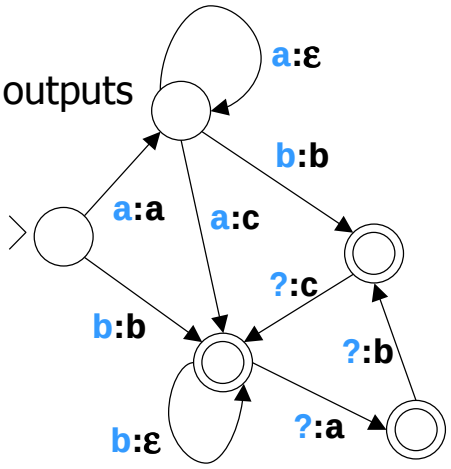
a little pre-talk review

Regular Relation (of strings)

- *Relation:* like a function, but multiple outputs ok
- *Regular:* finite-state
- *Transducer:* automaton w/ outputs

- b → ? a → ?
- aaaaa → ?

- Invertible?
- Closed under composition?



By way of review – or maybe first time for some of you!

Computes functions on strings – which is essentially what phonology (e.g.) is all about

(non-string representations in phonology can still be encoded as strings)

b -> {b}

a -> {} might look like -> {b} but it dead-ends.

aaaaa -> {ac, aca, acab, acabc}

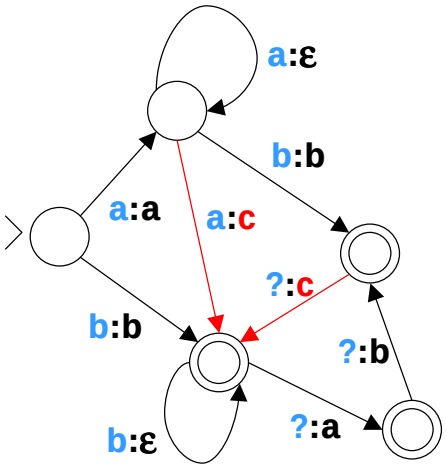
Notice that if we're not careful we get stuck – aaaaa path is no good.

These relations are closed under composition & invertible, very nice

a little pre-talk review

Regular Relation (of strings)

- Can weight the arcs: → vs. →
- a → {} b → {b}
- aaaaa → {ac, aca, acab, acabc}



- How to find best outputs?
  - For aaaaa?
  - For all inputs at once?

If we weight, then we try to avoid output c's in this case.

We can do that for input b easily.

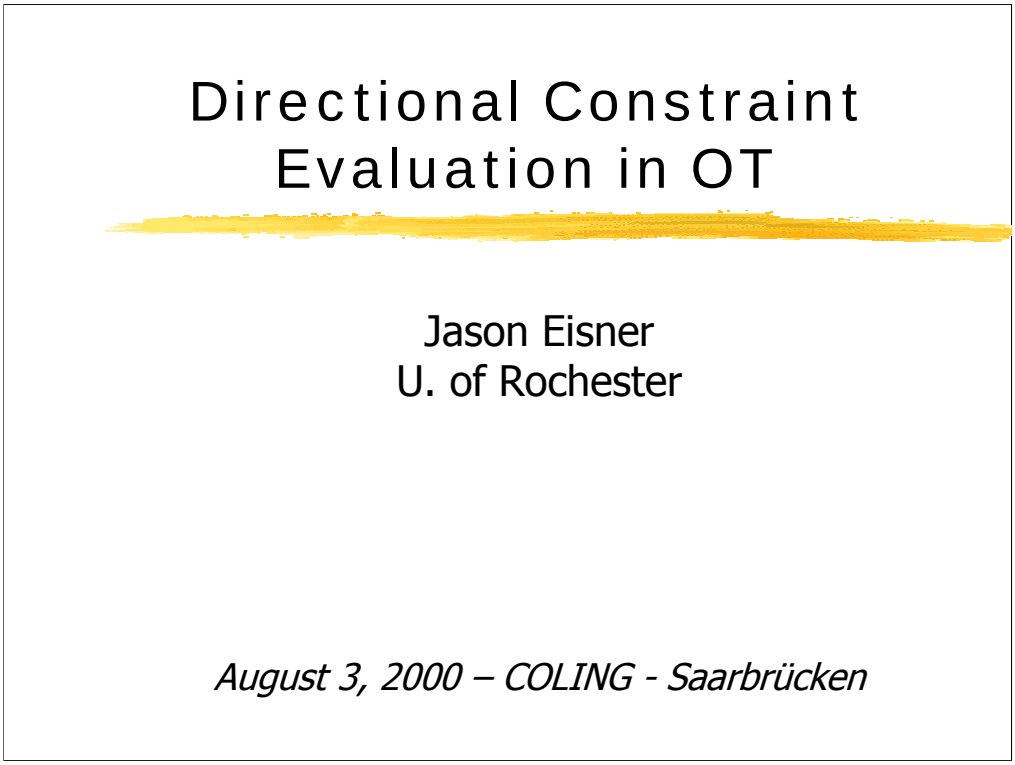
For output aaaaa we can't, since any way of getting to final state has at least one c.

But we prefer ac, aca, acab over acabc which has two c's.

For a given input string, we can find set of lowest-cost paths, here ac, aca, acab.

How? Answer: Dijkstra.

Do you think we can strip the entire machine down to just its best paths?



going to suggest a modification to OT, and here's why.

Synopsis: Fixing OT's Power

- Consensus: Phonology = regular relation
  E.g., composition of little local adjustments (= FSTs)
- Problem: Even finite-state OT is worse than that
  Global "counting" (Frank & Satta 1998)
- Problem: Phonologists want to add even more
  Try to capture iterativity by Gen. Alignment constraints
- Solution: In OT, replace counting by iterativity
  Each constraint does an iterative optimization

since Kaplan & Kay it's been believed that phonology is regular

In OT, even when you write your constraints using finite-state machines, the grammar doesn't compile into a regular relation. We'll see why that is, but basically it's because OT can count. It tries to minimize the number of violations in a candidate form.

Generalized Alignment, which fails to reduce to finite-state constraints, which in turn fails to reduce regular relations. So we have two levels of excess power to cut back.

Well, if counting is the bad thing, and iteration is something we want, let's replace counting by a kind of iterative mechanism. Not GA, something less powerful.

Iterative is a phonologist's word for directional – it means you write a for loop that iterates down the positions of the string and does the same thing at each position.

This is still going to be OT – there's no iterative mechanism to scan the string and assign syllable structure from left to right, or anything like that. We're still finding the optimal candidate under constraints. Constraints can't alter the string. But they can *evaluate* the string from left to right. That's the new idea - each constraint uses iteration to *compare* candidates.

Outline

- Review of Optimality Theory
- The new "directional constraints" idea
- Linguistically: Fits the facts better
- Computationally: Removes excess power

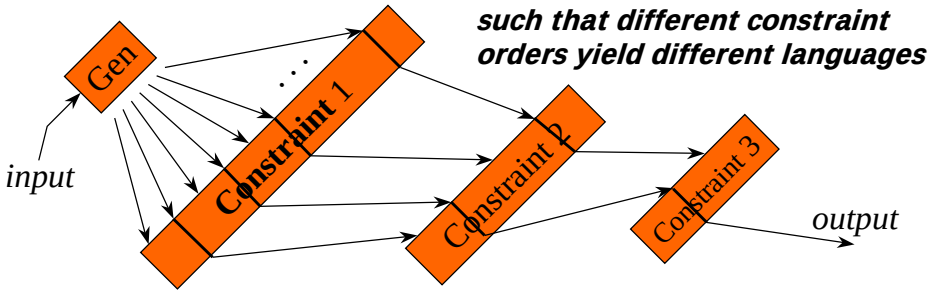
- Formal stuff
  - The proposal
  - Compilation into finite-state transducers
  - Expressive power of directional constraints

… The formal stuff is slightly intricate to set up, but not conceptually difficult and well worth it.

The central result is that directional constraints *can* indeed be compiled into regular relations, and they're more expressive than bounded constraints.

What Is Optimality Theory?

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



Been around for several years

Theory of the input-to-output map in phonology

Instead of changing input to output step by step, as we used to do, we generate a bunch of

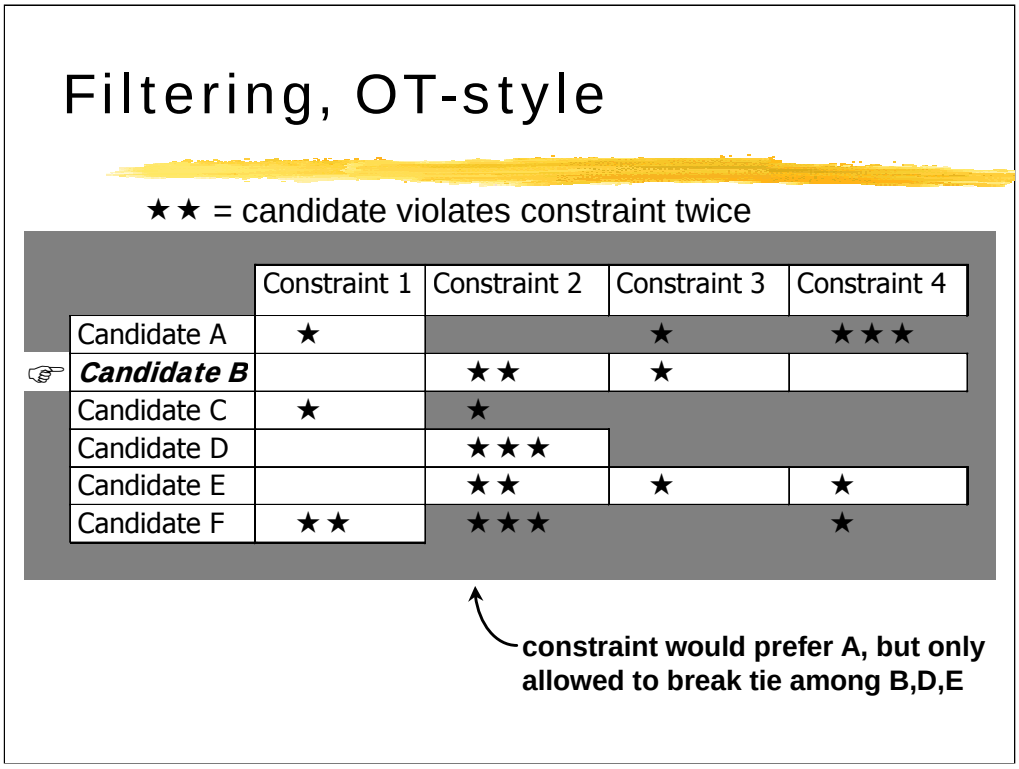
possible candidates, and filter them step by step.

So we give our input to a candidate generator, which produces a lot of possible pronunciations.

Constraint 1 passes only the ones it likes best, or hates least. It may not like any of them very much but it makes the best of a bad situation.

So all of these tied according to constraint 1. But constraint 2 may distinguish among them. It passes some on, and so forth.

And by the way, if you reorder these you get different languages. If you want to hear more about that, come to my talk on Sunday.



At which point, the notorious pointy hand of cognition comes and takes B and puts it on the tip of your tongue for ready access.

A Troublesome Example

Input: bantodibo

		Harmony	Faithfulness
	ban.to.di.bo	★	
	ben.ti.do.bu	★	★★★★
	ban.ta.da.ba		★★★
☞	bon.to.do.bo		★★

"Majority assimilation" – impossible with FST -- and doesn't happen in practice!

How would you implement each constraint with an FSA?

So we end up choosing bontodobo, with all o's because there were more o's to start with.

Can't do this with a finite state machine! You'd have to keep count of a's and o's in the input and at the end decide which count was greater. Maybe at some point you'd get up to 85 more a's than o's, which means that for o's to win you'd need 86 or more, so you need to encode 85 in the state. But then you need an unbdd # of states.

Can you do it with a CFG or PDA? Answer: if language only has 2 vowels (or 2 classes of vowels). But OT allows more vowels as above.

But harmony *never* works by majority, not even for two classes (the usual case) short strings.

Outline

- Review of Optimality Theory
- The new "directional constraints" idea
- Linguistically: Fits the facts better
- Computationally: Removes excess power

- Formal stuff
  - The proposal
  - Compilation into finite-state transducers
  - Expressive power of directional constraints

An Artificial Example

Candidates have 1, 2, 3, 4 violations of **NoCoda**

		NoCoda
☞	ban.to.di.bo	★
	ban.ton.di.bo	★★
	ban.to.dim.bon	★★★
	ban.ton.dim.bon	★★★★

NoCoda is violated by syllable codas.

And I've marked in red the codas *where* those violations occur.

The fourth candidate has four red codas, so it has four violations of NoCoda, as indicated by the stars.

The first candidate, ban.to.di.bo, is clearly the best pronunciation - it has the fewest codas, and it wins.

An Artificial Example

Add a higher-ranked constraint
This forces a tradeoff: ton vs. dim.bon

		C	NoCoda
	ban.to.di.bo	★	★
☞	ban.ton.di.bo		★★
	ban.to.dim.bon		★★★
	ban.ton.dim.bon		★★★★

But suppose a higher ranked constraint knocks it out.

The remaining three candidates leave us with an interesting choice about where the violations fall.

The second candidate violates on ton, the third on dim.bon. The last violates on both …

Traditionally in OT, we say, ton has only one violation, dim.bon has two, so we prefer ton. And that's what wins here.

But suppose NoCoda were sensitive not only to the number of violations, but also to their location.

An Artificial Example

Imagine splitting **NoCoda** into 4 syllable-specific constraints

		C	NoCoda σ1	NoCoda σ2	NoCoda σ3	NoCoda σ4
	ban.to.di.bo	★		★		
☞	ban.ton.di.bo			★★		
	ban.to.dim.bon			★★★		
	ban.ton.dim.bon			★★★★		

Imagine splitting it into four subconstraints - one for the first syllable, one for the second … ranked in that order.

And we'll put the stars in the columns where the violations actually fell.

An Artificial Example

Imagine splitting **NoCoda** into 4 syllable-specific constraints
Now ban.to.dim.bon wins - more violations but they're later

		C	NoCoda σ1	NoCoda σ2	NoCoda σ3	NoCoda σ4
	ban.to.di.bo	★	★			
	ban.ton.di.bo		★	★		
☞	ban.to.dim.bon		★		★	★
	ban.ton.dim.bon		★	★	★	★

(don't read the slide)

Well, that changes things!

All three candidates are equally bad on the first syllable. So they all stay in the running.

Second syllable -- oop! Sudden death. Two candidates violate, they're gone. So we have a winner already.

I constructed this example so that the winner would be different from before. That is, it's not the one with the fewest overall violations.

That is, a directional constraint is willing to take more violations so long as it can postpone the pain. Don't hurt me now! Do whatever you want to me later, just not now! It pushes them rightward.

This is directional constraint evaluation from left to right.

Sometimes, instead of ranking 1 2 3 4 …

An Artificial Example

For "right-to-left" evaluation, reverse order (σ4 first)

		C	NoCoda σ4	σ3	σ2	σ1
	ban.to.di.bo	★				★
☞	ban.ton.di.bo				★	★
	ban.to.dim.bon		★	★		★
	ban.ton.dim.bon		★	★	★	★

… we might want to rank 4 3 2 1, which gives us right-to-left evaluation.

The idea is, just as some constraints specify "evaluate me left to right," others specify right to left evaluation. That is, they want to push any violations to the left of the form.

Aaaah, I'll take the pain earlier, but don't hurt me late – I wanna die happy! This one is the deferred gratification strategy, and if you stayed in studying on Saturday nights, I guess you know all about it. Your reward awaits

Ok, so that's the intuition. I've suppressed some interesting detail, which I'll get to later. But first I'd like to motivate it.

Outline

- Review of Optimality Theory
- The new "directional constraints" idea
- Linguistically: Fits the facts better
- Computationally: Removes excess power

- Formal stuff
  - The proposal
  - Compilation into finite-state transducers
  - Expressive power of directional constraints

When is Directional Different?

- The crucial configuration:

		σ1	σ2	σ3	σ4
	ban.to.di.bo	★			
	ban.ton.di.bo	★	★		
☞	ban.to.dim.bon	★		★	★

solve location conflict by ranking locations
(sound familiar?)

- Forced location tradeoff
- Can choose where to violate, but must violate *somewhere*
- Locations aren't "orthogonal"

When does it even matter that we use directional constraints?

Here's the crucial configuration: …

… Doesn't that sound familiar? It's the same mechanism we use to resolve constraint conflict;

you can't satisfy both constraint, but one is more important. Strict ranking of constraints. Here we have location conflict within a constraint, so we do strict ranking of locations. Earlier in the form is more important than later in the form.

When is Directional Different?

- The crucial configuration:

		σ1	σ2	σ3	σ4
	ban.to.di.bo	★			
	ban.ton.di.bo	★	★		
☞	ban.to.dim.bon	★		★	★

- But if candidate 1 were available …

Well, it's because candidate 1 is missing!

When is Directional Different?

- But usually locations *are* orthogonal:

		σ1	σ2	σ3	σ4
☞	ban.to.di.bo	★			
	ban.ton.di.bo	★	★		
	ban.to.dim.bon	★		★	★

- Usually, if you can satisfy σ2 and σ3 separately, you can satisfy them together
- Same winner under *either* counting or directional eval. (satisfies everywhere possible)

If we still had candidate 1, then the tradeoff between candidates 2 and 3 would be meaningless

You don't have to choose between syllable 2 and syllable 3 - you can satisfy NoCoda on both!

And in fact that's the usual circumstance, that something like candidate 1 exists … Locations are orthogonal to each other. Satisfying one doesn't affect your ability to satisfy the other.

…

Linguistic Hypothesis

- Q: When is directional evaluation different?
- A: When something forces a location tradeoff.

- Hypothesis: Languages always resolve these cases directionally.

So this is not a very radical change to OT, since usually, directional and summing evaluation make the same predictions.

They only differ if some higher-ranked constraint kills off candidate 1.

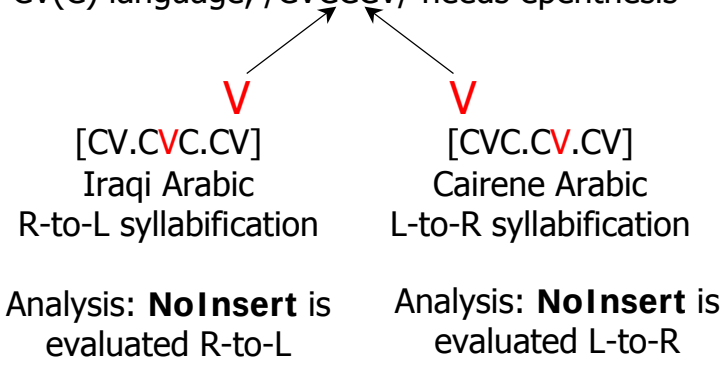
That leaves a candidate set with a location tradeoff.

And in that case, directional constraints predict that the phonology will push violations to one end of the form.

Test Cases for Directionality

- Prosodic groupings
  - Syllabification

In a CV(C) language, /CVCCCV/ needs epenthesis



Prosodic groupings - syllables and feet - sometimes lead to location tradeoffs.

Trying to get one edge of a prosodic constituent right can hurt you on the other edge.

Got to break up that triple-consonant cluster CCC, because it can't be syllabified otherwise.

We could insert V here, leading to this pattern, or here, leading to this pattern.

The traditional example of the latter (e.g. Ito 1989) is Cairene Arabic.

Traditionally called L-R syllabification, because it greedily syllabifies as much as it can of the input form, from left to right, until it is *forced* to insert something.

And in OT, we can think of this as a directional L-R constraint - insertion is bad, postpone it as long as possible.

Higher level constraints force us to insert sometime, because they require CV(C) structure and don't allow C deletion.

But we'll postpone that insertion. Imagine as before that *Insert is broken up into lots of little constraints, one for each position in the input form.

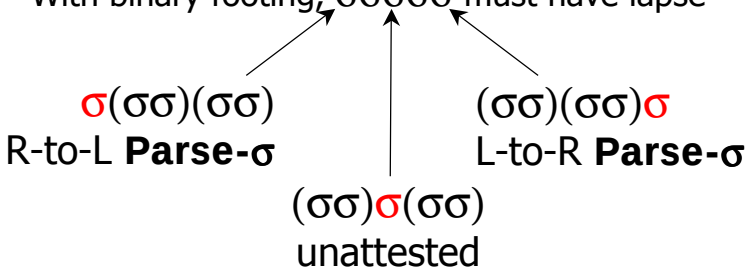
By the way, other pattern is also observed, and that's the same constraint evaluated right-to-left.

Of course, in OT, there's no *procedure* that goes right to left, syllabifying. We optimize over all possible syllabifications. But when we're comparing candidates to decide which is optimal, we *compare* them from right to left.

Test Cases for Directionality

- Prosodic groupings
  - Syllabification [CV.CVC.CV] vs. [CVC.CV.CV]
  - Footing

With binary footing, σσσσσ must have lapse



Footing is similar. We've got to group 5 syllables into pairs. One will be left over.

And again, languages seem to resolve this directionally.

A language can push the stray syllable to the right of the form - postpone violations of "Parse-syllable."

Another language can push it leftward.

What languages don't do is this.

Test Cases for Directionality

- Prosodic groupings
  - Syllabification [CV.CVC.CV] vs. [CVC.CV.CV]
  - Footing σ(σσ)(σσ) vs. (σσ)(σσ)σ
- Floating material
  - Lexical:
    - Tone docking ban.tó.di.bo vs. ban.to.di.bó
    - Infixation grumadwet vs. gradwumet
  - Stress "end rule" (bán.to)(di.bo) vs. (ban.to)(dí.bo)
    - OnlyStressFootHead, HaveStress » NoStress (L-R)
- Harmony and OCP effects

Prosodic groupings force location tradeoffs, as I said before.

Floating material is another source of location tradeoffs.

The floating material's got to dock somewhere, but we have a choice about where.

For example, floating tones. In this language, the high tone can show up on any non-low vowel: ton, di, bo.

But it's marked. That is, there's a constraint against high tone, which is why we don't get extra tones showing up besides this required one.

If that markedness constraint is evaluated right to left, it pushes the tone as early as possible; if left-to-right, it pushes that marked material to the end of the form.

And that *is* how tonal languages work.

Similarly, infixed material falls as early or as late as possible in the word, given higher-ranked constraints on syllable structure.

Primary stress: same thing. It falls on the first or last foot.

Let me be concrete. High-ranked constraints say we must have at least one primary stress, and tell us where stress is allowed to fall.

But we also need a lower-ranked constraint against primary stresses.

This not only eliminates extra stresses, so that we only get one, but if it's directional, it pushes that one toward the end of the form: ban.to.DI.bo

Finally - Harmony must say how far to spread a feature. OCP must say which copies of a feature to eliminate. So far as I know, these always behave directionally, too.

Generalized Alignment

- Phonology has directional phenomena
  - [CV.CVC.CV] vs. [CVC.CV.CV] - both have 1 coda, 1 V
- Directional constraints work fine
- But isn't Generalized Alignment fine too?
  - Ugly
    - Non-local; uses addition
  - Not well formalized
    - Measure "distance" to "the" target "edge"
  - Way too powerful
    - Can center tone on word, which is not possible using any system of finite-state constraints (Eisner 1997)

So all I've said is something you probably knew: phonology has directional phenomena.

And just to drive that home: Both these forms look equally bad - they both have one coda and one inserted vowel.

Arabic chooses between them based on the position of the coda, or the inserted vowel.

And I've argued that these new directional constraints capture these phenomena very naturally.

But people have been handling the same facts in OT for 8 years without directional constraints.

Very early in OT, Robert Kirchner remarked that they *could* be handled by constraints that count, or count in a weird way.

That's called Generalized Alignment and everyone uses it to describe these effects.

Very briefly, here's my diatribe about why that's a bad solution.

It's a hack that does things that are anathema to phonology

Various ways in which it's not well formalized

And most important, it's way too powerful. You can write a GA constraint that centers a floating tone on a word.

This drives up the computational power of the theory, and all for a phenomenon that's unattested. No language centers tones!

Outline

- Review of Optimality Theory
- The new "directional constraints" idea
- Linguistically: Fits the facts better
- → Computationally: Removes excess power

- Formal stuff
  - The proposal
  - Compilation into finite-state transducers
  - Expressive power of directional constraints

So on that computational note …

Computational Motivation

- Directionality not just a substitute for GA
- Also a substitute for counting
- Frank & Satta 1998:

OTFS > FST

(Finite-state OT is *more powerful* than finite-state transduction)

I just introduced directional constraints to take over the functions of Generalized Alignment.

But thinking GA is ugly isn't the only reason to like directional constraints …

I'll define in a little while what finite-state OT grammars are

Why OTFS > FST?

- It matters that OT can count
- **HeightHarmony** » **HeightFaithfulness**
- Input: to.tu.to.to.tu
- Output: to.to.to.to.to
  vs. tu.tu.tu.tu.tu

can both be implemented by weighted FSAs

prefer candidate with fewer faithfulness violations

- Majority assimilation (Baković 1999, Lombardi 1999)
- Beyond FST power - fortunately, unattested

to.to.to.to.to - I guess that's a foot
versus Archbishop tu.tu.tu.tu.tu

Has been described as majority assimilation - fewer u's than o's, so the o's have it.
Can't do that kind of counting with a transducer,
and fortunately in real phonology you shouldn't have to, since this is unattested.

Why Is OT > FST a Problem?

- Consensus: Phonology = regular relation
  - OT supposed to offer elegance, not power
- FSTs have many benefits!
  - Generation in linear time (with no grammar constant)
  - Comprehension likewise (cf. no known OTFS algorithm)
    - Invert the FST
    - Apply in parallel to weighted speech lattice
    - Intersect with lexicon
  - Compute difference between 2 grammars

Making OT=FST: Proposals

- Approximate by bounded constraints
  - Frank & Satta 1998, Karttunen 1998
  - Allow only up to 10 violations of **NoCoda**
  - Yields huge FSTs - cost of missing the generalization
- Another approximation
  - Gerdemann & van Noord 2000
  - Exact if location tradeoffs are between close locations
- Allow directional and/or bounded constraints only
  - Directional **NoCoda** correctly disprefers *all* codas
  - Handle location tradeoffs by ranking locations
  - Treats counting as a bug, not a feature to approximate

The first proposed solution was to bound the number of violations.

Finite-state machines can do bounded counting, so let's write a version of NoCoda that only counts to 10.

A candidate with 11 or 12 codas is no worse on this constraint.

If you set this bound high enough, you're fine.

But NoCoda needs a lot of states so it can count to 10. It's a big automaton. You combine it with big automata for all the other constraints, all furiously counting at once, and you get a huge transducer. It's huge because it misses the simple generalization that all codas are bad.

Another approximation, very cute, is the subject of an invited talk at Sunday's workshop.

It does allow unbounded violations

They bound something else instead.

Remember I mentioned location tradeoffs.

So long as location tradeoffs are between locations that are boundedly far apart, this approximation is exact.

But again, the bigger the bound you need, the bigger the FST.

My proposal is an extension to using bounded constraints. I'll show you can also throw in directional constraints without penalty.

For directional constraints, unlike bounded ones, adding violations always makes a candidate worse

Outline

- Review of Optimality Theory
- The new "directional constraints" idea
- Linguistically: Fits the facts better
- Computationally: Removes excess power

- Formal stuff
  - The proposal
  - Compilation into finite-state transducers
  - Expressive power of directional constraints

So let's redefine it!

Tuples

- Violation levels aren't integers like ★★★
- They're integer *tuples*, ordered lexicographically

		NoCoda			
		σ1	σ2	σ3	σ4
	ban.ton.di.bo	1	1	0	0
☞	ban.to.dim.bon	1	0	1	1
	ban.ton.dim.bon	1	1	1	1

Remember this example from earlier. This is a directional constraint,

so violation levels aren't integers. (Those stars are OT notation for 3. You know why? It's so we don't look like we're counting.)

For a directional constraint, a violation level is a tuple of integers. Let's change the notation.

You can see that the winning candidate has the smallest 4-tuple, under a lexicographic ordering.

Tuples

- Violation levels aren't integers like ★★★
- They're integer *tuples*, ordered lexicographically
- But what about candidates with 5 syllables?
  - And syllables aren't fine-grained enough in general

		NoCoda			
		σ1	σ2	σ3	σ4
	ban.ton.di.bo	1	1	0	0
☞	ban.to.dim.bon	1	0	1	1
	ban.ton.dim.bon	1	1	1	1

But why 4-tuples? How would a candidate with 5 syllables fit into this tableau? And syllables might work for NoCoda, but in general we need something more fine-grained.

Alignment to Input

- Split by input symbols, not syllables
- Tuple length = input string length + 1

Input:		b	a	n	t	o	d	i	b	o
Output:		b	a	n	t	o n	d	i	b	o
	0	0	0	1	0	1	0	0	0	0

For this input (length 9),
NoCoda assigns each output candidate a 10-tuple
Possible because output is aligned with the input
So each output violation associated with an input position

So really, we partition the violations not by syllable but by location in the input. The length of the tuple, for any given tableau, will be tied to the input length.

Alignment to Input

- Split by input symbols, not syllables
- Tuple length = input length + 1, for all outputs

Input:		b	a	n	t	o	d	i	b	o
Output:		b	a	n	t	o n	d	i	b	o
	0	0	0	1	0	1	0	0	0	0
Output:		b	a	n	t	o	d	i m	b	o n
	0	0	0	1	0	0	0	1	0	1

Here's another candidate. It's longer, but its performance is still described by a 10-tuple.

Alignment to Input

- Split by input symbols, not syllables
- Tuple length = input length + 1, for all outputs

Input:		b	a	n	t	o		d	i		b	o
Output:		b	a	n	t	o	n	d	i		b	o
	0	0	0	1	0	1		0	0		0	0
Output:		b	a	n	t	o		d	i	m	b	o	n
	0	0	0	1	0	0		0	1		0	1	
Output:	i	b	a	n	t	o	n	d	i	m	t	i	m	b	o	n	n	n
	0	0	0	1	0	0	1	0	2					0	3			

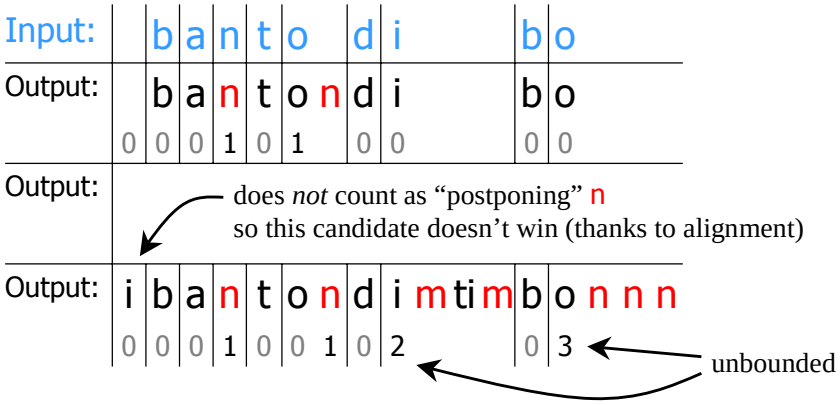
Likewise this candidate, i.ban.ton.dim.tim.bonnnn, which has 5 syllables and a lot of other junk besides.

This last candidate should be really bad - look at all that red!

But I want to use it to make a couple of points.

Alignment to Input

- Split by input symbols, not syllables
- Tuple length = input length + 1, for all outputs



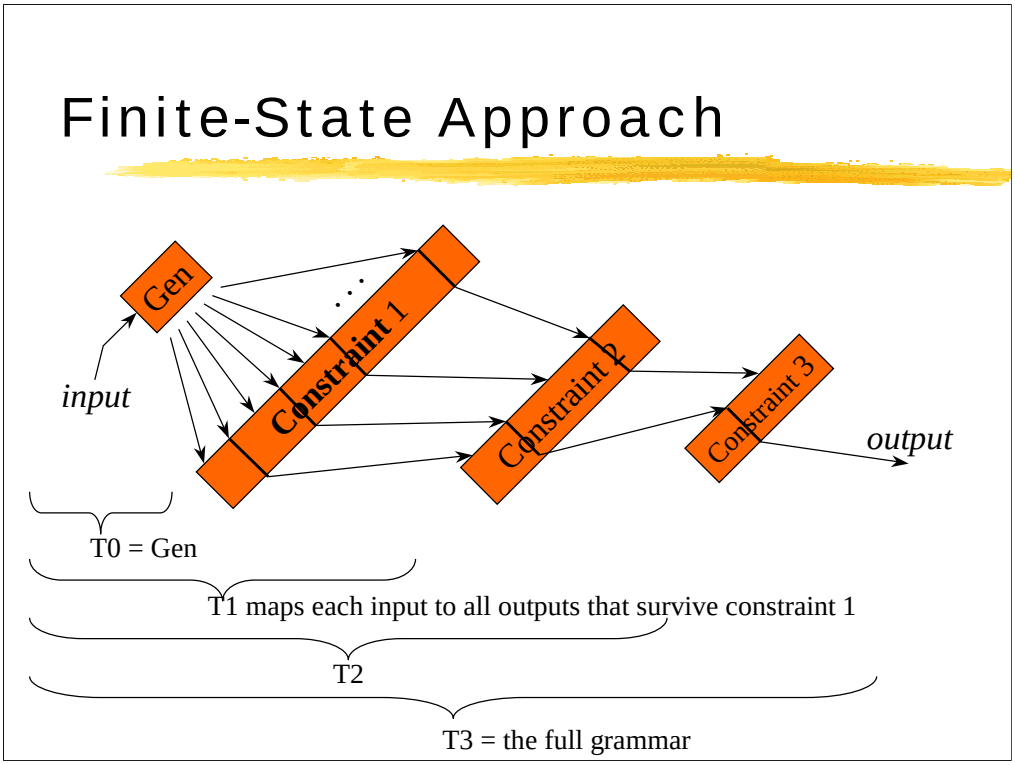
We inserted material at the start.

doesn't this postpone the first violation, the red "n"? It's certainly postponed on the surface, relative to the candidate above.

It doesn't show up till character 4. So under directional evaluation, which likes to postpone material, should this terrible candidate win?

The answer, thank God, is no. Both these candidates are aligned with the underlying form and hence with each other. In both cases, the violation, the red n, falls in bin 4. That's why alignment is important.

unbounded - not the case that once you have one coda in here you might as well take 2 or 3 or 17 - each additional coda causes additional pain.



Ok, that's the alignment idea.

But where does the alignment come from?

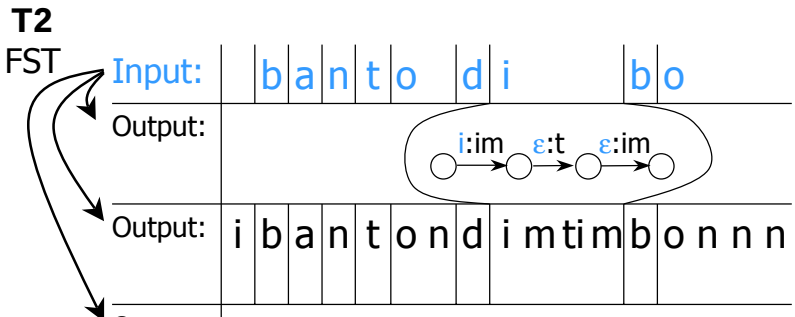
And where do those violation numbers come from?

Well, from a grammar. So I have to tell you how to write a grammar in terms of finite-state devices.

So to compile a grammar into a transducer, we'll build these up in sequence.

Finite-State Approach

- FST maps each input to set of outputs
  (nondeterministic mapping)
- The transducer gives an alignment

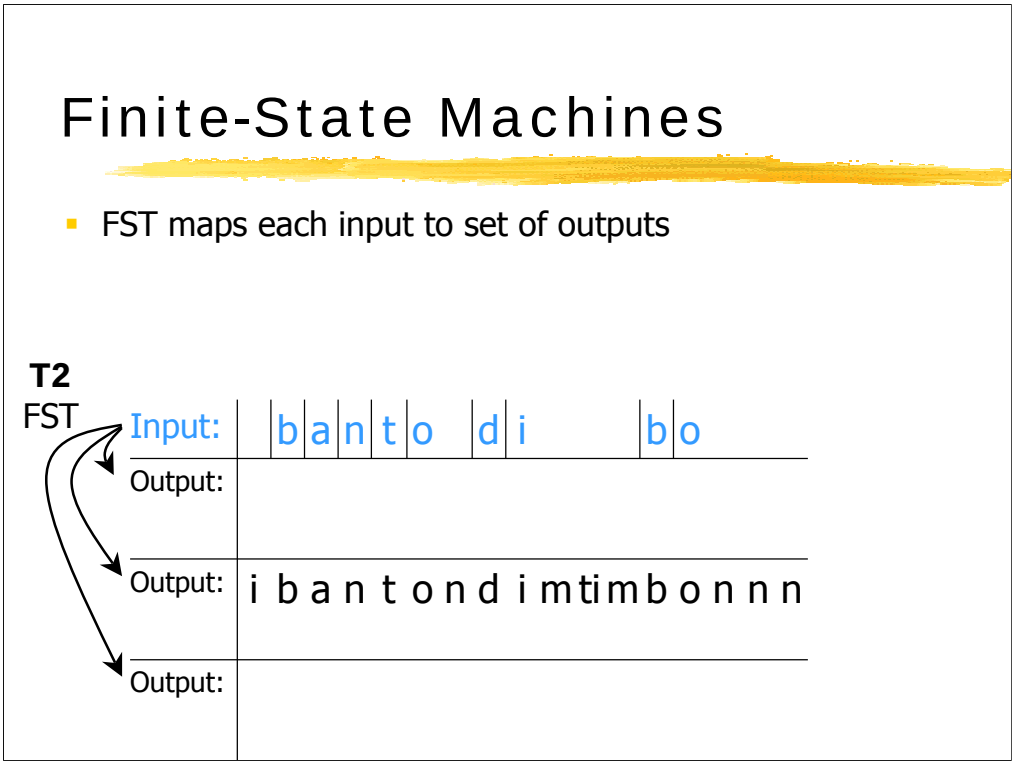


So T2, for example, is a transducer that maps each input to several outputs.

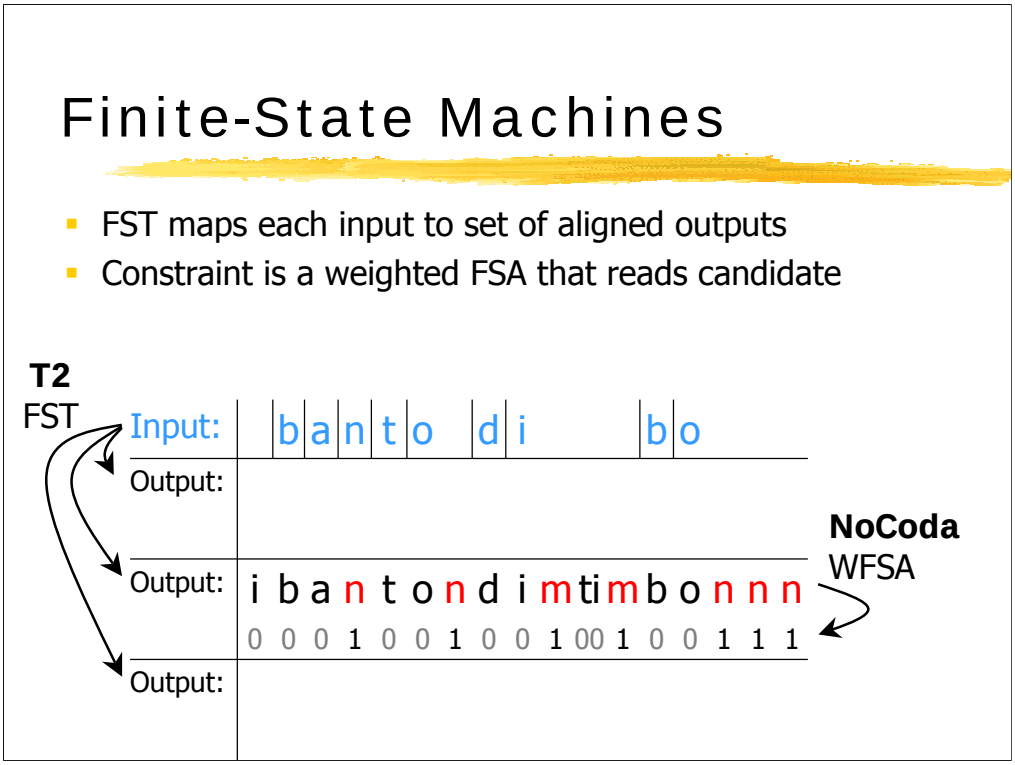
Important point: This gives an alignment!

the transducer reads “i”, and “imtim” is what it spits out before it reads the next input symbol “b”

That might all happen in one arc, or there might be a bunch of arcs, the first one reading i and the rest reading epsilon, and together they spit out imtim.

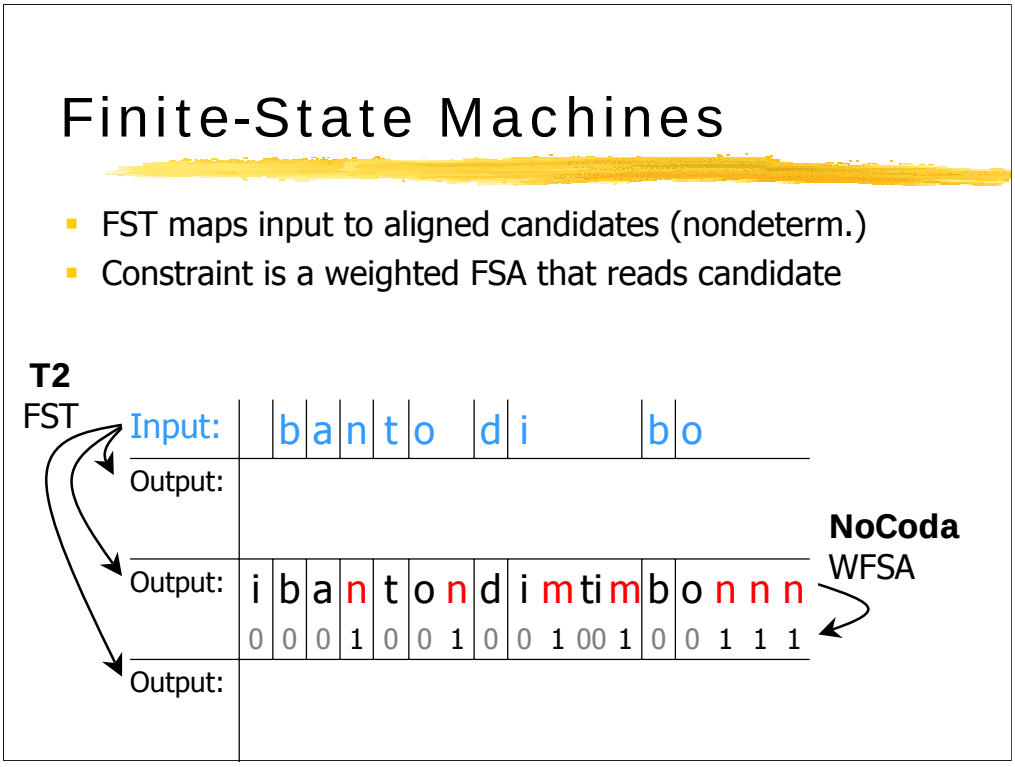


But let's ignore the alignment for now, because our constraints don't care about it.

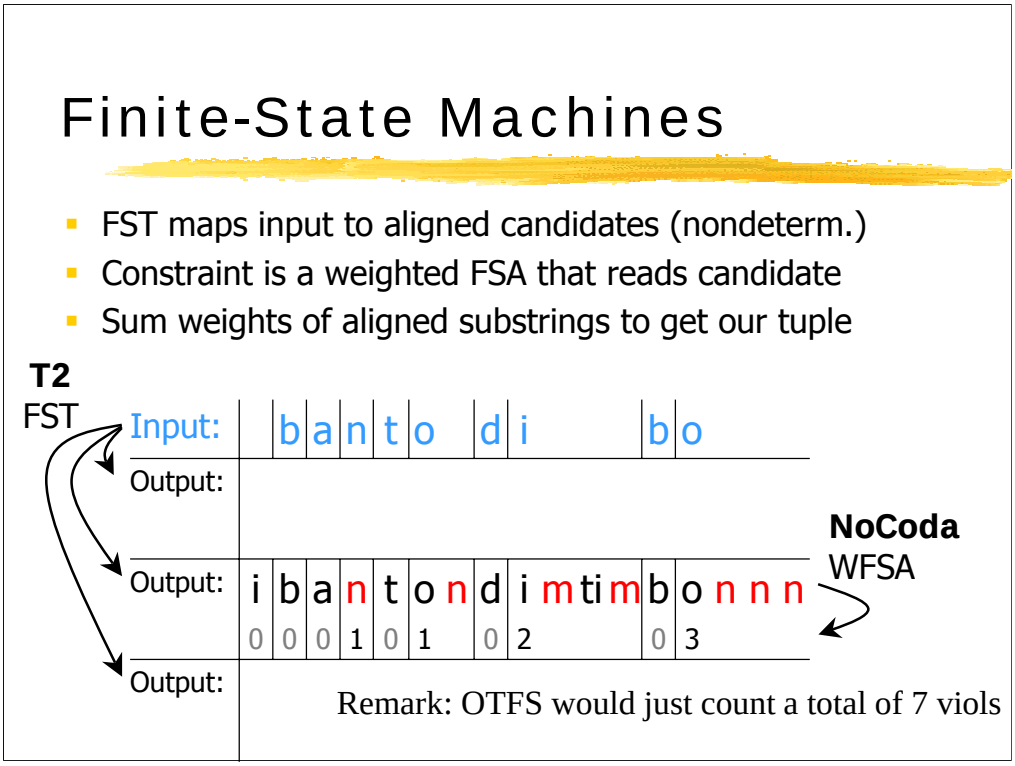


The constraint NoCoda is a weighted FSA (could be nondeterministic).
It's designed so that whenever it reads a coda, it does so on an arc of weight 1.
So it scans the candidate, assigning 1 to codas and 0 to everything else.

The automaton can be nondeterministic, so it could simulate lookahead.



Now let's remember the alignment, which divides the candidate into 10 substrings. If we add up the cost of each substring …



… we get our 10-tuple! For example, imtim has 2 codas, so the underlying position "i" has cost 2.

I should point out that standard FS constraints are *exactly the same* except that you sum over the whole string.

So now you know the finite-state formalism of directional constraints.

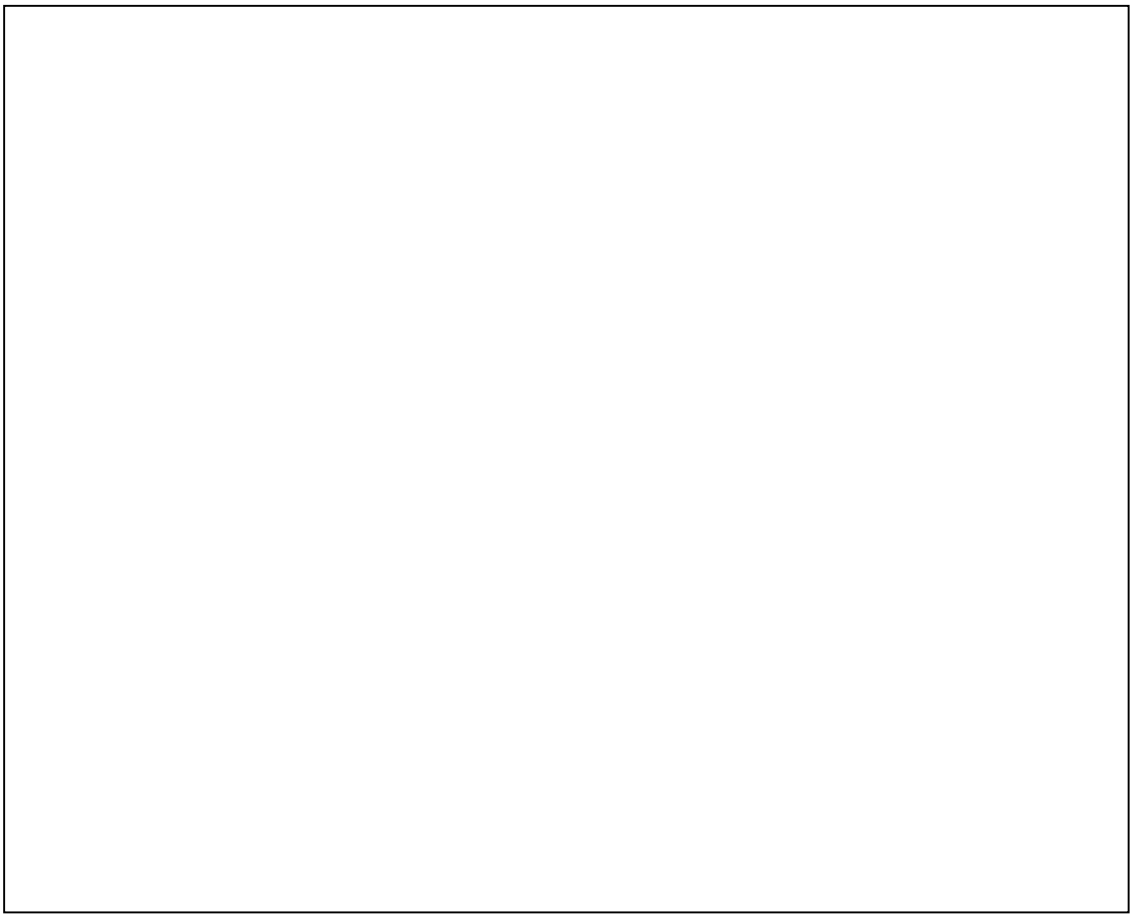
Similar Work

- Bounded Local Optimization
  - Walther 1998, 1999 (for DP)
  - Trommer 1998, 1999 (for OT)
  - An independent proposal
    - Motivated by directional syllabification
  - Greedy pruning of a candidate-set FSA
    - Violations with different prefixes are incomparable
    - No alignment, so insertion can postpone violations
    - No ability to handle multiple inputs at once (FST)

Outline

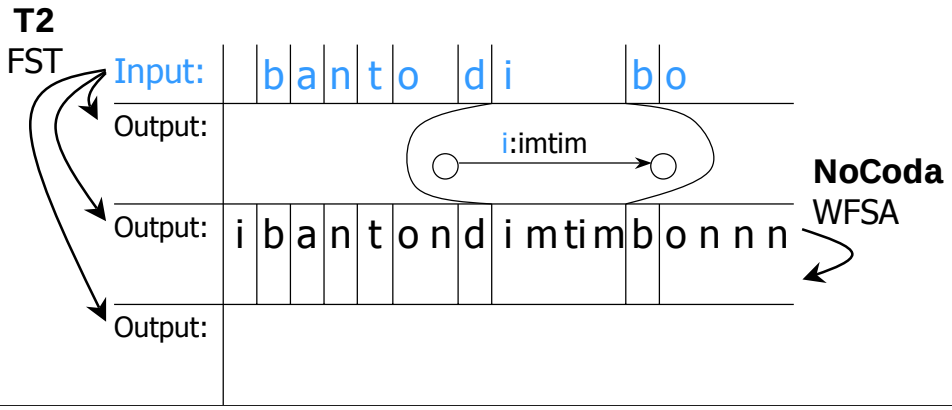
- Review of Optimality Theory
- The new "directional constraints" idea
- Linguistically: Fits the facts better
- Computationally: Removes excess power

- Formal stuff
  - The proposal
  - Compilation into finite-state transducers
  - Expressive power of directional constraints



The Construction

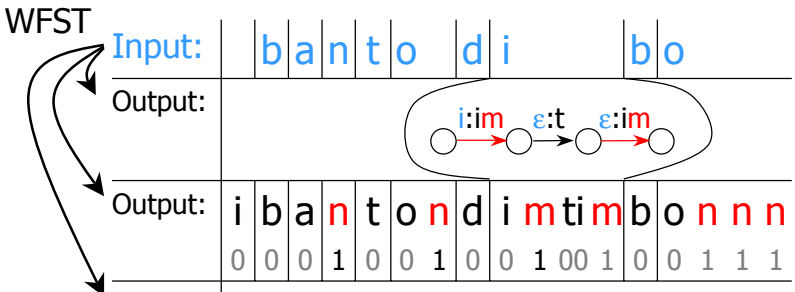
- Our job is to construct T3 - a "filtered" version of T2
  - First compose T2 with **NoCoda** …



Now we know the tuples are defined.

The Construction

- Our job is to construct T3 - a "filtered" version of T2
  - First compose T2 with **NoCoda** to get a weighted FST



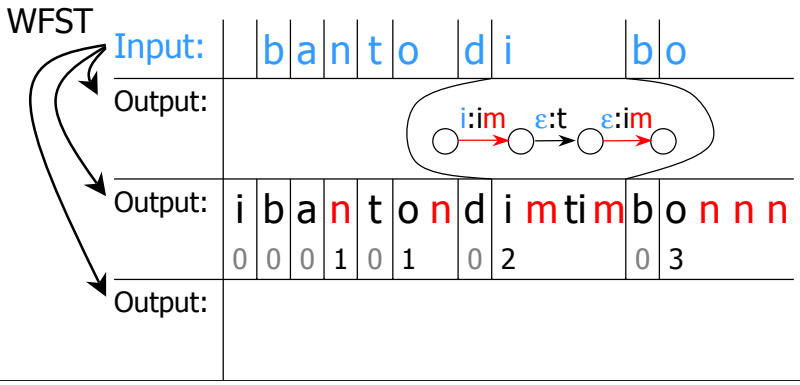
We compose T2 with the constraint, nocoda.

That pushes the weights onto the transducer, so now we

are transducing i to imtim on a subpath of weight 2. See the 2 red arcs.

The Construction

- Our job is to construct T3 - a "filtered" version of T2
  - First compose T2 with **NoCoda** to get a weighted FST
  - Now prune this weighted FST to obtain T3
  - Keep only the paths with minimal tuples: Directional Best Paths

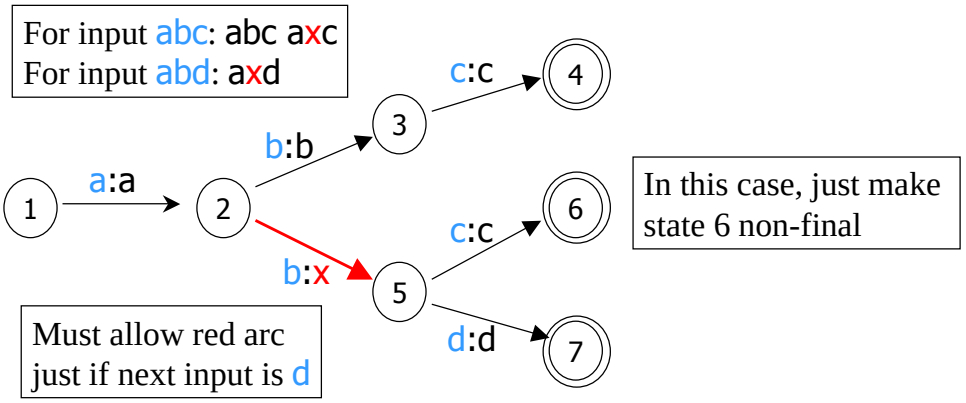


Now to get T3, we have to prune away the suboptimal paths for each input

In general, we'll have to split states to do this, because …

Directional Best Paths (sketch)

- Handle all inputs simultaneously!
- Must keep best outputs for each input: at least 1.



… because we need to handle all inputs simultaneously.

We have to keep at least one output for input, but the best outputs may have violations in various places; it depends on the input.

For example, this transducer maps abc to 2 candidates, abc and axc. Red is bad, so we prefer abc.

But it only maps abd to one candidate, axd. So for input abd we must accept a violation.

Suppose we try to be greedy.

We avoid the red arc. Avoid the violation. Ok - but now what? Next input symbol is d. Uh oh. There's no way to finish reading the input from here, that is, to get to a final state.

So on input abd, we can't afford to be greedy. We have to take the bad red arc and suffer a violation.

In fact that's obvious - all the abd candidates have a violation there.

But when we're at state 2, making the choice to take the red arc or not, we don't know yet whether the input will be abc or abd. We've only seen ab.

We need some kind of lookahead.

Specifically, we'll nondeterministically take both arcs from state 2, but if we get to state 6 on input abc, we'd better know that this was not the best path for this input. So it shouldn't be an accepting path. State 6 shouldn't be a final state because if we

Directional Best Paths (sketch)

- Must pursue counterfactuals
- Recall determinization (2^n states)
  - DFA simulates a parallel traverser of the NDFA
  - "What states could I be in, given input so far?"
- Simulate a neurotic traverser of the WFST
  - "If I had taken a cheaper (greedier) path on the input so far, what states could I be in right now?"
  - Shouldn't proceed to state *q* if there was a cheaper path to *q* on same input
  - Shouldn't terminate in state *q* if there was a cheaper terminating path (perhaps to state *r*) on same input
  - 3^n states: track statesets for equal and cheaper paths

I'll sketch the construction in general. It's cute.

It's sort of like the standard subset construction for determinization.

You're given an NDFA (that is, nondeterm), and you simulate it with a DFA (determ), whose states reflect the thought processes of someone trying to pursue all options in parallel.

The DFA reads the input, always asking: what states could I be in now? [And there are only 2^n possible answers, so we stay finite-state. It pursues all hypothetical paths at once].

[Can FSTs be determinized? Use odd-even construction to show they can't be, and show why it's folly to think we can merge states. Anyway, we're not trying to determinize but do best-paths.]

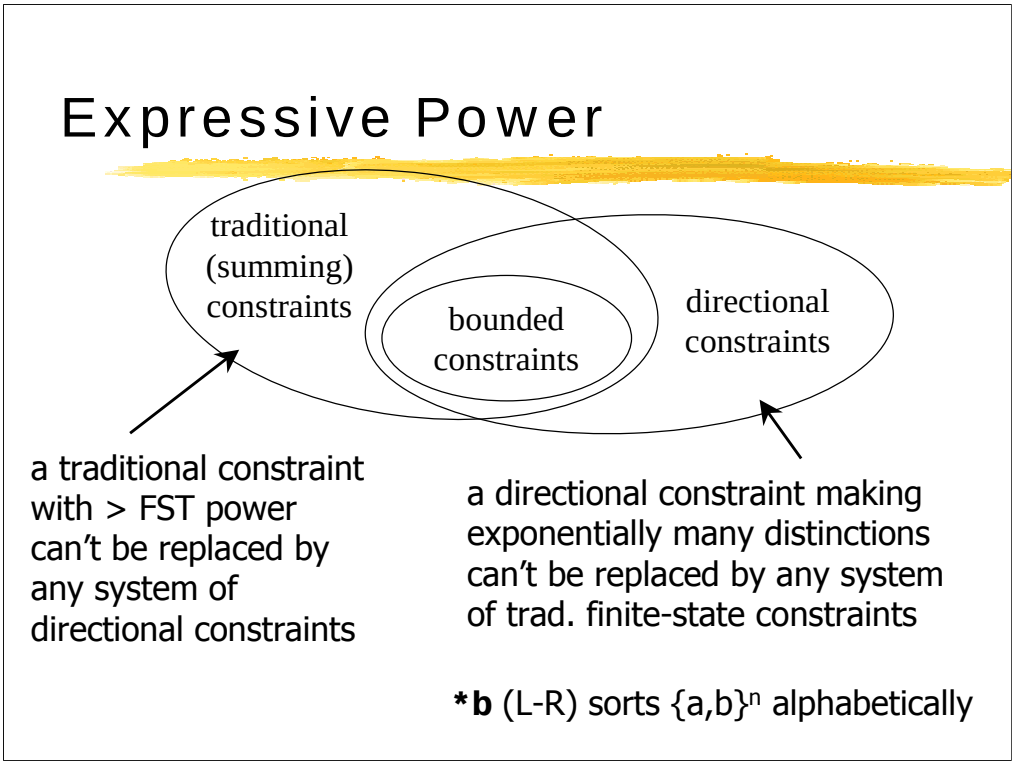
For best-paths we'll do something similar, but in a weighted version that happens to do transduction. We'll build a machine that's also nondeterministic, but unweighted, and traverses the weighted FST neurotically. What does that mean? It's really the same as the original, but whatever path it takes, it keeps fantasizing about the arcs it didn't take. So we'll live our life, but as we continue to read the input along some path, we keep tracking in our state what would have happened if we'd taken a cheaper path. A path is cheaper than our current one, in directional left-to-right terms, if it made even *one* good greedy decision earlier on. What would have happened to me if I'd run away and joined the circus? What would have happened if I'd dropped out of college? That would have been more fun! My life is only worth it if those alternate lives meet with untimely ends on this input. Suppose, as we lead our parallel lives, getting the same input, my circus self – who

Outline

- Review of Optimality Theory
- The new "directional constraints" idea
- Linguistically: Fits the facts better
- Computationally: Removes excess power

- Formal stuff
  - The proposal
  - Compilation into finite-state transducers
  - Expressive power of directional constraints

So on that note …



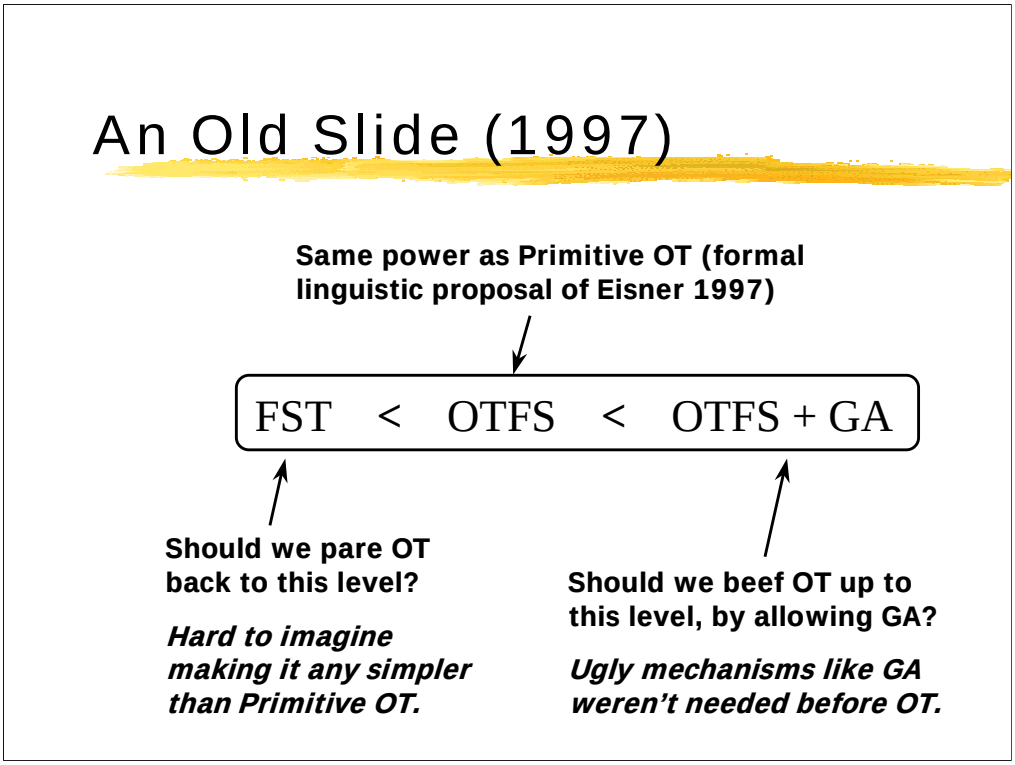
I'll just flash this up quickly.

The most important thing is that directional constraints are a strict extension to bounded constraints -

they can do things that bounded constraints or even traditional constraints can't - but they keep us within transducer power.

Future Work

- Further empirical support?
- Examples where 1 early violation trades against 2 late violations of the same constraint?
- How do directional constraints change the style of analysis?
- How to formulate constraint families? (They must specify precisely where violations fall.)



So let me summarize by saying where this work came from.

In 1997 I made a proposal that almost all linguistic work in OT could be fit into a very simple formalism, primitive OT.

Here's a slide from ACL '97.

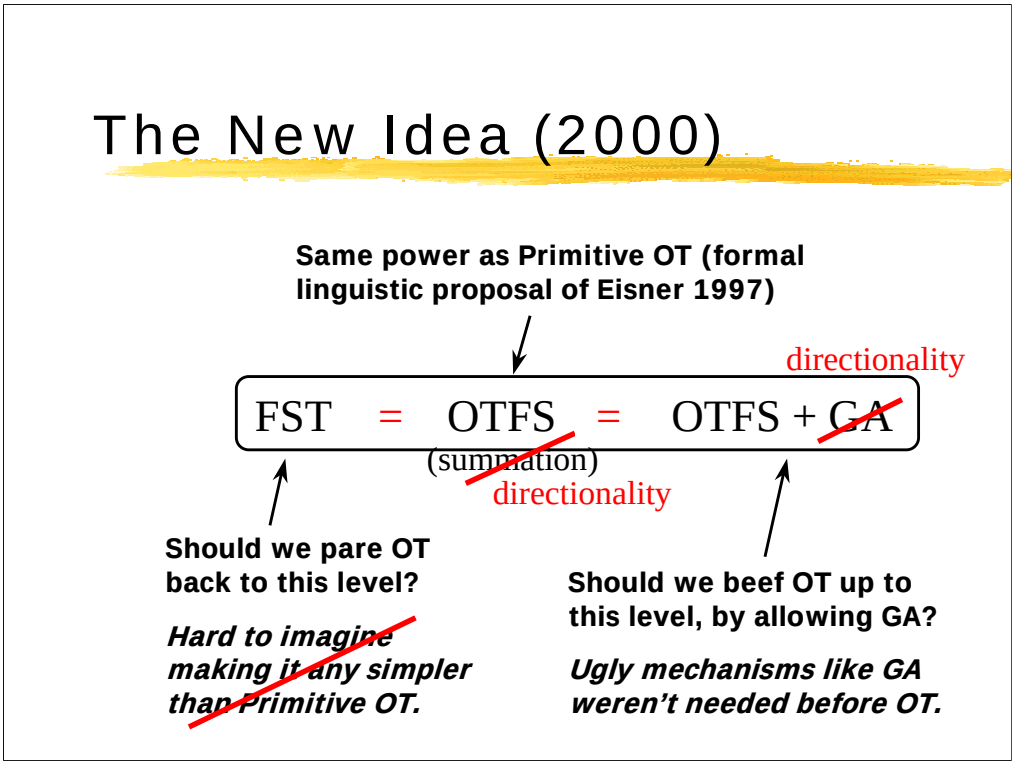
Just in terms of generative power [If we ignore questions of what's easy and hard to express in the formalism], it's equivalent to OTFS.

Too strong or too weak?

Frank and Satta had just shown it was too strong, worse than FSTs.

But my theory was already quite minimal; it did everything with just two kinds of constraints. And those were so simple and well attested that I didn't see how to pare it back.

On the other hand, was it strong enough? It didn't have GA. Mark Ellison, who first proposed finite-state OT, had argued that maybe we could get by without GA. So I took that line and did some interesting analyses without GA or directionality.



I think the jury's out on whether those analyses were right. But we certainly don't need all of GA. At most we need directionality.

And we don't seem to need summation, which was the problem with OTFS. All we need is directionality.

So now these three things are equal. And in particular, I was wrong about primitive OT. There *is* at least one way to make it simpler.