The Myhill-Nerode Theorem

Prakash Panangaden

21^{st} September 2018

The collection of strings over an alphabet Σ, i.e. Σ^* is an infinite set[1] with a binary operation called *concatenation* and written by placing the arguments next to each other as in xy; occasionally we write $x \cdot y$ when we want to emphasize the operation. This operation has two properties: *associativity*, which means $\forall x, y, z$ in Σ^*, $(xy)z = x(yz)$, so bracketing is not needed when performing repeated concatenations. The second property of concatenation is that it has a *unit element*, namely ε; $\forall x \in \Sigma^*, x\varepsilon = \varepsilon x = x$. Such a structure is called a monoid.

Definition 1. A **monoid** is a set S with a binary associative operation and an identity element for this operation.

Note that the operation is **not** necessarily commutative. If the operation happens to be commutative it is called a *commutative monoid* . The most basic example of a monoid is the set of natural numbers with addition as the operation and 0 as the identity element for this operation. Strings (words) over a non-empty alphabet with concatenation as the operation and the empty word as the identity is an example. As noted above, this is not commutative. Another example of a monoid, also not commutative, is the set of functions from a set to itself: the operation is, of course, function composition[2].

Given any set, say X, and an *equivalence relation* on X we have the set of equivalence classes of X; written $X/\sim$. The *number* of equivalence classes is called the *index* of the relation. For example, we can define an equivalence relation on integers by saying that two numbers are equivalent if they have the same remainder when divided by 5. This relation has 5 equivalence classes, hence its index is 5. Note that each equivalence class in this case is infinite; this means that it contains infinitely many members. Please note the difference between there being infinitely many distinct classes and each class containing infinitely many elements.

Now we consider what happens when these two concepts interact.

Definition 2. An equivalence relation R on Σ^* is said to be **right-invariant** if

$$\forall x, y \in \Sigma^*,\ xRy \Rightarrow \forall z \in \Sigma^*,\ xzRyz.$$

In other words, if we have a related pair of strings and we stick the same string on the right side of each we get another related pair, *no matter what string we put on the right side.* Of course, not every equivalence relation will have this property. We are going to look at two very crucial examples of relations that do have this property.

[1] Except when Σ is the empty set.

[2] None of these examples yield groups

Suppose we have a DFA $M = (Q, \Sigma, q_0, \delta, F)$. We have previously defined the function $\delta^* : Q \times \Sigma^* \longrightarrow Q$ by the inductive definition

$$\forall q \in Q, \delta^*(q, \varepsilon) = q, \text{ and } \forall a \in \Sigma, x \in \Sigma^*, \ \delta^*(q, ax) = \delta^*(\delta(q, a), x).$$

It follows from this definition that

$$\forall x, y \in \Sigma^*, \ \delta^*(q, xy) = \delta^*(\delta^*(q, x), y).$$

In the rest of this note x, y, z always stand for strings in Σ^*.
Definition 3. We define an equivalence relation R_M on Σ^* as follows,

$$xR_My \text{ if and only if } \delta^*(q_0, x) = \delta^*(q_0, y).$$

It is obviously an equivalence relation[3] and, in fact, is right invariant as the following little calculation shows: suppose xR_My, then

$$\forall z \in \Sigma^*, \delta^*(q_0, xz) = \delta^*(\delta^*(q_0, x), z) = \delta^*(\delta^*(q_0, y), z) = \delta^*(q_0, yz).$$

But this just means that xzR_Myz.

Now we define another such equivalence relation, this time based on languages rather than on machines.
Definition 4. Given any language $L \subseteq \Sigma^*$, not necessarily regular, we define an equivalence relation R_L on Σ^* as follows

$$xR_Ly \text{ if and only if } \forall z, xz \in L \Leftrightarrow yz \in L.$$

This says that two strings x and y are related if, for any third string z either both xz and yz are in the language or neither one is in the language. In particular, this means that both x and y are in L or neither one is in L; this follows from choosing $z = \varepsilon$.
Lemma 5. The equivalence relation R_L is right invariant.

Proof . Suppose that xR_Ly, we must show that for any choice of z xzR_Lyz. Let u be any string and suppose that $xzu(= x(zu)) \in L$, then using the definition of right invariance with zu (remember, the definition says *for every* z!) we get that $y(zu) \in L$. Similarly, if xzu is not in L, yzu is also not in L. What we have just shown is that if we consider xz and yz, then for any u, $(xz)u \in L$ iff $(yz)u \in L$. But this is exactly what xzR_Lyz means. In short, if xR_Ly then xzR_Lyz, *i.e.* R_L is right invariant. ∎

Now we are ready to state and prove the main theorem of this note.
Theorem 6 (Myhill-Nerode)**.** The following three statements are equivalent:

1. The language L is accepted by a deterministic finite automaton.
2. L is the union of *some of the* equivalence classes of *some* right-invariant equivalence relation of finite index.

[3] Make sure you see why!

3. The equivalence relation R_L has finite index. In fact any right-invariant equivalence relation R with the property that L is the union of some of the equivalence classes of R will have index greater than R_L.

Proof . **(1) implies (2)**:

Assume that L is accepted by some DFA $M = (Q, \Sigma, q_0, \delta, F)$. We have to find a right-invariant equivalence relation that satisfies the conditions of statement (2). We have the DFA M, let us use the one that is naturally associated with M, namely R_M defined above. We already know that R_M is right invariant. How many equivalence classes can there be? As many as there are reachable states from q_0. Since the DFA has finitely many states[4] the equivalence relation R_M has finite index. Now every reachable state of M defines an equivalence class. The equivalence class of words associated with the state q is the set

$$S_q \overset{\text{def}}{=} \{x | \delta^*(q_0, x) = q\}.$$

Now the language L is just

$$L = \bigcup_{q \in F} S_q$$

which is exactly what statement (2) claims.

(2) implies (3)

In order to show this we will show that the index of R_L is smaller than (or equal to) *any* equivalence relation of the type mentioned in statement (2). We will do this by showing that for any equivalence relation of the type mentioned in (2) the equivalence classes fit inside the equivalence classes of R_L: R_L has "bigger" equivalence classes, hence it must have fewer of them. Let R be any right-invariant equivalence relation with finite index and such that L is the union of some collection of R's equivalence classes. Let xRy, we will be done if we show that xR_Ly. First note that since L is the union of some set of equivalence classes of R if any string $x \in L$ then any string related by R to x must also be in L. Note also that, since R is right-invariant *for any* z we have $xzRyz$. If $xz \in L$ then xz belongs to one of the equivalence classes whose union is L and hence yz is also in L. The same argument holds with x and y interchanged so xR_Ly. Thus an equivalence class of R must fit inside an equivalence class of R_L. So the index of R_L is less than or equal to the index of R, which is finite, thus the index of R_L is also finite.

(3) implies (1)

We construct a DFA $(Q', \Sigma, q'_0, \delta', F')$ from the language L. Now we are assuming that R_L has finite index and we already know that it is right invariant from the earlier lemma. We take the collection of equivalence classes of R_L to be the set of states Q', we take the equivalence class of ε to be q'_0, we take δ' to be given by $\delta'([x], a) = [xa]$[5] and for F' we take $\{[x] | x \in L\}$. Now this is a DFA and

$$\delta'^*(q'_0, x) = \delta'^*([\varepsilon], x) = [x]$$

[4] That is what the "F" in DFA means.

[5] Why is this well defined? If I had chosen y from the equivalence class $[x]$ I would have gotten $[ya]$, but this is OK since we know that if xR_Ly then xaR_Lya so it would have given the same equivalence class.

so this machine accepts x if and only if $x \in L$ by the way we defined F'.

■

Now two machines may have different names for the states but be essentially the same. We call this concept an "isomorphism".

Definition 7. We say two machines $M = (Q, \Sigma, q_0, \delta, F)$ and $M' = (Q', \Sigma, q'_0, \delta', F')$ are **isomorphic** if there is a bijection $\phi : Q \longrightarrow Q'$ such that

1. $\phi(q_0) = q'_0$,
2. $\phi(\delta(q, a)) = \delta'(\phi(q), a)$, Expressed diagrammatically this is:

$$\begin{tikzcd}
q \arrow[r, "a"] \arrow[d, "\phi"] & \delta(q,a) \arrow[d, "\phi"] \\
\phi(q) \arrow[r, "a"] & \delta'(\phi(q),a) = \phi(\delta(q,a)),
\end{tikzcd}$$

3. and, $q \in F \Leftrightarrow \phi(q) \in F'$.

We will view two isomorphic machines as the *same* machine. So when I use the phrase, "the unique machine" I mean unique up to isomorphism.

Proposition 8. The machine described in the last part of the proof of the theorem is the *unique* minimal DFA that recognizes the language L.

Proof . We saw from the proof of (2) implies (3) that the number of states of any machine must be greater than or equal to the number of states of the machine constructed in part (3). Suppose that there is a machine $M = (Q, \Sigma, q_0, \delta, F)$ with the same number of states as the machine $M' = (Q', \Sigma, q'_0, \delta', F')$ constructed in part (3) of the theorem. Then we can define a mapping ϕ of the states of Q to the states of Q' as follows. Let $q \in Q$. There must be some string x such that $\delta^*(q_0, x) = q$, because if there weren't we could just remove q from the machine. We define $\phi(q) = [x]$; note that this definition is independent of the choice of x. Note that $\phi(\delta(q, a)) = [xa] = \delta'([x], a) = \delta'(\phi(q), a)$. Now for every "state" $[x]$ (equivalence class of words) of M' there is some $q \in Q$ with $\phi(q) = [x]$. Why? because $\phi(\delta^*(q_0, x)) = [x]$. Thus ϕ is surjective and, since the sets of states have the same size, ϕ is a bijection. It is easy to see that $\phi(q_0) = [\varepsilon]$. Suppose that $q \in F$ then there is some word x in L such that $\delta^*(q_0, x) = q$ so $\phi(q) = [x]$, but if $x \in L$ then $[x]$ is an accept state of M'; it is easy to reverse this argument to show that if $\phi(q) = [x]$ and $[x]$ is an accept state of M' it must be true that q is an accept state of M. Thus ϕ is an isomorphism of DFAs. ■

This proposition says that M states behave exactly like the corresponding states of M'. In other words if it is the same size as M' it looks just like M', the only difference is that the names of the states are different.

The algorithms for minimization produce exactly the machine described in the proof of this theorem. Thus we can algorithmically decide if two regular expressions define the same regular language by constructing NFAs from the regular expressions, determinizing them to get DFAs and then minimizing the resulting DFAs. If they describe the same language we must get the same minimal DFA. It is completely mistaken to use the algebra of regular expressions to reason about machines, one uses machines to reason about regular expressions. There is a rich and interesting algebraic theory of regular expressions and one can prove equations without using machine models but it is **much** harder. There is no such result for NFAs, there is not even a clear notion of what is meant by a minimal NFA.

Do not read beyond this: I mean it!

In the definition of isomorphism we can weaken the condition that we have a bijection and just say that we have a function. We call this a *homomorphism* of machines. If there is a homomorphism between two machines they accept the same language. Now we can define a category with machines as the objects and homomorphisms as the morphisms. In this category we can define an initial and a terminal object: they will not be finite-state machines any more. One can define other similar categories of machines not based on **Set**, thus one can define machines in **Top** or **Vect** to get topological machines or linear machines or machines in categories of measure spaces (which I do for a living). Thus the use of category theory allows one to lift the simple theory we are studying to more exotic settings. There is even a beautiful duality theory for automata which I aaaarrrggg.... gasp... OK, I promise not to tell them any more.