CSE 322 - Introduction to Formal Methods in Computer Science
The Myhill-Nerode Theorem

Dave Bacon
Department of Computer Science & Engineering, University of Washington

The pumping lemma for regular languages is nice, but it has one fatal drawback, and that is that it is not a complete characterization of whether a language is regular. In other words this means that there are languages which are not regular, but which obey the pumping lemma.

Here we will describe a condition which, unlike the pumping lemma, can be used as a complete test for whether a language is regular or not. It is called the Myhill-Nerode theorem.

I. TWO EQUIVALENCE RELATIONS

To begin let us start with two useful equivalence relations. The first equivalence relation is defined over any language. Let A be a language in Σ^*. Then we say that the strings x and y are *indistinguishable by* A if and only if, for every string $z \in \Sigma^*$, either both xz and yz are in the language or both xz and yz are not in the language. When two strings are indistinguishable with respect to A, we write $x \equiv_A y$.

Lets give an example. Consider the language $A = \{01^k | k \geq 0\} \cup \{00\}$. The two strings 01 and 011 are indistinguishable in A. Why? Because $01z$ is in A iff $z \in 1^*$. Similarly $011z$ is in A iff $z \in 1^*$. Thus the strings $01z$ and $011z$ are both in A at the same time, or both not in A at the same time.

Now, $\equiv_A$ is an equivalence relation. Recall that a relation is an equivalence relation if it is reflexive, symmetric, and transitive. The first of these means that $x \equiv_A x$ which is clearly true. The second of these means that $x \equiv_A y$ iff $y \equiv_A x$, which is also clearly true since our definition was symmetric. The final of these means that if $x \equiv_A y$ and $y \equiv_A z$, then $x \equiv_A z$ which is the only slightly tricky one, but if you give it a little thought you will see it is true. Now given that $\equiv_A$ is an equivalence relation what can we do? Recall that if we have an equivalence relation we can divide the set the relation is defined over into equivalence classes. These equivalence classes are subsets of the set we are working with which are disjoint (the intersection of any two is empty) and which are complete (the union of all of the equivalence classes is the entire set.) In other words equivalence relations give us a way to define equivalence classes, and these equivalence classes fully partition the set we are working on.

Lets work out an example for the language we considered above $A = \{01^k | k \geq 0\} \cup \{00\}$ (we work with the alphabet $\Sigma = \{0, 1\}$.) It is easy to see, by generalizing our example above, the $\{01^k | k > 0$ are equivalent to each other. Note that 0 is not equivalent over A to 01, since, 00 is in A, but 010 is not in A. Further, using similar reasoning 00 is not equivalent to 0. Further ε is not equivalent to any other string in A, since there is no portion of A which ends with arbitrary strings. But ε is equivalent to strings which are not in A. Indeed, all of the elements that are not in A are equivalent to each other (you should check this.) Thus the equivalence classes of A are $\{0\}$, $\{00\}$, $\{01^k | k > 0\}$, and $\Sigma^* - A$.

Next lets give another equivalence relation, but this time defined over a DFA M. To do this we will need to define a concept called delta-star. The transition function of a DFA, δ tell us, given a current state, and a symbol, what new state to transition to. δ^* tell us, given a current state, and a *string*, what new state would the DFA end up, starting at the current state and then reading that particular string. We can define δ^* inductively. First note that $\delta^* : Q \otimes \Sigma^* \to Q$. Then define this function as

Delta-star definition:
1. Base case: $\delta^*(q, \varepsilon) = q$
2. Inductive step: For $x \in \Sigma^*$ and $a \in \Sigma$, $\delta^*(q, xa) = \delta(\delta^*(q, x), a)$

Okay, given the definition of δ^* we can now begin to define our new equivalence relation. Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, we say that two string x and y are indistinguishable, if and only if $\delta^*(q_0, x) = \delta^*(q_0, y)$. In other words the two strings are indistinguishable if, starting at the initial state, one ends up at the same state upon reading either x or y. If this is true, we write $x \equiv_M y$.

Again, it is easy to check that $\equiv_M$ is and equivalence relation. Thus we can use this equivalence relation to partition Σ^* up into equivalence classes as before. Further note that $\equiv_M$ only has a finite number of equivalence classes, since the equivalence is defined by ending up in the same state, and there are only a finite number of states for a DFA. Further note that if the DFA is made up of reachable states, then the number of equivalence classes will just be equal to the number of states of the machine M.

II. A LEMMA AND A NEW METHOD FOR PROVING A LANGUAGE NOT REGULAR

Now lets prove a useful lemma relating the equivalence class of a machine M and the language accepted by this machine, $L(M)$. The lemma is as follows: If $A = L(M)$ for a DFA M, then for any $x, y \in \Sigma^*$, if $x \equiv_M y$ then $x \equiv_A y$. Notice that at this point this is only an if statement, not an if and only if (yet!). So why is this true? Lets prove it!

Let $A = L(M)$ and $M = (Q, \Sigma, \delta, q_0, F)$. Therefore $w \in A$ if and only if $\delta^*(q_0, w) \in F$. Suppose also that $x \equiv_M y$. Then $\delta^*(q_0, x) = \delta^*(q_0, y)$. Let $z \in \Sigma^*$. Then $\delta^*(q_0, xz) = \delta^*(yz)$ (since after we get to the state reached by both x and y the DFA will proceed deterministically.) Therefore $xz \in A$ if and only if $\delta^*(q_0, xz) \in F$. Further since $\delta^*(q_0, xz) = \delta^*(yz)$ this means that $xz \in A$ if and only if $\delta^*(q_0, yz) \in F$. But $\delta^*(q_0, yz) \in F$ if and only if $yz \in A$. Therefore $xz \in A$ if and only if $yz \in A$. It thus follows that $x \equiv_A y$.

What good is this lemma? Well, it tells us something nice. It says that whenever two elements arrive at the same state in M, they are in the same equivalence class of $\equiv_A$. This means that each equivalence class of $\equiv_A$ is a union of equivalence classes of $\equiv_M$. One nice consequence of this is, since $\equiv_M$ has equivalence classes which are finite in number, that we can now show that

> If A is regular, then $\equiv_A$ has a finite number of equivalence classes.

Why is this true? If A is regular, then there is a DFA M which recognizes the language A. Our lemma from above tells us that $\equiv_A$ has at most the number of equivalence classes of $\equiv_M$ (this largest case comes about when $\equiv_A$ equivalence classes are identical to the equivalence classes of $\equiv_M$.) Since the number of equivalence classes of $\equiv_M$ is finite, this implies that the number of equivalence classes of $\equiv_A$ is finite.

Okay well now that we've proved this, what good is it? Well now we have a new method for proving that languages are not regular! How? Well, again, by contradiction. We proceed by assuming that the language, A, we are considering is regular. Then if we can show that the number of equivalence classes under $\equiv_A$ is not finite, then we have proved a contradiction to our deduction above. To prove that the number of equivalence classes under $\equiv_A$ is not finite, we don't really need to identify all of the equivalence classes: all we really need to do is to provide an infinite sequence of strings and prove that they are not equivalent to each other.

Lets do an example! Consider the language $A = \{0^n 1^n | n \geq 0\}$. Lets prove that this language is not regular, using the corollary above. Assume that A is regular and therefore, by the above corollary, it has a finite number of equivalence classes. Lets examine the sequence of strings $x_1, x_2, \ldots$ where $x_i = 0^i$, where $i \geq 1$. Let us show that no two of these are equivalent to each other with respect to A. Consider 0^i and 0^j where $i \neq j$. Let $z = 1^i$. Now $0^i z = 0^i 1^i$ which is in A. But $0^j z = 0^j 1^i$ is not in A. Thus these two strings 0^i and 0^j are not equivalent to each other. Thus, since the sequence of string $x_1, x_2, \ldots$ is infinite, we have shown that the number of equivalence classes of A is infinite. This contradicts our assumption that the number of equivalence classes of A is finite. Thus A must not be regular. Wah-lah.

Okay, so we have a brand spanking new method for showing that a language is not regular. But does this method provide us anything more than the pumping lemma did? Well the answer to that is the reason I'm blabbing on about this. This is the fact that, unlike the pumping lemma, this method is always guaranteed to work. The statement of this is the Myhill-Nerode theorem.

III. MYHILL-NERODE THEOREM

Myhill-Nerode Theorem
A is regular if and only if $\equiv_A$ has a finite number of equivalence classes. In addition there is a DFA M with $A = L(M)$ having precisely one state for each equivalence class of $\equiv_A$.

We have already proved one direction of this theorem (that if A is regular, then it has a finite number of equivalence classes under $\equiv_A$.) Now lets prove the opposite direction: that if $\equiv_A$ has a finite number of equivalence classes, then A is regular. We will do this by our favorite trick, by showing that if $\equiv_A$ has a finite number of equivalence classes, then we can construct a DFA M which recognizes A.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be the DFA we are going to construct. Further let $A_1, A_2, \ldots, A_r$ denote the equivalence classes of $\equiv_A$. The A_i are disjoint and partition Σ^*. Define the states of the DFA as $Q = \{q_1, \ldots, q_r\}$. Our goal will be to define the transition function δ and q_0 and F such that $\delta^*(q_0, x) = q_r$ if and only if $x \in A_r$.

Lets begin by defining q_0. Let $q_0 = q_i$ such that $\varepsilon \in A_i$. In other words, the start state corresponds to the equivalence class that contains the empty string.

Now lets figure out how to construct the transition function. To see how to do this note that for an A_j and any $a \in \Sigma$, for every $x, y \in A_j$, xa and ya are both contained in the same equivalence class of $\equiv_A$. Why? Because if $x, y \in A_j$ then xz and yz, $z \in \Sigma^*$ are either both in A or both not in A. Therefore, in particular if we consider strings

of the form $z = aw$, $w \in \Sigma^*$, then xaw and yaw are either both in A or not both in A. This means that they xa and ya are in the same equivalence class with respect to $\equiv_A$.

Thus to figure out what $\delta(q_j, a)$ should be, we pick some $x \in A_j$ and find the one A_k such that $xa \in A_k$. Then we set $\delta(q_j, a) = q_k$. Notice that by our argument above, this will be independent of what x we choose.

Finally we need to pick the accept states. Note that either $A_j \subset A$ or $A_j \cap A = \emptyset$ because equivalence classes stratling A are not possible since they fail the equivalence definition for $z = \varepsilon$ appended onto these elements. In other words if A_j contained elements from A and from $\Sigma^* - A$, then we can pick string x from A and string y from $\Sigma^* - A$, and then $x\varepsilon$ is in A buy $y\varepsilon$ is not in A, contradicting the assumption that A_j is so constructed. Thus we define $F = \{q_j | A_j \subseteq A\}$.

It is easy to see that the above DFA recognizes A. A formal proof, which we will forgo, would proceed by induction to show this.

Thus we have seen that if $\equiv_A$ has a finite number of equivalence classes, then we can construct a DFA which recognizes A and thus A is regular. This, along with the other direction proved above proves the Myhill-Nerod Theorem. Further note that the DFA we constructed has precisely the number of equivalence classes of $\equiv_A$ states.

Further note that the DFA we have constructed has the minimal number of states possible for a DFA. Why is this so? Suppose that we have a machine M which accepts the language A. Then we know the equivalence classes of $\equiv_A$ are made up of unions of equivalence classes of $\equiv_M$. Thus the smallest the number of equivalence classes of $\equiv_M$ could be is the number of equivalence classes of $\equiv_A$. Since $\equiv_M$ is the number of (reachable) states in M this implies that M the smallest number of states is equal to the number of equivalence classes of $\equiv_A$. Our construction of a DFA from the equivalence classes of $\equiv_A$ has exactly such a number of states and is therefore the smallest DFA which will accept A.