Proceedings of the 35th Annual ACL and 8th European ACL, 313-320, Madrid, July 1997. [With minor corrections.]

Efficient Generation in Primitive Optimality Theory

Jason Eisner
Dept. of Computer and Information Science
University of Pennsylvania
200 S. 33rd St., Philadelphia, PA 19104-6389, USA
jeisner@linc.cis.upenn.edu

Abstract

This paper introduces primitive Optimality Theory (OTP), a linguistically motivated formalization of OT. OTP specifies the class of autosegmental representations, the universal generator Gen, and the two simple families of permissible constraints. In contrast to less restricted theories using Generalized Alignment, OTP's optimal surface forms can be generated with finite-state methods adapted from (Ellison, 1994). Unfortunately these methods take time exponential on the size of the grammar. Indeed the generation problem is shown NP-hard in this sense. However, techniques are discussed for making Ellison's approach fast in the typical case, including a simple trick that alone provides a 100-fold speedup on a grammar fragment of moderate size. One avenue for future improvements is a new finite-state notion, "factored automata," where regular languages are represented compactly via formal intersections $\cap_{i=1}^{k} A_i$ of FSAs.

1 Why formalize OT?

Phonology has recently undergone a paradigm shift. Since the seminal work of (Prince & Smolensky, 1993), phonologists have published literally hundreds of analyses in the new constraint-based framework of Optimality Theory, or OT. Old-style derivational analyses have all but vanished from the linguistics conferences.

The price of this creative ferment has been a certain lack of rigor. The claim for OT as Universal Grammar is not substantive or falsifiable without formal definitions of the putative Universal Grammar objects **Repns**, **Con**, and **Gen** (see below). Formalizing OT is necessary not only to flesh it out as a linguistic theory, but also for the sake of computational phonology. Without knowing what classes of constraints may appear in grammars, we can say only so much about the properties of the system, or about algorithms for generation, comprehension, and learning.

The central claim of OT is that the phonology of any language can be naturally described as *successive filtering*. In OT, a phonological grammar for a language consists of a vector $C_1, C_2, \ldots C_n$ of soft **constraints** drawn from a universal fixed set **Con**. Each constraint in the vector is a function that scores possible output representations (surface forms):

(1) $C_i : \mathbf{Repns} \rightarrow \{0, 1, 2, \ldots\} \qquad (C_i \in \mathbf{Con})$

If $C_i(R) = 0$, the output representation R is said to **satisfy** the ith constraint of the language. Otherwise it is said to **violate** that constraint, where the value of $C_i(R)$ specifies the degree of violation. Each constraint yields a filter that permits only minimal violation of the constraint:

(2) $\text{Filter}_i(Set) = \{R \in Set : C_i(R) \text{ is minimal}\}$

Given an underlying phonological input, its set of legal surface forms under the grammar—typically of size 1—is just

(3) $\text{Filter}_n(\cdots \text{Filter}_2(\text{Filter}_1(\mathbf{Gen}(input))))$

where the function **Gen** is fixed across languages and $\mathbf{Gen}(input) \subseteq \mathbf{Repns}$ is a potentially infinite set of candidate surface forms.

In practice, each surface form in $\mathbf{Gen}(input)$ must contain a silent copy of *input*, so the constraints can score it on how closely its pronounced material matches *input*. The constraints also score other criteria, such as how easy the material is to pronounce. If C_1 in a given language is violated by just the forms with coda consonants, then $\text{Filter}_1(\mathbf{Gen}(input))$ includes only coda-free candidates—regardless of their other demerits, such as discrepancies from *input* or unusual syllable structure. The remaining constraints are satisfied only as well as they can be given this set of survivors. Thus, when it is impossible to satisfy all constraints at once, successive filtering means early constraints take priority.

Questions under the new paradigm include these:

- *Generation.* How to implement the input-output mapping in (3)? A brute-force approach fails to terminate if **Gen** produces infinitely many candidates. Speakers must solve this problem. So must linguists, if they are to know what their proposed grammars predict.

- *Comprehension.* How to invert the input-output mapping in (3)? Hearers must solve this.
- *Learning.* How to induce a lexicon and a phonology like (1) for a particular language, given the kind of evidence available to child language learners?

None of these questions is well-posed without restrictions on **Gen** and **Con**.

In the absence of such restrictions, computational linguists have assumed convenient ones. Ellison (1994) solves the generation problem where **Gen** produces a regular set of strings and **Con** admits all finite state transducers that can map a string to a number in unary notation. (Thus $C_i(R) = 4$ if the C_i transducer outputs the string 1111 on input R.) Tesar (1995, 1996) extends this result to the case where **Gen**(*input*) is the set of parse trees for *input* under some context-free grammar (CFG).[1] Tesar's constraints are functions on parse trees such that $C_i([\mathrm{A}\ [\mathrm{B}_1 \cdots]\ [\mathrm{B}_2 \cdots]])$ can be computed from A, B_1, B_2, $C_i(\mathrm{B}_1)$, and $C_i(\mathrm{B}_2)$. The optimal tree can then be found with a standard dynamic-programming chart parser for weighted CFGs.

It is an important question whether these formalisms are useful in practice. On the one hand, are they expressive enough to describe real languages? On the other, are they restrictive enough to admit good comprehension and unsupervised-learning algorithms?

The present paper sketches **primitive Optimality Theory** (OTP)—a new formalization of OT that is explicitly proposed as a linguistic hypothesis. Representations are autosegmental, **Gen** is trivial, and only certain simple and phonologically local constraints are allowed. I then show the following:

1. *Good news:* Generation in OTP can be solved attractively with finite-state methods. The solution is given in some detail.
2. *Good news:* OTP usefully restricts the space of grammars to be learned. (In particular, Generalized Alignment is outside the scope of finite-state or indeed context-free methods.)
3. *Bad news:* While OTP generation is close to linear on the size of the input form, it is NP-hard on the size of the grammar, which for human languages is likely to be quite large.
4. *Good news:* Ellison's algorithm can be improved so that its exponential blowup is often avoided.

[1]This extension is useful for OT syntax but may have little application to phonology, since the context-free case reduces to the regular case (i.e., Ellison) unless the CFG contains recursive productions.

2 Primitive Optimality Theory

Primitive Optimality Theory, or OTP, is a formalization of OT featuring a homogeneous output representation, extremely local constraints, and a simple, unrestricted **Gen**. Linguistic arguments for OTP's constraints and representations are given in (Eisner, 1997), whereas the present description focuses on its formal properties and suitability for computational work. An axiomatic treatment is omitted for reasons of space. Despite its simplicity, OTP appears capable of capturing virtually all analyses found in the (phonological) OT literature.

2.1 Repns: Representations in OTP

To represent [mp], OTP uses not the autosegmental representation in (4a) (Goldsmith, 1976; Goldsmith, 1990) but rather the simplified autosegmental representation in (4b), which has no association lines. Similarly (5a) is replaced by (5b). The central representational notion is that of a **constituent timeline**: an infinitely divisible line along on which constituents are laid out. Every constituent has width and edges.

(4) a.
```
      voi
  nas/
   |/
   C   C
    \ /
    lab
```
b.
$$\begin{array}{llll} {}_{voi}[&]_{voi} & & \\ {}_{nas}[&]_{nas} & & \\ {}_{C}[& {}_{C}]{}_{C}[& &]_{C} \\ {}_{lab}[& & &]_{lab} \end{array}$$
⟶⟶timeline⟶⟶

For phonetic interpretation: $]_{voi}$ says to end voicing (laryngeal vibration). At the same instant, $]_{nas}$ says to end nasality (raise velum).

(5) a.
```
   σ   σ
  /|\ /|
  C V C V
```
b.
$$\begin{array}{llllllll} {}_{\sigma}[& & & & {}_{\sigma}]{}_{\sigma}[& & &]_{\sigma} \\ {}_{C}[&]_{C} & {}_{C}[& & &]_{C} & & \\ & {}_{V}[&]_{V} & & & {}_{V}[&]_{V} & \end{array}$$

A timeline can carry the full panoply of phonological and morphological constituents—anything that phonological constraints might have to refer to. Thus, a timeline bears not only autosegmental features like nasal gestures [*nas*] and prosodic constituents such as syllables [σ], but also stress marks [x], feature domains such as [ATR*dom*] (Cole & Kisseberth, 1994) and morphemes such as [*Stem*]. All these constituents are formally identical: each marks off an interval on the timeline. Let **Tiers** denote the fixed finite set of constituent types, {*nas*, σ, x, ATR*dom*, *Stem*, ...}.

It is always possible to recover the old representation (4a) from the new one (4b), under the convention that two constituents on the timeline are linked if their interiors overlap (Bird & Ellison, 1994). The **interior** of a constituent is the open interval that excludes its edges. Thus, *lab* is linked to both consonants C in (4b), but the two consonants are not linked to each other, because their interiors do not overlap.

By eliminating explicit association lines, OTP eliminates the need for faithfulness constraints on them, or for well-formedness constraints against gapping or crossing of associations. In addition, OTP can refer naturally to the edges of syllables (or morphemes). Such edges are tricky to define in (5a), because a syllable's features are scattered across multiple tiers and perhaps shared with adjacent syllables.

In diagrams of timelines, such as (4b) and (5b), the intent is that only horizontal order matters. Horizontal spacing and vertical order are irrelevant. Thus, a timeline may be represented as a finite collection S of labeled edge brackets, equipped with ordering relations $\prec$ and $\simeq$ that indicate which brackets precede each other or fall in the same place.

Valid timelines (those in **Repns**) also require that edge brackets come in matching pairs, that constituents have positive width, and that constituents of the same type do not overlap (i.e., two constituents on the same tier may not be linked).

2.2 Gen: Input and output in OTP

OT's principle of Containment (Prince & Smolensky, 1993) says that each of the potential outputs in **Repns** includes a silent copy of the input, so that constraints evaluating it can consider the goodness of match between input and output. Accordingly, OTP represents both input and output constituents on the constituent timeline, but on different tiers. Thus surface nasal autosegments are bracketed with ${}_{nas}[$ and $]_{nas}$, while underlying nasal autosegments are bracketed with ${}_{\underline{nas}}[$ and $]_{\underline{nas}}$. The underlining is a notational convention to denote input material. No connection is required between $[nas]$ and $[\underline{nas}]$ except as enforced by constraints that prefer $[nas]$ and $[\underline{nas}]$ or their edges to overlap in some way. (6) shows a candidate in which underlying $[\underline{nas}]$ has surfaced "in place" but with rightward spreading.

(6)
$$\begin{array}{lll} {}_{nas}[& &]_{nas} \\ {}_{\underline{nas}}[&]_{\underline{nas}} & \end{array}$$

Here the left edges and interiors overlap, but the right edges fail to. Such overlap of interiors may be regarded as featural Input-Output Correspondence in the sense of (McCarthy & Prince, 1995).

The lexicon and morphology supply to **Gen** an **underspecified timeline**—a partially ordered collection of *input* edges. The use of a *partial* ordering allows the lexicon and morphology to supply floating tones, floating morphemes and templatic morphemes.

Given such an underspecified timeline as lexical input, **Gen** outputs the set of *all* fully specified timelines that are consistent with it. No new input constituents may be added. In essence, **Gen** generates every way of refining the partial order of input constituents into a total order and decorating it freely with output constituents. Conditions such as the prosodic hierarchy (Selkirk, 1980) are enforced by universally high-ranked constraints, not by **Gen**.[2]

2.3 Con: The primitive constraints

Having described the representations used, it is now possible to describe the constraints that evaluate them. OTP claims that **Con** is restricted to the following two families of **primitive constraints**:

(7) $\alpha \rightarrow \beta$ ("implication"):
"Each α temporally overlaps some β."

Scoring: Constraint(R) = number of α's in R that do not overlap any β.

(8) $\alpha \perp \beta$ ("clash"):
"Each α temporally overlaps no β."

Scoring: Constraint(R) = number of (α, β) pairs in R such that the α overlaps the β.

That is, $\alpha \rightarrow \beta$ says that α's attract β's, while $\alpha \perp \beta$ says that α's repel β's. These are simple and arguably natural constraints; no others are used.

In each primitive constraint, α and β each specify a phonological event. An event is defined to be either a type of labeled edge, written e.g. ${}_{\sigma}[$, or the interior (excluding edges) of a type of labeled constituent, written e.g. σ. To express some constraints that appear in real phonologies, it is also necessary to allow α and β to be non-empty conjunctions and disjunctions of events. However, it appears possible to limit these cases to the forms in (9)–(10). Note that other forms, such as those in (11), can be decomposed into a sequence of two or

[2] The formalism is complicated slightly by the possibility of deleting segments (syncope) or inserting segments (epenthesis), as illustrated by the candidates below.

(i) Syncope ($\underline{CVC} \Rightarrow CC$): the $\underline{V}$ is crushed to zero width so the C's can be adjacent.

$$\begin{array}{lll} {}_{C}[&]_{C} &]_{C} \\ {}_{\underline{C}}[&]_{\underline{C}} &]_{\underline{C}} \\ & {}_{\underline{V}}|_{\underline{V}} & \end{array}$$

(ii) Epenthesis ($\underline{CC} \Rightarrow CVC$): the $\underline{C}$'s are pushed apart.

$$\begin{array}{lllll} & {}_{V}[& &]_{V} & \\ {}_{C}[&]_{C} & {}_{C}[& &]_{C} \\ {}_{\underline{C}}[&]_{\underline{C}} & {}_{\underline{C}}[& &]_{\underline{C}} \end{array}$$

In order to allow adjacency of the surface consonants in (i), as expected by assimilation processes (and encouraged by a high-ranked constraint), note that the underlying vowel must be allowed to have zero width—an option available to to input but not output constituents. The input representation must specify only ${}_{\underline{V}}[\preceq]_{\underline{V}}$, not ${}_{\underline{V}}[\prec]_{\underline{V}}$. Similarly, to allow (ii), the input representation must specify only $]_{\underline{C_1}} \preceq {}_{\underline{C_2}}[$, not $]_{\underline{C_1}} \simeq {}_{\underline{C_2}}[$.

more constraints.[3]

(9) (α_1 and α_2 and ...) $\rightarrow$ (β_1 or β_2 or ...)
Scoring: Constraint(R) = number of vectors of events $\langle A_1, A_2, \ldots\rangle$ of types $\alpha_1, \alpha_2, \ldots$ respectively that all overlap on the timeline and whose intersection does not overlap any event of type β_1 or β_2 or ....

(10) (α_1 and α_2 and ...) $\perp$ (β_1 and β_2 and ...)
Scoring: Constraint(R) = number of vectors of events $\langle A_1, A_2, \ldots, B_1, B_2, \ldots\rangle$ of types $\alpha_1, \alpha_2, \ldots, \beta_1, \beta_2, \ldots$ respectively that all overlap on the timeline.
(Could also be notated:
$\alpha_1 \perp \alpha_2 \perp \cdots \perp \beta_1 \perp \beta_2 \perp \cdots$.)

(11) $\alpha \rightarrow (\beta_1$ and $\beta_2)$ [cf. $\alpha \rightarrow \beta_1 \gg \alpha \rightarrow \beta_2$]
$(\alpha_1$ or $\alpha_2) \rightarrow \beta$ [cf. $\alpha_1 \rightarrow \beta \gg \alpha_2 \rightarrow \beta$]

The unifying theme is that each primitive constraint counts the number of times a candidate gets into some bad *local* configuration. This is an interval on the timeline throughout which certain events (one or more specified edges or interiors) are all present and certain other events (zero or more specified edges or interiors) are all absent.

Several examples of phonologically plausible constraints, with monikers and descriptions, are given below. (Eisner, 1997) shows how to rewrite hundreds of constraints from the literature in the primitive constraint notation, and discusses the problematic case of reduplication. (Eisner, in press) gives a detailed stress typology using only primitive constraints; in particular, non-local constraints such as FtBin, FootForm, and Generalized Alignment (McCarthy & Prince, 1993) are eliminated.

(12) a. Onset: $_\sigma[\rightarrow {}_C[$
"Every syllable starts with a consonant."
b. NonFinality: $]_{Word} \perp]_F$
"The end of a word may not be footed."
c. FtSyl: $_F[\rightarrow {}_\sigma[$, $]_F \rightarrow]_\sigma$
"Feet start and end on syllable boundaries."
d. PackFeet: $]_F \rightarrow {}_F[$
"Each foot is followed immediately by another foot; i.e., minimize the number of gaps between feet. Note that the final foot, if any, will always violate this constraint."
e. NoClash: $]_{\mathtt{x}} \perp {}_{\mathtt{x}}[$
"Two stress marks may not be adjacent."
f. ProgressiveVoicing: $]_{voi} \perp {}_C[$
"If the segment preceding a consonant is voiced, voicing may not stop prior to the consonant but must be spread onto it."
g. NasVoi: $nas \rightarrow voi$
"Every nasal gesture must be at least partly voiced."
h. FullNasVoi: $nas \perp {}_{voi}[$, $nas \perp]_{voi}$
"A nasal gesture may not be only partly voiced."
i. Max(voi) or Parse(voi): $\underline{voi} \rightarrow voi$
"Underlying voicing features surface."
j. Dep(voi) or Fill(voi): $voi \rightarrow \underline{voi}$
"Voicing features appear on the surface only if they are also underlying."
k. NoSpreadRight(voi): $voi \perp]_{\underline{voi}}$
"Underlying voicing may not spread rightward as in (6)."
l. NonDegenerate: $F \rightarrow {}_\mu[$
"Every foot must cross at least one mora boundary $_\mu[$."
m. TautomorphemicFoot: $F \perp {}_{\underline{Morph}}[$
"No foot may cross a morpheme boundary."

3 Finite-state generation in OTP

3.1 A simple generation algorithm

Recall that the generation problem is to find the output set S_n, where

(13) a. $S_0 = \mathbf{Gen}(input) \subseteq \mathbf{Repns}$
b. $S_{i+1} = \mathrm{Filter}_{i+1}(S_i) \subseteq S_i$

Since in OTP, the input is a partial order of edge brackets, and S_n is a set of one or more total orders (timelines), a natural approach is to successively refine a partial order. This has merit. However, not every S_i can be represented as a single partial order, so the approach is quickly complicated by the need to encode disjunction.

A simpler approach is to represent S_i (as well as *input* and **Repns**) as a finite-state automaton (FSA), denoting a regular set of strings that encode timelines. The idea is essentially due to (Ellison, 1994), and can be boiled down to two lines:

(14) **Ellison's algorithm** (variant).
$S_0 = input \cap \mathbf{Repns}$
= all conceivable outputs containing *input*
$S_{i+1} = \mathrm{BestPaths}(S_i \cap C_{i+1})$

Each constraint C_i must be formulated as an *edge-weighted FSA* that scores candidates: C_i accepts any string R, on a single path of weight $C_i(R)$.[4] BestPaths is Dijkstra's "single-source shortest paths" algorithm, a dynamic-programming algorithm that prunes away all but the minimum-weight paths in an automaton, leaving an unweighted automaton.

OTP is simple enough that it can be described in this way.[5] The next section gives a nice encoding.

[3] Such a sequence does alter the meaning slightly. To get the exact original meaning, we would have to decompose into so-called "unranked" constraints, whereby $C_i(R)$ is defined as $C_{i_1}(R)+C_{i_2}(R)$. But such ties undermine OT's idea of strict ranking: they confer the power to minimize linear functions such as $(C_1 + C_1 + C_1 + C_2 + C_3 + C_3)(R) = 3C_1(R) + C_2(R) + 2C_3(R)$. For this reason, OTP currently disallows unranked constraints; I know of no linguistic data that crucially require them.

[4] Weighted versions of the state-labeled finite automata of (Bird & Ellison, 1994) could be used instead.

[5] The converse is also true, as was shown at the talk accompanying this paper: given enough abstract tiers, OTP can simulate any finite-state OT grammar.

3.2 OTP with automata

We may encode each timeline as a string over an enormous alphabet Σ. If $|\mathbf{Tiers}| = k$, then each symbol in Σ is a k-tuple, whose components describe what is happening on the various tiers at a given moment. The components are drawn from a smaller alphabet $\Delta = \{$`[`, `]`, `|`, `+`, `-`$\}$. Thus at any time, the ith tier may be beginning or ending a constituent (`[`, `]`) or both at once (`|`), or it may be in a steady state in the interior or exterior of a constituent (`+`, `-`). At a minimum, the string must record all moments where there is an edge on some tier. If all tiers are in a steady state, the string need not use any symbols to say so. Thus the string encoding is not unique.

(15) gives an expression for *all* strings that correctly describe the single tier shown. (16) describes a two-tier timeline consistent with (15). Note that the brackets on the two tiers are ordered with respect to each other. Timelines like these could be assembled morphologically from one or more lexical entries (Bird & Ellison, 1994), or produced in the course of algorithm (14).

(15) $_x[\qquad {}_x]_x \qquad]_x \qquad \Rightarrow$ `-*[+*|+*]-*`

(16) $_x[\qquad\qquad {}_x]_x \qquad\qquad\qquad]_x$
$\qquad\quad {}_y[\qquad\qquad\quad]_y \quad {}_y[\quad]_y$

$\Rightarrow$ ⟨-,-⟩*⟨[,-⟩⟨+,-⟩*⟨+,[⟩⟨+,+⟩*⟨|,+⟩⟨+,+⟩* ⟨+,]⟩⟨+,-⟩*⟨+,[⟩⟨+,+⟩*⟨],]⟩

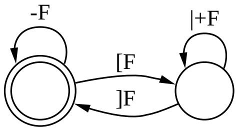
We store timeline expressions like (16) as deterministic FSAs. To reduce the size of these automata, it is convenient to label arcs not with individual elements of Σ (which is huge) but with subsets of Σ, denoted by predicates. We use conjunctive predicates where each conjunct lists the allowed symbols on a given tier:

(17) `+F`, `]`σ, `[|+-voi` (arc label w/ 3 conjuncts)

The arc label in (17) is said to **mention** the tiers $F, \sigma, voi \in \mathbf{Tiers}$. Such a predicate allows any symbol from Δ on the tiers it does not mention.

The input FSA constrains only the input tiers. In (14) we intersect it with **Repns**, which constrains only the output tiers. **Repns** is defined as the intersection of many automata exactly like (18), called **tier rules**, which ensure that brackets are properly paired on a given tier such as F (foot).

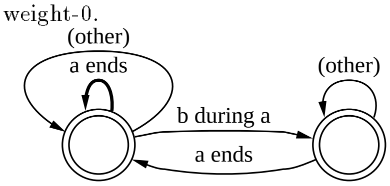
(18)



Like the tier rules, the constraint automata C_i are small and deterministic and can be built automatically. Every edge has weight 0 or 1. With some care it is possible to draw each C_i with two or fewer states, and with a number of arcs proportional to the number of tiers mentioned by the constraint. Keeping the constraints small is important for efficiency, since real languages have many constraints that must be intersected.

Let us do the hardest case first. An implication constraint has the general form (9). Suppose that all the α_i are interiors, not edges. Then the constraint targets intervals of the form $\alpha = \alpha_1 \cap \alpha_2 \cap \cdots$. Each time such an interval ends without any β_j having occurred during it, one violation is counted:

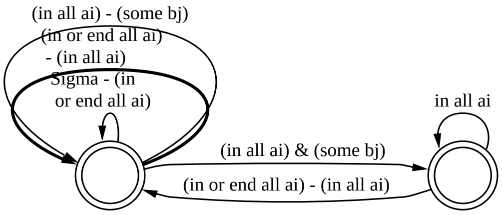
(19) Weight-1 arcs are shown in bold; others are weight-0.



A candidate that does see a β_j during an α can go and rest in the right-hand state for the duration of the α.

Let us fill in the details of (19). How do we detect the end of an α? Because one or more of the α_i end (`]`, `|`), while all the α_i either end or continue (`+`), so that we know we are leaving an α.[6] Thus:

(20)

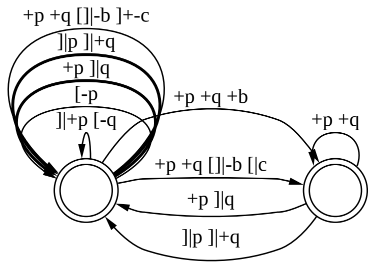


An unusually complex example is shown in (21). Note that to preserve the form of the predicates in (17) and keep the automaton deterministic, we need to split some of the arcs above into multiple arcs. Each β_j gets its own arc, and we must also expand set differences into multiple arcs, using the scheme $W - x \wedge y \wedge z = W \vee \neg(x \wedge y \wedge z) = (W \wedge \neg x) \vee (W \wedge x \wedge \neg y) \vee (W \wedge x \wedge y \wedge \neg z)$.

[6] It is important to take `]`, not `+`, as our indication that we have been inside a constituent. This means that the timeline ⟨[,-⟩⟨+,-⟩*⟨+,[⟩⟨+,+⟩*⟨],+⟩⟨-,+⟩*⟨-,]⟩ cannot avoid violating a clash constraint simply by instantiating the ⟨+,+⟩* part as ϵ. Furthermore, the `]` convention means that a zero-width input constituent (more precisely, a sequence of zero-width constituents, represented as a single `|` symbol) will often act as if it has an interior. Thus if $\underline{V}$ syncopates as in footnote 2, it still violates the parse constraint $\underline{V} \rightarrow V$. This is an explicit property of OTP: otherwise, nothing that failed to parse would ever violate PARSE, because it would be gone!

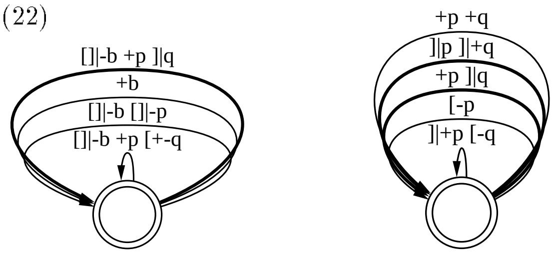
On the other hand, `]` does not have this special role on the right hand side of $\rightarrow$, which does not quantify universally over an interval. The consequence for zero-width consituents is that even if a zero-width $\underline{V}$ overlaps (at the edge, say) with a surface V, the latter cannot claim on this basis alone to satisfy FILL: $V \rightarrow \underline{V}$. This too seems like the right move linguistically, although further study is needed.

(21) (p and q) $\rightarrow$ (b or ${}_c[$)



How about other cases? If the antecedent of an implication is *not* an interval, then the constraint needs only one state, to penalize moments when the antecedent holds and the consequent does not. Finally, a clash constraint $\alpha_1 \perp \alpha_2 \perp \cdots$ is identical to the implication constraint (α_1 and α_2 and $\cdots$) $\rightarrow$ FALSE. Clash FSAs are therefore just degenerate versions of implication FSAs, where the arcs looking for β_j do not exist because they would accept no symbol. (22) shows the constraints (p and $]_q$) $\rightarrow$ b and $p \perp q$.

(22)



4 Computational requirements

4.1 Generalized Alignment is not finite-state

Ellison's method can succeed *only* on a restricted formalism such as OTP, which does not admit such constraints as the popular Generalized Alignment (GA) family of (McCarthy & Prince, 1993). A typical GA constraint is ALIGN(F, L, *Word*, L), which sums the number of syllables between each left foot edge ${}_F[$ and the left edge of the prosodic word. Minimizing this sum achieves a kind of left-to-right iterative footing. OTP argues that such non-local, arithmetic constraints can generally be eliminated in favor of simpler mechanisms (Eisner, in press).

Ellison's method cannot directly express the above GA constraint, even outside OTP, because it cannot compute a quadratic function $0 + 2 + 4 + \cdots$ on a string like $[\acute{\sigma}\sigma]_F\,[\acute{\sigma}\sigma]_F\,[\acute{\sigma}\sigma]_F\,\cdots$. Path weights in an FSA cannot be more than linear on string length.

Perhaps the filtering operation of any GA constraint can be simulated with a system of finite-state constraints? No: GA is simply too powerful. The proof is suppressed here for reasons of space, but it relies on a form of the pumping lemma for weighted FSAs. The key insight is that among candidates with a fixed number of syllables and a single (floating) tone, ALIGN(σ, L, H, L) prefers candidates where the tone docks at the *center*. A similar argument for weighted CFGs (using *two* tones) shows this constraint to be too hard even for (Tesar, 1996).

4.2 Generation is NP-hard even in OTP

When algorithm (14) is implemented literally and with moderate care, using an optimizing C compiler on a 167MHz UltraSPARC, it takes fully 3.5 minutes (real time) to discover a stress pattern for the syllable sequence ⌣⌣⌣—⌣⌣——⌣⌣⌣.[7] The automata become impractically huge due to intersections.

Much of the explosion in this case is introduced at the start and can be avoided. Because **Repns** has $2^{|\mathbf{Tiers}|} = 512$ states, S_0, S_1, and S_2 each have about 5000 states and 500,000 to 775,000 arcs. Thereafter the S_i automata become smaller, thanks to the pruning performed at each step by BestPaths. This repeated pruning is already an improvement over Ellison's original algorithm (which saves pruning till the end, and so continues to grow exponentially with every new constraint). If we modify (14) further, so that each tier rule from **Repns** is intersected with the candidate set only when its tier is first mentioned by a constraint, then the automata are pruned back as quickly as they grow. They have about 10 times fewer states and 100 times fewer arcs, and the generation time drops to 2.2 seconds.

This is a key practical trick. But neither it nor any other trick can help for all grammars, for in the worst case, the OTP generation problem is NP-hard on the number of tiers used by the grammar. The locality of constraints does not save us here. Many NP-complete problems, such as graph coloring or bin packing, attempt to minimize some global count subject to numerous local restrictions. In the case of OTP generation, the global count to minimize is the degree of violation of C_i, and the local restrictions are imposed by $C_1, C_2, \ldots C_{i-1}$.

Proof of NP-hardness (by polytime reduction from Hamilton Path). Given $G = \langle V(G), E(G) \rangle$, an n-vertex directed graph. Put $\mathbf{Tiers} = V(G) \cup \{\underline{Stem}, S\}$. Consider the following vector of $O(n^2)$ primitive constraints (ordered as shown):

(23) a. $\forall v \in V(G)$: ${}_v[\ \rightarrow\ {}_S[$ ["vertices are S's"]
b. $\forall v \in V(G)$: $]_v \rightarrow\ ]_S$ ["vertices are S's"]
c. $\forall v \in V(G)$: $\underline{Stem} \rightarrow v$ ["at least one"]
d. $\forall u, v \in V(G)$: $u \perp v$ ["disjoint"]
e. $\underline{Stem} \perp S$ ["no extra copies"]
f. $\forall u, v \in V(G)$ s.t. $uv \notin E(G)$: $]_u \perp {}_v[$
g. $]_S \rightarrow {}_S[$ ["single path"]

[7] The grammar is taken from the OTP stress typology proposed by (Eisner, in press). It has tier rules for 9 tiers, and then spends 26 constraints on obvious universal properties of moras and syllables, followed by 6 constraints for universal properties of feet and stress marks and finally 6 substantive constraints that can be freely reranked to yield different stress systems, such as left-to-right iambs with iambic lengthening.

Suppose the input is simply [$\underline{Stem}$]. Filtering **Gen**(*input*) through constraints (23a–e), we are left with just those candidates where $\underline{Stem}$ bears n disjoint constituents of type S, each coextensive with a constituent bearing a different label $v \in V(G)$. (These candidates satisfy (23a–d) but violate (23e) n times.) (23f) says that a chain of abutting constituents $[u][v][w]\cdots$ is allowed only if it corresponds to a path in G. Finally, (23g) forces the grammar to minimize the number of such chains. If the minimum is 1 (i.e., an arbitrarily selected output candidate violates (23g) only once), then G has a Hamilton path.

When confronted with this pathological case, the finite-state methods respond essentially by enumerating all possible permutations of $V(G)$ (though with sharing of prefixes). The machine state stores, among other things, the subset of $V(G)$ that has already been seen; so there are at least $2^{|\mathbf{Tiers}|}$ states.

It must be emphasized that if the grammar is *fixed in advance,* algorithm (14) is close to linear in the size of the input form: it is dominated by a constant number of calls to Dijkstra's BestPaths method, each taking time $O(|\text{input arcs}| \log |\text{input states}|)$. There are nonetheless three reasons why the above result is important. (a) It raises the practical specter of huge constant factors ($> 2^{40}$) for real grammars. Even if a fixed grammar can somehow be compiled into a fast form for use with many inputs, the compilation itself will have to deal with this constant factor. (b) The result has the interesting implication that candidate sets can arise that cannot be concisely represented with FSAs. For if all S_i were polynomial-sized in (14), the algorithm would run in polynomial time. (c) Finally, the grammar is *not* fixed in all circumstances: both linguists and children crucially experiment with different theories.

4.3 Work in progress: Factored automata

The previous section gave a useful trick for speeding up Ellison's algorithm in the typical case. We are currently experimenting with additional improvements along the same lines, which attempt to defer intersection by keeping tiers separate as long as possible.

The idea is to represent the candidate set S_i not as a large unweighted FSA, but rather as a collection $\mathcal{A}$ of preferably small unweighted FSAs, called **factors**, each of which mentions as few tiers as possible. This collection, called a **factored automaton**, serves as a compact representation of $\cap\mathcal{A}$. It usually has far fewer states than $\cap\mathcal{A}$ would if the intersection were carried out.

For instance, the natural factors of S_0 are *input* and all the tier rules (see 18). This requires only $O(|\mathbf{Tiers}| + |input|)$ states, not $O(2^{|\mathbf{Tiers}|} \cdot |input|)$. Using factored automata helps Ellison's algorithm (14) in several ways:

- The candidate sets S_i tend to be represented more compactly.
- In (14), the constraint C_{i+1} needs to be intersected with only certain factors of S_i.
- Sometimes C_{i+1} does not need to be intersected with the input, because they do not mention any of the same tiers. Then step $i+1$ can be performed in time independent of input length.

Example: *input* $= \smile\smile\smile-\smile\smile--\smile\smile\smile$, which is a 43-state automaton, and C_1 is $F \rightarrow \mathbf{x}$, which says that every foot bears a stress mark. Then to find $S_1 = \text{BestPaths}(S_0 \cap C_1)$, we need only consider S_0's tier rules for F and $\mathbf{x}$, which require well-formed feet and well-formed stress marks, and combine them with C_1 to get a new factor that requires stressed feet. No other factors need be involved.

The key operation in (14) is to find Bestpaths($\mathcal{A} \cap C$), where $\mathcal{A}$ is an unweighted factored automaton representing a candidate set, and C is an ordinary weighted FSA (a constraint). This is the **best intersection** problem. For concreteness let us suppose that C encodes $F \rightarrow \mathbf{x}$, a two-state constraint.

A naive idea is simply to add $F \rightarrow \mathbf{x}$ to $\mathcal{A}$ as a new factor. However, this ignores the BestPaths step: we wish to keep just the best paths in $F \rightarrow \mathbf{x}$ that are compatible with $\mathcal{A}$. Such paths might be long and include unrolled cycles in $F \rightarrow \mathbf{x}$. For example, a weight-1 path would describe a chain of optimal stressed feet interrupted by a single unstressed one where $\mathcal{A}$ happens to block stress.

A corrected variant is to put $I = \cap\mathcal{A}$ and run BestPaths on $I \cap C$. Let the pruned result be B. We could add B directly back to to $\mathcal{A}$ as a new factor, but it is large. We would rather add a smaller factor B' that has the same effect, in that $I \cap B' = I \cap B$. (B' will look something like the original C, but with some paths missing, some states split, and some cycles unrolled.) Observe that each state of B has the form $i \times c$ for some $i \in I$ and $c \in C$. We form B' from B by "re-merging" states $i \times c$ and $i' \times c$ where possible, using an approach similar to DFA minimization.

Of course, this variant is not very efficient, because it requires us to find and use $I = \cap\mathcal{A}$. What we really want is to follow the above idea but use a smaller I, one that considers just the relevant factors in $\mathcal{A}$. We need not consider factors that will not affect the choice of paths in C above.

Various approaches are possible for choosing such an I. The following technique is completely general, though it may or may not be practical.

Observe that for BestPaths to do the correct thing, I needs to reflect the sum total of $\mathcal{A}$'s constraints on F and $\mathbf{x}$, the tiers that C mentions. More formally, we want I to be the projection of the candidate set $\cap\mathcal{A}$ onto just the F and $\mathbf{x}$ tiers. Unfortunately, these constraints are not just reflected in the factors mentioning F or $\mathbf{x}$, since the allowed configurations of F and $\mathbf{x}$ may be mediated through additional factors. As an example, there may be a factor mentioning F and ψ, some of whose paths are incompatible with the input factor, because the latter allows ψ only in certain places or because only allows paths of length 14.

1. Number the tiers such that F and $\mathtt{x}$ are numbered 0, and all other tiers have distinct positive numbers.
2. Partition the factors of $\mathcal{A}$ into lists L_0, L_1, $L_2, \ldots L_k$, according to the highest-numbered tier they mention. (Any factor that mentions no tiers at all goes onto L_0.)
3. If $k = 0$, then return $\cap L_k$ as our desired I.
4. Otherwise, $\cap L_k$ exhausts tier k's ability to mediate relations among the factors. Modify the arc labels of $\cap L_k$ so that they no longer restrict (mention) k. Then add a determinized, minimized version of the result to to L_j, where j is the highest-numbered tier it now mentions.
5. Decrement k and return to step 3.

If $\cap\mathcal{A}$ has k factors, this technique must perform $k - 1$ intersections, just as if we had put $I = \cap\mathcal{A}$. However, it intersperses the intersections with determinization and minimization operations, so that the automata being intersected tend not to be large. In the best case, we will have $k - 1$ intersection-determinization-minimizations that cost $O(1)$ apiece, rather than $k-1$ intersections that cost up to $O(2^k)$ apiece.

5 Conclusions

Primitive Optimality Theory, or OTP, is an attempt to produce a a simple, rigorous, constraint-based model of phonology that is closely fitted to the needs of working linguists. I believe it is worth study both as a hypothesis about Universal Grammar and as a formal object.

The present paper introduces the OTP formalization to the computational linguistics community. We have seen two formal results of interest, both having to do with generation of surface forms:

- OTP's *generative power* is low: finite-state optimization. In particular it is more constrained than theories using Generalized Alignment. This is good news for comprehension and learning.
- OTP's *computational complexity*, for generation, is nonetheless high: NP-hard on the size of the grammar. This is mildly unfortunate for OTP and for the OT approach in general.

  It remains true that for a fixed grammar, the time to do generation is close to linear on the size of the input (Ellison, 1994), which is heartening if we intend to optimize long utterances with respect to a fixed phonology.

Finally, we have considered the prospect of building a practical tool to generate optimal outputs from OT theories. We saw above to set up the representations and constraints efficiently using deterministic finite-state automata, and how to remedy some hidden inefficiencies in the seminal work of (Ellison, 1994), achieving at least a 100-fold observed speedup. Delayed intersection and aggressive pruning prove to be important. Aggressive minimization and a more compact, "factored" representation of automata may also turn out to help.

References

Bird, Steven, & T. Mark Ellison. One Level Phonology: Autosegmental representations and rules as finite automata. *Comp. Linguistics* 20:55–90.

Cole, Jennifer, & Charles Kisseberth. 1994. An optimal domains theory of harmony. *Studies in the Linguistic Sciences* **24**: 2.

Eisner, Jason. In press. Decomposing FootForm: Primitive constraints in OT. Proceedings of SCIL 8, NYU. Published by MIT Working Papers. (Available at http://ruccs.rutgers.edu/roa.html.)

Eisner, Jason. What constraints should OT allow? Handout for talk at LSA, Chicago. (Available at http://ruccs.rutgers.edu/roa.html.)

Ellison, T. Mark. Phonological derivation in optimality theory. COLING '94, 1007-1013.

Goldsmith, John. 1976. Autosegmental phonology. Cambridge, Mass: MIT PhD. dissertation. Published 1979 by New York: Garland Press.

Goldsmith, John. 1990. Autosegmental and metrical phonology. Oxford: Blackwell Publishers.

McCarthy, John, & Alan Prince. 1993. Generalized alignment. *Yearbook of Morphology*, ed. Geert Booij & Jaap van Marle, pp. 79-153. Kluwer.

McCarthy, John and Alan Prince. 1995. Faithfulness and reduplicative identity. In Jill Beckman et al., eds., Papers in Optimality Theory. UMass, Amherst: GLSA. 259–384.

Prince, Alan, & Paul Smolensky. 1993. *Optimality theory: constraint interaction in generative grammar.* Technical Reports of the Rutgers University Center for Cognitive Science.

Selkirk, Elizabeth. 1980. Prosodic domains in phonology: Sanskrit revisited. In Mark Aranoff and Mary-Louise Kean, eds., *Juncture*, pp. 107–129. Anna Libri, Saratoga, CA.

Tesar, Bruce. 1995. Computational Optimality Theory. Ph.D. dissertation, U. of Colorado, Boulder.

Tesar, Bruce. 1996. Computing optimal descriptions for Optimality Theory: Grammars with context-free position structures. Proceedings of the 34th Annual Meeting of the ACL.