat that any $(v_1, S_A) \in A$ and $(v_2, S_B) \in B$ where $v_1 = v_2$ are collapsed into a single

[11] E.g. $l(E)$ is the conjunction of $|E|$ boolean vectors of length $|Con|$ describing the constraints in the L sets and E' is obtained by checking at most $|Con|$ constraints in each ERC to see if they occur in $l(E)$.

candidate $(v_1, S_A \cup S_B)$.[12] This ensures that the candidate set is a function from violation profiles to surface-form-sets and avoids duplicate computations in the contenders function.

If $\mathcal{R}$ is expressed with ERCs, there are as many $\sqcup_{\mathcal{R}}$ operators as there are ERC-sets over V_{Con}, but in every case $\sqcup_{\mathcal{R}}$ is a closed binary operator that takes two sets of candidates from $\mathbb{K}_{\mathcal{R}}$ and returns a set of candidates from $\mathbb{K}_{\mathcal{R}}$. Because union is commutative, so is $\sqcup_{\mathcal{R}}$ and because $A \sqcup_{\mathcal{R}} B \sqcup_{\mathcal{R}} C = \mathit{contenders}(A \sqcup B \sqcup C, \mathcal{R})$ the $\sqcup_{\mathcal{R}}$ operation is also associative.[13]

Candidates are constructed by merging violations and concatenating surface forms as in §3.3. Because we are now working with sets of candidates, the operation $A \times_{\mathcal{R}} B$ will create a new candidate from every pairing of a candidate from A with a candidate from B, and then unify this set to collapse any ties, and return the subset that are contenders under $\mathcal{R}$.

(29) $A \times_{\mathcal{R}} B = \mathit{contenders}\big(\bigsqcup\{(v_1 \uplus v_2, S_1 \cdot S_2) : (v_1, S_1) \in A,\ (v_2, S_2) \in B\}, \mathcal{R}\big)$

$\times_{\mathcal{R}}$ is a closed binary operator that maps a pair of contender sets to a contender set. The operator is associative because $\uplus$, $\cdot$, and $\times$ are associative, but it is not commutative.

Along with singleton sets containing the 'annihilator' candidate $(\{\infty\}, \emptyset)$ and the 'empty' candidate $(\varnothing, \{\epsilon\})$, these operators form the semiring of contenders for ERC-set $\mathcal{R}$, $\mathcal{C}_{\mathcal{R}}$.

(30) $\mathcal{C}_{\mathcal{R}} = (\mathbb{K}_{\mathcal{R}}, \sqcup_{\mathcal{R}}, \times_{\mathcal{R}}, \{(\infty, \emptyset)\}, \{(\varnothing, \{\epsilon\})\})$ obeys the following conditions:

1. $(\mathbb{K}_{\mathcal{R}}, \sqcup_{\mathcal{R}}, \{(\infty, \emptyset)\})$ is a commutative monoid with $\{(\infty, \emptyset)\}$ as identity,
   $\forall\, a, b, c \in \mathbb{K}_{\mathcal{R}}$, $\sqcup_{\mathcal{R}}$ is associative: $(a \sqcup_{\mathcal{R}} b) \sqcup_{\mathcal{R}} c = a \sqcup_{\mathcal{R}} (b \sqcup_{\mathcal{R}} c)$
   $\sqcup_{\mathcal{R}}$ is commutative: $(a \sqcup_{\mathcal{R}} b) = (b \sqcup_{\mathcal{R}} a)$
   $a \sqcup_{\mathcal{R}} \{(\infty, \emptyset)\} = \{(\infty, \emptyset)\} \sqcup_{\mathcal{R}} a = a$
2. $(\mathbb{K}, \times_{\mathcal{R}}, \{(\varnothing, \{\epsilon\})\})$ is a monoid with $\{(\varnothing, \{\epsilon\})\}$ as identity,
   $\forall\, a, b, c \in \mathbb{K}_{\mathcal{R}}$, $\times_{\mathcal{R}}$ is associative: $(a \times_{\mathcal{R}} b) \times_{\mathcal{R}} c = a \times_{\mathcal{R}} (b \times_{\mathcal{R}} c)$
   $a \times_{\mathcal{R}} \{(\varnothing, \{\epsilon\})\} = \{(\varnothing, \{\epsilon\})\} \times_{\mathcal{R}} a = a$
3. $\times_{\mathcal{R}}$ distributes over $\sqcup_{\mathcal{R}}$,
   $\forall\, a, b, c \in \mathbb{K}_{\mathcal{R}}$, $(a \sqcup_{\mathcal{R}} b) \times_{\mathcal{R}} c = (a \times_{\mathcal{R}} c) \sqcup_{\mathcal{R}} (b \times_{\mathcal{R}} c)$,
   $c \times_{\mathcal{R}} (a \sqcup_{\mathcal{R}} b) = (c \times_{\mathcal{R}} a) \sqcup_{\mathcal{R}} (c \times_{\mathcal{R}} b)$,
   $(a \sqcup_{\mathcal{R}} b) \times_{\mathcal{R}} c = (a \times_{\mathcal{R}} c) \sqcup_{\mathcal{R}} (b \times_{\mathcal{R}} c)$
4. $\{(\infty, \emptyset)\}$ is an annihilator for $\times_{\mathcal{R}}$,
   $\forall\, a \in \mathbb{K}_{\mathcal{R}}$, $a \times_{\mathcal{R}} (\{\infty\}, \{\}) = (\{\infty\}, \emptyset) \times_{\mathcal{R}} a = (\{\infty\}, \emptyset)$.

The contenders semiring is not commutative because the order matters for the $\times_{\mathcal{R}}$ operator, but it is idempotent (i.e., $x \sqcup_{\mathcal{R}} x = x$ for all $x \in \mathbb{K}_{\mathcal{R}}$). The idempotency follows from the distributivity law and the fact that $\{(\varnothing, \{\epsilon\})\} \sqcup_{\mathcal{R}} \{(\varnothing, \{\epsilon\})\} = \{(\varnothing, \{\epsilon\})\}$. This means that $\mathcal{C}_{\mathcal{R}}$ is monotonic and can be used in the dynamic programming strategy given in §3.2.

[12] Unification of candidate sets is a particularly simple case of the kind of unification used in Feature Unification Grammar, Lexical-Functional Grammar, and Generalized Phrase Structure Grammar, (cf. Kay (1979), Bresnan (1982), Gazdar et al. (1985)) in which the profiles in V_{Con} are the labels, $\mathbb{S}$ are the values, all of which are both atomic and compatible via union.

[13] Associativity follows from the fact that the boundedness of $x \in X$ is unaffected by the removal of any $y \in X$ that is harmonically bounded. For more on this, see Samek-Lodovici & Prince (2002).

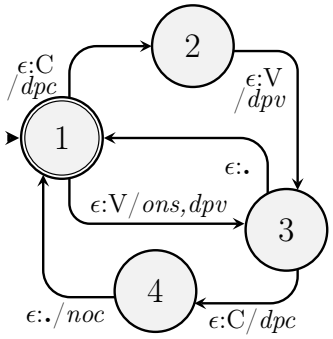
Though the $\bar{0}$ element $\{(\infty, \emptyset)\}$ is universally worse than every candidate set in $\mathbb{K}_{\mathcal{R}}$, the contender semiring is *not* bounded because $\bar{1}$ is not universally better in the sense that $\{(\varnothing, \{\epsilon\})\}$ is not an annihilator for $\sqcup_{\mathcal{R}}$ (i.e., other candidates can also have $\varnothing$ violations). As with the candidate semiring in §3.3, allowing infinite surface-form-sets is not problematic; the surface-sets can be represented with regular expressions which are well behaved under concatenation in the $\times_{\mathcal{R}}$ operation and subset-of-union in the $\sqcup_{\mathcal{R}}$ operation. Though these sets are formally well defined, they are hard to relate to real-world linguistic data.

If candidates are instead defined as $(v \in V_{Con},\ S \in \wp\Delta^*)$ pairs where $|S| \in \mathbb{N}$ then we have another contender semiring, $\ddot{\mathcal{C}}_{\mathcal{R}} \subset \mathcal{C}_{\mathcal{R}}$, in which surface-sets are finite. As in §3.3, if one assumes that $(\varnothing, \{\epsilon\})$ is the only candidate with $\varnothing$ violations (e.g. by virtue of *Struc), then it follows immediately that $\ddot{\mathcal{C}}_{\mathcal{R}}$ is bounded because $a \sqcup_{\mathcal{R}} \{(\varnothing, \{\epsilon\})\} = \{(\varnothing, \{\epsilon\})\}$ for all a. This is sometimes called the *economy* principle and it has analogs in the syntactic literature (e.g. Chomsky 1991). This idea has, however, been criticized in the domains of syntax and phonology by Grimshaw (2001) and Gouskova (2003) with the argument that economy should not be reified as an explicit mechanism of grammar but rather should emerge as a consequence of other mechanisms. Gouskova also points out that *Struc makes odd predictions when highly ranked. With respect to the latter critique, it is noteworthy that *Struc will render $\ddot{\mathcal{C}}_{\mathcal{R}}$ bounded even if it is universally dominated by all other constraints.

Even without the economy assumption, if a particular weighted automaton M has no free epenthetic loops (e.g. [[Eval]] in Fig. 9), then all candidates in $[\![\mathrm{M}]\!](x)$ are guaranteed to stay within $\ddot{\mathcal{C}}_{\mathcal{R}}$. Mohri (2002) calls weighted automata with this property **regulated**.

3.7 Optimization for all rankings simultaneously

Figure 18 presents the set of arcs in [[Eval]] whose input label is the empty string along with the adjacency matrix $\underset{\rightarrow}{\epsilon}$ containing the violation profiles and surface forms for the ϵ-arcs.



	1	2	3	4
1	$\varnothing, \{\epsilon\}$	$\wr dpc\wr, \{\mathrm{C}\}$	$\wr \overset{ons}{dpv} \wr, \{\mathrm{V}\}$	$\infty, \emptyset$
2	$\infty, \emptyset$	$\varnothing, \{\epsilon\}$	$\wr dpv\wr, \{\mathrm{V}\}$	$\infty, \emptyset$
3	$\varnothing, \{\epsilon\}$	$\infty, \emptyset$	$\varnothing, \{\epsilon\}$	$\wr dpc\wr, \{\mathrm{C}\}$
4	$\wr noc\wr, \{\bullet\}$	$\infty, \emptyset$	$\infty, \emptyset$	$\varnothing, \{\epsilon\}$

Figure 18: [[Eval]] for the input $/\epsilon/$ and the adjacency matrix for ϵ

As in §3.2, a Floyd-Warshal-style update rule can be used to find the optimal paths between every pair of states in [[Eval]]. When generating contenders, the notion of 'optimal' is more loosely construed to mean any path that is a contender. The update rule in (31) will generate sets of contenders.

(31) **The contenders update rule**
For each $y \in Q$, for each $(x, z) \in Q \times Q$: set $M_{x,z} = M_{x,z} \sqcup_{\mathcal{R}} (M_{x,y} \times_{\mathcal{R}} M_{y,z})$.

Iteratively applying the update rule to the matrix in Fig. 18 produces a matrix that encodes the contender candidates (under $\mathcal{R}$) for getting from any state in [[EVAL]] to any other state in [[EVAL]] by a sequence of insertions. This matrix, which I will call $\lambda_{\mathcal{R}}$, is given in Fig. 19.

$$\begin{bmatrix} \{(\varnothing, \{\epsilon\})\} & \{(\{dpc\}, \{C\})\} & \left\{ \begin{matrix} (\{ons, dpv\}, \{V\}) \\ (\{dpv, dpc\}, \{CV\}) \end{matrix} \right\} & \left\{ \begin{matrix} (\{ons, dpv, dpc\}, \{VC\}) \\ (\{dpv, 2dpc\}, \{CVC\}) \end{matrix} \right\} \\ \{(\{dpv\}, \{V.\})\} & \{(\varnothing, \{\epsilon\})\} & \{(\{dpv\}, \{V\})\} & \{(\{dpv, dpc\}, \{VC\})\} \\ \{(\varnothing, \{.\})\} & \{(\{dpc\}, \{.C\})\} & \{(\varnothing, \{\epsilon\})\} & \{(\{dpc\}, \{C\})\} \\ \{(\{noc\}, \{.\})\} & \{(\{noc, dpc\}, \{.C\})\} & \left\{ \begin{matrix} (\{ons, noc, dpv\}, \{.V\}) \\ (\{noc, dpv, dpc\}, \{.CV\}) \end{matrix} \right\} & \{(\varnothing, \{\epsilon\})\} \end{bmatrix}$$

Figure 19: $\lambda_{\mathcal{R}}$, the lambda matrix with contenders for $\mathcal{R} = \varnothing$ (i.e., all contenders)

In each of the $|Q|^3$ applications of the contenders update rule, the contenders function is used twice, once for $\sqcup_{\mathcal{R}}$ and once for $\times_{\mathcal{R}}$. Thus the cost of computing $\lambda_{\mathcal{R}}$ is dominated by $2|Q|^3$ applications of the *contenders* function.

As in §3.2, optimization is carried out via matrix multiplication. For two matrices of contenders $A \in \mathbb{K}_{\mathcal{R}}^{(m \times n)}$, and $B \in \mathbb{K}_{\mathcal{R}}^{(n \times p)}$, the product $[AB]_{\mathcal{R}} = C \in \mathbb{K}_{\mathcal{R}}^{(m \times p)}$, where the value $C_{i,j} = \big((A_{i,1} \times_{\mathcal{R}} B_{1,j}) \sqcup_{\mathcal{R}} \ldots \sqcup_{\mathcal{R}} (A_{i,n} \times_{\mathcal{R}} B_{n,j})\big)$. As before, matrix products are denoted by juxtaposition, but in this case the annotation $[\,]_{\mathcal{R}}$ indicates the $\mathcal{R}$ for $\sqcup_{\mathcal{R}}$, $\times_{\mathcal{R}}$, and $\mathbb{K}_{\mathcal{R}}$.

Contenders for the underlying form /VC/ under ranking conditions $\mathcal{R}$ are generated by the product $\big[\lambda \underset{v}{\rightarrow} \lambda \underset{c}{\rightarrow} \lambda\big]_{\mathcal{R}}$. For the case $\mathcal{R} = \varnothing$ (i.e., no ranking conditions imposed), the value of $VC_{1,1}$ is given on the left in Fig. 20. Because state 1 is the start and the only final state, these are the contenders among *all* complete candidates in [[EVAL]](VC) and thus they provide the candidates for the familiar OT tableau on the right in Fig. 20.

$VC_{1,1} =$

$$\left\{ \begin{matrix} (\{max, dpv\}, \{\text{CV.}\}) \\ (\{max, dpc\}, \{\text{CV.}\}) \\ (\{ons, noc\}, \{\text{VC.}\}) \\ (\{dpv, dpc\}, \{\text{CV.CV.}\}) \\ (\{ons, dpv\}, \{\text{V.CV.}\}) \\ (\{2max\}, \{\epsilon\}) \\ (\{ons, max\}, \{\text{V.}\}) \\ (\{noc, dpc\}, \{\text{CVC.}\}) \end{matrix} \right\}$$

/VC/	ONSET	NOC	MAX	DEPV	DEPC
a. CV.			*	*	
b. CV.			*		*
c. VC.	*	*			
d. CV.CV.				*	*
e. V.CV.	*			*	
f. ∅			**		
g. V.	*		*		
h. CVC.		*			*

Figure 20: All contenders for the underlying form /VC/ under $\mathcal{R} = \varnothing$

The complexity of contender generation is basically identical to that of optimization in §3.2 save for the fact that the computation is now dominated by calls to the *contenders* function.

For an underlying form of length n, if matrices of contenders are computed for $\lambda_{\mathcal{R}}$ and each input segment ahead of time, then contender generation requires $n|Q|^2$ calls to *contenders*, otherwise it requires $2n|Q|^2 + |Q|^3$. In either case, the number of calls is linear in the length of the underlying form.

The cost of each call can be quite large because there can be $|Con|!$ contenders. This value is a constant for any given constraint set and it is the bound on the size of the answer to the question: *What are the non-harmonically-bounded candidates for input* x*?* Thus, it is reasonable to pull this factor out in evaluating the efficiency of algorithms for generating contenders. Put differently, though the number of contenders can be astronomical, it is important to know how hard it is to generate n contenders for the cases where n is manageable.

3.8 Recursive typology construction

The tableau in Fig. 20 is essentially a micro-typology consisting of the eight 'languages' realized by the underlying form /VC/ for the constraints used to build $[\![\text{Eval}]\!]$. This general approach can be straightforwardly extended to sets of tableaux to generate the typology that is realized by any given lexicon of underlying forms.

I will take a typology $\mathcal{T}$ to be a set of pairs $(\mathcal{E}, L)$ where $\mathcal{E}$ is a set of ERCs and L is a **language** comprising a set of (i, k) pairs in which i is an underlying form and $k = (v, S)$ is the candidate (violation-profile, surface-form-set) that is optimal for i under the conditions described by $\mathcal{E}$. The recursive strategy in (32) will construct such a typology.

(32) **Recursive typology construction** – Given a V_{Con}-weighted transducer $[\![\text{Eval}]\!]$, an ERC-set $\mathcal{R}$ over Con, and $Lexicon \subset \Sigma^*$, a typology $\mathcal{T}$ can be constructed as follows:
- a. The 'base' typology $\mathcal{T}$ is $\{(\mathcal{R}, \emptyset)\}$.
- b. For each underlying form $i \in Lexicon$:
- c. for each $(\mathcal{E}, L) \in \mathcal{T}$, remove $(\mathcal{E}, L)$ from $\mathcal{T}$,
- d. then for each contender $k \in K = [\![\text{Eval}]\!]_{\mathcal{R}}(i)$,
- e. if $\hat{\mathcal{E}} = ercs(k, K) \cup \mathcal{E}$ is a *consistent* ERC-set,
- f. add language $\hat{L} = L \cup \{(i, k)\}$: $\mathcal{T} = \mathcal{T} \cup \{(\hat{\mathcal{E}}, \hat{L})\}$.

The complexity of this procedure depends on the number of contenders for each input and the number of points in the typology which are both bounded at $k!$ for k active constraints. This highlights the utility of focusing on a specific set of active constraints. That is, fixing the rankings for most of Con relative to a set of active constraints Con, can make it feasible to actually generate typologies that focus on the linguistic variation predicted by specific constraint interactions. Furthermore, building typologies from contenders is far more likely to reveal unexpected predictions and constraint interactions than building them from handcrafted sets of tableaux.

4 Conclusions

Without some restriction on what kinds of formal objects constraints are, we cannot know whether optimization can be done efficiently or even whether it is computable. A common solution to this quandary among computational phonologists has been to assume that *Gen* and all the constraints are rational (i.e., representable by weighted finite state automata). The approach here is slightly more nuanced. Without making assumptions about the formal complexity of *all* the constraints in the universal set Con, the complexity of optimization can be evaluated for specific sets of active constraints that are drawn from a specified complexity-class.

For rational constraint sets, the complexity of generating optimal candidates is a linear function of the length of the underlying form with a multiplicative constant $(|E|, |Q|)$ provided by the size of the finite state representation of the set of constraints. If the constraint set in question is the entire rational subset of the universal constraint set, then, even though it is a constant, $(|E|, |Q|)$ is likely to be so large as to preclude any practical strategy for optimization. On the other hand, if the active constraints are all rational and $(|E|, |Q|)$ is not too large then optima and contenders can be feasibly generated.

The approach presented here can be readily extended beyond finite-state constraints to those representable with context-free expressions. To do this, a chart-parsing strategy like the one Goodman (1998) uses to generate the n-most likely parses in a probabilistic context free grammar can be straightforwardly adapted to the OT case. All that is needed is to replace the function that selects the n-most-likely parse-fragments at each step with a function that selects the parse-fragments that are contenders (and, of course, the probability semiring is replaced with the violation semiring).

Using contenders can ensure that OT analyses are valid by guaranteeing that the conclusions follow from the premises, the premises in this case being the assumptions about the underlying forms and the active constraints. However, this does not guarantee that the analyses are sound, because that will depend ultimately on factors like having made the right assumptions about the underlying forms and active constraints.

References

Barton, Jr., G. Edward, Robert C. Berwick, & Eric Sven Ristad (1987) *Computational Complexity and Natural Language*. Cambridge, MA: MIT Press.

Becker, Michael, Joe Pater, & Christopher Potts (2007) OT-Help 1.2 software package, University of Massachusetts, Amherst.

Bresnan, Joan (1982) Control and Complementation. *Linguistic Inquiry* **13**: 343–434.

Chomsky, Noam (1991) *Some notes on economy of derivation and representation*, chap. 14. Cambridge, Mass.: MIT Press, 417–454.

Cormen, Leiserson, & Rivest (1990) *Introduction to Algorithms*. Cambridge Mass.: MIT Press.

Dijkstra, Edsger. W. (1959) A note on two problems in connexion with graphs. *Numerische Mathematik* **1**: 269–271.

Droste, Manfred & Werner Kuich (2007) Semirings and formal power series. In *Handbook of Weighted Automata*, Manfred Droste, Werner Kuich, & Heiko Vogler, eds., Tübingen: Springer-Verlag, 1–26.

Eisner, Jason (1997a) Efficient Generation in Primitive Optimality Theory. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, Madrid, 313–320.

Eisner, Jason (1997b) What Constraints Should OT Allow? Talk handout available online (22 pages), Linguistic Society of America (LSA), Chicago.

Ellison, T. Mark (1994a) Phonological derivation in optimality theory. In *Proceedings of the 15th conference on Computational linguistics*, Morristown, NJ, USA: Association for Computational Linguistics, 1007–1013.

Ellison, T. Mark (1994b) Phonological derivation in optimality theory. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, Kyoto, 1007–1013.

Fink, E. (1992) A survey of sequential and systolic algorithms for the algebraic path problem.

Gazdar, Gerald, Ewan Klein, Geoffrey K. Pullum, & Ivan Sag (1985) *Generalized Phrase Structure Grammar Cambridge*. Harvard University Press.

Gerdemann, Dale & Gertjan van Noord (2000) Approximation and Exactness in Finite State Optimality Theory. In *Coling Workshop Finite State Phonology*, Luxembourg.

Goldsmith, John (1993) *Harmonic phonology*. Chicago: University of Chicago Press, 221–269.

Goodman, J. (1998) *Parsing Inside-Out*. Ph.D. thesis, Harvard University, Harvard.

Gouskova, Maria (2003) *Deriving Economy: Syncope in Optimality Theory*. Ph.D. thesis, University of Massachusetts Amherst.

Grimshaw, Jane (2001) Economy of structure in ot.

Hayes, Bruce, Bruce Tesar, & Kie Zuraw (2003) OTSoft 2.3 software package: /www.linguistics.ucla.edu/people/hayes/otsoft/.

Heinz, Jeffrey, Gregory Kobele, & Jason Riggle (2009) Evaluating the complexity of Optimality Theory. *Linguistic Inquiry* **40**: 277–288, ROA 968-0508.

Hopcroft, John E. & Jeffrey D. Ullman (1979) *Introduction to automata theory, languages, and computation*. Reading, Mass.: Addison-Wesley, 78067950 John E. Hopcroft, Jeffrey D. Ullman. Addison-Wesley series in computer science. Includes index. Bibliography: p. 396-410.

Idsardi, William J. (2006) A Simple Proof That Optimality Theory Is Computationally Intractable. *Linguistic Inquiry* **37**(2): 271–275.

Kay, Martin (1979) Functional Grammar. In *BLS-79*, Berkeley, CA, 142–158.

Kay, Martin (1980) Algorithm schemata and data structures in syntactic processing. Tech. Rep. CSL-80-12,, Xerox PARC, Xerox PARC, Palo Alto, CA.

Kuich, Werner (1997) Semirings : A basis for a mathematical automata and language theory. In *Developments in Language Theory*, 49–60.

Legendre, Géraldine, Yoshiro Miyata, & Paul Smolensky (1990) Harmonic Grammar—A Formal Multi-Level Connectionist Theory of Linguistic Well-Formedness: Theoretical Foundations. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum Associates, 388–395.

Mohri, Mehryar (2002) Semiring Frameworks and Algorithms for Shortest-Distance Problems. *Journal of Automata, Languages and Combinatorics* **7**(3): 321–350.

Papadimitriou, Christon (1994) *Computational Complexity*. Addison Wesley.

Pater, Joe, Rajesh Bhatt, & Christopher Potts (2007a) Linguistic Optimization. Ms., ms. UMASS Amherst.

Pater, Joe, Christopher Potts, & Rajesh Bhatt (2007b) Harmonic Grammar with Linear Programming, ms.

Prince, Alan (2002) Entailed Ranking Arguments. *Rutgers Optimality Archive* **500**: 1–117, rOA-500. `roa.rutgers.edu`.

Prince, Alan & P. Smolensky (1993) *Optimality Theory*. Ph.D. thesis, Rutgers University and University of Colorado.

Riggle, Jason (2004) *Generation, Recognition, and Learning in Finite State Optimality Theory*. Ph.D. thesis, University of California, Los Angeles.

Riggle, Jason (2009) Violation Semirings in Optimality Theory. *Research on Language & Computation* **July 2009**: 1570–7075.

Roche, Emmanuel & Yves Schabes (1997) *Finite-State Language Processing*. Cambridge (MA): MIT Press.

Samek-Lodovici, Vieri (1992) *Universal constraints and morphological gemination: A crosslinguistic study*. Ph.D. thesis, Brandeis University.

Samek-Lodovici, Vieri & Alan Prince (1999) Optima, ms.

Samek-Lodovici, Vieri & Alan Prince (2002) Fundamental Properties of Harmonic Bounding. Tech. rep., Rutgers Center for Cognitive Studies, rutgers Center for Cognitive Science, RuCCS-TR-71.

Simon, Imre (1988) Recognizable Sets with Multiplicities in the Tropical Semiring. In *MFCS '88: Proceedings of the Mathematical Foundations of Computer Science 1988*, London, UK: Springer-Verlag, 107–120.

Smolensky, Paul & Géraldine Legendre (2006) *The Harmonic Mind: From Neural Computation to Optimality-Theoretic GrammarVolume I: Cognitive Architecture (Bradford Books)*. The MIT Press.

Tesar, Bruce (1995) *Computational Optimality Theory*. Ph.D. thesis, University of Colorado.

Tesar, Bruce & Paul Smolensky (1996) Learnability in Optimality Theory (long version). Tech. rep., The Center for Cognitive Science/Linguistics Department, Rutgers University.

Wareham, H.T. (1998) *Systematic Parameterized Complexity Analysis in Computational Phonology*. Ph.D. thesis, University of Victoria.