

## Class 2: Introducing Optimality Theory

- Human grammars: Unity in Variety – say more
- The ascendancy of the footnotes  
OT embodies the elevation (and in some case reification) of conditions that, in earlier generative (rule based and autosegmental) grammars were either overtly stated or merely lurking as generalizations Subjacency, the OCP, MSCs, (stray erasure?)
- There are two kinds of constraints in OT:
  - i) **Markedness constraints** apply to surface forms (never to input) and enforce universal drives like articulatory ease, perceptual distinctiveness, and other ‘natural’ desiderata.
  - ii) **Faithfulness constraints** apply to the relationship between the input and the output and enforce various metrics of similarity between the input and output.

Q: Can you come up with some markedness and faithfulness constraints?

1	1 DEP
2	2 MAX
3	3 IDENT
4	4
5	5

Q: How are constraints universal if they are untrue (massively violated) in some languages?

Q: Give me some examples of conflicting constraints and tell me how this conflict is resolved?

- How about conflicting markedness constraints?

Q: What can you tell me about strict domination?

- An example of strict domination you are familiar with is alphabetization. When determining where a word should be placed in an alphabetized list, the first letter in the word is strictly more important than the second which is strictly more important than the third and so on (e.g. [azzzzzzz] precedes [baaaaaaa]).

Q: Other theories of constraint interaction have supposed that the constraints are ‘weighted’ (this is actually common outside of linguistics). How would something like weighted constraints handle an alphabetization problem (e.g. with the words: axiom, banana, and tab).

Q: How are constraints grounded?

Give me some examples?

- **Where candidates come from: GEN**

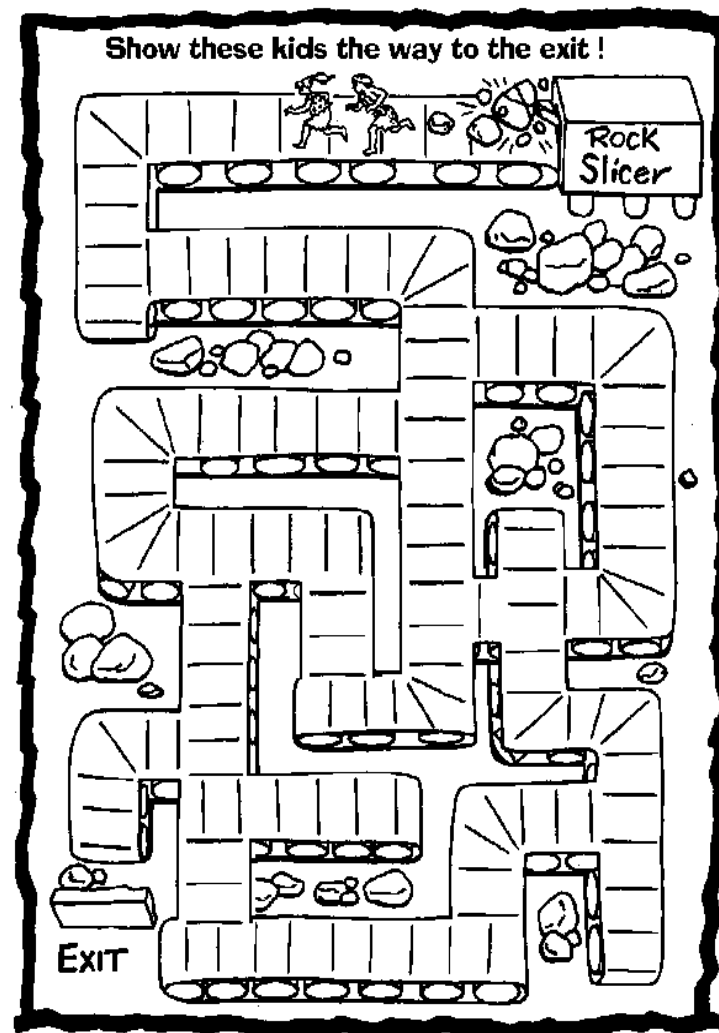
The idea is that we start with the input and then there is a function called **GEN** which generates the set of all possible candidates. **GEN** can **delete, insert, change, copy** and **rearrange** material in the input form.

Q: How big does this make the candidate set?  
How do the candidate sets for two different inputs differ?

- **Finding the optimal candidate**

The basic idea is that we take the candidate set and throw out all members that violate the highest ranked constraint, and then throw out all members that violate the next highest ranked constraint, and so on until there is only one candidate left (or possibly several that have exactly the same violations).

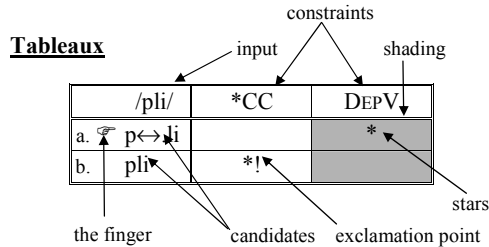
What is the naive strategy for finding the optimal candidate and can it work?



● **Showing your work**

In illustrating OT analyses we typically rely on a device that's a sort of visual anecdote used to illuminate how the crucial constraints work to pick out the observed form out of the vast sea of competitors. This is called a 'tableau' (and pluralized to tableaux).

In presenting your analysis you are trying to persuade your audience that the constraints and ranking that you have given select the desired candidate as optimal over all competitors. Nonetheless you must resist the temptation to use massive tableaux with huge numbers of constraints and candidates and instead try to find nice minimal cases with two or three candidates and two or three constraints that show how the constraints interact.



Q: How could you find *the* optimal candidate among the infinite range of possible candidates?

8. **Practice**

Fill in the violations, exclamation mark, shading, and finger of optimality in the following:

	/aórta/	MAXV	*VV	IDENT(stress)	DEPC
a.					
b.					
c.					
d.					
e.					

9. **How do you know which candidates and constraints to include in a particular tableau?**

This can actually be quite tricky! Though it's often hard to get exactly the right candidates it is pretty clear which ones aren't all that relevant.

Basically the rule of thumb is that you should consider candidates that use every reasonable way to satisfy each markedness constraint that you include in the tableau.

	/pli/	*CC	DEPV
a.	(☛) pɛ.li		*
b.	pli	*!	
c.	☛ pi		

The bomb icon means that the analysis is crashing and selecting the wrong winner (the one with the bomb). The parentheses around the pointing finger indicate that it is the desired winner but is not being chosen.

The skull and crossbones ☠ is also used when an undesirable winner is chosen. This symbol is often used to indicate that the selected candidate is not only wrong but somehow 'pathological' – a candidate that no language should ever select as optimal in the context that's been described.

**Exercise** – Fill in violations so that each of these candidates can win under some ranking

/input/	C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>
a. candidate a			
b. candidate b			
c. candidate c			
d. candidate d			
e. candidate e			
f. candidate f			
g. candidate g			

**Exercise** – Imagine that human language has only two symbols (units) to work with C and V and come up with all of the faithfulness constraints that you can.

**Faithfulness**

**Markedness**

**Come up with constraint rankings for the following languages:**

- Language A does not allow CC sequences

**Ranking:**

- Language B does not allow VV sequences

**Ranking:**

- Language C demands that C and V alternate

**Ranking:**

- Language D allows any sequence of C's and V's

**Ranking:**