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

evaluate a given candidate (good or bad? how bad?)

compilation

- Bad news (gives us something to work on):
  - OT grammars are not always finite-state

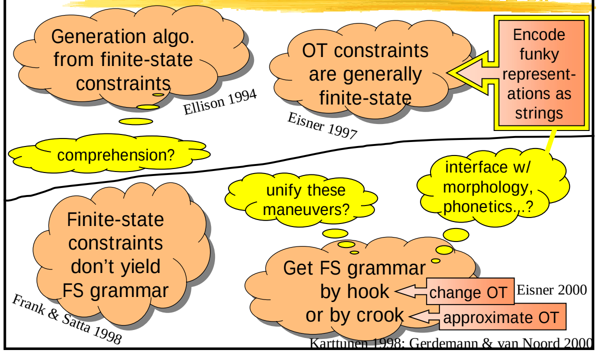
map each input to the best candidate
(aggregates several constraints (easy part)
and uses them to search (hard part))

Computational OT is Mainly Finite-State – Why?

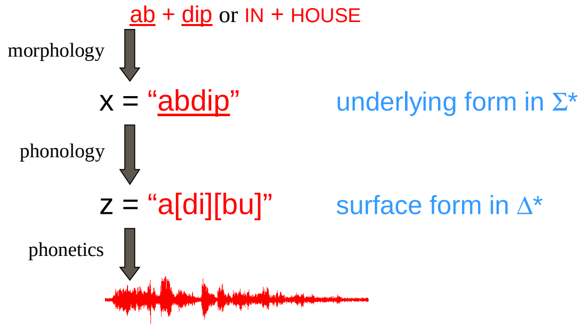
- Good news:
  - Individual OT constraints appear to be finite-state
- Bad news:
  - OT grammars are not always finite-state
  - Oops! Too powerful for phonology.
  - Oops! Don't support nice computation.
    - Fast generation
    - Fast comprehension
    - Interface with rest of linguistic system or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

---

Main Ideas in Finite-State OT



Phonology in the Abstract



ab + dip or IN + HOUSE

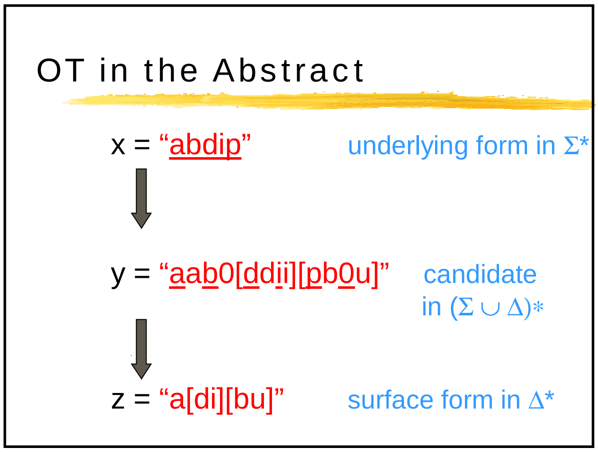
morphology

x = "abdip" underlying form in Σ*

phonology

z = "a[di][bu]" surface form in Δ*

phonetics

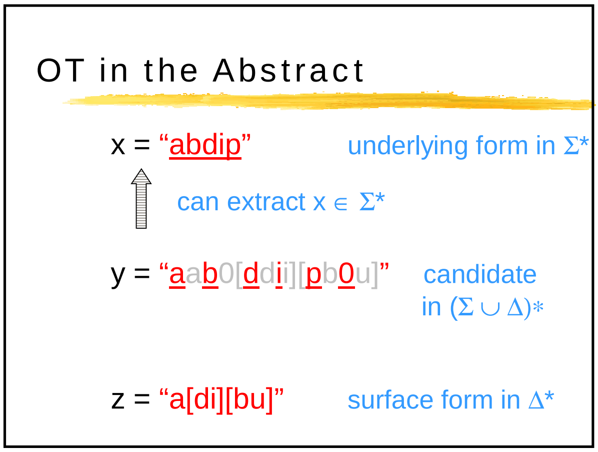


OT in the Abstract

x = "abdip" underlying form in Σ^*

y = "aab0[ddii][pb0u]" candidate in $(\Sigma \cup \Delta)^*$

z = "a[di][bu]" surface form in Δ^*



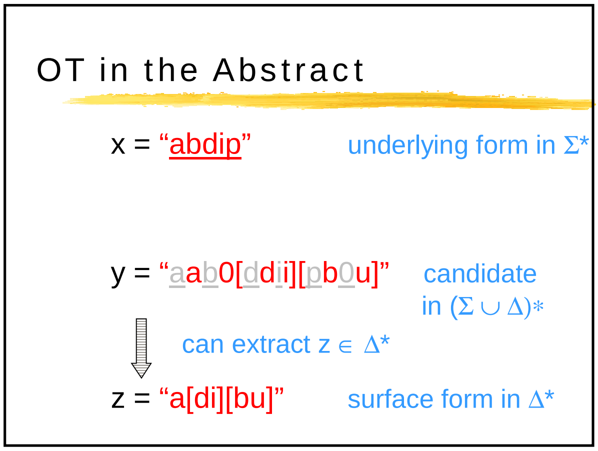
OT in the Abstract

x = "abdip" underlying form in Σ^*

can extract $x \in \Sigma^*$

y = "aab0[ddii][pb0u]" candidate in $(\Sigma \cup \Delta)^*$

z = "a[di][bu]" surface form in Δ^*



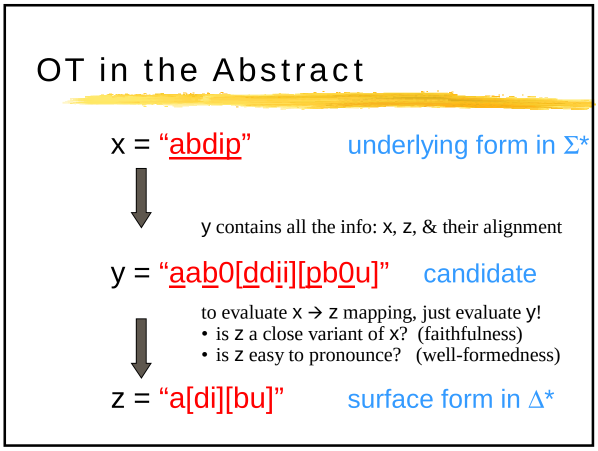
OT in the Abstract

x = "abdip" underlying form in Σ^*

y = "aab0[ddii][pb0u]" candidate in $(\Sigma \cup \Delta)^*$

can extract $z \in \Delta^*$

z = "a[di][bu]" surface form in Δ^*



OT in the Abstract

x = "abdip" underlying form in Σ^*

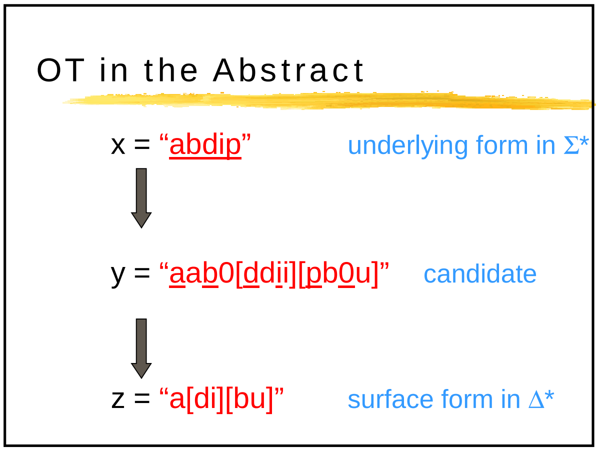
y contains all the info: x, z, & their alignment

y = "aab0[ddii][pb0u]" candidate

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^*

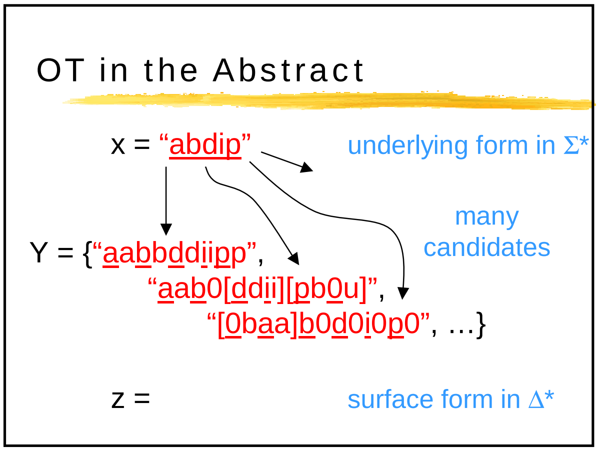


OT in the Abstract

x = "abdip" underlying form in Σ^*

y = "aab0[ddii][pb0u]" candidate

z = "a[di][bu]" surface form in Δ^*



OT in the Abstract

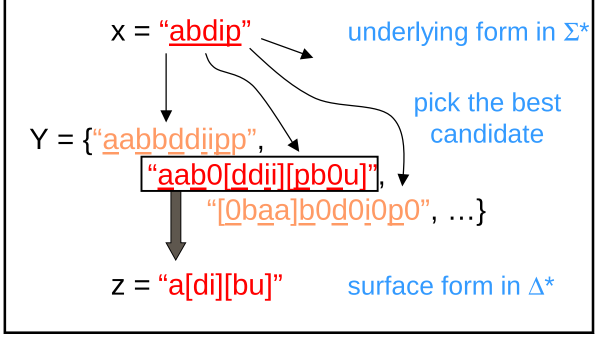
x = "abdip" underlying form in Σ^*

Y = {"aabbddiipp",
"aab0[ddii][pb0u]",
"[0baa]b0d0i0p0", …}

many candidates

z = surface form in Δ^*

OT in the Abstract



x = "abdip"

underlying form in Σ*

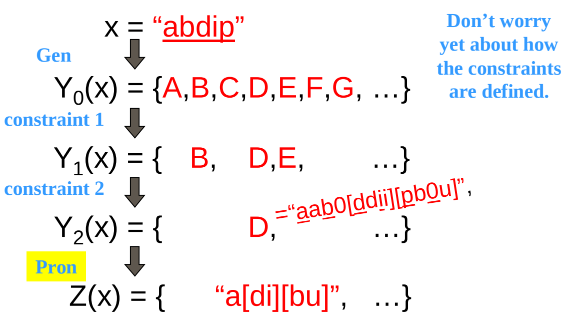
pick the best candidate

Y = {"aabbddiipp",
"aab0[ddii][pb0u]",
"[0baa]b0d0i0p0", …}

z = "a[di][bu]"

surface form in Δ*

OT in the Abstract



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

x = "abdip"

Gen

$Y_0(x)$ = {A,B,C,D,E,F,G, …}

constraint 1

$Y_1(x)$ = { B, D,E, …}

constraint 2

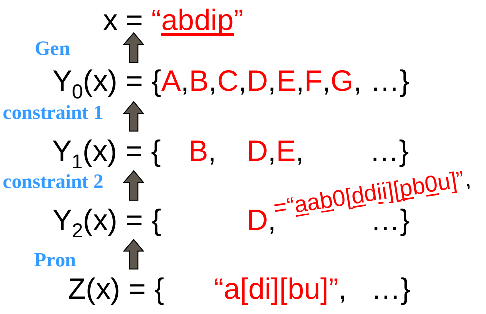
$Y_2(x)$ = { D, …}

="aab0[ddii][pb0u]",

Pron

Z(x) = { "a[di][bu]", …}

OT Comprehension? No …



x = "abdip"

Gen

$Y_0(x)$ = {A,B,C,D,E,F,G, …}

constraint 1

$Y_1(x)$ = { B, D,E, …}

constraint 2

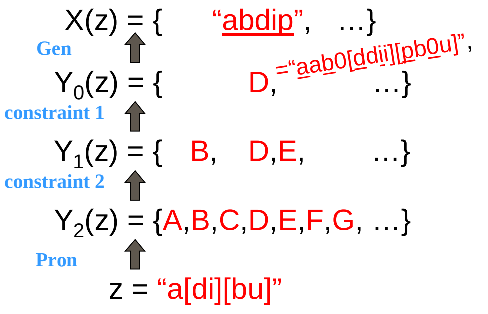
$Y_2(x)$ = { D, …}

="aab0[ddii][pb0u]",

Pron

Z(x) = { "a[di][bu]", …}

OT Comprehension? No …



X(z) = { "abdip", …}

Gen

$Y_0(z)$ = { D, …}

="aab0[ddii][pb0u]",

constraint 1

$Y_1(z)$ = { B, D,E, …}

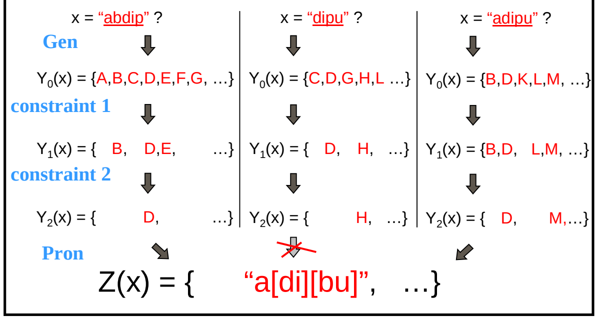
constraint 2

$Y_2(z)$ = {A,B,C,D,E,F,G, …}

Pron

z = "a[di][bu]"

OT Comprehension Looks Hard!



x = "abdip" ? | x = "dipu" ? | x = "adipu" ?

Gen

$Y_0(x)$ = {A,B,C,D,E,F,G, …} | $Y_0(x)$ = {C,D,G,H,L …} | $Y_0(x)$ = {B,D,K,L,M, …}

constraint 1

$Y_1(x)$ = { B, D,E, …} | $Y_1(x)$ = { D, H, …} | $Y_1(x)$ = {B,D, L,M, …}

constraint 2

$Y_2(x)$ = { D, …} | $Y_2(x)$ = { H, …} | $Y_2(x)$ = { D, M,…}

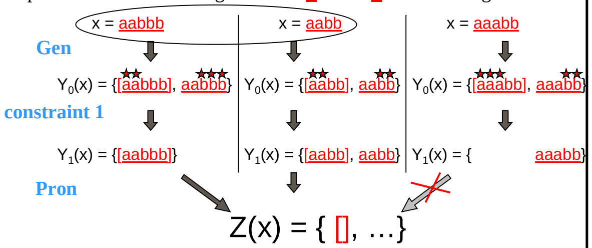
Pron

Z(x) = { "a[di][bu]", …}

OT Comprehension Is Hard!

Constraint 1: One violation for each a inside brackets (*[a])
or b outside brackets (*b)

possible x's are all strings where # a's ≤ # b's ! Not a regular set.



x = aabbb | x = aabb | x = aaabb

Gen

$Y_0(x)$ = {[aabbb], aabbb} | $Y_0(x)$ = {[aabb], aabb} | $Y_0(x)$ = {[aaabb], aaabb}

constraint 1

$Y_1(x)$ = {[aabbb]} | $Y_1(x)$ = {[aabb], aabb} | $Y_1(x)$ = { aaabb}

Pron

Z(x) = { [], …}

OT Comprehension Is Hard!

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

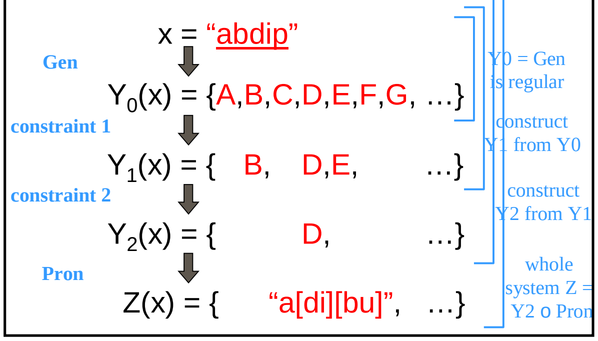
possible x's are all strings where # a's ≤ # b's ! Not a regular set.

- 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 → 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)



x = "abdip"

Gen

$Y_0(x)$ = {A,B,C,D,E,F,G, …}

constraint 1

$Y_1(x)$ = { B, D,E, …}

constraint 2

$Y_2(x)$ = { D, …}

Pron

Z(x) = { "a[di][bu]", …}

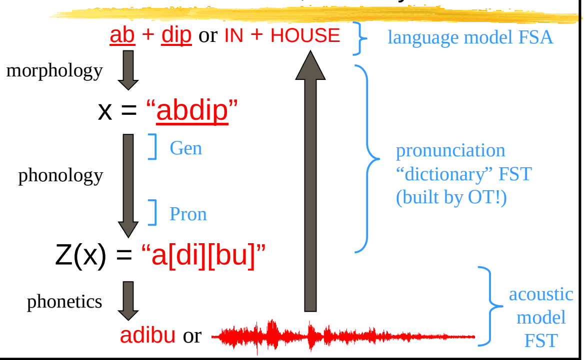
Y0 = Gen is regular

construct Y1 from Y0

construct Y2 from Y1

whole system Z = Y2 o Pron

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



ab + dip or IN + HOUSE

language model FSA

morphology

x = "abdip"

Gen

phonology

Pron

pronunciation "dictionary" FST (built by OT!)

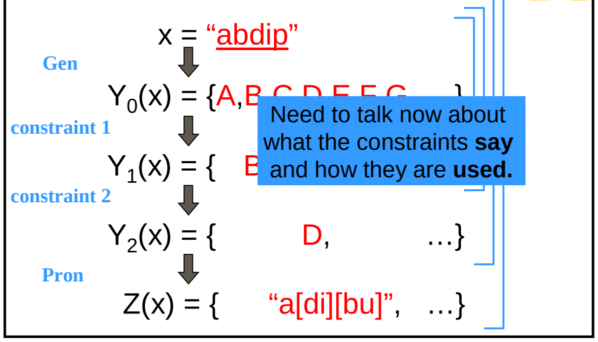
Z(x) = "a[di][bu]"

phonetics

adibu or

acoustic model FST

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



x = "abdip"

Gen

$Y_0(x)$ = {A,B

constraint 1

$Y_1(x)$ = { B

constraint 2

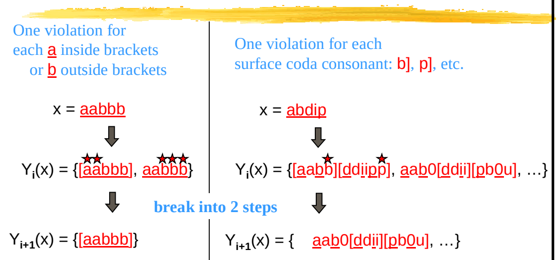
$Y_2(x)$ = { D, …}

Pron

Z(x) = { "a[di][bu]", …}

Need to talk now about what the constraints **say** and how they are **used.**

A General View of Constraints



One violation for each a inside brackets or b outside brackets

x = aabbb

$Y_i(x)$ = {[aabbb], aabbb}

$Y_{i+1}(x)$ = {[aabbb]}

One violation for each surface coda consonant: b], p], etc.

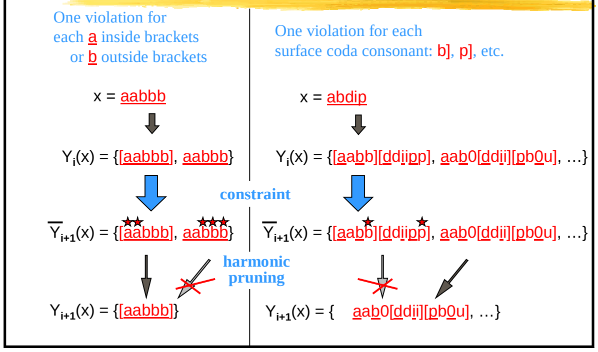
x = abdip

$Y_i(x)$ = {[aabb][ddiipp], aab0[ddii][pb0u], …}

$Y_{i+1}(x)$ = { aab0[ddii][pb0u], …}

break into 2 steps

A General View of Constraints



One violation for
each a inside brackets
or b outside brackets

x = aabbb

$Y_i(x)$ = {[aabbb], aabbb}

$\overline{Y}_{i+1}(x)$ = {[$\overset{\star}{a}\overset{\star}{a}$bbb], aa$\overset{\star}{b}\overset{\star}{b}\overset{\star}{b}$}

$Y_{i+1}(x)$ = {[aabbb]}

constraint

harmonic
pruning

One violation for each
surface coda consonant: b], p], etc.

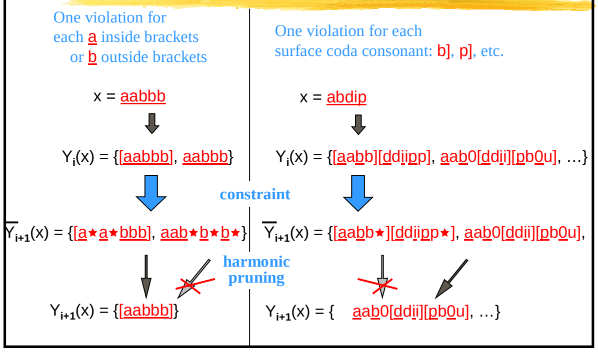
x = abdip

$Y_i(x)$ = {[aabb][ddiipp], aab0[ddii][pb0u], …}

$\overline{Y}_{i+1}(x)$ = {[aab$\overset{\star}{b}$][ddiip$\overset{\star}{p}$], aab0[ddii][pb0u], …}

$Y_{i+1}(x)$ = {   aab0[ddii][pb0u], …}

A General View of Constraints



One violation for
each a inside brackets
or b outside brackets

x = aabbb

$Y_i(x)$ = {[aabbb], aabbb}

$\overline{Y}_{i+1}(x)$ = {[a★a★bbb], aab★b★b★}

$Y_{i+1}(x)$ = {[aabbb]}

constraint

harmonic
pruning

One violation for each
surface coda consonant: b], p], etc.

x = abdip

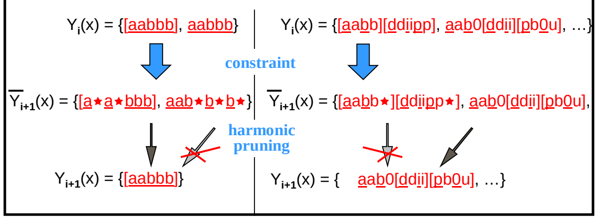
$Y_i(x)$ = {[aabb][ddiipp], aab0[ddii][pb0u], …}

$\overline{Y}_{i+1}(x)$ = {[aabb★][ddiipp★], aab0[ddii][pb0u],

$Y_{i+1}(x)$ = {   aab0[ddii][pb0u], …}

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], aabbb}

$\overline{Y}_{i+1}(x)$ = {[a★a★bbb], aab★b★b★}

$Y_{i+1}(x)$ = {[aabbb]}

constraint

harmonic
pruning

$Y_i(x)$ = {[aabb][ddiipp], aab0[ddii][pb0u], …}

$\overline{Y}_{i+1}(x)$ = {[aabb★][ddiipp★], aab0[ddii][pb0u],

$Y_{i+1}(x)$ = {   aab0[ddii][pb0u], …}

The Harmony Ordering

- An OT grammar really has 4 components:
  - Gen, Pron, harmony ordering, $\underbrace{\text{constraint seq.}}_{\text{language-specific}}$
- Harmony ordering compares 2 starred candidates that share underlying material:
  - Traditional OT says “fewer stars is better”
  - aab0[ddii][pb0u] > [aabb★][ddiipp★]   “0 beats 2”
  - [a★a★bbb] > aab★b★b★   “2 beats 3”
  - Unordered: [a★a★bbb], aab★b★   “2 vs. 2”
  - Unordered: aab0[ddii][pb0u], aab★b★b★   “abdip vs. aabbb”

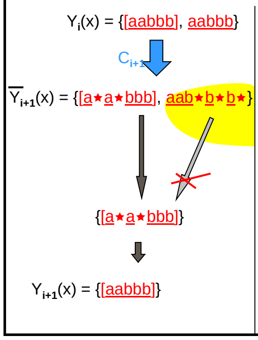
Regular Harmony Orderings

- 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) iff q > r (e.g., [a★a★bbb] > aab★b★b★)
- **H**(q) = range(q o H) = {r: q > r}
  “set of r’s that are worse than q”
- **H**(Q) = range(Q o H) = $\bigcup_{q\in Q}$ {r: q > r}
  “set of r’s that are worse than something in Q”
  (or if Q is an FST, worse than some output of Q)

range(Q o **H**) = $\bigcup_{q\in Q}$ {r: q > r}   (where **H** accepts (q,r) iff q > r)

“set of starred candidates r that are worse than some output of Q”

Using a Regular Harmony Ordering



$Y_i(x)$ = {[aabbb], aabbb}

C_{i+1}

$\overline{Y}_{i+1}(x)$ = {[a★a★bbb], aab★b★b★}

{[a★a★bbb]}

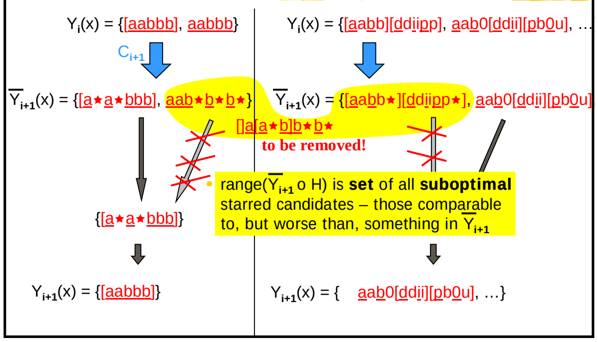
$Y_{i+1}(x)$ = {[aabbb]}

- Y_i is FST that maps each x to its optimal candidates under first i constraints
  By induction, assume it’s regular!
  Note: $Y_i(x) \cap Y_i(x') = \varnothing$
- $\overline{Y}_{i+1} = Y_i$ o C_{i+1} maps each x to same candidates, but starred by constraint i+1
- range($\overline{Y}_{i+1}$ o H) is **set** of all **suboptimal** starred candidates – those comparable to, but worse than, something in $\overline{Y}_{i+1}$

range(Q o **H**) = $\bigcup_{q \in Q}$ {r: q > r}   (where **H** accepts (q,r) iff q > r)

“set of starred candidates r that are worse than some output of Q”

Using a Regular Harmony Ordering



$Y_i(x)$ = {[aabbb], aabbb}

C_{i+1}

$\overline{Y}_{i+1}(x)$ = {[a★a★bbb], aab★b★b★}

{[a★a★bbb]}

$Y_{i+1}(x)$ = {[aabbb]}

$Y_i(x)$ = {[aabb][ddiipp], aab0[ddii][pb0u], …

$\overline{Y}_{i+1}(x)$ = {[aabb★][ddiipp★], aab0[ddii][pb0u]

[]a[a★b]b★b★

to be removed!

- range($\overline{Y}_{i+1}$ o H) is **set** of all **suboptimal** starred candidates – those comparable to, but worse than, something in $\overline{Y}_{i+1}$

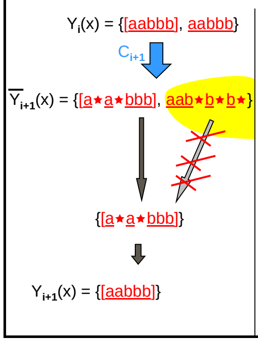
$Y_{i+1}(x)$ = {   aab0[ddii][pb0u], …}

---

range(Q o **H**) = $\bigcup_{q \in Q}$ {r: q > r}   (where **H** accepts (q,r) iff q > r)

“set of starred candidates r that are worse than some output of Q”

Using a Regular Harmony Ordering



$Y_i(x)$ = {[aabbb], aabbb}

C_{i+1}

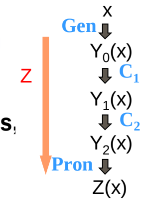
$\overline{Y}_{i+1}(x)$ = {[a★a★bbb], aab★b★b★}

{[a★a★bbb]}

$Y_{i+1}(x)$ = {[aabbb]}

- Y_i is FST that maps each x to its optimal candidates under first i constraints
  Note: $Y_i(x) \cap Y_i(x') = \varnothing$
- $\overline{Y}_{i+1} = Y_i$ o C_{i+1} maps each x to same candidates, but starred by constraint i+1
- range($\overline{Y}_{i+1}$ o H) is set of all **suboptimal** starred candidates – those comparable to, but worse than, something in $\overline{Y}_{i+1}$
- $\overline{Y}_{i+1}$ o ~range($\overline{Y}_{i+1}$ o H) maps x to just the C_{i+1}-optimal candidates in $\overline{Y}_{i+1}(x)$
- Delete ★'s to get Y_{i+1}
  (by composition with another FST)

What Have We Proved?



x

Gen

$Y_0(x)$

C_1

$Y_1(x)$

C_2

$Y_2(x)$

Pron

Z(x)

Z

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

Consequences:
A Family of Optimality Operators oo_H

- Y o C   Inviolable constraint (traditional composition)
- Y oo_H C   Violable constraint with harmony ordering H
- Y o+ C   Traditional OT: harmony compares # of stars
  Not a finite-state operator!
- Y oo C: Binary constraint: “no stars” > “some stars”
  q > r
  This H is a regular relation:
  Can build an FST that accepts (q,r)
  iff q has no stars **and** r has some stars,
  and q,r have same underlying x
  Therefore oo is a finite-state operator!
  If Y is a regular relation and C is a regular constraint,
  then Y oo C is a regular relation

Consequences:
A Family of Optimality Operators oo_H

- Y o C   Inviolable constraint (traditional composition)
- Y oo_H C   Violable constraint with harmony ordering H
- Y o+ C   Traditional OT: harmony compares # of stars
  Not a finite-state operator!
- Y oo C: Binary constraint: “no stars” > “some stars”
- Y oo_3 C   Bounded constraint: 0 > 1 > 2 > 3 = 4 = 5 …
  Frank & Satta 1998; Karttunen 1998
  Yields big approximate FSTs that count
- Y oo⊂ C   Subset approximation to o+ (traditional OT)
  Gerdemann & van Noord 2000
  Exact for many grammars, though not all
- Y o> C, Y <o C }   Directional constraint (Eisner 2000)
  Non-traditional OT – linguistic motivation

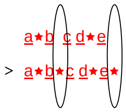
For each operator, the paper shows how to construct H as a finite-state transducer.

Consequences:
A Family of Optimality Operators oo_H

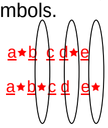
- For each operator, the paper shows how to construct H as a finite-state transducer.
- Z = (Gen oo_H C1 oo_H C2) o Pron becomes, e.g.,
  - Z = (Gen oo C1 oo_3 C2) o Pron
  - Z = (Gen o⊂ C1 oo C2) o Pron
  - Z = (Gen o> C1 <o C2) o Pron

Subset Approximation

- Y oo⊂ 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★b c d★e
> a★b★c d★e★

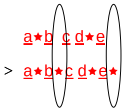
top candidate wins


a★b c d★e
a★b★c d e★

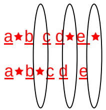
incomparable; both survive

Directional Constraints

- Y o> C
  Y <o C
  Directional constraint (Eisner 2000)
  Non-traditional OT – linguistic motivation
- As for many harmony orderings, ignores surface symbols. Just looks at underlying and starred symbols.


a★b c d★e
> a★b★c d★e★

always same result as subset approx if subset approx has a result at all


a★b c d★e ★
a★b★c d e

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.

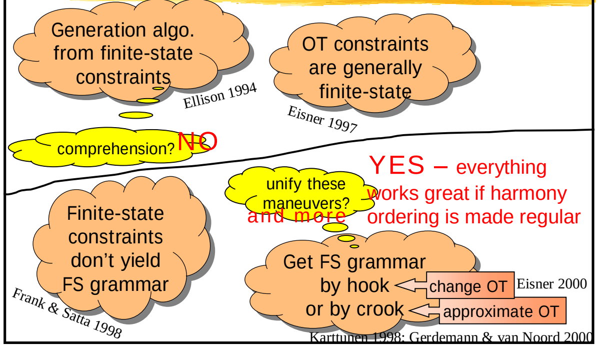
Directional Constraints

- So one nice outcome of our construction is an **algebraic** construction for directional constraints – much easier to understand than machine construction.

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

Ellison 1994

OT constraints are generally finite-state

Eisner 1997

comprehension? NO

unify these maneuvers?

and more

YES – everything works great if harmony ordering is made regular

Finite-state constraints don't yield FS grammar

Frank & Satta 1998

Get FS grammar by hook or by crook

change OT

Eisner 2000

approximate OT

Karttunen 1998; Gerdemann & van Noord 2000

FIN