Formal Language Theory and Phonology: Day 3

Jane Chandlee and Adam Jardine

July 10, 2023

1 Phonological processes as functions

So far we've been working with languages/stringsets, which as we've seen can be used to model phonotactic constraints.

For example, the pattern of 'final devoicing' can be represented with the SL-2 grammar {b$\ltimes$, d$\ltimes$, g$\ltimes$, z$\ltimes$, ... }, or the set of strings that do not contain any of these 2-factors.

But phonology is more than phonotactics. When strings *do* contain these 2-factors, some type of repair (usually devoicing) is employed. Depending on the theory we're working with, we might say the language has the following rule in its grammar:

(1) [−son] → [−voice] / __ $\ltimes$

Or, we might say it ranks these constraints in the following way:

(2) *[−son, +voice] $\ltimes$ >> Ident(voice)

These are **intensional** descriptions, both of which describe the following **extension** or **map**:

intensional
extension
map

(3) {(ab, ap), (ad, at), (ag, ak), (ba, ba), (da, da), (a, a), ... }

Notice we're still dealing with a set; it's just a set of **string pairs** (input, output) rather than strings.

string pairs

We can then ask the same kinds of questions, like: how computationally complex is this map? Or, how complex is the computation of an output string for a given input string? How much (or what kind of) information does the device that performs this computation need?

As before, we can start with regular. The regular languages have a counterpart called the **regular relations**. Decades ago it was shown by Johnson (1972) and (Kaplan and Kay, 1981, 1994) that SPE grammars describe regular relations provided the rules can't reapply to their own structural change.[1]

regular relations

[1]The need for this restriction is demonstrated with the rule $\emptyset \rightarrow$ ab / a __ b, which can otherwise generate $a^n b^n$.

But also as before, we can be more restrictive than regular. One hypothesis is that phonological maps are **subsequential functions** (Mohri, 1997). This hypothesis is too strong: some attested processes are more complex than subsequential, but *a lot* of them aren't. So again it's a good place to start.

subsequential functions

2 Subsequential functions

As with languages, there are multiple ways to represent maps, but we'll be using an abstract characterization based on the concept of **tails**.

tails

First recall that a prefix of a string is any substring that includes the beginning. For a set of strings S, we can define the **common prefixes** as the set of prefixes shared by all strings in S.

common prefixes

For example, if $S = \{abc, abcd, abaa, abdac\}$, then the common prefixes of S, or $\texttt{comprefs}(S)$ = $\{\lambda, a, ab\}$.

The **longest common prefix** of a set S (or the $\texttt{lcp}(S)$) is the longest string in $\texttt{comprefs}(S)$. In our example, $\texttt{lcp}(S) = ab$.

longest common prefix

For a given function $f : \Sigma^* \rightarrow \Delta^*$[2] and a string $x \in \Sigma^*$, we define $f^p(x)$ as follows:

[2]Note the distinction between an input alphabet and an output alphabet.

$$f^p(x) = \texttt{lcp}(\{f(xu) : u \in \Sigma^*\})$$

Let's use an example to unpack this definition. Let $\Sigma = \Delta = \{a, b, c\}$ and let f be the function such that an a immediately following a b is turned into a c.

$f(bac) = bcc$
$f(ba) = bc$
$f(abc) = abc$
etc.

Now what is $f^p(ab)$? First consider all the ways we can extend the string ab: aba, abb, abc, $abaa$, $abab$, $abac$, etc. There are of course an infinite number of expansions.

We take each of these strings, apply f to it, and then compile the resulting output strings into a set:

$\{ab, abc, abb, abc, abca, abcb, abcc, ...\}$

What is the $\texttt{lcp}$ of this set? It's ab![3] So $f^p(ab) = ab$.

[3]Notice how we don't need to enumerate the entire (infinite) set to determine this $\texttt{lcp}$. For any input string that starts with ab, the output will also start with ab, because f does not alter this sequence.

Now we define the **tail function** f_x as follows:

tail function

$$f_x(u) = v \text{ such that } f^p(x)v = f(xu)$$

Again let's unpack this definition. In the above example, again with $x = ab$, what is:

$f_{ab}(a)$?

$f_{ab}(b)$?

$f_{ab}(c)$?

Now let $x = ba$. What is:

$f_{ba}(a)$

$f_{ba}(b)$

$f_{ba}(c)$

So this means the strings ab and ba have different tail functions. A given function will have some set of these tail functions.

How many tail functions does the identity function ($f(x) = x$) have?

This (finally!) leads us to our definition of subsequential:

Definition 1 (Subsequential function)**.** *A function f is subsequential iff the set $\{f_x : x \in \Sigma^*\}$ is finite.*

This looks familiar! Remember the regular languages are those with a finite set of equivalence classes. Two strings are equivalent w.r.t. a language if they 'behave' the same no matter how you extend them.

Tail functions also establish equivalence classes. Two strings are equivalent w.r.t. a function if they have the same tail function, meaning each possible input extension has the same effect on the output.

Note now there is a directionality built into the tail functions: we're always looking at what comes *to the left*.
There is a **right-subsequential** class that we get by reversing the (left-)subsequential functions.

right-subsequential

3 Non-subsequential functions

A common process in tone is **unbounded tone plateauing (UTP)** (Kisseberth and Odden, 2003; Hyman, 2011).

unbounded tone plateauing (UTP)

(4) UTP in Luganda (Hyman and Katamba, 2010)
- a. /ki-kópo/ [kikópo] 'cup'
- b. /ki-sikí/ [kisikî] 'log'
- c. /mu-tund-a/ [mutunda] 'seller' (from /-tund-/ 'to sell')
- d. /mu-tém-a/ [mutéma] 'chopper' (from /-tém-/ 'to chop')
- e. /mu-tund-a bi-kópo/ [mu-tund-a bi-kópo] 'cup seller'
- f. /mu-tém-a bi-sikí/ /mu-tém-á bí-síkî/ 'log chopper'

Over strings of TBUs, this looks something like this:

(5) UTP as a string map
H∅∅∅ → H∅∅∅
∅∅∅H → ∅∅∅H
H∅∅∅H → HHHHH
...

Let's call this map p and assume an input and output alphabet of $\Sigma = \{\text{H}, \varnothing\}$. This function p is not subsequential. Let's see why.

(It's also not right-subsequential. Can you think of why?)

4 Hierarchy of functions

We talked about the subregular hierarchy of languages, but is there a subregular hierarchy of functions?

Of course there is!

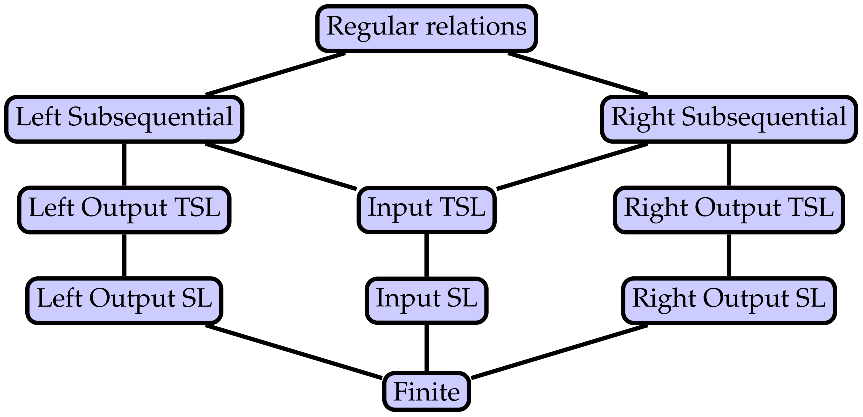

Figure 1: Subregular hierarchy of functions

Why is this useful? As we noted on Day 1: one reason is typology. The vast majority of phonological processes are subsequential (or below).[4] Non-subsequential functions appear to be the exception, not the rule, and these categories give us a way of formally characterizing the nature of that exceptionality.

[4]Of course, we could have a discussion about how we're quantifying 'majority' here.

Put another way, even though non-subsequential phonological maps exist, subsequentiality is still a useful basis from which to study the nature of phonological computation. Consider the following quotation from Howard (1972), defending his theory of directional rules even though it left some attested patterns without a satisfactory account:

> "We must understand that the theories we offer are merely steps toward the right answer and that each theory must be judged in comparison with alternative theories in terms of their ability to deal with the knowledge currently available. Most importantly, we must regard a theory as a research tool. By attempting to force what we know and what we believe to be true into a single logical framework we become more aware of the pieces that don't fit in, of the internal inconsistencies, and of problems that remain unresolved" (Howard, 1972, 2-3).

Another reason subregularity matters is learning. The regular relations are not formally learnable from only positive data. But everything below them in the hierarchy *can* be learned in this way. More on that in our final class!

5 Next time

- **Reading:** Nowak et al. (2002)
- **Task:** Come up with three distinct meanings of the word 'learning'.
- Also, we plan/hope to reserve some time at the end of the last class for open discussion, so feel free to prepare some clarification questions or topics of interest that we haven't been able to cover.

References

Howard, I. (1972). *A directional theory of rule application in phonology*. PhD thesis, MIT.

Hyman, L. (2011). Tone: Is it different? In Goldsmith, J. A., Riggle, J., and Yu, A. C. L., editors, *The Blackwell Handbook of Phonological Theory*, pages 197–238. Wiley-Blackwell.

Hyman, L. and Katamba, F. X. (2010). Tone, syntax and prosodic domains in Luganda. In Downing, L., Rialland, A., Beltzung, J.-M., Manus, S., Patin, C., and Riedel, K., editors, *Papers from the Workshop on Bantu Relative Clauses*, volume 53 of *ZAS Papers in Linguistics*, pages 69–98. ZAS Berlin.

Johnson, C. (1972). *Formal Aspects of Phonological Description*. Mouton, The Hague.

Kaplan, R. and Kay, M. (1981). Phonological rules and finite state transducers. Paper presented at ACL/ LSA Conference, New York.

Kaplan, R. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20:371–387.

Kisseberth, C. and Odden, D. (2003). Tone. In Nurse, D. and Philippson, G., editors, *The Bantu Languages*. New York: Routledge.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:269–311.

Nowak, M. A., Komarova, N. L., and Niyogi, P. (2002). Computational and evolutionary aspects of language. *Nature*, 417:611–617.