Class 13: Systems similar to OT

First, finish up Albro’s 3 formal extensions to OTP

(1) Self-conjunction

∧C1/G is violated any time there are two violations of C1 in domain G (a self-conjunction can be further conjoined with itself, so we can specify any number of violations).

Albro gives an algorithm for self-conjunction of C1 in domain G (interior of tier G):

1. Weaken the constraint:
   • Make 2 copies of C1’s FSA.
   • For every non-zero-weighted arc in the first copy, subtract one from its weight and make it go to the start state of the second copy.
   This means that the first violation of the constraint doesn’t count, but subsequent ones do.
   Call the resulting machine $T$.

2. Make an FSA that accepts anything with one or more complete $+$s for the domain G (it can begin with garbage—not sure if this is crucial)

3. Intersect $T$ and $M$
   Except give a weight of zero to any of the following arcs
   • intersection of $M$’s $-]|G$ arc and any arc from $T$
   • intersection of $M$’s $[G$ arc and any arc from $T$ that contains no reference to a left edge
   • intersection of $M$’s $]|G$ arc and any arc from $T$ that contains no reference to a right edge
   This means that violations of C1 outside the domain G won’t count at all.
And if, in one of the above three cases, the \( T \) arc was one that went to the start state of the second copy in \( T \), treat the intersected arc as if it went to the start state of the first copy in \( T \).

This resets the weakened counter every time a new \( G \) domain is entered, so that the first violation of \( C1 \) within the new domain isn’t counted.

Shall we try an example?

(2) *Conjoining distinct constraints*

\[ C1 \land C2/G \text{ is violated any time both } C1 \text{ and } C2 \text{ are violated in domain } G \]

Albro suggests that to get something like \( C1 \land C2/G \), we take \( \land(C1 \lor C2)/G \) to get a “nearly equivalent” result.

What would be the difference?

**OT and connectionism**


(3) *Basic connectionism*

- Nodes, with individual activation values and pairwise connection weights
- Positive weight --> nodes should have same activation
- Negative weight --> nodes should have opposite activations (say, 1 and \(-1\)*)
To quote Smolensky p. 2:

- “Mental representations are distributed patterns of numerical activity.
- Mental processes are massively parallel transformations of activity patterns by patterns of numerical connections.
- Knowledge acquisition results from the interaction of
  - innate learning rules
  - innate architectural features
  - modification of connection strengths with experience”

(4) Running the network

- Input is a set of activation values on certain nodes—for language generation, these must be ‘clamped’ (not allowed to change)
- Activations flow through the net according to the connection weights
- Output is given by the activation values of certain nodes at the end

(5) Harmony function
(type of Lyapunov function)

- Assigns a number to the entire state of the network that will increase over time as the network ‘runs’.
- $= \sum_{i,j} a_i \times a_j \times w_{ij}$ for all neuron pairs $a_i$ and $a_j$, the activation of $a_i$ times the activation of $a_j$ times the connection weight between the two.
- Number is higher when modes with strong positive connections have the same activation and when nodes with strong negative connections have opposite activations.
- In the linguistic case, this might represent how well various constraints are satisfied.

(6) Tensor product representations
For encoding hierarchical structure in a 2D matrix

Have a vector that means ‘left’: <0,1,0,1, .5, 0>
and a vector that means ‘right’: <.5, 0, .5, .5, .5, 1>
to encode $x \otimes y$, take the tensor product of $x$ with ‘left’ and $y$ with ‘right’:

<table>
<thead>
<tr>
<th>'left'</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>.5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>.5</td>
<td>0</td>
<td>.5</td>
<td>0</td>
<td>.5</td>
<td>.25</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>.5</td>
<td>0</td>
<td>.5</td>
<td>0</td>
<td>.5</td>
<td>.25</td>
<td>0</td>
</tr>
<tr>
<td>.5</td>
<td>0</td>
<td>.5</td>
<td>0</td>
<td>.5</td>
<td>.25</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>.5</td>
<td>0</td>
<td>.5</td>
<td>0</td>
<td>.5</td>
<td>.25</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>'right'</th>
<th>.5</th>
<th>0</th>
<th>.5</th>
<th>.5</th>
<th>.5</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>.5</td>
<td>.25</td>
<td>0</td>
<td>.25</td>
<td>.25</td>
<td>.25</td>
</tr>
<tr>
<td>1</td>
<td>.5</td>
<td>0</td>
<td>.5</td>
<td>.5</td>
<td>.5</td>
<td>.5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>.5</td>
<td>.25</td>
<td>0</td>
<td>.25</td>
<td>.25</td>
<td>.25</td>
<td>.5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>.5</td>
<td>.25</td>
<td>0</td>
<td>.25</td>
<td>.25</td>
<td>.25</td>
<td>.5</td>
</tr>
</tbody>
</table>

then add the two:

<table>
<thead>
<tr>
<th>.25</th>
<th>.50</th>
<th>.25</th>
<th>.75</th>
<th>.50</th>
<th>.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>.50</td>
<td>0</td>
<td>.50</td>
<td>.50</td>
<td>.50</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>.50</td>
<td>0</td>
<td>.50</td>
<td>.25</td>
<td>0</td>
</tr>
<tr>
<td>.25</td>
<td>.50</td>
<td>.25</td>
<td>.75</td>
<td>.50</td>
<td>.50</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>.25</td>
<td>.50</td>
<td>.25</td>
<td>.75</td>
<td>.50</td>
<td>.50</td>
</tr>
</tbody>
</table>

Why is this recoverable?

(7) *Harmonic grammar*
(Legendre, Miyata, and Smolensky)

- nodes correspond to linguistic structures
- define a set of soft constraints
  - structure $i$ requires structure $j$ = positive connection weight
  - structure $i$ repels structure $j$ = negative connection weight
- domination is not strict—ganging up is allowed
  - high absolute value of connection weight = strong constraint
(8) Connectionist OT?

How do we get strict domination?

- Prince & Smolensky (1997) suggest making “the numerical strength of a constraint [...] much lower than the strengths of those constraints ranked lower than it in the hierarchy; so much so that the combined force of all the lower-ranked constraints can never exceed the force of the higher-ranked constraint.” (p. 1608)

- But if there’s no upper bound on the number of violations a given form can have, ‘never’ can’t be guaranteed.

OT and stochastic constraints


Frisch, Stefan, Michael Broe, and Janet Pierrehumbert (in prep). Similarity and phonotactics in Arabic. Revision of submission to Natural Language and Linguistic Theory.

(9) Lexical regularities

Arabic OCP-Place, for example: consonants with the same place of articulation are not strictly forbidden within a root. Rather, the more similar two consonants in the same place category are, the less likely they are to co-occur.

(10) Acceptability functions

- Functions from some numerical phonological characteristic to an acceptability value between 0 and 1.
- Acceptability should then be reflected in lexical frequency.
- The functions are of the form \( \text{acceptability} = \frac{1}{1 + e^{-kxx}} \)
- Parameter \( s \) determines the steepness of the boundary between acceptable and unacceptable, and parameter \( k \) determines the location of the boundary
(11) **Constraint conflict**

To get an overall acceptability rating for a trilateral root, the acceptability functions for the three consonant pairs are multiplied together (to get a good fit, numerators other than 1 are used).

Each function may have a different steepness or cutoff point.

No strict domination—ganging up is possible.

(12) **Generation?**

Frisch 1996: “[the stochastic constraint] does not influence what the output is for any particular input, but rather it constrains the space of possible inputs and outputs in a probabilistic manner.” (p. 92)

These constraints aren’t intended to do generation, but can we adapt them so that they do?

What if we choose the most acceptable from a set of candidates...
McCarthy and Prince (1995) Balangao allows codas freely allowed in the base and inside a reduplicant, but not at the end of a disyllabic reduplicant.  

<table>
<thead>
<tr>
<th>/RED-tagtag/</th>
<th>B-R Contiguity</th>
<th>I-O Contiguity</th>
<th>Max-IO</th>
<th>No-Coda</th>
<th>Max-BR</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. → tagta-tagtag</td>
<td></td>
<td></td>
<td>***</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>b. tagtag-tagtag</td>
<td></td>
<td></td>
<td>****</td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>c. tagta-tagta</td>
<td></td>
<td></td>
<td>!</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>d. tata-tagtag</td>
<td></td>
<td></td>
<td>!</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>e. tata-tata</td>
<td></td>
<td></td>
<td>!( )</td>
<td>!( )*</td>
<td></td>
</tr>
<tr>
<td>f. tata-tagta</td>
<td></td>
<td></td>
<td>!</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Can we translate the OT tableau into Frischian stochastic constraints?

- Each constraint must assign an acceptability value between 0 and 1 to each candidate (the acceptability values will then be multiplied together, resulting in a total acceptability value between 0 and 1).

- If a candidate has no violations of a constraint, the acceptability value for that constraint will be 1. The more violations, the lower the acceptability value.

How to weight values for each constraint so that violation of a high-ranked constraint is more damaging to the total acceptability value?

- Since the values are to be multiplied together, merely multiplying each value by some factor will be useless (by commutativity and associativity, there is no difference between (10 * value1)*(2*value2) and (2*value1)*(10*value2)).

- Steepness and cut-off point don’t work either: if we make a higher-ranked constraint steep, we have to commit to what the fatal number of violations is.

- Let’s try acceptability = \( 1 - \frac{\# \text{ of violations}}{\# \text{ of violations} + a} \) \( \frac{\text{a}}{\text{a}} \) where \( a \) is a positive number, the value of which determines how strict (highly ranked) the constraint is.

- A small value for \( a \) means that acceptability approaches zero quickly as the number of violations grows (i.e., acceptability is very low with just one violation)—i.e., the constraint is very strict.

1 Compare Tagalog, where a reduplicant-final coda is allowed only if the reduplication is total (i.e., for two-syllable reduplication of a disyllabic base). A constraint like TOTAL-BR or edge-anchoring must outrank NOCODA.
Letting $a=0.1$ for the top three constraints, $a=10$ for NoCoda, and $a=20$ for MaxB-R, acceptability values for all the candidates in the tableau are:

<table>
<thead>
<tr>
<th>candidate</th>
<th>acceptability</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagta-tagtag</td>
<td>.73</td>
</tr>
<tr>
<td>tagtag-tagtag</td>
<td>.71</td>
</tr>
<tr>
<td>tagta-tagta</td>
<td>.076</td>
</tr>
<tr>
<td>tata-tagtag</td>
<td>.069</td>
</tr>
<tr>
<td>tata-tata</td>
<td>.004</td>
</tr>
<tr>
<td>tata-tagta</td>
<td>.079</td>
</tr>
</tbody>
</table>

The candidates that violate the undominated constraints are clear losers, but tagtag-tagtag is not such a clear loser.

Can we make tagtag-tagtag worse?

- Setting $a=100$ for MaxB-R improves the winner's score to .76 (and does not change the score for tagtag-tagtag), but the improvements diminish exponentially as $a$ is increased for MaxB-R (the winner’s score increases towards a limit of .769).

- The problem is that any value of $a$ for NoCoda that makes the winner tagta-tagtag (with three violations) highly acceptable must also make tagtag-tagtag, (with four violations) fairly acceptable.

It looks like phenomena in which some candidates are uniformly unsuccessful despite obeying all undominated constraints aren’t amenable to a straightforward OT-to-Stochastic-Constraint adaptation.

Course summary

*Why are we interested in computation?*

- Theoretical reasons—our theories of how the mind works have to be computable
- Practical reasons—we want to build machines that can carry out our theories

*Computational issues*

- Is a problem solvable at all?
- In finite time?
- In feasible time?
- By what kind of machine?
**Tasks in OT: Generation**

Problem: infinite candidate set

Solutions:
- Tesar: dynamic programming/chart parsing
  Consider only a small set of partial candidates at a time.
  Requires property like the syllabic cycle (no need to insert a whole syllable).
- Ellison, Eisner: manipulate regular expressions/finite-state automata instead of actual lists of candidates

**Tasks in OT: Comprehension**

Problem: infinite set of possible inputs if not restricted to existing words

Solution? Maybe manipulating regular expressions, or other compact descriptions, instead of lists of possible inputs could work.

**Tasks in OT: Learning the ranking**

Problem: factorial number of grammars

Solutions:
- Tesar & Smolensky: constraint demotion (CD)—works only for invariable grammars.
- Boersma: gradual learning algorithm (GLA)—works for variation, robust to errors, but because the GLA is probabilistic, it’s harder to study its computational properties.

**Tasks in OT: Learning the lexicon**

Problem: infinite set of possible inputs

Solution:
- Lexicon optimization: input is identical to output unless alternations occur (but how to resolve them?)

Problem: How to learn grammar without knowing lexicon, lexicon without knowing grammar?

Solution:
- Tesar: robust interpretive parsing interleaved with constraint demotion

**Enhancing our formal understanding of OT**

- Orderings of candidates—standard OT (Samek-Lodovici & Prince) and OT with targeted constraints (Wilson)
- Defining the optimal candidate(s)
- What kind of languages are OT-grammar-defined languages? Frank & Satta: they are regular languages if constraints are regular and if we can put an upper bound on number of constraint violations (perhaps through local evaluation).
Tools
Computing devices
• finite-state automata
• pushdown automata
• Turing machines

Allow us to investigate the computational properties of an algorithm (e.g., if it can be implemented with a FSA, various facts follow).
Allow us to manipulate infinite sets with finite means.

Learnability theory
Gold paradigm
• Learner tries to identify language from exposure to (ideally a finite portion of) a text consisting of all strings in language, in random order and with repetitions allowed.
• Some languages are identifiable from all possible texts, others from just some texts, others maybe not at all.
• Learners that know when they’re done are self-monitoring—maybe not necessary for human language.

PAC learning
• Learner tries to identify language from exposure to possibly frequency-weighted text, with strings labeled as in or out of the language (not realistic for human language).
• Learner’s task is to achieve, some minimum percentage of the time, some minimum level of accuracy (in categorizing new strings as in or out).

Complexity and parameterized complexity
How much time/memory does a task take?
Which aspects of the task make it take a long time, and can they be controlled?

Stochastic and variable grammars
Free ranking—true ties vs. variable ranking
Probabilistic ranking: free, but with preferences

Open questions
• Comprehension!
• Constructing a lexical entry from a set of allomorphs
• Primitive OT: is it powerful enough to capture all the constraints we want?

What else can you think of?

The End. Have a good break.