Efficient Generation in Primitive Optimality Theory

Jason Eisner
University of Pennsylvania
ACL - 1997

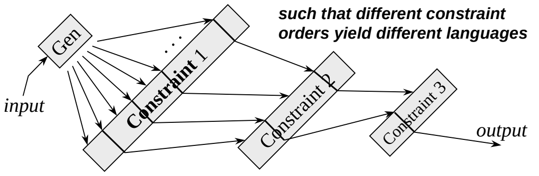
Overview

- **A new formalism**
  - – What is Optimality Theory? (OT)
  - – Primitive Optimality Theory (OTP)
- **Some results for OTP**
  - – Linguistic fit
  - – Formal results
  - – Practical results on generation

2

What Is Optimality Theory?

- Prince & Smolensky (1993)
- Alternative to stepwise derivation
- **Stepwise winnowing of candidate set**



such that different constraint orders yield different languages

3

Filtering, OT-style

★★ = candidate violates constraint twice

	Constraint 1	Constraint 2	Constraint 3	Constraint 4
Candidate A	★		★	★★★
Candidate B		★★	★	
Candidate C	★	★		
Candidate D		★★★		
Candidate E		★★	★	★
Candidate F	★★	★★★		★

constraint would prefer A, but only allowed to break tie among B,D,E

4

Formalisms in phonology

Two communities with different needs ...

	Linguists	Computer Scientists	
SPE (1968)	string rewrites (restricted)	finite-state transducers	*(equivalent)*
Autosegmental phonology (1979)	tier-local rewrites	finite-state transducers	*(equivalent)*
OT (1993)	? **informal English** ?	**OTFS (finite-state)**	

5

Unformalized OT isn't a theory

	Linguists	Computer Scientists
OT (1993)	?	OTFS (finite-state)

We need a formalism here, not informal English.

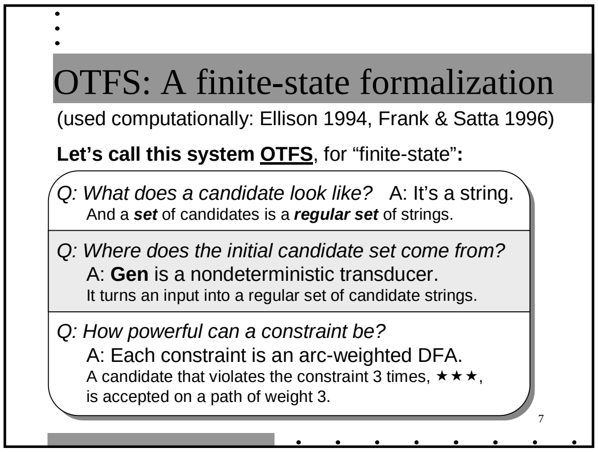
Using English, can express *any* constraint

⇒ *describe impossible languages*

⇒ *specify any grammar with 1 big constraint*
(undermines claim that typology = constraint reranking)

⇒ *no algorithms (generation, parsing, learning)*

6



OTFS: A finite-state formalization

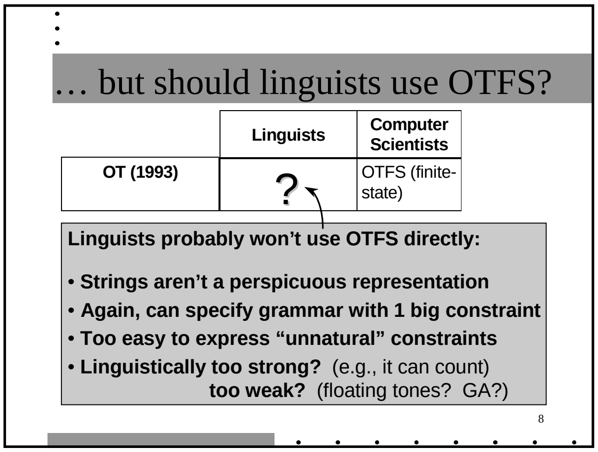
(used computationally: Ellison 1994, Frank & Satta 1996)

Let's call this system <u>OTFS</u>, for "finite-state"**:**

Q: What does a candidate look like? A: It's a string.
And a ***set*** of candidates is a ***regular set*** of strings.

Q: Where does the initial candidate set come from?
A: **Gen** is a nondeterministic transducer.
It turns an input into a regular set of candidate strings.

Q: How powerful can a constraint be?
A: Each constraint is an arc-weighted DFA.
A candidate that violates the constraint 3 times, ★★★,
is accepted on a path of weight 3.

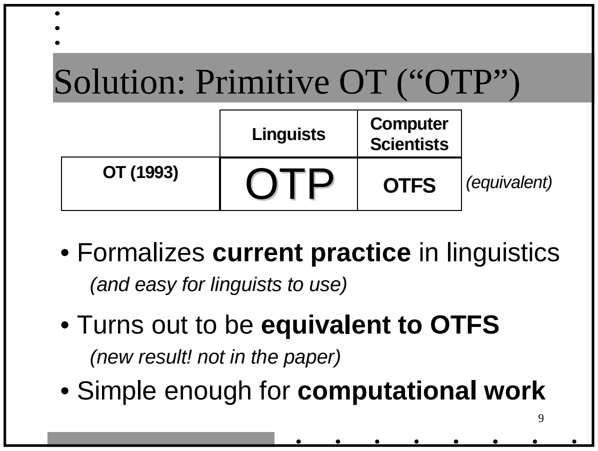


… but should linguists use OTFS?

	Linguists	**Computer Scientists**
OT (1993)	?	OTFS (finite-state)

Linguists probably won't use OTFS directly:

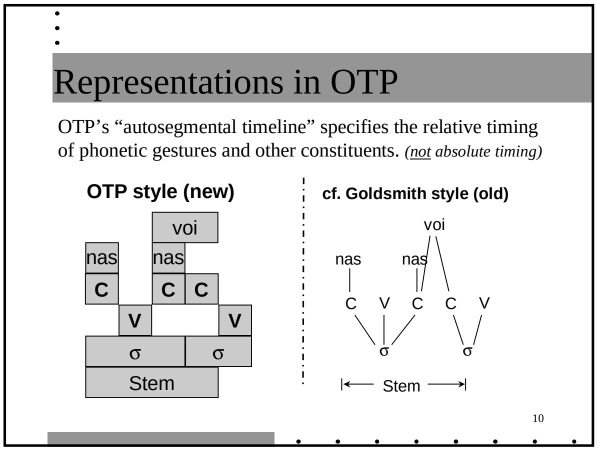
- **Strings aren't a perspicuous representation**
- **Again, can specify grammar with 1 big constraint**
- **Too easy to express "unnatural" constraints**
- **Linguistically too strong?** (e.g., it can count)
  too weak? (floating tones? GA?)



Solution: Primitive OT ("OTP")

	Linguists	**Computer Scientists**	
OT (1993)	OTP	**OTFS**	*(equivalent)*

- Formalizes **current practice** in linguistics
  (and easy for linguists to use)
- Turns out to be **equivalent to OTFS**
  (new result! not in the paper)
- Simple enough for **computational work**

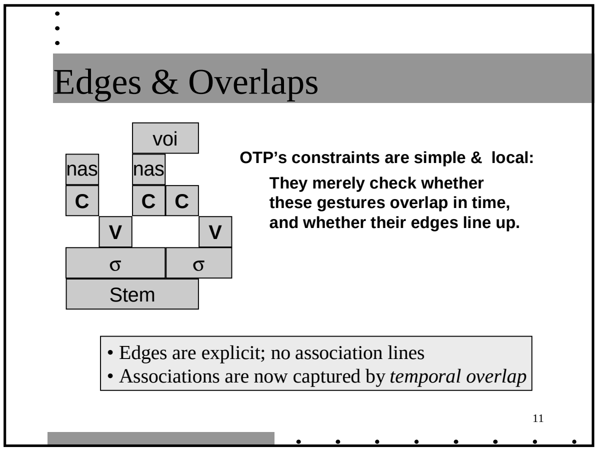


Representations in OTP

OTP's "autosegmental timeline" specifies the relative timing of phonetic gestures and other constituents. *(<u>not</u> absolute timing)*

OTP style (new)

cf. Goldsmith style (old)

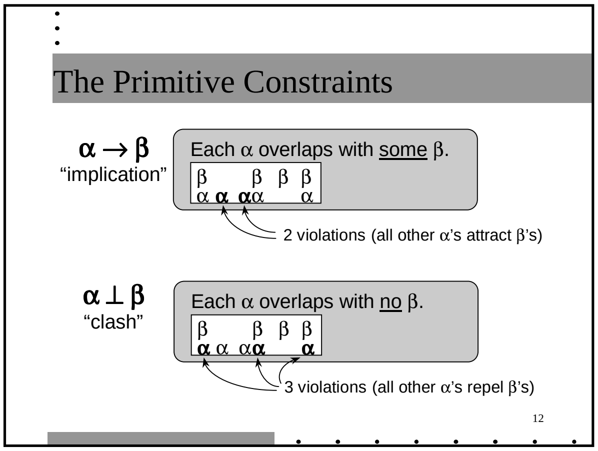


Edges & Overlaps

OTP's constraints are simple & local:

They merely check whether these gestures overlap in time, and whether their edges line up.

- Edges are explicit; no association lines
- Associations are now captured by *temporal overlap*



The Primitive Constraints

α → β
"implication"

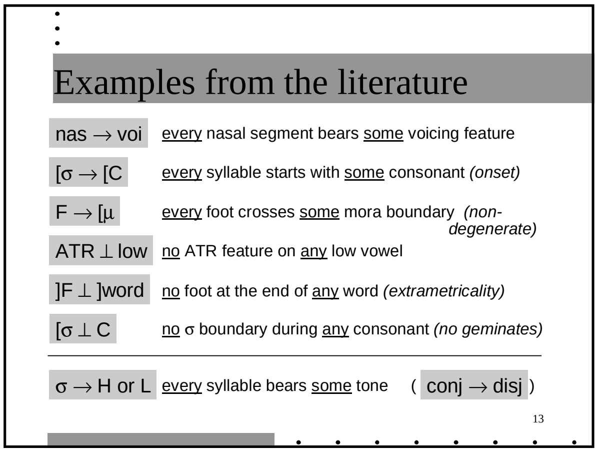
Each α overlaps with <u>some</u> β.

2 violations (all other α's attract β's)

α ⊥ β
"clash"

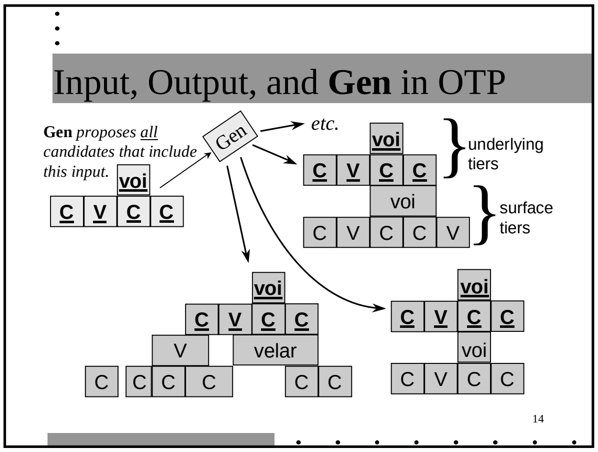
Each α overlaps with <u>no</u> β.

3 violations (all other α's repel β's)



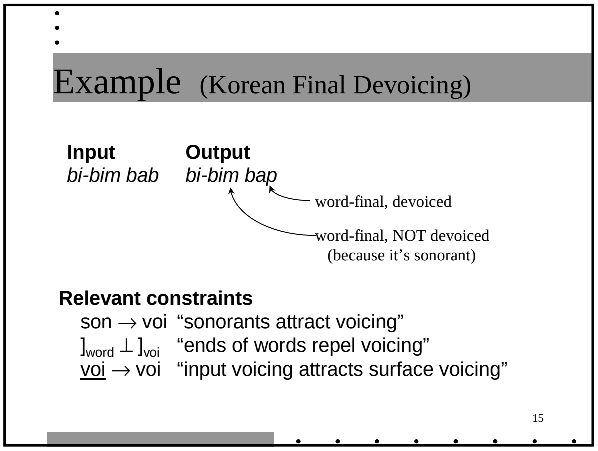
Examples from the literature

nas → voi	every nasal segment bears some voicing feature
[σ → [C	every syllable starts with some consonant *(onset)*
F → [μ	every foot crosses some mora boundary *(non-degenerate)*
ATR ⊥ low	no ATR feature on any low vowel
]F ⊥]word	no foot at the end of any word *(extrametricality)*
[σ ⊥ C	no σ boundary during any consonant *(no geminates)*
σ → H or L	every syllable bears some tone (conj → disj)



Input, Output, and **Gen** in OTP

Gen *proposes all candidates that include this input.*



Example (Korean Final Devoicing)

Input *bi-bim bab*

Output *bi-bim bap*

word-final, devoiced

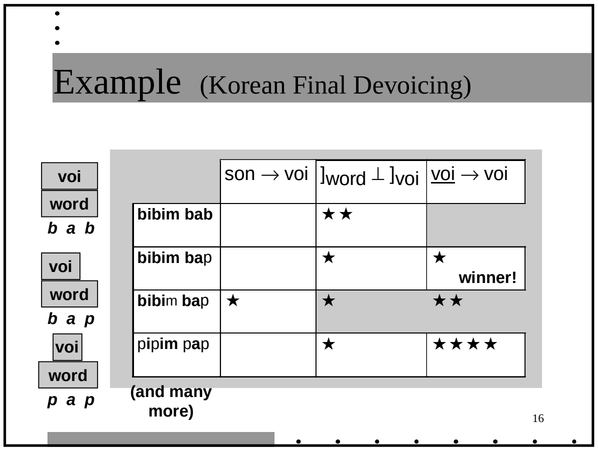
word-final, NOT devoiced
(because it's sonorant)

Relevant constraints

son → voi "sonorants attract voicing"

$]_{\text{word}} \perp]_{\text{voi}}$ "ends of words repel voicing"

$\underline{\text{voi}}$ → voi "input voicing attracts surface voicing"

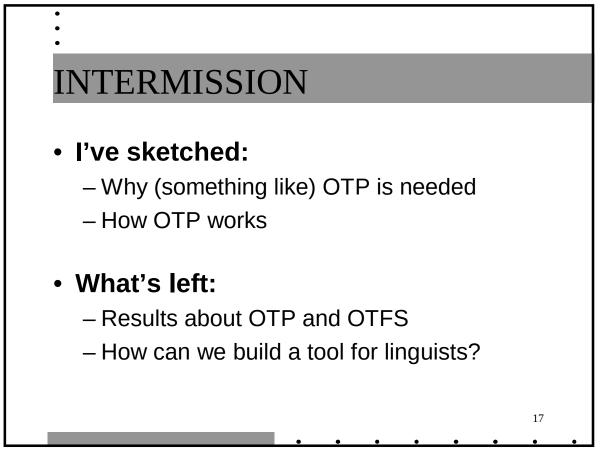


Example (Korean Final Devoicing)

voi
word
b a b
voi
word
b a p
voi
word
p a p

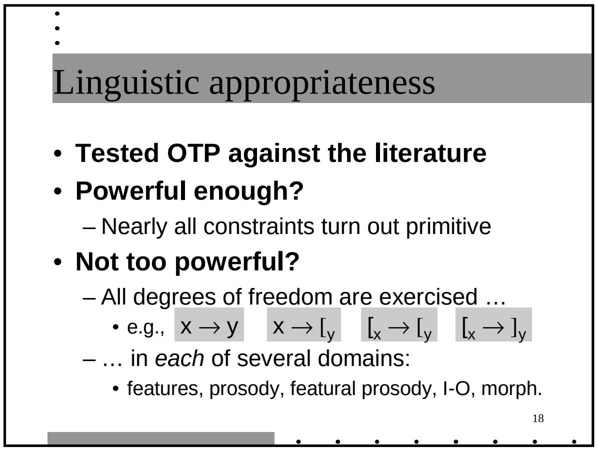
	son → voi	$]_{\text{word}} \perp]_{\text{voi}}$	$\underline{\text{voi}}$ → voi
bibim bab		★★	
bibim bap		★	★ **winner!**
bibim **ba**p	★	★	★★
pipi**m** pap		★	★★★★

(and many more)



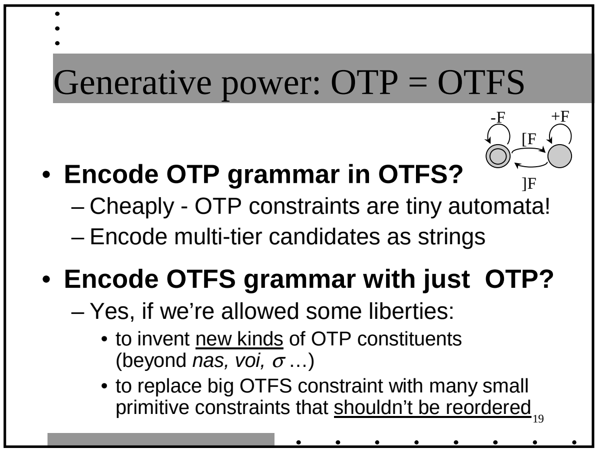
INTERMISSION

- **I've sketched:**
  - – Why (something like) OTP is needed
  - – How OTP works
- **What's left:**
  - – Results about OTP and OTFS
  - – How can we build a tool for linguists?



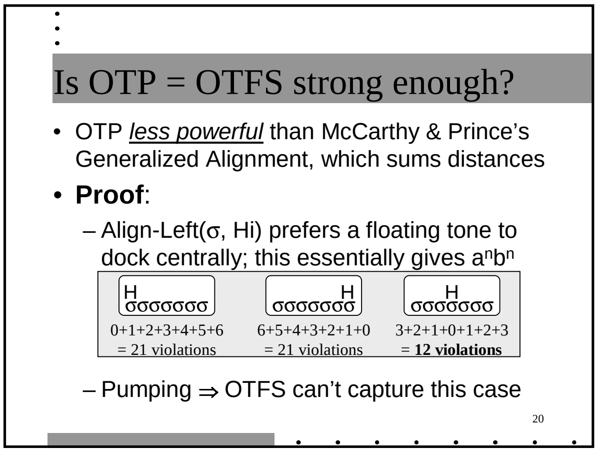
Linguistic appropriateness

- **Tested OTP against the literature**
- **Powerful enough?**
  - – Nearly all constraints turn out primitive
- **Not too powerful?**
  - – All degrees of freedom are exercised …
    - e.g., $x \rightarrow y$ $x \rightarrow [_y$ $[_x \rightarrow [_y$ $[_x \rightarrow]_y$
  - – … in *each* of several domains:
    - features, prosody, featural prosody, I-O, morph.



Generative power: OTP = OTFS

- **Encode OTP grammar in OTFS?**
  - Cheaply - OTP constraints are tiny automata!
  - Encode multi-tier candidates as strings
- **Encode OTFS grammar with just OTP?**
  - Yes, if we're allowed some liberties:
    - to invent <u>new kinds</u> of OTP constituents (beyond *nas, voi, σ …*)
    - to replace big OTFS constraint with many small primitive constraints that <u>shouldn't be reordered</u>

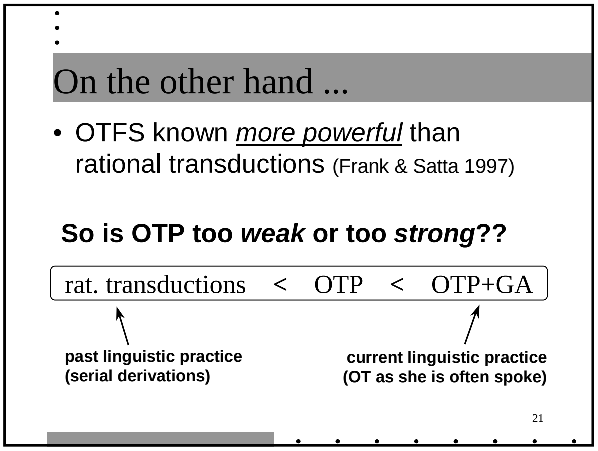


Is OTP = OTFS strong enough?

- OTP *<u>less powerful</u>* than McCarthy & Prince's Generalized Alignment, which sums distances
- **Proof**:
  - Align-Left(σ, Hi) prefers a floating tone to dock centrally; this essentially gives $a^n b^n$

H σσσσσσσ	H σσσσσσσ	H σσσσσσσ
0+1+2+3+4+5+6 = 21 violations	6+5+4+3+2+1+0 = 21 violations	3+2+1+0+1+2+3 = **12 violations**

  - Pumping ⇒ OTFS can't capture this case



On the other hand ...

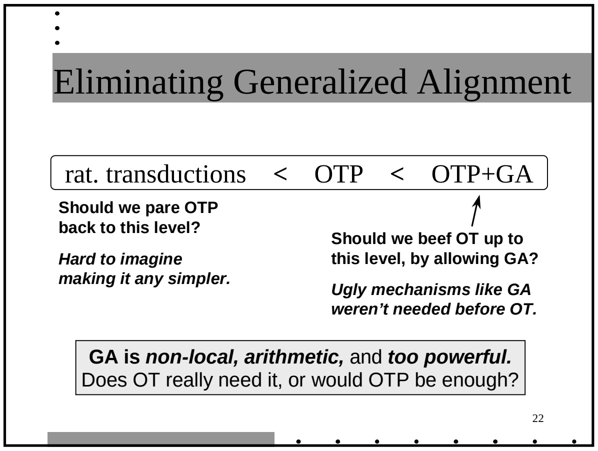
- OTFS known *<u>more powerful</u>* than rational transductions (Frank & Satta 1997)

So is OTP too *weak* or too *strong*??

rat. transductions < OTP < OTP+GA

past linguistic practice (serial derivations)

current linguistic practice (OT as she is often spoke)



Eliminating Generalized Alignment

rat. transductions < OTP < OTP+GA

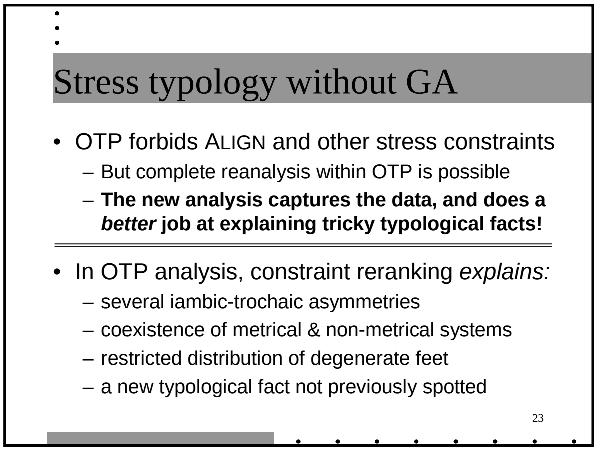
Should we pare OTP back to this level?

Hard to imagine making it any simpler.

Should we beef OT up to this level, by allowing GA?

Ugly mechanisms like GA weren't needed before OT.

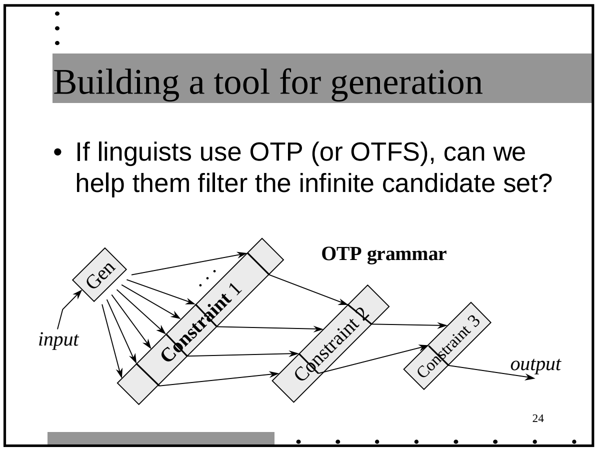
GA is *non-local, arithmetic,* and ***too powerful.***
Does OT really need it, or would OTP be enough?



Stress typology without GA

- OTP forbids ALIGN and other stress constraints
  - But complete reanalysis within OTP is possible
  - **The new analysis captures the data, and does a *better* job at explaining tricky typological facts!**

- In OTP analysis, constraint reranking *explains:*
  - several iambic-trochaic asymmetries
  - coexistence of metrical & non-metrical systems
  - restricted distribution of degenerate feet
  - a new typological fact not previously spotted

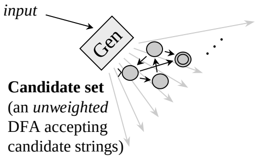


Building a tool for generation

- If linguists use OTP (or OTFS), can we help them filter the infinite candidate set?

Ellison's generation method (1994) (simplified)

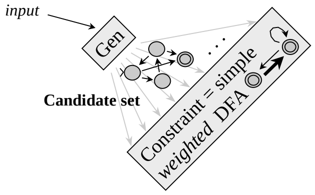
• Encode every candidate as a string



Candidate set
(an *unweighted*
DFA accepting
candidate strings)

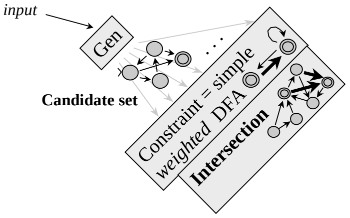
Ellison's generation method (1994)

• Encode every candidate as a string
• A constraint is an arc-weighted DFA that evaluates strings
– Weight of the accepting path = degree of violation



Ellison's generation method (1994)

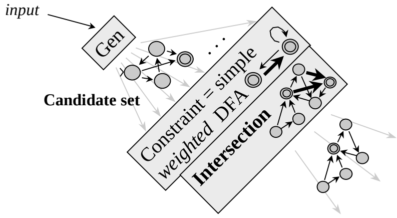
• Encode every candidate as a string
• A constraint is an arc-weighted DFA that scores strings
– Weight of accepting path = degree of violation



yields *weighted*
DFA that accepts
the candidates and
scores each one

Ellison's generation method (1994)

• Encode every candidate as a string
• A constraint is a weighted DFA that scores strings
– Weight of accepting path = degree of violation



Prune back to
min-weight
accepting paths
(best candidates)

Alas - Explosion of states

- Ellison's algorithm is ***impractical*** for OTP
- Why? Initial candidate set is **huge DFA**
  - 2^k states: An intersection of *many* orthogonal 2-state automata
  - For every left edge on any tier, there must be a right edge
  - So state must keep track: "I'm **in C**, and **in nas**, but **out of σ**..."
- Mostly the same work gets duplicated at nasal and non-nasal states, etc.
  - Wasteful: stress doesn't care if foot is nasal!

Solution: Factored automata

- **Clumsy big automata arise in OTP when we intersect many small automata**

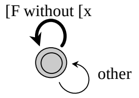
- **Just maintain the list of small automata**
  - Like storing a large integer as a list of prime factors
  - Try to compute in this "factored" domain for as long as possible: defer intersection

Solution: Factored automata

Candidate set

nas tier is well-formed
∩
x tier is well-formed
∩
F tier is well-formed
∩
input material
∩
word never ends
on voiced obstruent
etc.

new constraint $[_F \rightarrow [_x$

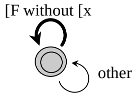


intersect
<u>candidate set</u>
with <u>new constraint</u>
and prune back
to lightest paths

Solution: Factored automata

Candidate set

nas tier is well-formed
∩
x tier is well-formed
∩
F tier is well-formed
∩
input material
∩
word never ends
on voiced obstruent
etc.



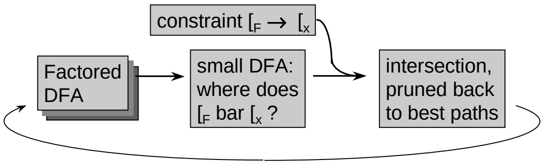
Just add this as a new factor?

No, must follow heavy arc as rarely as possible.

***CERTAIN* of the existing factors force us to take heavy arc. Ignore the other factors!**

Factored automata

- **Filter candidates via "best intersection"**
  - Candidate set = unweighted factored DFA
  - Constraint = simple weighted DFA
  - Goal: Winnow candidate set (i.e., add new factor)


constraint $[_F \rightarrow [_x$
Factored DFA
small DFA: where does $[_F$ bar $[_x$?
intersection, pruned back to best paths

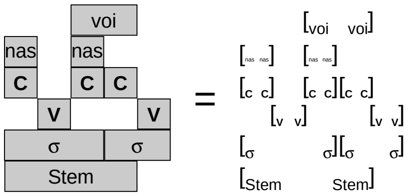
Good news & bad news

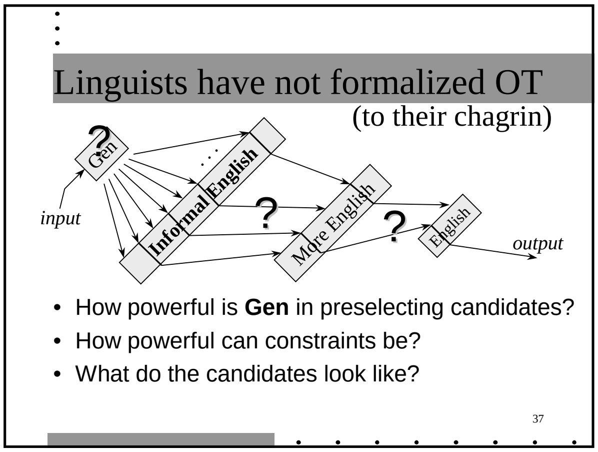
- **Factored methods work correctly**
- **Can get 100x speedup on real problem**
- **But what is the worst case?**
  - O(n log n) on the size of the input
  - but <u>**NP-complete**</u> on the size of the grammar!
    - can encode Hamilton Path as an OTP grammar
  - Significant if grammar keeps changing
    - learning algorithms (Tesar 1997)
    - tools for linguists to develop grammars

Summary

- **OTP: A clean formalism for linguists**
  - simple, empirically plausible version of OT
  - good fit to *current* linguistic practice
  - can force fruitful *new* analyses
- **Formal results**
  - transducers **<** OTFS **=** OTP **<** OTP+GA
  - the generation problem is NP-complete
- **Practical results on generation**
  - use factored automata for efficiency

Representation = Edge Ordering


voi
nas
nas
C
C
C
V
V
σ
σ
Stem
=
$[_{voi}\ \ _{voi}]$
$[_{nas}\ _{nas}]$ $[_{nas}\ _{nas}]$
$[_C\ _C]$ $[_C\ _C][_C\ _C]$
$[_V\ _V]$ $[_V\ _V]$
$[_\sigma\ \ _\sigma][_\sigma\ \ _\sigma]$
$[_{Stem}\ \ _{Stem}]$

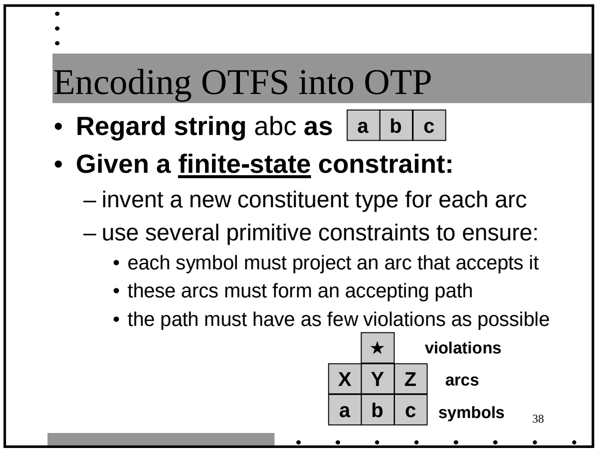


Linguists have not formalized OT

(to their chagrin)

- How powerful is **Gen** in preselecting candidates?
- How powerful can constraints be?
- What do the candidates look like?

37



Encoding OTFS into OTP

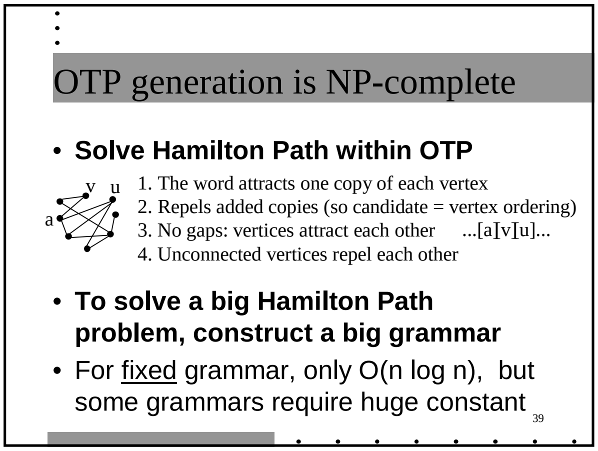
- **Regard string** abc **as**

a	b	c

- **Given a <u>finite-state</u> constraint:**
  - – invent a new constituent type for each arc
  - – use several primitive constraints to ensure:
    - each symbol must project an arc that accepts it
    - these arcs must form an accepting path
    - the path must have as few violations as possible

	★		**violations**
X	Y	Z	**arcs**
a	b	c	**symbols**

38



OTP generation is NP-complete

- **Solve Hamilton Path within OTP**

1. The word attracts one copy of each vertex
2. Repels added copies (so candidate = vertex ordering)
3. No gaps: vertices attract each other ...[a[v[u]...
4. Unconnected vertices repel each other

- **To solve a big Hamilton Path problem, construct a big grammar**
- For <u>fixed</u> grammar, only O(n log n), but some grammars require huge constant

39