Comprehension & Compilation in Optimality Theory

Jason Eisner
Johns Hopkins University
July 8, 2002 — ACL

Introduction
- This paper is batting cleanup.
  - Pursues some other people’s ideas to their logical conclusion. Results are important, but follow easily from previous work.
    - Comprehension: More finite-state woes for OT
    - Compilation: How to shoehorn OT into finite-state world
- Other motivations:
  - Clean up the notation. (Especially, what counts as "underlying" and "surface" material and how their correspondence is encoded.)
  - Discuss interface to morphology and phonetics.
  - Help confused people. I get a lot of email. ☺

Computational OT is Mainly Finite-State – Why?
- Good news:
  - Individual OT constraints appear to be finite-state
- Bad news (gives us something to work on):
  - OT grammars are not always finite-state

Computational OT is Mainly Finite-State – Why?
- Good news:
  - Individual OT constraints appear to be finite-state
- Bad news:
  - OT grammars are not always finite-state
  - Oops! Too powerful for phonology.
  - Oops! Don’t support nice computation.
    - Fast generation
    - Fast comprehension
    - Interface with rest of linguistic system or NLP/speech system

Main Ideas in Finite-State OT
- Generation algo. from finite-state constraints
- OT constraints are generally finite-state
- Comprehension?
- Finite-state constraints don’t yield FS grammar
- Get FS grammar by hook or by crook
- Unify these maneuvers?
- Change OT
- Encode funky representations as strings

Phonology in the Abstract
- ab + dip or IN + HOUSE
- x = “abdip” underlying form in Σ∗
- z = “a[di][bu]” surface form in Δ∗
OT in the Abstract

x = “abdip” underlying form in Σ*

y = “aab0[ddij][pb0u]” candidate in (Σ ∪ Δ)*

z = “a[di][bu]” surface form in Δ*

OT in the Abstract

x = “abdip” underlying form in Σ*

can extract x ∈ Σ*

y = “aab0[ddij][pb0u]” candidate in (Σ ∪ Δ)*

z = “a[di][bu]” surface form in Δ*

OT in the Abstract

x = “aab0[ddij][pb0u]” candidate in (Σ ∪ Δ)*

can extract z ∈ Δ*

z = “a[di][bu]” surface form in Δ*

OT in the Abstract

x = “aab0[ddij][pb0u]” candidate in (Σ ∪ Δ)*

to evaluate x → z mapping, just evaluate y!
• is z a close variant of x? (faithfulness)
• is z easy to pronounce? (well-formedness)

z = “a[di][bu]” surface form in Δ*

OT in the Abstract

x = “aab0[ddij][pb0u]” candidate in (Σ ∪ Δ)*

Y = {“aabddiipp”, “aab0[ddij][pb0u]”, “[0baa]b0d0]0p0”, …}

many candidates

z = “a[di][bu]” surface form in Δ*
OT in the Abstract

\[ x = \text{"abdip"} \] underlying form in \( \Sigma \)

\[ Y = \{ \text{"aabddipp",} \] pick the best candidate

\[ \text{"aab0[ddij][pb0u]"} \]

\[ z = \text{"a[di][bu]"} \] surface form in \( \Delta \)

OT in the Abstract

\[ x = \text{"abdip"} \]

\[ Y_0(x) = \{ A,B,C,D,E,F,G, \ldots \} \]

\[ Y_1(x) = \{ B, D,E, \ldots \} \]

\[ Y_2(x) = \{ \text{"aab0[ddij][pb0u]"}, \ldots \} \]

Don’t worry yet about how the constraints are defined.

OT Comprehension? No ...

\[ x = \text{"abdip"} \]

\[ Y_0(x) = \{ A,B,C,D,E,F,G, \ldots \} \]

\[ Y_1(x) = \{ B, D,E, \ldots \} \]

\[ Y_2(x) = \{ \text{"aab0[ddij][pb0u]"}, \ldots \} \]

OT Comprehension? No ...

\[ x = \text{"abdip"} \]

\[ Y_0(x) = \{ A,B,C,D,E,F,G, \ldots \} \]

\[ Y_1(x) = \{ B, D,E, \ldots \} \]

\[ Y_2(x) = \{ \text{"aab0[ddij][pb0u]"}, \ldots \} \]

OT Comprehension Looks Hard!

\[ x = \text{"abdip"} \] ?

\[ Y_0(x) = \{ A,B,C,D,E,F,G, \ldots \} \]

\[ Y_1(x) = \{ B, D,E, \ldots \} \]

\[ Y_2(x) = \{ \text{"aab0[ddij][pb0u]"}, \ldots \} \]

OT Comprehension Is Hard!

Constraint 1: One violation for each \( a \) inside brackets (\( \text{"[a]"} \)) or \( b \) outside brackets (\( \text{"b"} \))

possible \( x \)'s are all strings where \( \# a \)'s \( \leq \) \( \# b \)'s ! Not a regular set.
**OT Comprehension Is Hard!**

Constraint 1: One violation for each \( a \) inside brackets or \( b \) outside brackets

possible x’s are all strings where \( \# a’s \leq \# b’s \) ! Not a regular set.

- The constraint is finite-state (we’ll see what this means)
- Also, can be made more linguistically natural
- If all constraints are finite-state:
  - Already knew: Given \( x \), set of possible \( z \)'s is regular (Ellison 1994)
  - That’s why Ellison can use finite-state methods for generation
  - The new fact: Given \( z \), set of possible x’s can be non-regular
    - So finite-state methods probably cannot do comprehension
    - Stronger than previous Hiller-Smolensky-Frank-Satta result that the relation (\( x,z \)) can be non-regular

---

**Possible Solutions**

1. Eliminate nasty constraints
   - Doesn’t work: problem can arise by nasty grammars
   - Of nice constraints (linguistically natural or primitive-OT)

2. Allow only a finite lexicon
   - Then the grammar defines a finite, regular relation
   - In effect, try all x’s and see which ones \( \rightarrow \) \( z \)
   - In practice, do this faster by precompilation & lookup
   - But then can’t comprehend novel words or phrases
   - Unless lexicon is “all forms of length < 20”; inefficient?

3. Make OT regular “by hook or by crook”

---

**In a Perfect World, Y0, Y1, Y2, ... Z Would Be Regular Relations (FSTs)**

\[
\begin{align*}
X &= \text{“abdip”} \\
Y_0(x) &= \{A,B,C,D,E,F,G, \ldots\} \\
Y_1(x) &= \{ B, D,E, \ldots\} \\
Y_2(x) &= \{ D, \ldots\} \\
Z(x) &= \{ \text{“a[di][bu]”, \ldots}\}
\end{align*}
\]

**In a Perfect World, Compose FSTs To Get an Invertible, Full-System FST**

**How Can We Make Y0, Y1, Y2, ... Z Be Regular Relations (FSTs) ?**

\[
\begin{align*}
X &= \text{“abdip”} \\
Y_0(x) &= \{A,B,C,D,E,F,G, \ldots\} \\
Y_1(x) &= \{ B, D,E, \ldots\} \\
Y_2(x) &= \{ D, \ldots\} \\
Z(x) &= \{ \text{“a[di][bu]”, \ldots}\}
\end{align*}
\]

**A General View of Constraints**
**A General View of Constraints**

- One violation for each symbol inside brackets
- One violation for each surface feature outside brackets

\[
Y_i(x) = \{aabbb, aab\} \quad \text{constraint}
\]

\[
Y_{i+1}(x) = \{aab\} \quad \text{harmonic pruning}
\]

**Why Is This View “General”?**

- Constraint doesn’t just count *’s but marks their location
- We might consider other kinds of harmonic pruning
  - Including OT variants that are sensitive to location of *

\[
Y_i(x) = \{aabbb, aab\} \quad \text{constraint}
\]

\[
Y_{i+1}(x) = \{aab\} \quad \text{harmonic pruning}
\]

**Regular Harmony Orderings**

- A harmony ordering \(>\) is a binary relation
- If it’s a regular relation, it can be computed by a finite-state transducer \(H\)
- \(H\) accepts \((q,r)\) if \(q > r\) (e.g., \([aab•bbb] > aab•b•b•*)
- \(H(q) = \text{range}(q \circ H) = \{r: q > r\}\)
  - “set of *’s that are worse than \(q\)”
- \(H(Q) = \text{range}(Q \circ H) = \bigcup_{q \in Q} \{r: q > r\}\)
  - “set of *’s that are worse than something in \(Q\)”

**The Harmony Ordering**

- An OT grammar really has 4 components:
  - Gen, Pron, harmony ordering, constraint seq.
  - Language-specific

- Harmony ordering compares 2 starred candidates that share underlying material:
  - Traditional OT says “fewer stars is better”
  - \([aab0[dd]bgbu] > [aab0[dd]bgbu]^{*}\) “0 beats 2”
  - \([a•a•b•b] > aab•b•b•*\) “2 beats 3”
  - Unordered: \([a•a•b•b] > aab•b•b•*\) “2 vs. 2”
  - Unordered: \([aab0[dd]bgbu] > [aab0[dd]bgbu]^{*}\) “aabb vs. aabb”

**Using a Regular Harmony Ordering**

\[
Y_i(x) = \{aabbb, aab\}
\]

\[
Y_{i+1}(x) = \{aab\}
\]
A Family of Optimality Operators $\text{OO}_H$

- $Y \circ C$: Inviolable constraint (traditional composition)
- $Y \circ o C$: Violable constraint with harmony ordering $H$
- $Y \circ o + C$: Traditional OT: harmony compares # of stars
- Not a finite-state operator!
- $Y \circ o C$: Binary constraint: "no stars" > "some stars"
  This $H$ is a regular relation:
  - Can build an FST that accepts $(q,r)$ iff $q > r$
  - $Y \circ o C$ is a regular relation, and $C$ is a regular constraint, then $Y \circ o C$ is a regular relation
- $Y \circ o C$: Subset approximation of $o +$ (traditional OT)
- Gerdemann & van Noord 2000
- Exact for many grammars, though not all
- $Y \gg C$: Directional constraint (Eisner 2000)
- $Y \ll C$: Non-traditional OT – linguistic motivation

Using a Regular Harmony Ordering

- $Y_i(x) = \{aabbb\} \cup \{abbb\}$
- $Y_i(x) = \{aab\}$
- $Y_i(x) = \{aab\}$
- $Y_i(x) = \{aab\}$
- $Y_i(x) = \{aab\}$
- $Y_i(x) = \{aab\}$
- $Y_i(x) = \{aab\}$

Consequences:

- For each operator, the paper shows how to construct $H$ as a finite-state transducer.

Consequences: A Family of Optimality Operators $\text{OO}_H$

- $Y \circ C$: Inviolable constraint (traditional composition)
- $Y \circ o C$: Violable constraint with harmony ordering $H$
- $Y \circ o + C$: Traditional OT: harmony compares # of stars
- Not a finite-state operator!
- $Y \circ o C$: Binary constraint: "no stars" > "some stars"
  This $H$ is a regular relation:
  - Can build an FST that accepts $(q,r)$ iff $q > r$
  - $Y \circ o C$ is a regular relation, and $C$ is a regular constraint, then $Y \circ o C$ is a regular relation
- $Y \circ o C$: Subset approximation of $o +$ (traditional OT)
- Gerdemann & van Noord 2000
- Exact for many grammars, though not all
- $Y \gg C$: Directional constraint (Eisner 2000)
- $Y \ll C$: Non-traditional OT – linguistic motivation

What Have We Proved?

- An OT grammar has 4 components:
  - $\text{Gen}$, Pron, constraints, harmony ordering
- Theorem (by induction):
  - If all of these are regular relations, so is the full phonology $Z$.
- $Z = (\text{Gen} \circ o C, \text{C1} \circ o C, \text{C2}) \circ \text{Pron}$
  where $Y \circ o C = Y \circ o C \circ \text{range}(Y \circ o C \circ H) \circ D$
- Generalizes Gerdemann & van Noord 2000
- Operator notation follows Karttunen 1998

For each operator, the paper shows how to construct $H$ as a finite-state transducer.
Subset Approximation

- $Y o \subset C$ Subset approximation to $o+$ (traditional OT) Gerdemann & van Noord 2000 Exact for many grammars, not all
- As for many harmony orderings, ignores surface symbols. Just looks at underlying and starred symbols.
- $a \preceq b \preceq c \preceq d \preceq e$
- $a \preceq b \preceq c \preceq d \preceq e$
  top candidate wins
  incomparable; both survive

Directional Constraints

- $Y o> C$ Directional constraint (Eisner 2000)
- $Y <o C$ Non-traditional OT - linguistic motivation
- As for many harmony orderings, ignores surface symbols. Just looks at underlying and starred symbols.
- $a \preceq b \preceq c \preceq d \preceq e$
- $a \preceq b \preceq c \preceq d \preceq e$
  always same result as subset approx if subset approx has a result at all
  if subset approx has a problem, resolves constraints directionally
  top candidate wins under $o>$
  bottom candidate wins under $<o$
  Seems to be what languages do, too.

Interesting Questions

- Are there any other optimality operators worth considering? Hybrids?
- Are these finite-state operators useful for filtering nondeterminism in any finite-state systems other than OT phonologies?

Summary

- Generation algo. from finite-state constraints
- OT constraints are generally finite-state
- Finite-state constraints don’t yield FS grammar
  works great if harmony ordering is made regular
  change OT
  approximate OT
- YES - everything
- NO
  unify these maneuvers?
  and more

FIN