CSE396 Thursday, March 4: Myhill-Nerode Examples and Implications

Once you pick up the pattern that was scripted in "fill-in-the-blank" fashion, the proofs become really quick---and even redundant. In fact, sometimes you can re-use a proof that was done for a different language. **[The actual lecture did a lot more change-arounds than these notes represent, some ad-libbed from excellent questions, but at least these notes give some flavor of that. I'm not sure I've put them all the way back the way they were.]**

Example $2'$: Prove that $L_a = \left\{0^n 1 0^n : n \geq 0\right\}$ is nonregular

Proof: literally copied-and-pasted from last lecture's posted notes on the PAL language:

Take S = _________ $(00)^* 1$ ______________. "Clearly S is infinite."

Let any $x, y \in S$ (such that $x \neq y$) **be given**. Then we can helpfully write x = ___ $(00)^m 1$ ____ and y = ____ $(00)^n 1$ ______ where ____ $m \neq n$ _____. ~~[optional: where __________ "wlog."]~~

Take z = ____ $(00)^m$ ______. Then $L_a(xz) \neq L_a(yz)$ because _____ $xz = (00)^m 1 (00)^m$ which is in L_a, but $yz = (00)^n \cdot 1(00)^m$ which is not in L_a since $m \neq n$ ______.

Since x and y are an arbitrary pair from S, S is PD for L_a, and since S is infinite, L_a is nonregular by the Myhill-Nerode Theorem. ☒.

Actually, on the chalkboard I took $S = 0^* 1$. The proof worked the same except that it used:

given any general $x = 0^m 1$ and $y = 0^n 1$, taking $z = 0^m$,

Then $xz = 0^m 1 0^m \in L_a$ but $yz = 0^n 1 0^m \notin L_a$ since $m \neq n$. I could also have just taken $S = 0^*$. Then we would have:

given any general $x = 0^m$ and $y = 0^n$, taking $z = 10^m$,

with the same outcomes for xz and yz. Including the `1' in the definition of S was merely being "proactive", insofar as the 1 being the only possible dividing point is the key point of the logical reasoning. Here is the proof compactly:

> Take $S = 0^* 1$. Clearly S is infinite. Let any $x, y \in S$ $(x \neq y)$ be given. Then we can generally write $x = 0^m 1$, $y = 0^n 1$ where $m \neq n$. Take $z = 0^m$. Then $xz = 0^m 1 0^m \in L_a$, whereas $yz = 0^n 1 0^m \notin L_a$ since $m \neq n$. Thus $L_a(xz) \neq L_a(yz)$, and since $x, y \in S$ are arbitrary, S is PD for L_a. And since S is infinite, L_a is non-regular by the Myhill-Nerode Theorem. ☒

Exactly the same proof worked for the PAL language. But suppose we defined the language $EPAL = \{xx^R : x \in \{0,1\}^*\}$ instead. This requires palindromes to have even length. Then $xz = 0^m10^m$ would be no good. But taking $z = 10^m$ to make $xz = 0^m110^m$ belongs to EPAL as well.

The proof in the notes doing $S = (00)^*1$, so $x = (00)^m1$ and $y = (00)^n1$, with $z = 1(00)^m$, was harmless overkill for all these languages---but might be useful for something else. In general, the more restricted you can make S while keeping it infinite, the better.

The other kind of re-use is exemplified by taking $L_b = \{0^m10^n : m \leq n\}$ instead. Now to avoid another possible confusion about what "m" and "n" stand for and which side of the middle 1 they apply to. We can rewrite the definition of the language to read $L_b = \{0^i10^j : i \leq j\}$. Now we roll:

Take $S = 0^*1$. Clearly S is infinite. Let any $x, y \in S$ $(x \neq y)$ be given. Then we can generally write $x = 0^m1, y = 0^n1$ where **wlog.** $m < n$. Take $z = 0^m$. Then $xz = 0^m10^m \in L_b$, since $m \leq m$, whereas $yz = 0^n10^m \notin L_b$ since $n > m$. Thus $L_b(xz) \neq L_b(yz)$, and since $x, y \in S$ are arbitrary, S is PD for L_b. And since S is infinite, L_b is non-regular by MNT. ☒

Example 5: $BAL = \{x \in \{(,)\}^* : x$ is a balanced string of parentheses$\}$.

[Side example: If you have ((())), then this is string is actually $\sim_{BAL}$ equivalent to the empty string. If you follow it by ()()() then the whole thing ((()))()()() is balanced just like ()()() is by itself.
And if you have (((())), then this is string is actually $\sim_{BAL}$ equivalent to the string "(" If you follow it by ()()() then the whole thing (((()))()()() has an excess of one (just like (()()() is by itself.]

MNT proof:
Take $S = (^*$. Clearly S is infinite. Let any $x, y \in S$, $x \neq y$, be given. Then we can write $x = (^m$, $y = (^n$ where $m, n \geq 0$ and $m > n$ wlog. Take $z =)^m$. Then $xz = (^m)^m$ is in BAL', but $yz = (^n)^m$ is not in BAL' since $m > n$ makes have too many right parens to close. So $BAL'(xz) \neq BA'L(yz)$. Since $x, y \in S$ are arbitrary, S is PD for BAL', and since S is infinite, BAL' is nonregular by MNT. ☒.

Example 6: How about $BAL' = \{x \in \{(,)\}^* : x$ can be closed to make a balanced string of parentheses$\}$? This is the same as saying $(\exists u \in \{)\}^*) xu \in BAL$. The above proof doesn't work since we could have $m < n$ so that $yz = (^n)^m$ can be closed by appending $u =)^{n-m}$. But if $n < m$ (which you can alternatively assert "wlog.") that wouldn't work. In fact, I prefer to keep $m < n$ to mimic alphabetical order and change the choice of z in the proof instead.

Take $S = (^*$. Clearly S is infinite. Let any $x, y \in S$, $x \neq y$, be given. Then we can write $x = (^m$, $y = (^n$ where $m, n \geq 0$ and wlog. $m < n$. Take $z =)^n$. Then $xz = (^m)^n$ is not in BAL' since the excess of right parens cannot be fixed, but $yz = (^n)^n$ is in BAL', so $BAL'(xz) \neq BAL'(yz)$. Since $x, y \in S$ are arbitrary, S is an infinite PD set for BAL', so BAL' is nonregular by MNT. ☒.

In fact, BAL' is really the same as the language of "spears and dragons with unlimited spears", reading '(' as a spear, ')' as a dragon, and ignoring empty rooms.

Example 7: To come back to an example in the text, try $A = \left\{ww : w \in \Sigma^* : |w| \text{ is odd}\right\}$ where again $\Sigma = \{0, 1\}$. Over any alphabet of size 2 or more, this language is often called DOUBLEWORD. When we cover the "CFL Pumping Lemma" in ch. 2 we will see that it is not even a "CFL" (which includes all regular languages), but for now we'll just prove it's nonregular. We can essentially ~~plagiarize~~ re-use the proof for the palindrome language (but by the way: PAL **is** a CFL).

Take $S =$ _________ 0^* ______________. "Clearly S is infinite."

Let any $x, y \in S$ (such that $x \neq y$) **be given**. Then we can helpfully write $x =$ ___ 0^m ____ and $y =$ ____ 0^n ______ where ____ $m \neq n$ _____.

Take $z =$ ____ 10^m1 ______. Then $A(xz) \neq A(yz)$ because _____ $xz = 0^m10^m1$ which is in A, but $yz = 0^n10^m1$ which is not in A since $m \neq n$ and the only possible way to make a double word is to break after the first 1.

Since x and y are an arbitrary pair from S, S is PD for A, and since S is infinite, A is nonregular by the Myhill-Nerode Theorem. ☒.

While cutting the mouse-copied fill-in-the-blank format, we can make the proof a little more elegant by defining S slightly differently:

Take $S = (00)^*1$. Clearly S is infinite. Let any $x, y \in S$ $(x \neq y)$ be given. Then we can write $x = (00)^m1$ and $y = (00)^n1$ where $m \neq n$. Take $z = (00)^m1$. Then $A(xz) \neq A(yz)$ because $xz = (00)^m1(00)^m1 \in A$ and $|(00)^m1| = 2m + 1$ which is always odd, but $yz = (00)^n1(00)^m1 \notin A$ since $m \neq n$ and the only possible way to make a double word is to break after the first 1. Since $x, y \in S$ are arbitrary, S is PD for A, and since S is infinite, we're done. ☒

What on earth is $\mathcal{REG}$? The curly font means it is a set of languages, which we call a **class**. So $\mathcal{REG}$ stands for the class of regular languages. Beware: the language

$A'' = \{x \cdot y : \#0(x) = \#1(y)\}$ **is** in $\mathcal{REG}$.
One help is to rewrite sets so they have only one named object to the left of the : (or |)
$A'' = \{w : w \text{ can be broken as } w =: xy \text{ such that } \#0(x) = \#1(y)\}$
Similarly, you can avoid confusing A^2 with $\{xx : x \in A\}$ by remembering the definition of

$A \cdot B = \{w : w \text{ can be broken as } w =: x \cdot y \text{ with } x \in A \text{ and } y \in B\}$. So $A^2 = \{w : w \text{ can be broken as } w =: x \cdot y \text{ with } x \in A \text{ and } y \in A\}$.

[Example 8 got covered a different way, in response to a question about the original DOUBLEWORD language. The gist was the same; I used the numbers 5 and 7 in lecture.]

Example 8: What about DOUBLEWORD over a single-letter alphabet, say $\Sigma = \{a\}$? It is still defined via $A = \left\{ww : w \in \Sigma^*\right\}$. Let's try the same kind of strategy:

"**Poof**": Take $S = a^*$. Clearlty S is infinite. Let any $x, y \in S$ $(x \neq y)$ be given. Then we can write $x = a^m$ and $y = a^n$ where $m \neq n$. Take $z = a^m$. Then $xz = a^m a^m$ which is clearly a double-word, but $yz = a^n a^m$ which is not since $n \neq m$. So $A(xz) \neq A(yz)$, so S is PD for A, and since S is infinite, $A \notin \mathcal{REG}$ by the Myhill-Nerode Theorem. ☒

But wait: a string over $\{a\}$ is a double-word if and only if it is an even number of a's, so it matches $(aa)^*$, so A is regular after all. What is wrong with the proof? Note that $m = 3, n = 5$ is a possible pair from S, that is, $x = a^3, y = a^5$ which makes $z = a^3$. Clearly $xz = a^3a^3$ is a double-word, but it looks like $yz = a^5a^3$ isn't. At least that's the *intent* of writing a^5a^3, and (here comes a jargon word) the ***intension*** by which we may read it. But the ***extension*** is that a^5a^3 is the string of eight a's, which without the power abbreviations is $aaaaaaaa$. This can be broken a different way as $aaaa \cdot aaaa$, whose intension is $a^4 \cdot a^4$. Thus the string a^5a^3 is a double-word after all, so the conclusions $yz \notin A$ and thus $A(xz) \neq A(yz)$ were wrong. Poof!

And since the language A is regular after all the proof can't be fixed. Another common mistake is to "fudge" by restricing pairs from S in ways that *do lose generality*. For instance, if you asserted "Then we can write $x = a^m$ and $y = a^n$ where one of m and n is even and the other is odd," then the conclusions $xz \in A$ and $yz \notin A$ giving $A(xz) \neq A(yz)$ would work---but you wouldn't have represented all the possibilities in the $(\forall x, y \in S,\ x \neq y)$ requirement fairly.

[The next example got skipped and will reappear next Tuesday.]

Does that mean all languages over a single-letter alphabet are regular? Our last lecture example shows not. The "wlog. $m < n$" part isn't strictly necessary, but it is convenient.

Example 9: Define $A = \left\{a^N : N \text{ is a perfect square}\right\}$, which equals $\left\{a^{n^2} : n \in \mathbb{N}\right\}$.

Take $S = a^*$. Clearlty S is infinite. Let any $x, y \in S$ $(x \neq y)$ be given. Then we can write $x = a^m$ and $y = a^n$ where wlog. $m < n$. Put $k = n - m$. The key numerical fact about perfect squares is that the gaps between successive squares grow bigger and bigger. So we can find r such that $(r+1)^2 - r^2 > k$, and for good measure, such that $r^2 > m$. Take $z = a^{r^2 - m}$. Then

$xz = a^m a^{r^2-m} = a^{r^2}$, which belongs to A. But

$$yz = a^n a^{r^2-m} = a^{k+m} a^{r^2-m} = a^{r^2+k},$$

which is not long enough to get up to $a^{(r+1)^2}$, which is the next member of A. So $yz \notin A$, giving $A(xz) \neq A(yz)$ legitimately this time. So S is PD for A, and since S is infinite, $A \notin \mathcal{REG}$ by the Myhill-Nerode Theorem. ☒

By the way, one can state the MNT as "if A has an infinite PD set S then A is not regular." This is, however, "Has-A" in the OOP sense, not in the sense of S being a subset of A. In example 9, S is actually a *superset* of A.

Consequences of MNT for Languages that *Are* Regular and Their DFAs

[The first part of this will also reappear in the next lecture. I jumped straight to the "L_k" example and called it "example 9."]

The full Myhill-Nerode Theorem---including the converse direction---says that every regular language A has a DFA M_A whose states are the equivalence classes of the relation $\sim_A$. No DFA can have any fewer states than the number k of those classes---that follows from the original forward direction. Moreover, the components of M_A are all completely dictated by the relation. Let us write $[x]$ to denote the equivalence class of a string x (with the language A understood); note that when $x \sim_A y$ we have $[x] = [y]$ even though $x \neq y$. Then we have $s_A = [\epsilon]$ as the start state of M_A and $F_A = \{[x] : x \in A\}$ for the final states. The part I didn't say before is the rule

$$\delta_A([x], c) = [xc],$$

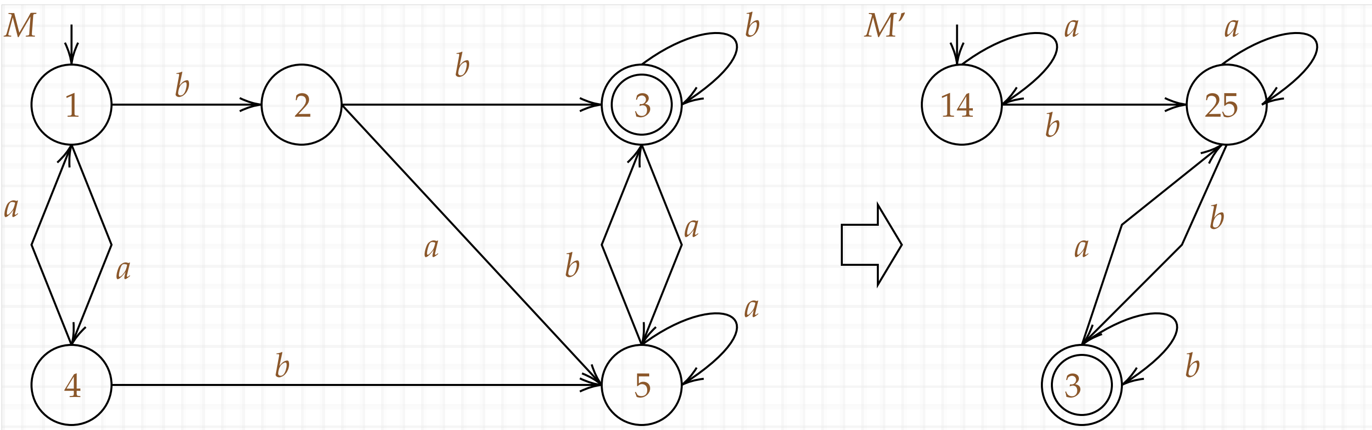
which is valid because using y in place of x when $[y] = [x]$ doesn't change the value, because then $[yc] = [xc]$ anyway, since $x \sim_A y$ implies $xc \sim_A yc$ for any char c. Anyway, the point is that δ_A too is completely dictated. The upshot is the following statement:

Corollary (to the MNT): Every regular language A has a minimum-size DFA M_A that is unique.

Unfortunately, the MNT does not do much to help you build an efficient *algorithm* to *find* M_A. The one thing we do know is that once you *have* a DFA $M = (Q, \Sigma, \delta, s, F)$ such that $L(M) = A$, no matter how wasteful, you can always efficiently *refine* it down to the unique optimal M_A. The key definition uses the auxiliary notation $\delta^*(q, z)$ to mean the state that M (being a DFA) uniquely ends at upon processing the string z from state q. Inductively, $\delta^*(q, \epsilon) = q$ for any q, and further for any string w and char c, $\delta^*(q, wc) = \delta\left(\delta^*(q, w), c\right)$.

Definition: Two *states* p and q in a DFA M are **distinguishable** if there exists a string z such that one of $\delta^*(p, z)$ and $\delta^*(q, z)$ belongs to F and the other does not. Otherwise they are **equivalent**.

Two equivalent states must either be both accepting or both rejecting, because if they are one of each then they are immediately distinguished by the case $z = \epsilon$. There is a simple sufficient condition: If p and q are bopth accepting or both rejecting, and if they go to the same states on the same chars (that is, if $\delta(p, c) = \delta(q, c)$ for all $c \in \Sigma$), then they are equivalent. But otherwise, it can be hard to tell equivalence. There is an algorithm for determining this that is covered in some texts, and also appears in some Algorithms texts as an example of "dynamic programming."



States 2 and 3 are distinguished by ϵ since 2 is rejecting and 3 is accepting. Ditto 3 and 5. States 2 and 5 are equivalent, however, because both are rejecting and both go to 5 on a and to 3 on b. States 2 and 4 are distinguished by $z = b$ because 2 goes to an accepting satte on b but 4 does not. Ditto states 1 and 2. But states 1 and 2 are equivalent. This is harder to see immediately, but is because they go to each other on a and go to equivalent states on b. The unique minimum DFA M' such that $L(M') = M$ is shown at right.

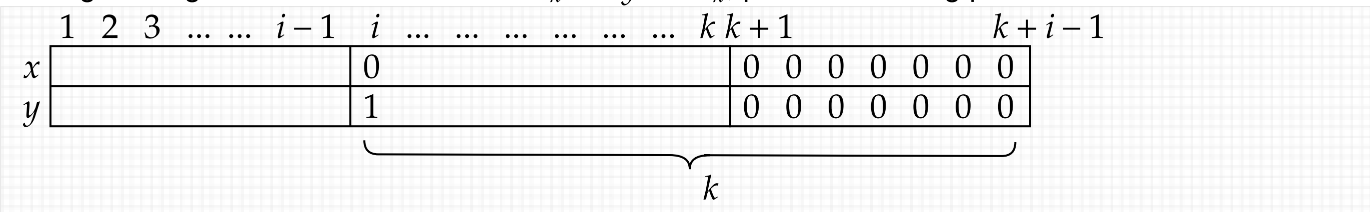
We will focus on the distinguishing side instead. The following are good self-study points about any DFA M with language $A = L(M)$:

- If $x \nsim_A y$ (in words, if x and y are distinctive for the language A) then in any DFA M such that $L(M) = A$, $\delta^*(s, x)$ and $\delta^*(s, y)$ must be distinguishable states---not just different states.
- If $x \sim_A y$, then $\delta^*(s, x)$ and $\delta^*(s, y)$ must be equivalent states.
- If $\delta^*(s, x)$ and $\delta^*(s, y)$ are distinguishable states, then $x \nsim_A y$.
- If S is a PD set for A, then the strings in S must all get processed to different states from s.

The last point leads us to consider PD sets in cases where languages *are* regular. Let us revisit the languages $L_k = (0 + 1)^* 1 (0 + 1)^{k-1}$ for all $k \geq 1$. Recall that L_k always has an NFA N_k of $k + 1$ states that mainly guesses when to jump out of its start state when reading a 1.

Proposition: For all $k \geq 1$, the set $S_k = \{0, 1\}^k$ is a PD set of size 2^k for L_k.

Proof: Clearly $|\{0,1\}^k| = 2^k$. Let any $x, y \in S_k$, $x \neq y$, be given. Since they are different binary strings, there must be a bit place i (numbering $1 \dots k$) in which they differ. Without loss of generality, let "x" refer to the string that has 0 in posiiton i and "y" to the string with a 1 there. Take $z = 0^{i-1}$, which is a legal string since $i \geq 1$. Then $xz \notin L_k$ but $yz \in L_k$, per the following picture:



Thus $L_k(xz) \neq L_k(yz)$, and since $x, y \in S_k$ are arbitrary, S_k is PD for L_k. Hence the minimum DFA M_k such that $L(M_k) = L_k$ must have at least 2^k states (and in fact, can be designed with that many states, e.g., since $[\epsilon] = [0^r]$ for any number r, we can re-use the start state for $\delta(s, 0) = s$, and so on). ☒

This finally proves there are cases of "exponential blowup" in the NFA-to-DFA construction.

[The Thu. 2/19 lecture ended here. The next section is already set up to segue nicely into the topic of Context-Free Grammars. Beteen that section and the introduction of CFGs, I will make some speculative assertions about human brain function and try to convince you of them.]

The Class of Regular Languages: What It means to be Regular

Given a DFA $M = (Q, \Sigma, \delta, s, F)$, let us use the notation $\delta^*(p, x) =$ the state q that M is in after processing x from state p. (We could have used Δ^* for the DFA in the NFA-to-DFA proof.) Note that

$$x \in L \iff \delta^*(s, x) \in F,$$

where $L = L(M)$, so

$$x \notin L \iff \delta^*(s, x) \notin F,$$

which is the same as writing

$$x \in \widetilde{L} \iff \delta^*(s, x) \in \widetilde{F}.$$

The upshot is that the DFA $M' = \left(Q, \Sigma, \delta, s, \widetilde{F}\right)$ gives $L(M') = \widetilde{L}$. This trick of complementing accepting and nonaccepting states does not, however, work for a general NFA. For example, if you try this on the NFAs N_k given for the languages L_k of binary strings whose kth bit from the end is a 1, then the new machine has an accepting loop at the start state on both 0 and 1 and so accepts every string, not just those in the complement of L_k. [I spent some time showing this from the picture of N_k in the previous lecture.] But thanks to Kleene's Theorem, being able to do it for DFAs is enough to prove:

Theorem 1: The complement of a regular language is always regular. ☒

Theorem 2: The class of regular languages is closed under all Boolean operations.

Actually, we already could have said this right after Theorem 1 about complements. This is because OR is a native regular expression operation. OR and negation ($\neg$) form a complete set of logic operations. For instance, a AND $b \equiv \neg((\neg a)$ OR $(\neg b))$ by DeMorgan's laws.

What kind of machine or formal system can have a non-regular language? Next week in Chapter 2 we will explore context-free grammars (CFGs). Just for preview, the CFG $G = S \rightarrow 0S1 \mid \epsilon$ gives $L(G) = \{0^n 1^n : n \geq 0\}$, and $G' = S \rightarrow \epsilon \mid 0S \mid \$S \mid \$SDS$ generates all strings in the spears-and-dragons game with unlimited spears in which the "Player" survives.