Lecture 10

Beyond Phonology

At the very beginning of this course, we split language into the subareas phonetics, phonology, morphology, syntax, semantics, and pragmatics. Now that we have concluded our investigation of phonology, it is time to move on to a new area. We will start with morphology, which is very similar to phonology on a computational level, before moving on syntax, which will occupy us for the rest of the course.

1 Morphology

1.1 Morphology in Natural Language

Morphology is concerned with word formation, which can be further divided into two distinct subsystems. *Inflectional* morphology regulates the overt marking of agreement with other words in a sentence. For example, English has a very limited form of person and number agreement between the subject and the finite verb of a sentence.

(1) a. The man like**s** the children.
b. * The man like the children.
c. The men like the children.
d. * The men like**s** the children.

Other languages have a much richer system of inflectional morphology. In Icelandic, for example, adjectives agree with the noun they modify in number, gender, and case. In addition, they also agree in definiteness with the determiner. And Icelandic is still relatively tame from a typological perspective, thanks to its restriction to only two numbers, three person, three genders, and four cases. Other languages have over ten distinct genders, three numbers (singular-dual/paucal-plural), possibly four distinct persons, and a myriad of cases.

Derivational morphology consists of the rules for deriving new words from existing ones. This includes compounding — the adjective *smart* and the noun *phone* combine into the compound *smartphone*, which didn't exist until a few years ago — as well as changes in POS, such as turning the noun *sea lion* into the verb *to sea lion* (another neologism that refers to polite yet intrusive attempts to engage somebody in a debate). Derivational morphology thus provides a system for dynamically extending the lexicon of a language.

The term *to sea lion* was coined in 2014 following a popular Wondermark comic strip: http://wondermark.com/1k62/.

Note that the distinction between inflectional and derivational morphology has nothing to say about how these processes are marked. Number agreement in German, for instance, can be indicated via a suffix (*Frau* 'woman', *Frauen* 'women'), via a sound change (*Laden* 'store' and *Läden* 'stores'), a suffix and a sound change (*Haus* 'house', *Häuser* 'houses'), or neither (*Schlüssel* 'key' and 'keys'). Other languages may use prefixes, circumfixes, or infixes instead. The same goes for derivational morphology. English often uses a suffix (*quick* and *quickly*, *the haste* and *to hasten*), sometimes a shift in stress (*the export* and *to export*), and sometimes no marker at all (*the anger* and *to anger*). In addition, compounding has no overt marking in many languages beyond the adjacency of the words, while many languages use *reduplication* to create new words (Marshallese *kagir* 'belt' and *kagirgir* 'to wear a belt'). In sum, morphological processes can be realized overtly via affixation, compounding, reduplication, or some phonological process.

The interaction of morphology and phonology is symmetric, though, in the sense that morphological structure can also suspend or trigger phonological constrains and processes. For instance, consonant clusters can have greater complexity if they span a morpheme boundary. This can be seen in German, where /tstpR/ cannot occur in any monomorphemic word but is perfectly acceptable in the compound *Arztpraxis* 'doctor's office'. The interaction of morphology and phonology is also known as *morphophonology* or, slightly shortened, *morphonology*. The close interaction of those two domains, with morphology sometimes relying phonological processes and phonology being sensitive to morphological information, means that the two are often treated by one and the same mechanism in real-life applications.

1.2 Two-Level Morphology

Why is this a better strategy than simply listing every word with all its inflected forms?

Just like phonology, morphology has been analyzed in terms of rewrite rules for the largest part of recent history. To the best of my knowledge, morphologists instinctively followed the same ban against a rule rewriting its own output that guaranteed the regularity of phonology. This makes it very likely that morphology, or at least a large part of it, is regular, too. So the finite state methods we used for phonology can be used just as well for morphology, which makes it possible to have one big rewrite grammar that intersperses phonological and morphological rewriting to capture the interaction of the two. This collection of rewrite rules can then be converted into a single FST thanks to the closure of finite state transductions under composition.

Contrary to what one might expect, though, this is not quite the approach taken in most industrial applications. This is mostly for historical reasons. Composing a large number of transducers, many of which may have dozens of states, simply wasn't computationally feasible before the 90s. In addition, people had not figured out yet how such a transducer can be used efficiently for morphological analysis, rather than generation.

Suppose you have a phonological surface form *s* for a word that sounds like *wipeboard*. Then *s* could be simply a compound of *wipe* and *board*, or a compound of *white* and *board* where /t/ was replaced by /p/ due to the following /b/. Now if you take the FST for your rewrite grammar and construct its inverse, which maps surface forms to underlying forms, it can return either option as a possible underlying form (since the FST is non-deterministic, even getting this set of all possible underlying forms is not trivial). In order to determine the correct underlying form, you then have to lookup each form in the lexicon, where you will eventually find an entry for *white board* but none for *wipe board*. Keep in mind that a lexicon can contain hundreds of thousands of entries, so this search may take a long time unless one uses a hashtable,

Marker	**Process type**	**Example**	**Language**
no marking	derivational	*the anger — to anger*	
	inflectional	*I eat — you eat*	
		this sheep — these sheep	
prefix	derivational	*do — undo*	
	inflectional	*i-ausipu* (3SG-put.down)	Nuaulu
suffix	derivational	*do — doable*	
		quick — quickly	
	inflectional	*I eat — he eats*	
		dog — dogs	
infix	derivational	*unbelievable — un-fucking-believable*	English (marginal)
		Óscar — Osquítar (diminutive of 'Oscar')	Nicaraguan Spanish
	inflectional	*hafal* 'celebrate' — *h-t-afal* 'celebrated'	Arabic
		(person and gender affixes omitted)	
circumfix	derivational	*adil* 'fair' — *ke-adil-an* 'fairness'	Malay
	inflectional	*kauf-en* 'to buy' — *ge-kauf-t* 'bought.PSTPART'	German
sound change	derivational	*biegen* 'to bend' — *Bogen* 'bow, arch'	German (not productive)
	inflectional	*Laden* 'store' — *Läden* 'stores'	German
		woman — women	English (not productive)
prosodic change	derivational	*the export — to export*	
	inflectional	*standen* 'stand.PST' — *stünden* 'stand.SUBJ'	German
reduplication	derivational	*to like — to like-like* 'love'	
		"Rules, schmules!"	American English
		kagir 'belt' — *kagir-gir* 'to wear a belt'	Marshallese
	inflectional	*kanak* 'child' — *kanak kanak* 'children'	Indonesian

Table 10.1: Overview of morphological marking strategies

which wasn't feasible back then due to memory limitations.

Nowadays the solution is obvious: take the identity function over the lexicon and compose it with the FST for the grammar. If this composed FST is run in reverse, it only outputs items in the lexicon, so that no search is needed. But this simple trick wasn't worked out until the mid 90s, and instead *two-level morphology* was developed as a more practical tool for morphological analysis (Koskenniemi 1983).

From a formal perspective, two-level morphology is a very simple modification of the rewrite paradigm. Rather than applying rewrite rules sequentially, one after another, they are all applied in parallel. Consequently, the grammar isn't a cascade of FSTs but runs all FSTs in parallel instead. The overall transduction is the intersection of these FSTs. Recall that the intersection of arbitrary FSTs is not guaranteed to be an FST. However, the class of ε-free FSTs is closed under intersection, and two-level morphology exploits this fact by using a special symbol 0 to indicate unpronounced nodes. That way, deletion of a symbol σ amounts to relabeling it as 0, and insertion of σ is emulated by relabeling 0 as σ.

The process of pluralizing *index* to *indices*, for instance, is modeled as first composing the stem *index* and the plural marker +00, and this string *index*+00 is rewritten as *indic*0*es*, which is pronounced *indices*. Formally, the transduction contains the pair $\langle index+00, indic0es\rangle$, and since both strings have the same length, we are dealing with an equal length relation, which instead can be viewed as a string of pair symbols

$$\binom{i}{i}\binom{n}{n}\binom{d}{d}\binom{e}{i}\binom{x}{c}\binom{+}{0}\binom{0}{e}\binom{0}{s}$$

This shift in perspective turns the relation into a regular language, and closure under intersection holds. A rather inelegant trick, but it gets the job done.

As is implied by the name, two-level morphology posits only two levels, one for underlying forms, one for surface forms. The two forms are padded with zeros if necessary to ensure they have the same length, wherefore they can always be analyzed as a single string of pair symbols as above. The rewrite rules operate over such pair symbols $u : s$, where u is the underlying segment and s the surface segment. The rewrite rules are of the form $u : s \diamond \alpha\_\beta$, where α and β are strings over pair symbols (and may also include SPE-style notation like brackets for optionality and + for iteration). The symbol $\diamond$ is a placeholder for two different types of rewrite arrows: $\Rightarrow$ and $\Leftarrow$. If $\diamond$ is replaced by $\Rightarrow$, then the rule states $u : s$ can occur only in the specified context. This requirement is weakened with $\Leftarrow$, which states that if both $u : s$ occurs in the specified context and the surface form has an s, then the underlying form must have u. This is also called *surface coercion*. Sometimes $\Leftrightarrow$ is used as a combination of the two to express that $u : s$ occurs only in the given context and no distinct form $u' : s$ may occur there.

With a little bit of ingenuity, each rewrite rules can be converted into a regular language of pair symbols, and the whole grammar is simply the intersection of these regular languages. Analyzing a surface form is tantamount to finding a pair string whose second component matches the surface form, whereas generation instead looks for a pair string whose first component matches the desired surface form. Since a language of pair strings can be converted into an ε-free FST, this search is simply a matter of running the FST over the underlying form, or its reverse over the surface form.

In sum, two-level morphology may look very different from SPE or OT, yet it is just another way of defining finite state transductions. While it can generate all regular languages, just like SPE and OT, it is weaker than those two in the sense that it cannot handle deletion and insertion in an elegant way and must instead rely on padding out strings via the special symbol 0.

1.3 Complexity of Morphology

Two-level morphology has been successfully applied to a variety of typologically diverse languages, including English, Finnish, Turkish, and Japanese. Just like the overwhelming descriptive success of SPE is strong evidence that phonology is at most regular, the wide usage of two-level morphology suggests that morphology is regular, too.

This is actually not all that surprising, considering the local and finitely bounded nature of most morphological processes. In almost all cases it is sufficient to know the modified stem and which morphological process took place most recently. In addition, some morphemes can only be instantiated exactly once. All of these things fall under the purview of regular languages.

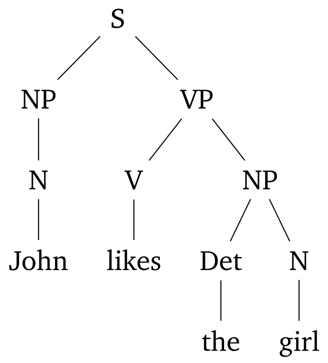
Only two aspects of morphology might be problematic. First, circumfixion requires the presence of both a prefix and a suffix of a specific type. This is illustrated by the German past participle, which consists of the stem of the verb and the circumfix *ge- -t* as in *ge-kauf-t* ‘bought’. If this process were unbounded, then German would contain words of the form *ge*^{n}*-kauf-t*^{n}, but not, say, *ge-ge-kauft-t-t-t*. In our discussion of the complexity of syntax later on we will see that such a pattern is not regular. Intuitively, that’s because an FST generating this pattern would have to first inserts *n* instances of *ge-*, followed by *n* instances of *-t*, but since the FST has only a finite number of states it can’t keep track of the exact number *n* past a certain threshold and may end up inserting too few or too many suffixes. As you might expect, though, German past participle formation is not an unbounded process (probably because additional circumfixes would serve no function). To the best of my knowledge, unbounded circumfixion is universally unattested, lending strong support to the hypothesis that morphology is regular and thus incapable of generating such patterns.

The only remaining challenge to the regularity hypothesis is reduplication. If there is no upper bound on the size of the reduplicant, we run into a similar memory problem for the FST as above: with *n* states, the FST can memorize only a fixed amount of information, so if the material that should be reduplicated is so long that it does not fully fit in the state memory, the FST cannot insert an exact copy. Reduplication data has proven very difficult to analyze, and at this point it is not clear whether there are any cases of unbounded reduplication where the reduplicant must be an exact copy. In the cases where reduplication seems to be unbounded, it usually interacts with phonology in various ways that make it difficult to discern what the morphological well-formedness criterion is. Putting aside reduplication, though, all of known morphology seems to fall within the class of finite state transductions. And just like in phonology, the overwhelming majority of processes do not come close to exploiting the full power FSTs.

2 Syntax

2.1 A Weak Argument That Syntax is Not Regular

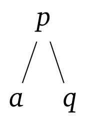
Syntax regulates the well-formedness of sentences, which includes basic factors like word-order but also much more arcane properties such as the distribution of so-called negative polarity items (*ever*) and anaphors (*himself*, *herself*, *itself*). It is usually believed to be more complex than phonology and morphology. For example, the basic data structure is assumed to be trees rather than strings, so that a simple sentence like *John likes Mary* actually involves a lot of hidden structure:



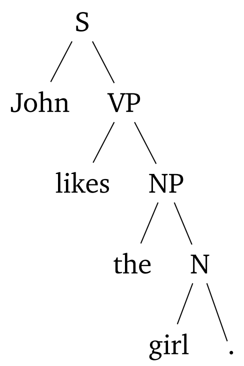
Trees of this form are generated by phrase structure grammars, which are finite sets of rewrite rules without context specifications. The tree above is generated by the grammar below:

S	→	NP VP	Det	→	the
NP	→	(Det) N	N	→	John | girl
VP	→	V NP	V	→	likes

At first sight it seems that this shift to trees — if it is indeed called for — renders our current tools useless because they generate strings, not trees. But this is not quite correct. Consider the transition $\langle p, a, q \rangle$ some FSA. We can represent this as a tree with p as the root and a and q as left and right sibling, respectively.



And every such tree corresponds to a rewrite rule $p \to a\ q$. So every FSA corresponds to a phrase structure grammar where every node has exactly two daughters, the first of which is a symbol from the alphabet whereas the second one is a state symbol. That is not enough to produce the tree above, but it allows us to generate a very similar version (the dot indicates the final state).

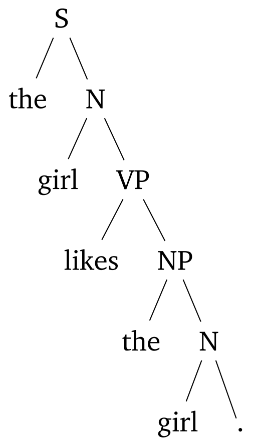


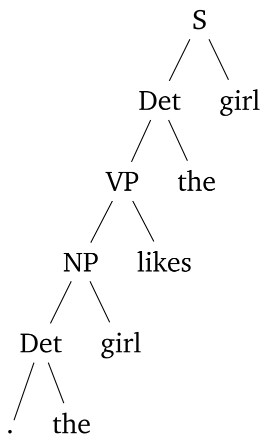
Such a tree is called *strictly right-branching*, and its mirror image would be *strictly left-branching*. This terminology also applies to phrase structure grammars that only generate trees of this kind.

Every FSA can also be converted into a grammar that generates strictly left-branching trees. What does this translation look like?

An argument that FSAs are insufficient for syntax cannot simply invoke the fact that syntax uses trees. Every FSA can be converted into a grammar that generates trees, albeit strictly right-/left-branching ones. Consequently, a tree-based argument has to show that these types of trees are unsuitable for natural language. That is a lot more difficult than one would expect.

For example, one may object that both the strictly right-branching and the strictly left-branching tree do not represent the true structure of *The girl likes the girl*.





But how does one determine what the true structure should be? In contrast to phonology, where we could check the generated strings against the observed surface forms, we have no direct access to the tree structure of a sentence, assuming it even exists. All we have is indirect evidence. In the case at hand, the two instances of *the girl* have different structures but seem to exhibit very similar behavior. For example, both can be coordinated with another noun phrase.

(2) a. The girl likes the girl.
  b. The girl and the boy like the girl.
  c. The girl likes the girl and the boy.
  d. The girl and the boy like the girl and the boy.

Linguists have an enormous battery of tests to probe the *constituency* of a sentence, i.e. how words combine into subtrees. It is important to keep in mind, though, that these tests are theory-laden, starting with the very basic assumption that if two substrings show similar behavior, this is indicative of a structural similarity between the subtrees. Often it is also implicitly assumed that constituency tests identify the *unique* structure of a substring, rather than one of several varieties. There are licit scientific assumptions, but they invariably weaken any results obtained this way — if the assumptions are rejected or shown faulty, then the array of findings that build on them, no matter how impressive, is at risk of collapsing like a house of cards. A proof that does not rely on any of these assumptions is more reliable and general and insightful, and thus to preferred.

For example, many debates about English possessive constructions focus on whether the possessor or the possessee is a sibling of the possessive marker *'s*, when the simplest account is simply that both options are licit.

Instead of focussing on how well regular methods may describe the tree structure of sentences, it is a lot simpler to verify whether the string languages are even regular. If the set of well-formed English sentences, understood as strings of words, is not regular, then obviously no FSA can generate this set and it follows immediately that FSAs cannot produce the necessary tree structures, either.

2.2 A String-Based Proof that Syntax is Not Regular

In our discussion of morphology it was mentioned that a pattern like *ge*^{n}-*kauft*-*t*^{n}, where n instances of *ge* must be followed by n instances of *t*, is not regular. While it is unclear that such a pattern is ever instantiated in morphology, it is actually fairly easy to construct for syntax.

(3) a. That John surprised me annoyed me.
    b. That that John surprised me surprised me annoyed me.
    ⋮
    c. That^n John (surprised me)^n annoyed me.

The pattern above relies on the fact that each *that* introduces a sentential subject, and since each one of them must have its finite verb, the number of occurrences of *that* must be exactly the number of finite verbs, *modulo* the finite verb in the highest clause. The sentence is fairly unnatural, but the basic idea can easily be generalized to other constructions.

(4) a. The cheese was rotten.
    b. The cheese that the mouse ate was rotten.
    c. The cheese that the mouse that the cat chased ate was rotten.
    d. The cheese that the mouse that the cat that the dog dislikes chased ate was rotten.
    ⋮
    e. The cheese (that the *noun*)^n (*transitive verb*)^n was rotten.

Quite generally, any kind of *center embedding construction* will give rise to to these kinds of number matching conditions $a^n b^n$.

We still have to proof, though, that $a^n b^n$ and related languages are not regular. The intuition that FSAs do not have enough memory homes in on the essential problem, but it does not constitute a proof. We will flash it out via a so-called *pumping lemma*. First, suppose that A is an FSA with k states that generates an infinite language $L(A)$. Then for any $w \in L(A)$ with $|w| > k$ it must be the case that two nodes m and n of w are assigned the same state q by A. But then A must contain a loop that leads from q back to q. More precisely, let $w[m : n]$ be the substring of w that spans from m to n. Then $w[m : n]$ describes a path through A from q to q. Clearly this path can be taken multiple times. Each instance of following this loop is called a *pump* p. Hence if $w := x \cdot w[m:n] \cdot y$ belongs to $L(A)$, so does every *pumped string* $x \cdot w[m:n]^p \cdot y$ for all $p \geq 1$.

Theorem 10.1 (Regular Pumping Lemma). If a language L is regular, then there exists an integer $k \geq 1$ such that every $w \in L$ of length at least k has a decomposition $w := x \cdot z \cdot y$, where

- $|z| \geq 1$,
- $|x| + |y| < k$,
- $xz^p y \in L$, $p \geq 1$. ⌟

Proof. If L is finite, the claim is trivial. For L infinite, it follows from the representation via FSAs as discussed above. □

Note that the pumping lemma is a necessary condition for a language to be regular, but not a sufficient one. That is to say, there are non-regular languages that satisfy the pumping lemma. Consequently, the pumping lemma cannot be used to prove that a language is regular, but failure of the pumping lemma does prove non-regularity.

With the pumping lemma, it is fairly easy to show that $a^n b^n$ is not regular. All we have to do is consider all the possible decompositions of $a^k b^k$ into xzy and show that we never end up with the right kind of pump, irrespective of the value for k. First, if x contains at least one b, then z can only contain bs and thus pumping z introduces new bs but not new as, so we got from $a^k b^k$ to $a^k b^m$, $m > n$, which is not in $a^n b^n$. An analogous argument holds if y contains at least one a. So z must be some string of the form $a^i b^j$. But then $z^2 = a^i b^j a^i b^j$, and no string with z^2 as a substring can be contained in $a^n b^n$. It follows that $a^n b^n$ is not regular.

Showing that $a^n b^n$ is not regular is not enough to demonstrate that English is not regular, though. We have to translate the English patterns into the formal language $a^n b^n$. To this end, we let τ be a finite state transduction that deletes the first two words *the cheese*, maps each instance of *that the noun* to a, each instance of a transitive verb to b, and deletes the last two words *was rotten*. This turns the language *the cheese (that the noun)*^{n} *(transitive verb)*^{n} *was rotten* into $a^n b^n$. As regular languages are closed under finite-state transductions, $a^n b^n$ is regular if the fragment of English is regular. Since the former is not, the latter is not either.

It is tempting to stop at this point and conclude that English is not regular, but this would be a grave (and unfortunately very common) mistake. Showing that a fragment of a language has a certain complexity does not imply that the whole language is of the same complexity. For example, Σ^* is strictly 1-local even though it contains the fragment $(aa)^+$, which is regular. The whole language can be much simpler than a given fragment. In the case at hand, English may actually be regular if we can also have sentences of the form *the cheese that the mouse was rotten* or *the cheese that the mouse chased ate was rotten*, where the number of NPs and transitive verbs no longer match. The crucial property — which is implicit in our claim that English center embedding is captured by the pattern $a^n b^n$ — is that all strings that deviate from this pattern in a specific way are ungrammatical (obviously there are strings that deviate in some other way yet are well-formed, e.g. *John likes Mary*). Directly defining this set of related yet ungrammatical strings is tricky, so often it is more efficient to use closure properties to extend the proof to the entire language.

In the case at hand, we construct a fragment of English that contains all relevant center embedding constructions via regular intersection. Evidently the language given by the pattern *the cheese (that the noun)* (transitive verb)* was rotten* is regular, and its intersection with English is the language *the cheese (that the noun)^n (transitive verb)^n was rotten*. But we already know that this language is not regular and as regular languages are closed under intersection, it follows that English cannot be regular. And since English is a natural language, we can finally conclude that natural language syntax is not adequately modeled by FSAs and FSTs.

2.3 Myhill-Nerode Characterization of Regular Languages

The pumping lemma has the disadvantage that it may yield false positives in the sense that some non-regular languages may satisfy it. A much stronger result is the *Myhill-Nerode theorem*, which gives a full characterization of the regular languages.

Suppose that A is a deterministic FSA with state set Q that recognizes L. Then we can identify with every state $q \in Q$ a set $tails(q)$ of *good tails* t such that t describes a path from q to a final state of A. In the same vein, we can identify the set $heads(q)$ of *good heads* that describe a path from the initial state of A to q. Concatenating a good head of q with one of its good tails produces a string of L. In fact, L is exactly $\bigcup_{q \in Q} heads(q) \cdot tails(q)$. The Myhill-Nerode theorem generalizes this idea so that it can be stated over strings without referencing automata or states.

Given two strings u and v, we write $u \equiv_L v$ iff u and v have the same good tails in L:

$$u \equiv_L v \Longleftrightarrow \{x \mid u \cdot x \in L\} = \{x \mid v \cdot x \in L\}$$

It is easy to see that $\equiv_L$ is an equivalence relation.

Background Equivalence relation

An equivalence relation is a binary relation R over a set S that satisfies three conditions:

reflexive for all $x \in S$, $x \, R \, x$

symmetry for all $x, y \in S$, $x \, R \, y$ implies $y \, R \, x$,

transitivity for all $x, y, z \in S$, $x \, R \, y$ and $y \, R \, z$ jointly imply $x \, R \, z$.

The equivalence relation partitions L into equivalence classes of strings that have the same good tails. But if these strings all have the same good tails, we can paraphrase this in FSA-terms: they all lead to the same state. So each equivalence class is the (possibly infinite) set of paths that lead to a state q. This basic insight is enough to show that a language is regular iff $\equiv_L$ induces a finite number of equivalence classes. One also says that $\equiv_L$ has *finite index*.

Definition 10.2 (Index). Let R be an equivalence relation over set S, and let $[a] = \{b \mid a \, R \, b\}$. The *index* of R is $|\{[a] \mid a \in S\}|$.

Theorem 10.3 (Myhill-Nerode). A string language L is regular iff the corresponding equivalence relation $\equiv_L$ has finite index. ⌟

Proof. If L is regular, then we have $u \equiv_L v$ if they lead to the same state in some deterministic FSA A. Hence, if A has k states, the index of $\equiv_L$ must be less than k and thus finite.

In the other direction, we use $\equiv_L$ to construct a deterministic FSA A. First, pick one string w from each equivalence class of $\equiv_L$. The state set of A contains q_w for every such w. We mark q_w as final iff $w \in L$, and q_w is initial iff $[w]$ contains ε. Finally, $\delta(q_w, a) = q_z$ iff there are $u \in [w]$ and $v \in [z]$ such that $v = u \cdot a$. Verifying that A is deterministic and recognizes L is left as an exercise to the reader. □

The Myhill-Nerode theorem furnishes another way of showing that $a^n b^n$ is not regular, one that is much closer to the original intuition that an FSA does not have enough memory for this pattern. For every string a^n, its set of good tails is given by $\left\{a^i b^j \mid i \leq 0, j = n + i\right\}$. Hence we have $a^m \equiv_L a^n$ iff $m = n$, i.e. each substring of as has its own equivalence class. Consequently, $\equiv_L$ cannot be of finite index.

So now we have yet another characterization of regular languages, but this one is particularly noteworthy because it completely abstracts away from grammars, automata and such devices to instead gives a direct characterization of regular languages. In that respect, it is very similar to our characterization of the strictly local languages via substring substitution closure.

Corollary 10.4. Let L be a string language. Then the following claims are equivalent:

1. L is generated by a refined strictly 2-local grammar,
2. L is a projection of a strictly 2-local language,
3. L is recognized by a finite-state automaton,
4. L is the output language of a finite-state transducer,
5. L is generated by a strictly right-/left-branching grammar,
6. $\equiv_L$ has finite index,
7. L is regular. ⌟