Class 6: Optimality Theory, part I

To do
- Finish Prince & Smolensky excerpt (SQs due Thursday)
- Start working on beginning OT problem (due Tuesday)

1. The "conceptual crisis" (Prince & Smolensky p. 1)

Since Kisseberth 1970, constraints were taking on a bigger and bigger role. But…

- What happens when there's more than one way to satisfy a constraint? We need to prioritize the rules that could be triggered.
- Why aren't constraints always obeyed?
- Relatedly, what happens when constraints conflict? What if one constraint wants to trigger a rule, but another wants to block it? We need a way of prioritizing constraints.
- Should a rule be allowed to look far ahead in the derivation to see if applying alleviates a constraint violation? Or does the alleviation have to be immediate?
- Can a constraint be against making a certain type of change, rather than against a certain structure?

2. Prince & Smolensky's solution: Optimality Theory

rule-based grammar with constraints	*OT grammar*
start with UR/input (from mental lexicon)	
apply rules in sequence—intermediate representation is known at all times	apply all possible rules, producing a (large!) set of *candidate outputs*
constraints may block or trigger rules	constraints pick the best candidate
look-ahead is nonexistent or sketchy	since the candidate outputs are all potential surface forms, there is full look-ahead to the end of each possible derivation
interaction of constraints is nonexistent or sketchy	constraints interact through *strict domination*
similarity to UR is the result of not applying too many rules and not having too many constraints	similarity to UR is enforced by *faithfulness* constraints
end with SR/output (send it to the phonetic system)	

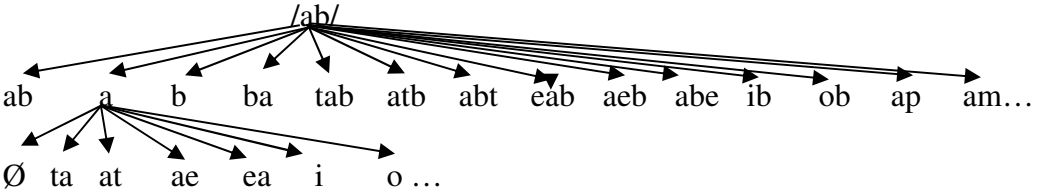
3. Gen()

This is the function that creates the set of candidate outputs from the input.

One way to think of it:[1] apply all possible rules to the input, any number of times. Each underlying segment can be deleted or have its features changed; extra segments can be inserted anywhere; underlying segments can change their order

[1] This is what P&S call 'anharmonic serialism,' except with a universal set of rules that's broad enough that the result is the "all possible variants" that P&S propose.

Gen(/ab/) = {[ab], [a], [b], [ba], [], [ta], [at], [ae], …}



- Why is the resulting set of candidates infinite (assuming a finite alphabet of symbols)?

4. Constraints

In standard OT, we can think of each constraint as a function from a candidate output to a natural number (the number of violations).

NoCoda([bak]) = 1
NoCoda([tik.pad]) = 2

Alternatively, we can think of each constraint C_i as imposing a strict partial ordering $\succ_i$ ("is more harmonic than with respect to C_i") on a set of candidates, with the following additional properties:

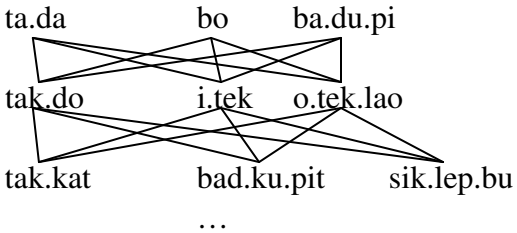
- The ordering is *stratified*: If $a \nsucc b$ and $b \nsucc a$, then for any i $\succ a$, $i \succ b$ too; and for any j such that $a \succ j$, $b \succ j$ too. (We can say that if $a \nsucc b$ and $b \nsucc a$, a and b are of equivalent harmony.)
- There exists some a such that there is no $i \succ a$. (That is, one or more candidates are the most harmonic; there are not necessarily one or more least harmonic candidates, though.)

A strict partial ordering is *transitive*, *irreflexive*, and *asymmetric*:

- Transitive: if $a \succ b$ and $b \succ c$, then $a \succ c$.
- Irreflexive: $a \nsucc a$.
- Asymmetric: If $a \succ b$, then $b \nsucc a$.

(In Colin Wilson's targeted-constraints variant of OT, the stratification requirement is relaxed.)

NoCoda:



- Why does assigning a non-unique natural number (0, 1, 2, …) to each candidate meet the ordering requirements above?
- If you're ahead in the reading, can you recall a case from P&S where numbers of violations weren't used?
- Why are there no least-harmonic candidates for NoCoda?

5. Eval()

Eval() is a function from a set of output candidates and an ordered list (*Con*) of constraints to the subset of the candidates that is optimal (typically, that subset contains just one candidate).

Eval(Gen(/input/),Con) = {[output]} (or, e.g., {[out$_1$], [out$_2$]} in the case of a two-way tie)

Eval() does this by taking the orderings imposed by the various constraints and assembling them into one giant ordering. We can think of many ways this could be done…*strict ranking* is the mechanism used in standard OT for adjudicating harmony disagreements among constraints.

6. Constraint interaction through strict ranking

Strict ranking is like alphabetization: to find the alphabetically earliest member of a list, pick out all the members of the list that have the earliest first letter—and throw the rest of the list away; from the new, smaller list, pick the members that have the earliest second letter, etc. Once a word is out of the running, it can't redeem itself by, e.g., having lots of *a*s later on:

axiom	axiate	tab	axicle	caba
banana	azalea	axolotl	zabaglione	baa

Eval works the same way. If you have *n* constraints…

- Pick out the candidates that tie for being 'best' on the top-ranked constraint C_0; discard the rest of the candidates.
- Move to the next constraint, C_1, and do the same—pick out the candidate(s) that are best with respect to the set that is left.
- Repeat for C_2,…,C_n.
- Whatever candidates are still left at the end are tied for being the winner (if you have enough constraints, there is normally just one winner).

Q: How can that be computable? Wouldn't you have to go through an infinite list of candidates just to do the first step?

A: For that reason, most computational implementations of OT (Ellison, Eisner, Albro, Riggle) represent the candidate set as a regular expression, which is a finite way to represent a certain class of infinite sets. For example, *ab*a* is the set {*aa*, *aba*, *abba*, *abbba*, *abbbba*, …}. These expressions can then be manipulated algorithmically, either in a fairly literal translation of the above or by other means.

More declaratively, a candidate *a* is optimal iff, for any *b* and C_j such that $b \succ_j a$, there exists some C_i such that $i < j$ (i.e., C_i is higher ranked than C_j) and $a \succ_i b$.
In words, for *a* to be optimal, any candidate that does better than *a* on some constraint must do worse than *a* on some higher-ranked constraint.

- Can you imagine some other ways that constraints could conceivably interact?

7. Two types of constraint

In pre-OT approaches to constraints, constraints were all *markedness* constraints: they penalized certain surface structures, such as CCC clusters.

So, on first hearing about OT, many people's second reaction (the first was worrying about infinity) was to wonder why, if it's all about constraints, every word isn't maximally unmarked.

- In rule+constraint theories, what prevents every word from coming out [baba] (or whatever the least marked word is)?
- For those who know OT or have read ahead, how do P&S prevent every word from coming out [baba]?

Markedness constraints are constraints on the *output* (they could be thought of as requiring articulatory ease, or perceptual clarity, or some other "natural" type of unmarkedness[2]). The simplest ones can be defined by the structural description that they ban: *[+voice]#, *$C]_{\sigma}$.

You can (and should!) give a constraint a helpful mnemonic name, like NOCODA for *$C]_{\sigma}$, as long as you clearly define the constraint somewhere. A good constraint definition should make it clear not just what is banned, but **how the number of violations is assessed**.

Faithfulness constraints are constraints on the *relationship* between the input and the output (the standard ones require similarity but we can imagine other possibilities).

P&S use PARSE (roughly, don't delete) and FILL (roughly, don't insert), but this has been superseded by McCarthy & Prince's correspondence approach, which you'll learn about in more detail in 201:

> MAX-X: don't delete X (e.g., MAX-C, MAX-V)
> DEP-X: don't insert X (e.g., DEP-C, DEP-V)
> IDENT-F: don't change a segment's value for the feature F

People often have a hard time at first with IDENT-F. The most common confusion is thinking it means "don't delete a segment that is +F". The next most common is thinking it means "don't alter a segment that is +F (e.g., by changing its values for some other feature G)".

8. Exposition: the tableau

In a utopic future (which may not be that far away), we will all check our analyses with software that evaluates the infinite candidate set. In the meantime, we illustrate our analyses for the reader with a *tableau*[3] showing a finite subset of candidates that have been chosen to demonstrate aspects of the constraint ranking. (The danger here is obvious—what if you didn't think of some important candidate?)

---

[2] Or maybe they are just arbitrary and learned by speakers in response to whatever cards history has dealt them. Or, maybe both natural and unnatural constraints are possible, but learners treat them differently. This is a matter of current debate. See Elliott Moreton's WCCFL paper (to appear—see his webpage) for a very clear overview.

[3] French for 'table'. The singular *tableau* is pronounced [tabló] in French, [t^hæblóʊ] in English. The plural *tableaux* is pronounced [tabló] in French, [t^hæblóʊ] or [t^hæblóʊz] in English.

This tableau shows a *ranking argument*: we have two candidates that differ in that NOCODA prefers *a* (the winner), whereas DEP-V prefers *b*. If that's the only difference between the candidates—there is no other constraint that prefers *a* over *b*—then NOCODA must outrank (>>) DEP-V.

/at+ka/	NOCODA	DEP-V
☞ *a* [a.tə.ka]		*
b [at.ka]	*!	

Parts of the tableau:

- input
- output candidates
- constraints (highest-ranked on left)
- asterisks
- exclamation marks
- shading
- pointing finger (or you can use an arrow)

} These three don't add any new information, but are there for the convenience of the reader.

9. How do I know which candidates and constraints to include in my tableaux?

Here is a procedure that usually works reasonably well:

- Start with the winning candidate and the fully faithful candidate.
- If the winning candidate ≠ the fully faithful candidate…
  - Add the markedness constraint(s) that rule out the fully faithful candidate.
  - Add the faithfulness constraints that the winning candidate violates.
  - Think of other ways to satisfy the markedness constraints that rule out the fully faithful candidate. Add those candidates, and the faithfulness and markedness constraints that rule them out. You have to use your judgment in deciding how far to take this step.
- If the winning candidate = the fully faithful candidate, then you are probably including this example only to show how faithfulness prevents satisfaction of a markedness constraint that, in other cases, causes deviation from the underlying form.
  - Add that markedness constraint.
  - Add one or more candidates that satisfy that markedness constraint.
  - Add the faithfulness constraints that rule out those candidates.

- Let's try it for /atka/ → [atəka].

- One of the candidates below is unnecessary in arguing for the constraint ranking. Why?

/at+ka/	*CC	DEP-V
☞ *a* [atəka]		*
b [atka]	*!	
c [atəkəa]		**!

A candidate is *harmonically bounded* if it could not win under any constraint ranking.

10. Comparative tableaux

An innovation of Alan Prince. They convey the same information, but in a different form

/at+ka/ → [atəka]		*CC	DEP-V
a	[atəka] vs. [atka]	W	L
b	[atəka] vs. [atəkəa]		W

Each line compares the winner to one losing candidate, and shows whether each constraint prefers the winner (W) or the loser (L)

Comparative tableaux are nice because you can easily see if your ranking is correct: the first non-blank cell in each row must say *W*.
We also see easily why [atəkəa] is irrelevant to the ranking.

11. Exercise: Metaphony (the two easy cases)

- Develop an OT account of these two metaphony systems.

Foggiano/Pugliese

pɛ́te	'foot'	píti	'feet'
móʃʃa	'soft (fem.)'	múʃʃu	'soft (masc.)'
kjéna	'full (fem.)'	kjínu	'full (masc.)'
grɔ́ssa	'big (fem.)'	grússu	'big (masc.)'

Veneto

védo	'I see'	te vídi	'you see'
kóro	'I run'	te kúri	'you run'
prɛ́te	'priest'	prɛ́ti	'priests'
bɛ́lo	'beautiful (masc. sg.)'	bɛ́li	'beautiful (masc. pl.)'
mɔ́do	'way'	mɔ́di	'ways'
gáto	'cat'	gáti	'cats'

12. Another exercise: English regular plurals

pi-z	'peas'	bloʊk-s	'blokes'
tʰoʊ-z	'toes'	kʰɑf-s	'coughs'
dɑl-z	'dolls'	glɑs-ɨz	'glasses'
pʰæn-z	'pans'	fɪz-ɨz	'fizzes'
dɑg-z	'dogs'	bɹæntʃ-ɨz	'branches'
læb-z	'labs'	bædʒ-ɨz	'badges'
kʰɪln-z	'kilns'	wɪʃ-ɨz	'wishes'
kʰlæsp-s	'clasps'	gəɹɑʒ-ɨz	'garages'
mɪt-s	'mitts'		