Lecture 7

Non-Local Dependencies

Our project of devising a computational model of phonological processes has made great progress. We started out with phonology as a list of well-formed surface forms, but quickly realized that this cannot account for essential properties like linguistic creativity or that there are strong constraints on what is a possible phonological system in natural language. So we moved to a more abstract perspective where phonology is still a list, but rather than full surface forms we list the n-grams that may occur in a string. A string's well-formedness is no longer an idiosyncratic property determined by a single list lookup — it now is contingent on the well-formedness of its parts.

We saw that this perspective equates phonological systems with strictly local languages, which allows us to explore formal properties of phonology through the study of the class of strictly local languages. This class seems to be a good approximation of local processes: every local phonological process is captured by some strictly local language, and the class exhibits closure properties that replicate very basic typological facts. Given a UG-specified restriction on the size of locality domains, we even get a positive learnability result under highly adversarial circumstances. And a natural algebraic generalization in terms of monoids revealed that many of these insights do not hinge on a categorical understanding of well-formedness, they also apply in a probabilistic setting.

But linguists have discovered a lot of evidence that there is more to phonology than just local processes. If strictly local languages do not encompass all of phonology, then the properties we have discovered so far characterize only a subpart of phonology. The question is whether there are indeed aspects of phonology that are not strictly local, and if so, how we could revise our model in an empirically adequate fashion while maintaining the majority of the insights that we have worked so hard for.

1 Long-Distance Dependencies in Phonology

1.1 Navajo Sibilant Harmony is not Strictly Local

While many phonological processes have clearly delineated application domains of bounded size — e.g. two adjacent segments, within a syllable or across a single word boundary — some processes seem to apply across arbitrary distances. Navajo, for instance, exhibits *sibilant harmony*, where all sibilants in a word must match the anteriority of the right-most one. The sibilants in the Navajo phoneme inventory are specified for anteriority as follows (Martin 2005:9):

[+anterior]	[−anterior]
s	ʃ
z	ʒ
ts^h	$tʃ^h$
ts	tʃ
ts’	tʃ’

The presentation here abstracts away from certain complicating factors. In particular, the likelihood of sibilant harmony decreases with distance, but crucially sibilant harmony never becomes ungrammatical.

Sibilant harmony applies in a variety of cases in Navajo, but it is most apparent in compounds (Martin 2005:10; the data is presented using a mixture of Navaja orthography and IPA).

xoʃ	‘cactus’	tʃaa	‘ear’
ts’óóz	‘slender’	nééz	‘long’
xosts’óóz	a specific type of cactus	tsaanéez	‘mule’

Since there is no upper bound on how many words a compound may consist of, and since the evidence suggests sibilant harmony can hold across an unbounded number of words within a compound, we are dealing with a truly unbounded process. We can express this as a formal theorem.

Theorem 7.1. Sibilant harmony is not strictly local. ⌟

Proof. This could be proved by exhibiting a productive pattern of Navajo compounding such that two sibilants with distinct anteriority values can be arbitrarily far apart from each other. Instead, we first convert sibilant harmony into a more abstract string language, and we then show that this string language is not strictly k-local for any choice of k.

Let l be a relabeling that replaces anterior sibilants by a, non-anterior sibilants by b, and all other segments by c. Then sibilant harmony is satisfied in all strings over alphabet $\{a, b, c\}$, and only those, that do not contain both a and b. Let L be the set of all such strings. Then L contains strings $s_{a,i} := ac^i a$ and $s_{b,i} := bc^i b$. Suppose L is strictly k-local. By k-local suffix substitution closure, $s_{a,k} \in L$ and $s_{b,k} \in L$ jointly imply $ac^k b \in L$. But this string violates sibilant harmony and thus cannot be a member of L, showing that L is not strictly k-local. Since k was arbitrary, L is not strictly local. □

Note that the proof starts out with a relabeling. At an earlier point we proved that strictly local languages are not closed under relabelings, so strictly speaking this proof cannot establish that sibilant harmony is not strictly local. Rather, it shows that sibilant harmony is not strictly local if one assumes that segments that aren’t sibilants contribute no useful information for determining well-formedness with respect to sibilant harmony.

From a linguistic perspective, this means that any tricks that might allow us to decompose the long-distance dependency into a local one are banned unless they have a visible effect on the surface form. For example, we cannot use anything like “invisible spreading” where the right sibilant has some feature that invisibly spreads to the adjacent segment to the left, and from there to the next one, and so one, until it reaches another sibilant and changes the value of anteriority feature. This is important to keep in mind: our claims about the complexity of various phonological processes are stated with the shared understanding that no hidden structure or distinctions can be invoked. Such hidden structures might exist, of course, but positing them blurs important boundaries and ties our claims to a very specific theoretical interpretation of

the data — one that might easily be false. Before we try to unify distinct phenomena by recourse to more elaborate structures, we should try to discern to which degree these phenomena are actually distinct.

1.2 Strictly Piecewise Languages

Now that we have encountered a process that is not strictly local, we have to start looking for a model that is sufficiently powerful but not too different from strictly local languages and grammars. Let us take another gander at the proof above — determining which property of strictly local languages was used to establish their inadequacy might give us an idea what needs to be changed.

Adopting for a moment the perspective of a strictly k-local scanner, we quickly see the problem. In order to regulate the dependency between a and b, both have to be within the scanner's search window, but that is impossible whenever the distance between the two exceeds the size of the search window. But what if our search window wasn't in one solid piece, but could be broken up into smaller parts that move around independently? That is to say, what if a strictly k-local scanner didn't have 1 window of size k, but k windows of size 1 so that we could inspect the string not in contiguous blocks but rather *piece by piece*?

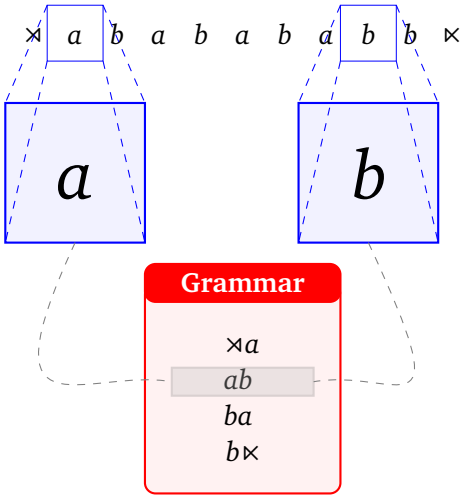

Figure 7.1: A scanner with multiple search windows

Suppose that we have such a scanner, with 2 windows of size 1. Then a string satisfies sibilant harmony iff it can never be the case that one of the windows is on an anterior sibilant a and the other window on a non-anterior sibilant b. And this is the case iff the string contains no *subsequence* ab.

Definition 7.2 (Subsequence). Given strings u and $v := v_1 \cdots v_m$, v is a *subsequence* of u iff every v_i occurs in u and for all $1 \leq i, j \leq m$, if $i < j$ then v_i precedes v_j in u (but they need not be adjacent). If v has length $k \geq 0$, it is a *k-subsequence*. The set of all k-subsequences of u is denoted k-seqs(u).

Example 7.1 Subsequences and Najavo Sibilant Harmony

The word xoʃ has the subsequences xo, oʃ, and xʃ, as well as x, o, ʃ, and xoʃ. Note that every substring is also a subsequence, but not the other way round. The 2-subsequences of tsaanéézare listed below:

ts	sa	aa	né	éé
ta	sn	an	nz	éz
tn	sé	aé		
té	sz	az		
tz				

Remember that strictly k-local grammars are defined in such a way that a string is well-formed iff every substring of length k is contained in the grammar. It seems that in order to account for long distance processes like Navajo sibiliant harmony, we have to look at subsequences instead. So all we have to do is make a minor change to the definition of strictly local grammars.

Definition 7.3 (Strictly Piecewise). A *strictly k-piecewise grammar* is a finite set of k-grams. A positive strictly k-piecewise grammar ${}^{+}G$ generates the language $L(G) := \{w \mid k\text{-seqs}(w) \subseteq G\}$. A negative strictly k-piecewise grammar ${}^{-}G$ generates the language $L(G) := \{w \mid k\text{-seqs}(w) \cap G = \emptyset\}$. A language L is *strictly k-piecewise* iff it is generated by some strictly k-piecewise grammar. The class SP of *strictly piecewise languages* is given by $\bigcup_{k\geq 1}\{L \mid L \text{ is strictly } k\text{-piecewise}\}$.

As you can see the definition for strictly piecewise is almost exactly the one for strictly local, except that the definition of the generated language use k-seqs(w) instead of k-grams(w).

Example 7.2 A Negative Strictly 2-Piecewise Grammar for Sibilant Harmony

Navajo sibilant harmony is violated whenever a string contains a 2-subsequence consisting of sibilants with opposite anteriority features. Since there's only finitely many sibilants, we can list all forbidden 2-subsequences (for the sake of brevity let's assume that affricates count as two segments rather than one).

sʃ	sʒ	$s\int^h$		ʃs	ʒs	$\int^h s$
zʃ	zʒ	$z\int^h$		ʃz	ʒz	$\int^h z$
$s^h\int$	$s^h\text{ʒ}$	$s^h\int^h$		$\int s^h$	$\text{ʒ}s^h$	$\int^h s^h$

A negative strictly 2-piecewise grammar with these 2-subsequences will never generate a string that violates sibilant harmony.

1.3 Monoid-Based Implementation

In the previous chapter we generalized the bigram scanner for strictly 2-local languages to a monoid-based scanner that can carry out a variety of tasks besides acting as a recognizer. The code is repeated here for your convenience:

```python
def monoid_scanner(base_value, compose, grammar, w):
    """
    Generalized bigram scanner that assigns strings
    a value over a given monoid.

    Arguments:
    base_value -- maps a bigram to an element of the monoid
    compose    -- implements the monoid operation
    grammar    -- set of licit bigrams
    w          -- the input string
    """
    # [Base Step]
    # value of a single bigram
    if len(w) == 2:
        return base_value(w, grammar)
    # [Recursion]
    # scan string from left to right and compose values
    else:
        return compose(
            base_value(w[0:2], grammar),
            monoid_scanner(base_value, compose, grammar, w[1:len(w)]))
```

With a little bit of ingenuity, the code can also be used to construct a bigram scanner for strictly piecewise languages. The idea is that the base value function returns two values, a set with the currently read bigram and a boolean indicating whether this subsequence is licensed by the grammar. The composition function then uses this information to construct a list of all seen subsequences and checks for each newly constructed subsequence that it is licensed by the grammar. So in a sense we actually have two monoids operating in a synchronized fashion: one for computing sets of seen subsequences, and the other one for computing string well-formedness based on these sets. The full code for strictly 2-piecewise languages is given in Listing 7.1 on the following page.

1.4 Properties of Strictly Piecewise Languages

Since strictly piecewise languages are the result of coupling the format of strictly local grammars with a different method for determining string well-formedness — precedence instead of adjacency — all the proofs that rely only on the grammar format carry over without modifications. This implies immediately that we can freely convert between positive and negative SP grammars, and that SP_k is learnable in the limit from positive text for every k. Other properties require only a little bit of extra work. The proof for closure under intersection is easily adopted, as is the proof for non-closure under union (and thus relative complement). The argument that all finite languages are strictly local also establishes that they are strictly piecewise, which furthermore entails that SP is not learnable in the limit from positive text.

The strictly piecewise languages thus retain all essential properties of the strictly local languages, which isn't too surprising seeing how they are both n-gram formalisms. Their properties line up very well with what we know about phonology — they do not give a full characterization of phonology, but what they do claim seems to be along the right track. It is very tempting to surmise that natural language phonology is somewhere in the union of SL_j and SP_k for some UG-specified values of j and k. In other words, phonology can be factored into a finite number of local and non-local processes that each correspond to some strictly local or strictly piecewise language

```python
def piecewise_base(w, grammar):
    """
    Check that input bigram is in the grammar and return:

    1) set of seen bigrams (= just the input),
    2) boolean indicating well-formedness, and
    3) the grammar so that it can be referenced by the compose function
    """
    return set([w]), w in grammar, grammar


def piecewise_compose(s, t):
    """
    Each argument is a triple of values:

    first component  -- set of 2-subsequences
    second component -- boolean indicating well-formedness
    third component  -- 2-gram grammar for which well-formedness is tested

    Based on this argument, the function computes:

    1) the set of all 2-subsequences that have been seen so far,
    2) the compound boolean.
    """

    # check that nothing went wrong with the grammar and give it a name
    if s[2] == t[2]:
        grammar = s[2]
    else:
        raise Exception("grammars of arguments differ")

    # retrieve values
    set_s = s[0]
    set_t = t[0]
    boolean_s = s[1]
    boolean_t = t[1]

    # don't do anything if we already have an ill-formed subsequence
    if (boolean_s is False) or (boolean_t is False):
        return s[0], False, s[2]
    # otherwise proceed with subsequence construction
    else:
        checked_bigrams = set_s.union(set_t)
        new_bigrams = set([])

        for i in checked_bigrams:
            for j in checked_bigrams:
                subsequence = i[0] + j[1]
                if subsequence not in checked_bigrams:
                    if subsequence in grammar:
                        # found a new licit subsequence
                        new_bigrams.add(subsequence)
                    else:
                        # found a new illicit subsequence
                        return s[0], False, s[2]
        return checked_bigrams.union(new_bigrams), True, grammar
```

Listing 7.1: Monoid functions for a strictly piecewise scanner

(closure under intersection ensures that these can all be combined into two systems of local and non-local processes, respectively).

Jeffrey Heinz has advocated this position in a series of papers (Heinz 2010; Heinz and Idsardi 2011), but only for segmental phonology. Suprasegmental phenomena, in particular stress, are exempt from this claim. And this is a good thing since some suprasegmental phenomena fall outside the purview of SL and SP.

Jeffrey Heinz

2 Suprasegmental Phonology

2.1 Unbounded Stress

One phenomenon we haven't considered so far is stress assignment. Every language comes with a fixed set of rules that determine which syllable in a word gets primary stress and is thus prosodically most prominent. Sometimes primary stress assignment can even resolve lexical ambiguities, such as in English with the noun *'export* and verb *ex'port*.

Most languages use rather simple stress assignment rules, such as *stress the first syllable* (which we will call **[0]** following Python's indexation system) or *stress the antepenultimate* (abbreviated **[-3]**). These simple systems are strictly local. Consider **[0]**, and assume that our alphabet consists of s and u for stressed and unstressed syllables, respectively. Then the set of (non-empty) strings that are well-formed with respect to **[0]** is exactly the language of the positive strictly 2-local grammar $\{\rtimes s, su, uu, u\ltimes\}$. For **[-3]**, we need a strictly 4-local grammar such that $suu\ltimes$ is the only 4-gram containing $\ltimes$ and no other 4-gram starts with s. Since there is an upper bound on the length of syllables (probably around 9, since some languages have very complex syllables like CCCVVCCCC) these grammars can be generalized to our standard string models where each segment represents a sound rather than a syllable. The values for k will be high (18 and 36, respectively), but there is a finite upper bound that renders these stress patterns strictly local.

The high values for k might actually indicate that phonological structures have two levels, one for segmental phonology and one for suprasemental phonology, with local processes on each level limited to some low k. This would explain why stress patterns require a high k over segmental structures while all other local segmental processes operate within considerably smaller domains.

But there are some stress patterns where the position of primary stress is more flexible and varies with the structure of the word. Kwakwala, for instance, exhibits what Hayes (1995) calls a *Leftmost Heavy Otherwise Right* (LHOR) pattern:

LHOR Primary stress falls on the leftmost heavy syllable in a word, and if there are no heavy syllables, on the final syllable.

Suppose that this pattern were strictly 2-local, given an alphabet of H and L for unstressed heavy and non-heavy syllables, respectively, and Ĥ and Ĺ for their stressed counterparts. Then the corresponding positive SL_2 grammar must contain at least the bigrams Ĺ$\ltimes$ (stress falls on the final syllable) and ĤL (stress falls on a heavy syllable). But this grammar will also accept strings where stress falls on a heavy syllable that is not leftmost. Since there is no principled upper bound on the distance between two heavy syllables, or the distance between a heavy syllable and the left edge of a word, we cannot fix this issue by moving to an SL_k grammar. So we need to bring in aspects of SP. To this end, we combine the positive SL_2 grammar with a negative SP_2 grammar that consists only of the bigram HĤ. Now we correctly capture the fact that the only valid targets for primary stress are the final syllable and the leftmost heavy syllable.

But this is not enough to enforce LHOR: The SL_2 grammar and the SP_2 grammar accept strings where primary stress falls on both the leftmost heavy syllable and the

final syllable. In order to rule out this case, we add the bigrams ŕĤŕ and ŕĤĹ to the SP_2 grammar, so that a string can only contain one primary stress. Unfortunately this is still insufficient, because both grammars will accept strings containing no stressed syllable at all. Try as we might, there is no way around this due to how these grammars calculate well-formedness: if w is well-formed, then so is every string w' with k-grams$(w') \subseteq k$-grams(w). For any string w with a stressed syllable, the grammars also accept every string w' that can be built from those k-grams of w that mention no stressed syllable. So SL allows us to tie stress to absolute positions in the string, and SP can identify relative positions as well as put an upper bound on how many stressed syllables a word may contain, but neither can enforce a lower bound in the general case. Our grammars can *allow* specific configurations to occur in a string, but they cannot *force* them to occur if a local alternative is available.

2.2 Culminativity and Threshold Testability

The problematic property of stress patterns like LHOR is that every string has to contain exactly one primary stress, which Heinz (2014) calls *culminativity*. The union of SL and SP can ensure the presence of at most one primary stress but fails to enforce the "at least one" half of the requirement. What we need, then, is a way to enforce lower bounds like this.

The most obvious solution is to pair a grammar G with a function τ that associates with each k-gram g a threshold $\tau(g)$. This threshold establishes the minimum number of times that a k-gram must appear in a string.

Definition 7.4 (Strict Threshold Testability). A *strictly k-local/piecewise m-threshold testable grammar* is a pair $\langle G, \tau \rangle$ such that

- G is a positive strictly k-local/k-piecewise grammar, and
- $\tau : G \to [0, m]$ is a total function assigning to each k-gram of G a *threshold* less than m.

Given a string w, $|g|_w$ denotes the number of occurrences of k-gram g in w. The language generated by G is $L(G) := \{w \mid |g|_w \geq \tau(g)\}$. A language L is *strictly threshold testable* (STT) iff it is strictly k-local/piecewise m-threshold testable for some fixed k and m.

Assuming an abstract alphabet where we distinguish only stressed and unstressed syllables, culminativity is expressed by the strictly 1-local 1-threshold testable grammar $G := \{\langle s, 1\rangle, \langle u, 0\rangle\}$ (this is a more compact representation that defines G and τ at the same time). Since the 1-gram s has threshold 1, every string must contain at least one primary stress, whereas it need not contain any unstressed syllables. The negative strictly piecewise grammar from the previous section blocks all strings with more than 1 strings, so the intersection of the languages generated by these two grammars contains only strings with exactly one primary stress. So culminativity can be factored into a strictly piecewise grammar and a 1-local 1-threshold testable grammar — in other words, it requires merely the smallest conceivable extension of the strictly local and strictly piecewise languages.

A problem arises, though, if we want both grammars to use the same alphabet and thus distinguish between stressed and unstressed heavy and non-heavy syllables. In

this case G needs to be expanded into $G := \{\langle \acute{H}, 1\rangle, \langle \acute{L}, 1\rangle, \langle H, 0\rangle, \langle L, 0\rangle\}$. However, this grammar force every string to contain at least one stressed heavy syllable and at least one stressed non-heavy syllable. This is clearly not what we want; there isn't even a single string that can satisfy this requirement and at the same time be generated by the strictly piecewise grammar. Culminativity thus is strictly threshold testable, but only over an impoverished alphabet. This is, in a certain sense, the inverse of previous cases where certain languages turned out to be strictly local only if one uses a refined alphabet that provides hidden information. Nonetheless it shows once more that the choice of alphabet is an important factor for determining the complexity of a process or constraint.

Another debatable assumption is that the application domain of phonology is a single word. This is clearly not the case for segmental processes, which can apply across word boundaries. The default assumption is that suprasegmental processes thus should not be limited to a single word, either. But that means that the input may include, among other things, a sequence of two mono-syllabic words, both of which must carry primary stress.The string could thus contain, say, $\rtimes\acute{H}\ltimes\rtimes\acute{L}\ltimes$. This is blocked by the SP_2 grammar's bigram $\acute{H}\acute{L}$ but can be remedied by moving to a positive SP_4 grammar whose 4-grams may contain two stressed syllabes only if they are separated by $\ltimes\rtimes$. More problematically, the ill-formed $\rtimes\acute{H}\ltimes\rtimes L\ltimes$is predicted to be grammatical — neither the SP_4 grammar nor the strictly threshold testable grammar block it.

Our approach to culminativity thus hinges on specific assumptions about the alphabet and the domain within which suprasegmental phonological processes apply. Following our standard methodology of conservatively adding new mechanisms and testing their limits we could carefully map out a space of increasingly more powerful formalisms and see how culminativity fits into these classes depending on our assumptions. This would reveal that culminativity over the expanded alphabet requires the power of so-called *locally testable grammars* (a generalization of strictly local 1-testable grammars), while extending the domain beyond words means moving to the much more powerful class of *star-free grammars*. Instead of exploring all these subtly different formalisms, we will turn our attention to a more principled investigation of the role of hidden structure as it is commonly entertained by phonologists. In particular, how much power can we obtain by distinguishing identical phones via a hidden alphabet?