Week 10: Formalizing OT


*loser* = candidate that loses under every ranking
*winner* = candidate that wins under some ranking

- How does telling winners from losers save us from infinity?
- How can Recursive Constraint Demotion (RCD) be used to tell if a candidate is a loser?

**Formal preliminaries**

**Constraints**
- previous view: constraint is a function from candidate to number
- new view: constraint is a function from a candidate set to subset of that set.

Comparative tableau: rows are candidate pairs, cells tell you which member of the pair is preferred by the constraint (if no cells with candidate $z$, it’s a loser—but not the converse).

Relative harmony (p. 38): reminds us that a ranking doesn’t just select an optimal candidate; it orders all the candidates (this fact is exploited in Colin Wilson’s work on opacity). The same is true of any sub-part of the ranking.

A constraint defines a *partial ordering* of candidates.

Poset (partially ordered set) vs. stratified hierarchy

- strata: $\neg(a>b \lor b>a) \rightarrow ((a>x \rightarrow b>x) \& (y>a \rightarrow y>b))$
- poset: set ordered by a reflexive, anti-symmetric, transitive relation

Strata work for candidates as ordered by a constraint, but posets (more general) may be necessary for constraints as ordered by a ranking.

**Review: properties of relations**

reflexive: $(\forall a) (aRa)$
irreflexive: $(\forall a) (\neg aRa)$
nonreflexive: $(\exists a) (\neg aRa)$

symmetric: $(\forall a)(\forall b) (aRb \rightarrow bRa)$
asymmetric: $(\forall a)(\forall b) (aRb \rightarrow \neg bRa)$
antisymmetric: $(\forall a)(\forall b) ((aRb \land bRa) \rightarrow a=b)$
nonsymmetric: $(\exists a)(\exists b) (aRb \land \neg bRa)$
transitive: \((\forall a)(\forall b)(\forall c) ((aRb \land bRc) \rightarrow aRc)\)
intransitive: \((\forall a)(\forall b)(\forall c) ((aRb \land bRc) \rightarrow \neg aRc)\)
nontransitive: \((\exists a)(\exists b)(\exists c) ((aRb \land bRc \land \neg aRc)\)

An equivalence relation is one that is reflexive, symmetric, and transitive.

*Composition*
\[ f \circ g (x) = f(g(x)) \]

The composition of functions \(f\) and \(g\) is itself a function. It’s the result of applying \(f\) to the result of applying \(g\).
In other words, to apply the composition of \(f\) and \(g\) to \(x\), first apply \(g\) to \(x\), then apply \(f\) to the result. Order matters!

**Bounding**

*Harmonic bounding*
A candidate \(a \in K\) is harmonically bounded relative to the constraint set \(\Sigma\) if there exists some other candidate \(b \in K\) that does at least as well as \(a\) on all constraints, and better than \(a\) on at least one.

*Bounding set:* set of candidates that collectively bounds some candidate
A set of candidates \(B \subseteq K\) is a bounding set for \(a \in K\) relative to a constraint set \(\Sigma\) iff:
- Strictness: Every member of \(B\) is better than \(a\) on at least one constraint in \(\Sigma\)
- Reciprocity: If \(a\) is better than some member of \(B\) on a certain constraint \(C \in \Sigma\), then some other member of \(B\) beats \(a\) on \(C\).

(a one-member bounding set is the degenerate case)
A candidate is a loser iff it has a non-null bounding set.

- Discuss p. 10: Misleadingly universalized weak bounding is too strong, and pseudo-reciprocity is too weak.

**Minimal bounding set**
If \(B\) is a bounding set for \(a\) on \(\Sigma\), then \(B\) is a minimal bounding set for \(a\) on \(\Sigma\) iff no strict, non-empty subset of \(B\) is a bounding set for \(a\) on \(\Sigma\).

*Bounding Theorem* (p. 11)
Every loser has a non-null bounding set (and every winner’s only bounding set is null).

**Finding a bounding set**
It would take too long too check all possible sets.

*Aside: Covering set*
\(K\) is a covering set for candidate \(a\) iff for all constraint rankings, some candidate \(x \in K\) is more harmonic than \(a\).
The covering set must contain within it a bounding set.
• When is a covering set not itself a bounding set? (p. 12)

*S-L&P’s basic strategy*
Check candidates to exclude them from the bounding set.

*Favor*
A constraint $C$ favors a candidate $a$ with respect to a set of candidates $K$ if $a$ does better on $C$ than any other member of $K$.

*Constructing the favoring hierarchy* (p. 22)
1. First stratum: all constraints that favor $a$ with respect to the candidate set $K$ (the set of favoring constraints of $a$ in $K$).
   
   Eliminate from the possible bounding set any candidates that are not favored by all the constraints in the stratum—that is, take the intersection of the top strata of all the constraints in the stratum.

2. Next stratum: all remaining constraints that favor $a$ with respect to the candidates that have not been eliminated.
   
   Eliminate from the remaining possible bounding set any candidates that are not favored by all the constraints in the stratum.

3. Repeat 2 until you run out of constraints (then $a$ is a winner), or until none of the remaining constraints favor $a$ (then $a$ is a loser and the remaining candidates are the bounding set).

   The favoring hierarchy turns out to be the hierarchy that the RCD would get in trying to construct a ranking that chooses $a$.

• Let’s try some examples: Gnanadesikan, Curtin

• Compare to RCD (pp. 25-26)

*Size of the Bounding Set*

*Residual candidates*
Candidates that are better than $a$ on some constraint. The residual candidates are a bounding set

strictness: by definition
reciprocity: for every residual constraint, some residual candidate is better than $a$

This puts an *upper bound* on the size of the minimal bounding set.

*Disfavoring system*
A disfavoring system for $a$ is a collection of constraints in which, for each constraint, $a$ loses to some other candidate
Blocking set
For each constraint in a disfavoring system for $a$, pick one candidate that beats $a$. The blocking set will have the same size as the number of constraints in the disfavoring system.

A blocking set is a bounding set (p. 31).

The residual constraints are a disfavoring system for $a$, so a blocking set constructed from them is a bounding set for $a$.

So, we can always construct a bounding set for $a$ that is no bigger than the number of residual constraints for $a$.

• Let’s apply the examples we worked above

Aside: If all the constraints are binary, then the smallest possible bounding set for any loser has just one member (p. 12).

Discussion
• So this gets us a finite number of candidates to worry about (for learning, generation), but of course the initial procedure deals with an infinite set of candidates, so we still need to finite-ize that.

• Can we adapt this procedure to identify not just loser candidates, but loser languages?

• It would be fun to make an implementation of this.

Next time
• Primitive OT

For next time
• Reading
  o Eisner 1997 LSA handout “What Constraints Should OT Allow”—no study questions but see below.
• Homework
  o Construct the favoring hierarchy for each candidate in some tableau—perhaps one of the tableaux from the example you used in the Recursive Constraint Demotion or Gradual Learning Algorithm homeworks. Show intermediate steps, and don’t forget to include the tableau.
  o Using the same tableau if possible, rewrite the constraints in Eisner’s Primitive OT (see pp. 6-13 for examples), and draw the candidates using Eisner’s timeline notation (see 15 on p 4). Bring 7 copies to class.
1. Pick one of the theorems in Section 4 or Section 5 of “Optima” and retell it in your own words, giving examples. Limit: one page.