Class 6: Optimality Theory, part I

To do
- Work on Korean rules+constraints **homework** (due Friday)
- Take a look on the course webpage at the **project** instructions and examples, and start being open to a topic coming to you. We'll get serious about looking for topics next week.

0. A couple of things to discuss about the Palauan homework
- underlying schwas and stress
- C_0V_0 notation
- CC-initial stems
- the data at the end

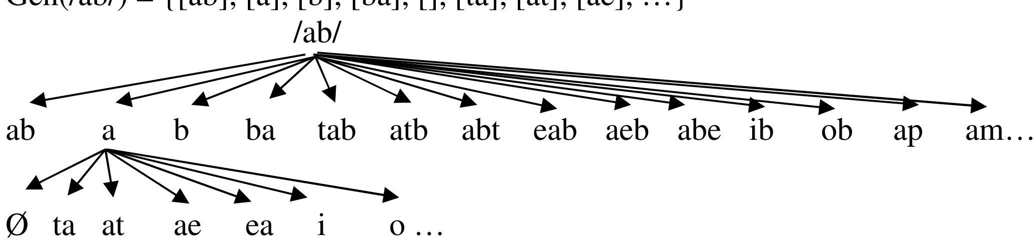
1. Recall the "conceptual crisis" (Prince & Smolensky 2004, p. 1)
- On the one hand, we want constraints in our theory
- On the other hand, we can't decide exactly how they're supposed to work.

2. Prince & Smolensky's solution: Optimality Theory

rule-based grammar with constraints	*OT grammar*
start with UR/input (from mental lexicon, maybe after morphology)	
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: nonexistent or sketchy	candidate outputs are (potential) surface forms => full **look-ahead** to end of each possible derivation
interaction of constraints: nonexistent or sketchy	constraints **interact** through *strict domination*
similarity to UR results from not applying too many rules, 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(): function that creates set of candidate outputs from input
- One way to think of it:[1] apply all possible rules to the input, any number of times (deletion, insertion, feature changing, maybe changing order).

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



[1] This is what P&S call 'anharmonic serialism,' but with a set of rules broad enough to get "all possible variants".

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

4. Constraints

- In standard OT, a markedness constraint can be a function from a candidate output to a natural number (the number of violations). A lower number means greater **harmony** (goodness):

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

- Similarly, a faithfulness constraint can be a function from input-output pair to natural number:

  DON'TDELETE(/bak/, [ba]) = 1 DON'TDELETE(/bak/, [bak]) = 0

- More generally, a constraint C_i is a function that imposes a strict partial order $\succ_i$ ("is more harmonic than with respect to C_i") on a set of candidates...
  - Transitive: if $a \succ_i b$ and $b \succ_i c$, then $a \succ_i c$.
  - Irreflexive: $a \nsucc_i a$.
  - Asymmetric: if $a \succ_i b$, then $b \nsucc_i a$
  - Show that asymmetry follows from the other two properties.

  - Show that irreflexivity follows from asymmetry.

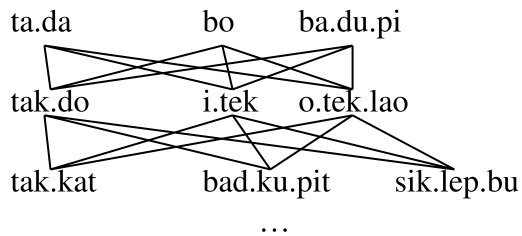
  - ...with these additional properties:
    - "Stratified":[2] if $a \nsucc_i b$ and $b \nsucc_i a$, then for any $x \succ_i a$, $x \succ_i b$ too; and for any y such that $a \succ_i y$, $b \succ_i y$ too. (In other words, if $a \nsucc b$ and $b \nsucc a$, then a and b are of equivalent harmony.)

    (In Wilson 2001, the stratification requirement is relaxed.)

[2] I don't know if there's a real math term for this. Samek-Lodovici & Prince 1999 use this term, following Tesar 1995, who uses it to describe partial orderings of constraints rather than of candidates.

- Bounded from above: There exists some *a* such that there is no $x \succ_i a$. (I.e., even in an infinite set of candidates, one or more are the most harmonic; there's not necessarily a set of *least*-harmonic candidate, though.)

NOCODA:



- Let's verify that assigning a (non-unique) natural number (0, 1, 2, …) to each candidate meets all these ordering requirements.

- Why are there no least-harmonic candidates for NOCODA?

- Can you recall a case from P&S where numbers of violations weren't used?

5. Eval()

- *Eval()* is a function
  - arguments of the function: input,[3] set of output candidates, ordered list (*Con*) of constraints
  - output of function: subset of the candidates that is optimal

- Typically we use it this way:
  - Eval(input, Gen(/input/),Con) = {[output]}
- But *Eval()* also can work on a smaller set of candidates:
  - Eval(/bak/, {[bak],[ba]}, <NoCoda, Don'tDelete>) = {[ba]}
- And, the output set can have a tie:
  - Eval(/bak/, {[bak],[ba], [bo]}, <NoCoda, Don'tDelete>) = {[ba], [bo]}

- *Eval()* takes the orderings imposed by the various constraints and assembles them into one giant ordering (with the same properties: transitive, irreflexive, asymmetric, stratified, bounded above).

- 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.

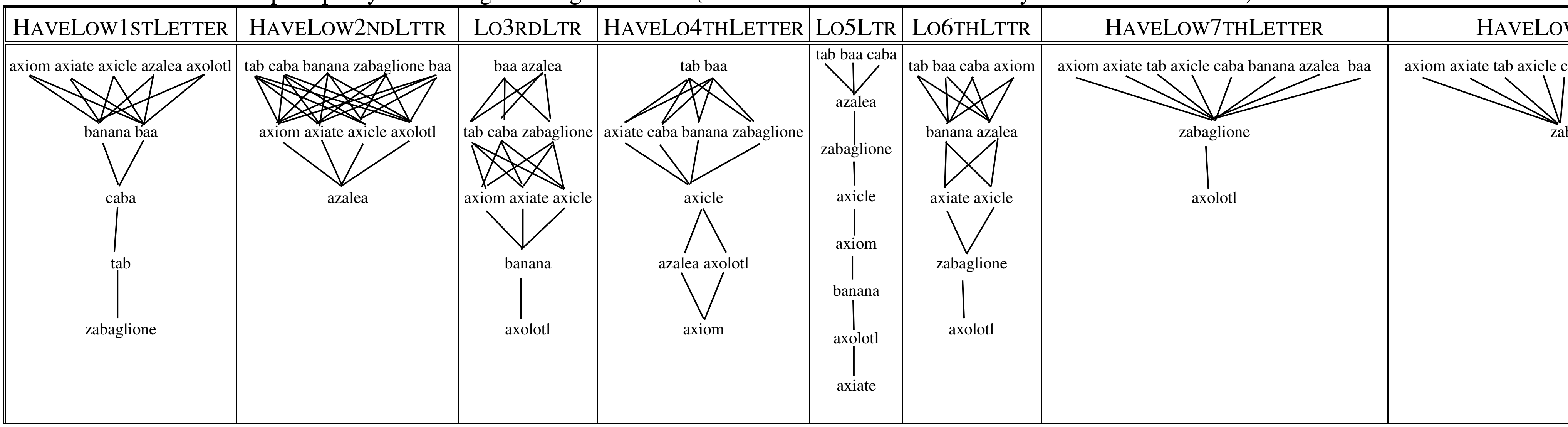
---

[3] In the original P&S manuscript, the output candidate always contains all the information about the input, so we don't need to include the input as an argument to *Eval()*.

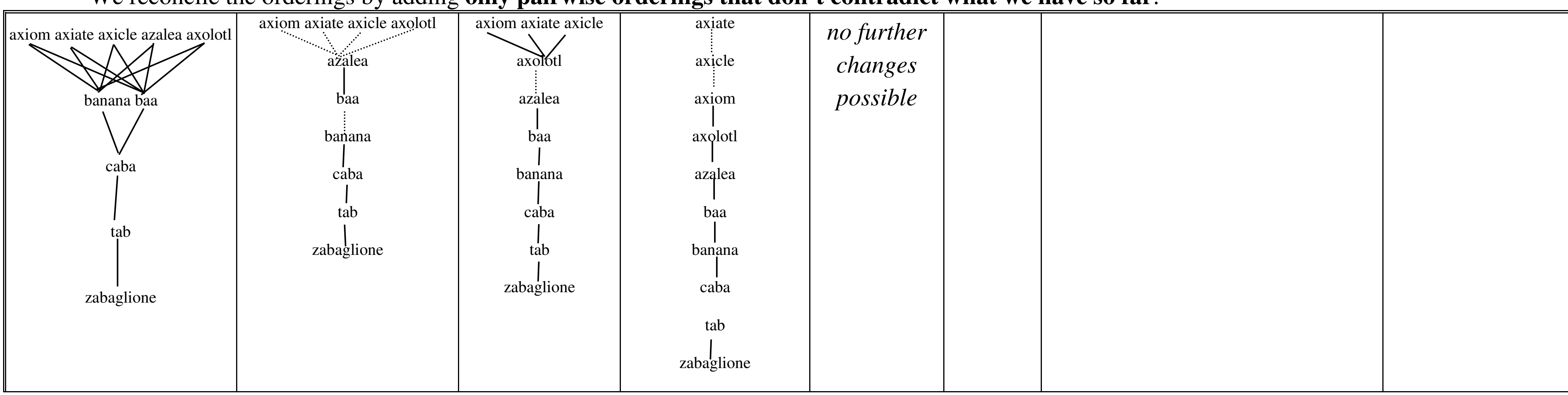
6. Alphabetization as strict ranking

axiom axiate tab axicle caba banana azalea axolotl zabaglione baa

- Constraints impose partly conflicting orderings on words (I know the last column isn't fully visible—wouldn't fit):



We reconcile the orderings by adding **only pairwise orderings that don't contradict what we have so far**:



7. How about finding just the *first* word?

- find the members that have the earliest first letter—and discard the rest
- from the new, smaller set, pick the members that have the earliest second letter, etc.

- Once a word is ruled out, it can't redeem itself by, e.g., having lots of *a*s later on.

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

8. *Eval()* works the same way

- To find just the winners, if you have *n* constraints…
  - Find the candidates that tie for being 'best' on the top-ranked constraint C_1; discard the rest.
  - Of the remaining candidates, find those the next constraint, C_2, deems best; discard the rest.
  - Repeat for $C_3,\ldots,C_n$.
  - 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 (Albro 2005, Eisner 1997, Ellison 1994, Riggle 2004) 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 (as in Eisner 1997) 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 another, higher-ranked constraint.

9. 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.

- o In rule theories, what prevents every word from coming out [baba] (or whatever the least marked word is)?

- o How do P&S prevent every word from coming out [baba]?

- Markedness constraints look at the surface representation.
  - The simplest ones can be defined by the structural description that they ban: *[+voice]#, $*C]_\sigma$.
  - Typical markedness constraints reflect articulatory ease, or perceptual clarity, rhythmic organization, or other "natural" drives.[4]

  - You can (and should!) give a constraint a helpful mnemonic name, like NOCODA for $*C]_\sigma$, as long as you precisely 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**.

  - o What are some different ways that NOCODA might count violations?

- Faithfulness constraints look at the *relationship* between the underlying and surface representations (the standard ones require similarity but we can imagine other possibilities).

  - P&S's PARSE (≈ don't delete) and FILL (≈ don't insert), were quickly superseded by McCarthy & Prince's correspondence constraints (the theory behind which we'll see next time), so let's start using the newer names now:

    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 mistake is thinking it means "don't alter a segment that is +F (e.g., by changing its values for some other feature G)".

[4] 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. See Moreton 2008.

10. Exposition: the tableau

- Someday, we'll all check our analyses with software that evaluates the infinite candidate set.[5]
  - In the meantime, we illustrate an analysis with a *tableau*[6] 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?)

- This tableau shows a *ranking argument*:
  - NOCODA prefers *a* (the winner), whereas DEP-V prefers *b*.
  - If that's the only difference between the candidates—no other constraint not known to be ranked below DEP-V 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 (not all structure shown)
- constraints (highest-ranked on left)
- asterisks
- exclamation marks
- shading
- pointing finger (you can use an arrow)

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

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

This procedure 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. How far to take this step is a matter of judgment .
- 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.

[5] See Jason Riggle's page for some software along these lines: http://hum.uchicago.edu/~jriggle/riggleDiss.html
[6] French for 'table'. The singular *tableau* is pronounced [tabló] in French; a typical English adaptation is [t^hæblóʊ]. The plural *tableaux* is also pronounced [tabló] in French, [t^hæblóʊ] or [t^hæblóʊz] in English.

- Let's try it for /atka/ → [atəka].

- One of the candidates below is unnecessary in arguing for the constraint ranking. Why? *plickers: A, B, or C*

/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.

- Here's a subtler case of harmonic bounding—explain:

/at+kap+so/	*CC	DEP-V
a atkapso	*!*	
b atkapəso	*!	*
☞*c* atəkapəso		**

12. 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—explain.

- Draw a comparative tableau for /at+kap+so/ too. Then try to make one where *b* wins.

Next time: Practice with OT; correspondence theory; targets vs. processes

References

Albro, Daniel. 2005. A large-scale LPM-OT analysis of Malagasy.. UCLA phd dissertation.

Eisner, Jason. 1997. *Efficient generation in Primitive Optimality Theory*.. Philadelphia.

Ellison, Mark T. 1994. Phonological derivation in Optimality Theory. *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, 1007–1013. Kyoto.

Moreton, Elliott. 2008. Modeling modularity bias in phonological pattern learning.. In Natasha Abner, Jason Bishop, & Kevin M Ryan (eds.), *Proceedings of WCCFL 27*, 1–16.

Prince, Alan & Paul Smolensky. 2004. *Optimality Theory: Constraint interaction in generative grammar*.. Malden, Mass., and Oxford, UK: Blackwell.

Riggle, Jason. 2004. Generation, recognition, and learning in finite state Optimality Theory.. University of California, Los Angeles ph.d. dissertation.

Samek-Lodovici, Vieri & Alan Prince. 1999. *Optima*.. London and New Brunswick, NJ.

Tesar, Bruce. 1995. Computational Optimality Theory.. University of Colorado.

Walker, Rachel. 2005. Weak Triggers in Vowel Harmony. *Natural Language & Linguistic Theory* 23(4). 917–989.

Wilson, Colin. 2001. Consonant Cluster Neutralisation and Targeted Constraints. *Phonology* 18(1). 147–197.