Formal Language Theory and Phonology: Day 1

Jane Chandlee and Adam Jardine

July 3, 2023

1 Why formal language theory?

Key takeaways from (Heinz, 2018)):

Heinz, J. (2018). The computational nature of phonological generalizations. In Hyman, L. and Plank, F., editors, *Phonological Typology*, Phonetics and Phonology, chapter 5, pages 126–195. De Gruyter Mouton

There are important computational generalizations about phonological patterns that are missed by existing theories of the phonological grammar.

The goal is to characterize phonology in a way that is both sufficiently *expressive* to account for cross-linguistic variation while also being sufficiently *restrictive* to distinguish 'phonologically-possible' from 'logically-possible' patterns. In addition, we want these characterizations to help us understand how phonology is learned.

FLT offers an approach to these goals that is 'about as atheoretical as one can get', because it explicitly distinguishes the **intensional description** of a pattern from its **extension**.

intensional description
extension

Given a phonological pattern (e.g., a phonotactic constraint, a process, even an entire grammar), we *categorize* it based on its computational complexity. Theoretical computer science provides many such complexity categories, but we want to zero in on those that are sufficient for phonological patterns in particular.

Doing this provides 1) falsifiable hypotheses for what is and isn't possible in phonological systems, and 2) a theory-neutral understanding of what phonology *is* (i.e., universal properties).

- NOTE: This course will focus on the language-theoretic characterizations of formal languages and string-to-string mappings. There is a great deal of work on equivalent characterizations in other formalisms including finite-state automata, first order logic, and algebra. We won't have time to cover these, but a further reading list will be provided. Importantly, all of these characterizations converge to describe the *same* formal classes.

2 Basic concepts and notations

Let A be a **set** of objects, for example $A = \{1, 2, 3\}$. Then $\in$ indicates an object is in the set ($1 \in A$), and $\notin$ indicates an object is not in the set ($4 \notin A$). The unique **empty set** that includes no objects is $\emptyset$.

set

empty set

A **subset** D of a set E is a set whose objects are all also in E. For example, $B = \{1, 2\}$ is a subset of A (defined above) because $1 \in A$ and $2 \in A$. We write $B \subseteq A$. But $C = \{3, 4\}$ is not a subset of A, because $4 \notin A$. So we write $C \nsubseteq A$.

subset

We use Σ to designate an **alphabet**, or a set of symbols. When working with phonology these symbols are typically segments and boundaries, e.g. Σ = {æ, ɑ, ə, ɛ, i, n, m, ŋ, ..., +, ...}, but for basic examples we often use lowercase letters starting from the beginning of the alphabet, e.g., $\Sigma = \{a, b, c\}$.

alphabet

A **string** is a finite[1] sequence of symbols from Σ. We typically use lowercase letters near the end of the alphabet as variables for strings, e.g., $z = abca$.

string

[1]Infinite strings are a thing, but not in this course!

The **length** of a string w is designated with $|w|$ and indicates the number of symbols, e.g., $|z| = 4$. This same notation is used for the number of objects in a set, so $|A| = 3$.

length

The unique **empty string** is represented with λ and is the string with no symbols, e.g., $|\lambda| = 0$.

empty string

The set of *all* strings of symbols from Σ of *any* length is Σ^* ('Sigma-star'). The set of all strings of symbols from Σ of length *at least one* is Σ^+. Note these are both *infinite* sets.

Σ^*

Σ^+

What string is in Σ^* but not Σ^+?

The set of all strings of length n is Σ^n and the set of all strings of lengths up to and including n is $\Sigma^{\leq n}$. Note these are both finite sets.

If $\Sigma = \{a, b\}$, what is Σ^3? What is $\Sigma^{\leq 2}$?

We use a **concatenation** operator $\cdot$ to combine strings. So if $x = ab$ and $y = cd$, $x \cdot y = abcd$ and $y \cdot x = cdab$. But this operator is sometimes omitted, in which case we use xy and yx to mean the same thing. **concatenation**

We can also use concatenation to break a string into substrings. Consider the string $w = abcd$. This string can be see as the concatenation of smaller strings, for example, $a \cdot bc \cdot d$. In fact we can break this string apart in many ways:

	x	y	z
1	λ	a	bcd
2	λ	ab	cd
3	λ	abc	d
4	λ	abcd	λ
5	a	bcd	λ
6	a	bc	d
7	a	b	cd
8	ab	cd	λ
9	ab	c	d
10	abc	d	λ
11	abcd	λ	λ

Formally, a **substring** of a string w is any string y such that there exist (possibly empty) strings x and z such that $w = x \cdot y \cdot z$. **substring**

A **prefix** of a string is a substring that starts from the beginning. So looking at just the y column of the above table, rows 1, 2, 3, 4, and 11 include the prefixes of w. Similarly, a **suffix** of a string is a substring that includes the end. In rows 4, 5, 8, 10, and 11, y is a suffix of w. **prefix** **suffix**

Note that these terms have no morphological meaning. Also note that the term 'substring' does not necessarily mean shorter than the original string. As defined here a given string is a substring, prefix, and suffix of itself.

If $z = welcome!$, what are the prefixes of z? What are the suffixes?

Lastly, in practice we often want to explicitly refer to the start and end of a string. We use the symbols $\rtimes$ ('open fish'; start of string) and $\ltimes$ ('close fish'; end of string) for this purpose. These symbols are not included in the alphabet, but rather augment strings created from the alphabet. Formally we represent this with $\{\rtimes\} \cdot \Sigma^* \cdot \{\ltimes\}$. (Note the use of the concatenation operator with sets.)

3 Languages, language classes, grammars

A **language**, L, is a set of strings (we might actually use the terms *language* and *stringset* interchangeably). Formally, $L \subseteq \Sigma^*$.

language

Because a set is a subset of itself, Σ^* is itself a language. It can be thought of as an *unrestricted* language, since it includes any combination of alphabet symbols of any length. In practice the languages we'll be working with do place restrictions on strings, so they are **proper subsets** of Σ^*.

proper subsets

For example, we might define a language that only includes strings of even length:

$$L_e = \{w \in \Sigma^* : |w| \texttt{ is even}\}$$

As linguists we also generally assume the languages we work with are infinite. But infinite objects are kind of difficult to look at, which is why we also assume the existence of a **grammar**. A grammar is a finite representation of the language.

grammar

Grammars are both **recognizers** and **generators**. They recognize or distinguish strings that are in the language from those that are not, and they also generate all and only the strings in the language. Any device that performs these tasks correctly is a grammar for the language. What that device looks like and how it performs this work is essentially a choice of implementation.

recognizers
generators

As noted above, in this class we are going to be focused more on languages than grammars, but we still assume a grammar exists. This is important, because not all languages have grammars! So as a starting point, we can use this fact to define a **class of languages**, $\mathcal{L}$.

class of languages

$$\mathcal{L} = \{L : L \texttt{ is a language and L has a grammar}\}$$

This language class has a name: **computably enumerable**.[2] Its definition includes a restriction, but this restriction is quite weak when it comes to natural languages in general and phonology in particular. So from here we're going to 'zoom in' more and define further restrictions.

computably enumerable

[2]It's also called recursively enumerable.

A couple of additional language classes that we're going to skip over are the **context-sensitive** languages and the **context-free** languages. Both of these are included in $\mathcal{L}$ defined above, but they include additional restrictions.

context-sensitive
context-free

The important thing is that when we study individual phonological patterns (e.g., final devoicing in German, sibilant harmony in Navajo, etc.), we represent them as languages. But more broadly we want to know the *class* of languages these patterns belong to.

When it comes to phonology, the class of **regular languages** is an important one, so we'll start there.

regular languages

4 What are the regular languages?

We will give a **language-theoretic characterization** of the regular languages. A language-theoretic characterization is *abstract*, meaning that it is either true or not true *of a language itself*, and not tied to any particular grammar formalism.

language-theoretic characterization

We'll use the language $(ab)^n$ to illustrate:

$(ab)^n$

$$(ab)^n = \{\lambda, ab, abab, ababab, abababab, ...\}$$

Some more notation:

- Two strings w and v are **equivalent with respect to L** if and only if, for *all* suffixes u,

  equivalent with respect to L

  $$wu \in L \text{ if and only if } vu \in L$$

  We write $w \equiv_L v$ if w and v are equivalent with respect to L.

- For each string w below, what are some strings v such that $w \equiv_{(ab)^n} v$?

  ab λ a bb

- Now consider are $a^n b^n$:

  $a^n b^n$

  $$(ab)^n = \{\lambda, ab, aabb, aaabbb, aaaabbbb, ...\}$$

  For each string w below, what are some strings v such that $w \equiv_{a^n b^n} v$?

  ab λ a aa $aaab$ $aabbb$

- Think about *sets* of strings that are equivalent under with respect to each language. How do these sets differ for $(ab)^n$ and $a^n b^n$?

The **Myhill-Nerode theorem** (Nerode, 1958; Rabin and Scott, 1959) is a language-theoretic characterization of the regular languages:

Myhill-Nerode theorem

Nerode, A. (1958). Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544

Rabin, M. O. and Scott, D. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125

Theorem 1 (The Myhill-Nerode theorem) *A language L is regular if and only if $\equiv_L$ categorizes (more technically,* partitions*) all strings in Σ^* into a* finite *number of nonempty categories such that for any two strings w and v in the same category, $w \equiv_L v$.*

Note that this is true for $(ab)^n$ but *not* for $a^n b^n$.

This is a deep, fundamental property of the regular languages. Many specific grammar formalisms capture exactly the regular languages:

- right-linear (Type III) grammars (Chomsky, 1956)
- regular expressions (Kleene, 1956)
- finite-state machines (Kleene, 1956)
- monadic second-order logic (Büchi, 1960; Trakhtenbrot, 1961)
- ...

However, the Myhill-Nerode Theorem shows that they all somehow capture the same deep, structural property of the regular languages.

5 Next time

Readings: Rogers and Pullum (2011) and Heinz (2010)

Task: Given the language $a^n b^n$, how many columns do we need to place every string in Σ^* in a column of strings that are equivalent to each other with respect to $a^n b^n$? (This is essentially the first bullet point at the top of this page.)

References

Büchi, J. R. (1960). Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathmatik*, 6:66—92.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, pages 113–124.

Heinz, J. (2010). Learning long-distance phonotactics. *Linguistic Inquiry*, 41(4):623–661.

Heinz, J. (2018). The computational nature of phonological generalizations. In Hyman, L. and Plank, F., editors, *Phonological Typology*, Phonetics and Phonology, chapter 5, pages 126–195. De Gruyter Mouton.

Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–42. Princeton: Princeton University Press.

Nerode, A. (1958). Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544.

Rabin, M. O. and Scott, D. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125.

Rogers, J. and Pullum, G. (2011). Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, (20):329–342.

Trakhtenbrot, B. A. (1961). Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 140:326–329.