Week 12: Computability and Complexity

(First finish discussion of OTP from last week)

Basic definitions

Algorithm/program: function that maps some INPUT to some OUTPUT. (I’ll use small caps to distinguish from the OT sense)

A problem can be thought of as some specification of the desired behavior of the algorithm.

Example: OUTPUT “yes” if some constraint ranking exists under which the input-to-output mappings in the INPUT would be selected as optimal; OUTPUT “no” otherwise.

Minimal criteria for any OT algorithm (strict view)

• Algorithm must never produce an incorrect answer.

• Algorithm must always settle on an OUTPUT after a finite number of steps.

• The number of data that the algorithm is storing is finite at all times (though it may not be possible to place an upper bound on the amount of memory that the algorithm ever needs).

• Algorithm itself must be finite—can be written down in a finite number of characters.

What if we want to implement an algorithm on a computer?

• We do need an absolute bound on the amount of memory needed—or else accept that certain INPUTS can’t be handled.

• The number of steps needed mustn’t ever get too big (we want the results in a reasonable amount of time).

*How big is too big?*

If the number of steps needed is $2^n$, where $n$ is the length of the INPUT, then the algorithm is effectively useless unless we know that INPUTS will never be long.
What about algorithms that we think humans run?

- The algorithm doesn’t always have to produce a correct answer.
  - Error in generation → slip of the tongue
  - Error in comprehension → mis-hearing
  - Error in learning → language change

- The algorithm can take advantage of parallelism.
  - Many candidates’ constraint violations can be computed at once.
  - Many candidate pairs can be compared at once.
  - Many partial parses can be entertained by the listener at once.

- The algorithm doesn’t always have to terminate. (For example, when you can’t think of a word, higher-order parts of the mind tell you to stop worrying about it and move on).

- The amount of memory needed must usually be below some amount, but it’s OK if sometimes too much memory is required (you just get an error).

- Perhaps there are some OT grammars for which generation, comprehension, and/or learning are not computable. Can gaps in the factorial typology be explained computationally? Can the inventory of correspondence constraints be explained computationally (e.g., arguments against ANCHOR-RIGHT and ALIGN-RIGHT)? Many good research topics here.

Next time

- Computability of standard OT

For next time

- Reading
  - Todd Wareham, Systematic Parametrized Complexity Analysis in Computational Phonology, pp. 1-25 and 153-169 (I’ve included the whole dissertation in the readings box in case you want to refer to other sections)
  - Jason Eisner, Efficient Generation in Primitive Optimality Theory

- Homework
  - Prepare a 1-page summary of the Wareham reading, pp. 1-25 only!
  - The second section (pp. 153-169) is very dense—we’ll go over it in detail in class—so prepare for yourself (not to turn in) any questions you have about it.