Lecture 7

The Myhill-Nerode Theorem

We have seen that regular languages can be characterized by regular expressions, DFAs, and NFAs. In this lecture we will show a fourth elegant characterization of regular languages, using what's known as the Myhill-Nerode theorem. This characterization will be useful not only for showing that languages are regular, but also for proving that languages are not regular.

Let $A \subseteq \Sigma^*$ be a language over Σ, and let $x, y \in \Sigma^*$ be two strings (not necessarily strings in A). We say that a string $z \in \Sigma^*$ is a *distinguishing extension* for x and y if one of xz and yz is in A and the other is not. That is, if we extend x and y by the string z, then they become different in terms of their membership in A.

Next, we say that two strings $x, y \in \Sigma^*$ are equivalent with respect to A, denoted $x \equiv_A y$, if there is no distinguishing extension between x and y. That is, two strings are equivalent with respect to A if no matter what string z you append to them, the strings xz and yz are either both in A or both not in A.

Observe that $x \equiv_A x$ and if $x \equiv_A y$ then $y \equiv_A x$. Also, if $x \equiv_A y$ and $y \equiv_A w$, then $x \equiv_A w$. That's because otherwise, there would be a distinguishing extension z for x and w, which would mean that exactly one of xz and wz are in A; but then regardless of whether yz is in A or not, it must be the case that z distinguishes y from either x or w, which contradicts our assumption that $x \equiv_A y$ and $y \equiv_A w$.

These three properties mean that $\equiv_A$ is what's called an *equivalence relation*. This equivalence relation splits the set Σ^* of all strings over Σ into different *equivalence classes*. That is, for any $x \in \Sigma^*$, the equivalence class of x with respect to A will be a the set of strings $C \subseteq \Sigma^*$ defined by $C = \{y \in \Sigma^* : x \equiv_A y\}$. This is the set of all strings equivalent to x under the relation $\equiv_A$.

Note that for any two strings $x, y \in \Sigma^*$, their equivalence classes with respect to A are either identical (if $x \equiv_A y$) or disjoint. That's because if the equivalence classes overlapped, there would be some string w which is a member of both, but then w would be equivalent to all strings in both classes, and hence all the strings in both classes would be equivalent to each other—making the classes equal, given the way we defined them. Also note that we do not consider the empty set to be an equivalence class; any equivalence class must contain at least one string, plus all the strings equivalent to that string with respect to A.

These equivalence classes with respect to a language A have a lot of nice properties:

1. By definition, all the equivalence classes are nonempty.

2. As we saw, any two equivalence classes with respect to A are either exactly equal or else disjoint.

3. The union of all the equivalence classes with respect to A is Σ^* (where Σ is the alphabet of A). This is because any string $x \in \Sigma^*$ is in some equivalence class—the class of all strings equivalent to x.

4. For any equivalence class C with respect to A, we have either $C \subseteq A$ or else $C \cap A = \varnothing$. This is because if $C \cap A \neq \varnothing$, then there is some $x \in C \cap A$; but then for any $y \in C$, we have $x \equiv_A y$, which means that there is no distinguishing extension between x and y, and in particular, $z = \epsilon$ is not such a distinguishing extension. Hence either both $xz = x$ and $yz = y$ in A or both are not in A; but since $x \in A$, we must have $y \in A$. Since $y \in C$ was arbitrary, we have $C \subseteq A$.

5. For any two equivalence classes C and D with respect to A, if there are strings $x \in C$ and $z \in \Sigma^*$ with $xz \in D$, then all strings $y \in C$ also have $yz \in D$. This is because otherwise, if $yz \notin D$ for some $y \in C$, it would mean yz is not equivalent to $xz \in D$, which means there is some distinguishing extension w, i.e. exactly one of yzw and xzw is in A. But in this case, the string zw is a distinguishing extension for x and y, which contradicts the assumption that $x, y \in C$.

Now that we have introduced the equivalence classes defined by a language A, we can state the Myhill-Nerode theorem, which gives a new characterization of regular languages in terms of the number of such classes.

Theorem 7.1. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language over Σ. Then A is regular if and only if there are finitely many equivalence classes with respect to A. Moreover, if A is regular, then the number of equivalence classes with respect to A is exactly the minimum number of states in a DFA for A.*

Why does the Myhill-Nerode theorem hold? Well, note that a DFA for A (if it exists) has only finitely many states. For each state q, consider the class of strings that cause the DFA to reach that state. Then these strings are all equivalent: if the DFA reaches q when run on both x and y, then no matter which string z you append to x and y, the DFA will reach the same state when run on xz and on yz. Therefore, they are either both in A (if the reached state is an accept state) or both not in A (if the reached state is not an accept state).

The intuition should therefore be that the each equivalence class with respect to a regular language A corresponds to a state of a DFA for A, with a string x belonging to the class if and only if the DFA reaches that state when run on x. Of course, there are many possible DFAs for a regular language; the equivalence classes turn out to correspond to the states of the *smallest* possible DFA for A, and the number of equivalence classes is the minimum number of states in a DFA for A. If A is not regular, this "minimum number of states" is infinity, because there is not finite number of states that suffices to represent A (as there is no DFA at all). This corresponds to there being infinitely many equivalence classes with respect to A.

We now prove the Myhill-Nerode theorem formally.

Proof. First, suppose that A is regular, and let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing A. For each $q \in Q$, let $C_q \subseteq \Sigma^*$ be the set of all strings x such that M reaches state q when run on x; that is, $C_q = \{x \in \Sigma^* : \delta^*(q_0, x) = q\}$. We claim that for each $q \in Q$ and each $x, y \in C_q$, we have $x \equiv_A y$. This is because for any $z \in \Sigma^*$, when M is run on xz it reaches $\delta^*(q_0, xz) = \delta^*(q, z)$, and when M is run on yz it also reaches $\delta^*(q, z)$; thus if $\delta^*(q, z) \in F$ we have $xz, yz \in A$, and if $\delta^*(q, z) \notin F$ we have $xz, yz \notin A$. In both cases, the string z does not distinguish x and y, so they have no distinguishing extension and $x \equiv_A y$.

Now, we know that all the strings inside C_q are equivalent for each $q \in Q$. Also note that the sets C_q are all disjoint. Now, if E is an equivalence class with respect to A, then if E contains any string in C_q, it must contain all of C_q (since all strings in C_q are equivalent with respect to A). Hence E must be a union of some of the sets C_q. We conclude that the equivalence classes with respect to A are unions of the sets C_q, so there can only be finitely many of them (in fact, at most $|Q|$, since they must be disjoint from each other). Hence if A is regular, it has finitely many equivalence classes, and the number of such classes is at most the minimum number of states in any DFA for A (and hence at most the number of states in the minimum DFA for A).

In the other direction, suppose that A has finitely many equivalence classes. Let k be the number of such classes, and let $C_1, C_2, \ldots, C_k$ be the equivalence classes themselves, with $C_1 \cup C_2 \cup \cdots \cup C_k = \Sigma^*$ and $C_i \cap C_j = \varnothing$ whenever $i \neq j$. We construct a DFA M for A. It will have one state p_i for each equivalence class C_i, so that its set of states $Q = \{p_1, p_2 \ldots, p_k\}$ has size k. The alphabet of M will be Σ. Its start state q_0 will be the state p_ℓ corresponding to the equivalence class C_ℓ containing the empty string, that is, the set C_ℓ with $\epsilon \in C_\ell$.

We define the set of accept states to be $F = \{p_i : C_i \subseteq A\}$. Finally, we define the transition function δ. To do so, recall that for any $a \in \Sigma$, for any equivalence class C_i, and for any $x, y \in C_i$, the equivalence class of xa and ya is the same. Let this equivalence class (defined by a and C_i) be C_j. We then set $\delta(p_i, a) = p_j$. Note that this will mean that $\delta^*(q_0, x)$ will be the state corresponding to the equivalence class of x. We can prove this by induction; it is true by definition for $x = \epsilon$ (the base case), and if it holds for strings of length k and x is a string of length $k + 1$, then we can write $x = wa$ with $|w| = k$, so that $\delta^*(q_0, w)$ corresponds to the equivalence class of w, and then $\delta^*(q_0, x) = \delta(\delta^*(q_0, w), a)$ is the equivalence class of wa, completing the induction argument.

We now claim that this DFA recognizes A. To see this, first suppose that $x \in A$. Then $\delta^*(q_0, x) = p_i$ with $x \in C_i$, and since $x \in A$, we have $C_i \cap A \neq \varnothing$ which means $C_i \subseteq A$. Hence by the definition of F, we have $p_i \in F$, so the DFA accepts x. Conversely, suppose that x is accepted by the DFA. Then $\delta^*(q_0, x) \in F$, which means that the equivalence class C_i of x satisfies $C_i \subseteq A$. Since $x \in C_i$, we must have $x \in A$, as desired.

We conclude that A is regular if and only if it defines a finite number of equivalence classes, and if it is regular, the number of equivalence classes is the minimum number of states in a DFA for A. □

7.1 Using the Myhill-Nerode theorem

The Myhill-Nerode theorem gives us yet another characterization of when a language is regular. It can be used to show languages are regular, simply by listing out all the equivalence classes if there are only finitely many (and proving that this list is correct). However, in practice this turns out to be quite similar to just constructing a DFA for the language. Therefore, for the purpose of showing that a language is regular, the Myhill-Nerode theorem usually only gives a hint for how to build a DFA, but isn't fundamentally easier than constructing a DFA or NFA directly.

Interestingly, however, the Myhill-Nerode theorem can be used to show that languages are *not* regular. To show that A is not regular, we need to show it defines infinitely many equivalence classes. This is the same as showing that there are infinitely many strings, $x_1, x_2, x_3, \ldots$, such that no two of them are equivalent with respect to A. In other words, it suffices to find a sequence of strings $x_1, x_2, x_3, \cdots \in \Sigma^*$ such that for each $i \neq j$, we have some $z \in \Sigma^*$ with exactly one of $x_i z$ and $x_j z$ being in A. Note that the strings x_i do not themselves have to be in the language A; they can be arbitrary strings over Σ.

As an example, let's reprove that $A = \{0^k 1^k : k \in \mathbb{N}\}$ is not regular. We will pick a sequence of

strings, in this case $x_i = 0^i$ for all $i \in \mathbb{N}$. Then for any $i \neq j$, we can take $z = 1^i$, and then we'll have $x_i z \in A$ but $x_j z \notin A$. This is enough to conclude that A is not regular.

Example where the pumping lemma fails. Let's try the following example. Let $B = \{01^k2^k : k \in \mathbb{N}\} \cup L((\epsilon \cup 000^*)1^*2^*)$. Can we show that B is not regular?

It turns out that we cannot use the pumping lemma directly. The reason is that B actually satisfies the pumping lemma: there is a pumping length $n = 2$ such that for all strings $w \in B$ of length at least 2, we can decompose $w = xyz$ with the desired properties. If $w = 01^k2^k$, we will pick $x = \epsilon$, $y = 0$, $z = 1^k2^k$ and observe that $xy^iz = 0^i1^k2^k$, which is in B for all i. On the other hand, if $w = 0^m1^k2^\ell$ with $m \geq 2$, we can pick $y = 00$, $x = \epsilon$, $z = 0^{m-2}1^k2^\ell$, which again lets us pump y. Finally, if $w = 1^k2^\ell$, we can pick $x = \epsilon$, let y be the first symbol of w, and let z be the rest of the symbols. This again lets us pump y, so the pumping lemma is satisfied in all cases.

One way to prove that B is not regular (despite satisfying the pumping lemma) is to use the Myhill-Nerode theorem; we can pick $x_i = 01^i$ for all $i \in \mathbb{N}$, and observe that whenever $i \neq j$, we can take $z = 2^i$ and have $x_i z \in B$, $x_j z \notin B$.

Another way to prove that B is not regular is to use closure properties. We would like to zoom in on the strings in B that starts with only one 0, since those are the strings that look non-regular. To do so, we will set $C = L(01^*2^*)$ and consider $B \cap C$. If B were regular, then $B \cap C$ would be regular (since regular languages are closed under intersection and since C is regular, as it has a regular expression). However, $B \cap C = \{01^k2^k : k \in \mathbb{N}\}$, which we can show is non-regular using the pumping lemma.

7.2 Some common mistakes

It's worth pointing out a few things that are *not* true about regular languages, but that are common mistakes. First of all, as we saw, while every regular language satisfies the pumping lemma, some non-regular languages also satisfy it; therefore, it doesn't always work by itself.

Second, don't confuse the pumping length of a regular language with the minimum number of states in a DFA for the language. We saw that the pumping length n is always *at most* the minimum number of states in a DFA for a regular language, but it can be smaller. A simple example where it is smaller is the language $L(0^*)$ over the alphabet $\Sigma = \{0, 1\}$. It is not hard to check that this language has two equivalence classes: the all-zero strings, and the strings with a 1 in them. Therefore, the minimum size of a DFA for this language is 2. On the other hand, the minimum pumping length of this language is actually 1: for any string w of length at least 1 which is in the language $L(0^*)$, we can write $w = 0^k$ and set $x = \epsilon$, $y = 0$, $z = 0^{k-1}$, and then $xy^iz \in L(0^*)$ for all $i \in \mathbb{N}$. In other words, while the pumping lemma is useful, it doesn't equate to the language having a DFA as nicely as the Myhill-Nerode theorem does.

Third, when using closure properties, never assume that the *irregular* languages are closed under anything. It is possible to have languages A and B that are both irregular but $A \cap B$ is regular. It is also possible for A and B to be irregular and for $A \cup B$ and A^* to be regular. If A is regular and B is irregular, then $A \cap B$ is sometimes regular and sometimes irregular, with no general way of telling which. The only thing we've shown is that the *regular* languages satisfy nice closure properties; if you start with two *regular* languages, many ways of combining them will give rise to a regular language, as we've seen.

Finally, as mentioned in the first lecture, when we say that the regular languages are closed under union, this only means that the union of *two* regular languages is regular, or equivalently (using induction) that the union of *finitely many* regular languages is regular. It is not true that

a union of infinitely many regular languages must be regular. In fact, every language is an infinite union of regular languages (as we saw in a previous lecture). The same thing happens with other operations under which the regular languages are closed, such as intersection.

7.3 Next steps

This will be the last lecture on regular languages. As we've seen, regular languages are those that can be recognized by a machine (DFA) with only a finite amount of memory that does not depend on the input size; that is, machines that use "constant" memory and cannot even store the input. The regular languages turn out to have many elegant equivalent formulations, and show up repeatedly in computer science.

Next class, we will introduce a larger class of languages, called the context-free languages, which will correspond to a more powerful computational model. This computational model will have more memory, but it will be restricted in how it can access that memory. Later in the course, we will introduce Turing machines, which are like DFAs that have full access to memory, and can compute anything you can do on an ordinary computer.