Strict Locality & Phonological Maps

Key Points:

- Input Strictly Local functions can model phonological UR-SR maps that involve simultaneous rule applications.
- Iterative/spreading processes with two-sided contexts and long-distance processes are also local in a sense and can be modeled with Output Strictly Local functions.
- These ISL functions are subclasses of the regular relations that can be learned in the limit from positive data.

1. **Finite State Acceptors (FSAs) and Finite State Transducers (FSTs)**
   FSAs (discussed in class): a string is accepted, i.e. in the language, if the state the FSA is in is a final state (double circle).

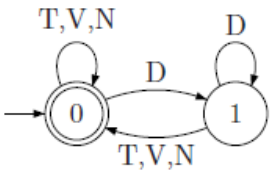

Figure 2

FSTs: produces an output string that could be different from the input as it proceeds through the transitions.
E.g. Final devoicing: tad → tat, but tada ok (for d to be in the final state, 2, it has to surface as a t)

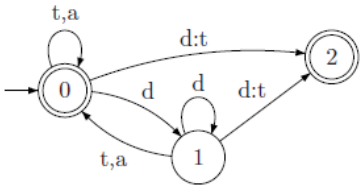

Figure 3: FST for final devoicing

- FSTs do not restrict the input: assuming that there are only {a, d, t} in this language, final-d will always surface as [t] regardless of what the input string looks like (e.g. ddd, tad) ➔ similar to the Richness of the Base in OT (you can have whatever input candidates you want)
- SR→UR is a regular relation, but it is not a strong enough restriction ➔ how do we exclude regular relations that are unattested and typologically odd?
- Earlier proposal: subsequential FSTs (SFSTs) → deterministic: at any given state there is only one outgoing transition per input symbol in the alphabet.

Input Strictly Local Functions: a more restrictive class of functions that is a proper subset of both the regular relations and the subsequential functions.

2. Input Strictly Local Functions

One alphabet can only belong to one state which can be a class of alphabets or just this particular alphabet), determined by the input, not the output. The available paths from the state of an alphabet are the set of tails (they come from that exact same state).

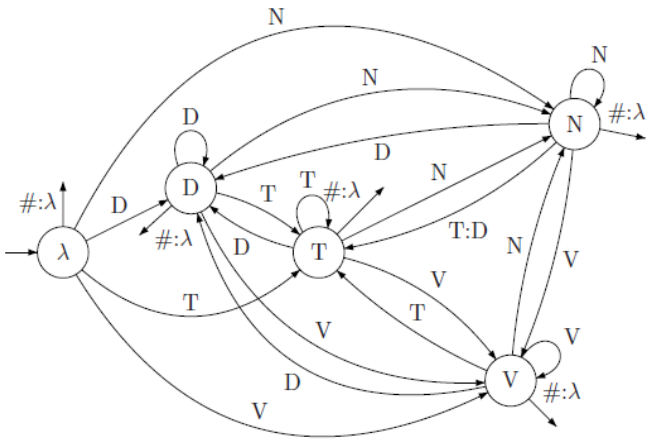

Figure 6

T = voiceless obstruents, D = voiced obstruents, N = nasals, V = vowels
Quechua Post-nasal Obstruent Voicing:
/kampa/ → [kamba] 'yours'
⇨ An ISL FST that maps NT to ND

How it works:

- It takes the input string one symbol at a time: the first symbol, k, takes the path to State T. At this stage, there are different paths (aka tails) to different states (T is the prefix of these subsequent states). As the next symbol is a, it goes to State V. Then it goes to State N. At this state, the function needs to "memorize" this nasal and look out for the sequence NT. The path it chooses is still from State N to State T, but this function maps the T to D before landing on State T.
- This one, k = 2: the output depends on the current input symbol and the preceding 2-1=1 symbol of the input string.

What is a non-ISL FST ➔ from one state, an alphabet has more than one path.

3. ISL Phonological Maps

German final-devoicing (one rule)

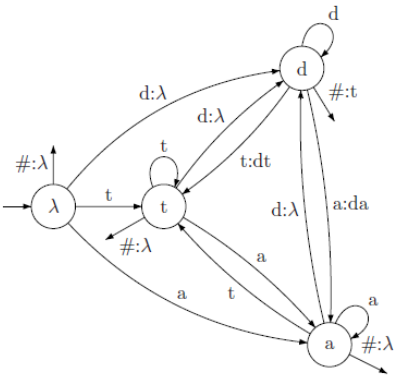

Figure 8: 2-ISL FST for final devoicing, with $\Sigma = \{d, t, a\}$

Dutch schwa-epenthesis (one rule): between [l, r] and [-cor] C

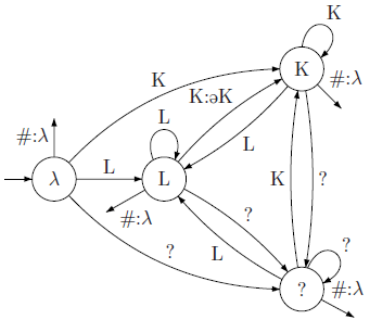

Figure 9: 2-ISL FST for Dutch ə-epenthesis, with $\Sigma = \{L, K, ?\}$

Rotuman -VC metathesis (k = 4 → look for three symbols)

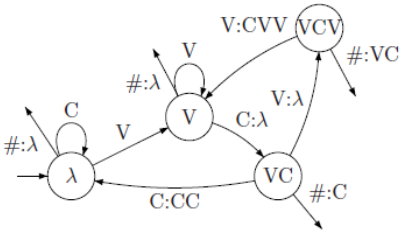

Figure 11: 4-ISL FST for Rotuman metathesis, with $\Sigma = \{V,C\}$

Remark: of approximately 5500 phonological patterns from over 500 languages, 95% are ISL.

Multiple rules: Consider this UR-SR map: /CVnp/ → [CVmb]

1. Post-nasal obstruent voicing (p → b)
2. Nsal place assimilation (n → m)

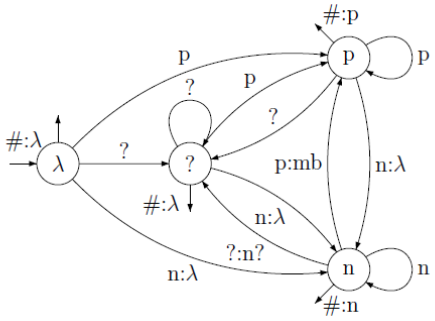

Figure 12: 2-ISL FST representing both post-nasal obstruent voicing and nasal place assimilation, with $\Sigma = \{p, n, ?\}$

4. Output Strictly Local

The rules introduced in the previous section can be applied simultaneously or iteratively. But it's not always the case, e.g. spreading: if the rules apply simultaneously, it's 2-ISL. If iterative, it's not ➔ OSL.

The major difference between OSL and ISL is that the states of the OSL are NOT the output but the most recently modified input. The path from such a state looks at this modified input and there is another OSL function that modifies this.

5. ISL and Phonological Learning

Given a set of (UR, SR) (and a given k):

1. Build a prefix tree
2. Merge states

Given an input, it starts by building different states (remember each input symbol can only belong to one state), from the start of the input symbol to the end. At this point, these states form a prefix tree. This prefix tree is FST, not FSA because it's not about whether a string is in the language or not, it's to map the input-output. The learner merges the states that share a suffix of k-1 until it converges on an FST (one path?) that describes the function that the data set samples. hy

6. Implication for Phonological Theories

Major criticism against OT: There seems to be a conspiracy among phonological processes with local triggers that they all have the property of being ISL or OSL function, but OT has no clear way to account for this.