CS 361 Meeting 12 — 3/6/20

Announcements

1. Homework 3 solutions are online.
2. Homework 4 is due today.
3. Homework 5 will be available over the weekend.
4. Our midterm will occur during the week of 3/16.

Quite Distinguished

1. Last time I introduced the notion of strings that were distinguished by a language:

   Definition: We say that $w, v \in \Sigma^*$ are *distinguishable* by a language L if for some $z \in \Sigma^*$, exactly one of wz and vz is a member of L.

2. Of course, the opposite of distinguished is indistinguishable:

   Definition: We say that $w, v \in \Sigma^*$ are *indistinguishable* by language L if for all $z \in \Sigma^*, wz \in L \iff vz \in L$ and we write $w \equiv_L v$.

3. Two strings are indistinguishable relative to a language if in some sense they are equivalent as prefixes of strings in the language.

4. The relation "indistinguishable by L" defined by

   Definition: We say that $w, v \in \Sigma^*$ are *indistinguishable* by language L if for all $z \in \Sigma^*, wz \in L \iff vz \in L$ and we write $w \equiv_L v$.

   is an equivalence relation on strings.

5. The number of distinct equivalence classes of strings under the indistinguishable relationship has an interesting relationship to regularity.

Click here to view the slides for this class

- Given a language L and a set X of strings over L's alphabet, we say that the X is pairwise distinguishable by L if every pair of strings in X is distinguishable by L.
- The *index* of a language L is the size of the largest set of strings X that is pairwise distinguishable by L. Equivalently, the index of a language is the size of the set of equivalence classes induced by the "indistinguishable" relation relative to the language.

The Myhill-Nerode Theorem

1. This bring us to the big Theorem (introduced through problem 1.52 in Sipser):

   Theorem (Myhill-Nerode): A language L is regular iff it has finite index and each regular language is accepted by a DFA whose description includes as many states as the index of the language.

2. We start by showing that regularity implies finite index.

   Proof:

   (a) Suppose that L is regular.
   - Then there is some DFA $D = (Q, \Sigma, \delta, s, F)$ such that $L = L(D)$.
   - Suppose that X is a set of strings that is pairwise distinguishable by L with $w, v \in X$.
   - Consider $\hat{\delta}(s, w)$ and $\hat{\delta}(s, v)$.
   - If $\hat{\delta}(s, w) = \hat{\delta}(s, v)$ then for all $z \in \Sigma^*$, $\hat{\delta}(s, wz) = \hat{\delta}(s, vz)$. But $wz \in L \iff \hat{\delta}(s, wz) \in F \iff \hat{\delta}(s, vz) \in F \iff vz \in L$ which would imply that w and z were indistinguishable by L.
   - Since the members of X are pairwise distinguishable, this cannot be the case so for all w, v it must be the case that $\hat{\delta}(s, w) \neq \hat{\delta}(s, v)$.
   - This implies that the number of elements in X cannot exceed the number of elements in Q since otherwise there would have to be at least two strings

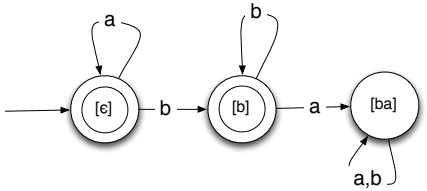
in X such that $\hat{\delta}(s, w) = \hat{\delta}(s, v)$. Therefore, if L is regular it is of finite index.

3. Now we need to consider the other direction of the if and only if ...

Proof (continued):

b) Suppose L is of finite index.

- First, let's look at a very simple concrete example: a^*b^*.
  - We have seen that this language has index 3 since $\{a, ab, ba\}$ forms a maximal set of distinguishable strings relative to the language.
  - Any other collection of representatives of the equivalence classes associated with these strings also form maximal distinguishable sets.
  - For example, $\{\epsilon, b, ba\}$ is another maximal distinguishable set for a^*b^*.
  - Consider the following machine which uses the obvious set of states to recognize a^*b^*, and names those states, as we have often done, with "representatives" of the strings that would move the machine to each state:



  - This suggests the following general construction.
- Let X be a maximal set of strings pairwise distinguishable by L.
- Construct a DFA

$$D = (\{[x] \mid x \in X\}, \Sigma, \delta, [\epsilon], \{[x] \mid x \in L \cap X\})$$

where $\delta([x], a) = [xa]$.

- We claim that $L = L(D)$. To justify this claim, we need to show that
  - δ is well-defined. In particular that if $a \in \Sigma, x, x' \in [x]$ then $[xa] = [x'a]$.
  - for all $w \in \Sigma^*$, $\hat{\delta}([\epsilon], w) = [w]$, and
  - $[w] \in \{[x] \mid x \in L \cap X\} \iff w \in L$.
- The first condition is true because of the way the indistinguishable relation is defined. If $[xa] \neq [x'a]$, then xa and $x'a$ must be distinguishable by L which would imply that for some z, one of the strings xaz and $x'az$ belonged to L and the other didn't. In that case, however, x and x' would be distinguished by the string az. If $[x] = [x']$, x and x' must be indistinguishable. Thus, δ is well defined.
- We can show the third condition by induction on the length of w. It is clearly true for $w = \epsilon$. Suppose it is true for w and consider the a string of the form wx. By definition, $\hat{\delta}([\epsilon], wx) = \delta(\hat{\delta}([\epsilon], w), x) = \delta([w], x) = [wx]$.
- For the final condition, suppose that $[w] \in \{[x] \mid x \in L \cap X\}$. We know that there is some $x \in L \cap X$ such that $[w] = [x]$ which implies that $w \equiv_L x$. Therefore, for any $z \in \Sigma^*$, wz and xz must either both belong to L or neither be in L. Consider $z = \epsilon$. This implies that $w \in L$. In the opposite direction, if $w \in L, w \equiv_L x$ for some $x \in X$ and therefore $[w] = [x] \in \{[x] \mid x \in L \cap X\}$

Minimization of DFAs

1. Given that we now know that for any regular language there is a DFA of size equal to the index of the language it recognizes, we would like to have a way to algorithmically find this DFA given any precise description of the language (i.e., a DFA, an NFA, or a regular expression).

2. Given a regular expression, we can construct a NFA for the language

using the constructions embedded in the proofs that regular languages are closed under the union, concatenation and closure.

3. Given an NFA, we can build an equivalent DFA using the subset construction presented earlier.

4. All we need is a way to convert a non-minimal DFA into one of minimal size (or to realize that the one we started with was already minimal).

5. We can precisely specify when two states can be merged by defining state equivalence formally for two states $p, q \in Q$ as:

$$p \approx q \iff \forall w \in \Sigma^*, \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$$

6. It should be clear that this notion of equivalence of states is in fact reflexive, symmetric and transitive. Therefore, it partitions the set of states of a DFA into equivalence classes.

7. This suggests a way that we could use the equivalence relation on the states of a DFA to determine the minimal DFA. Namely, if $[q]$ denotes the equivalence class of state q induced by the state equivalence relation we just defined:

   Given $M = (Q, \Sigma, \delta, s, F)$ define

$$M' = (\{[q] \mid q \in Q\}, \Sigma, \delta', [s], \{[f] \mid f \in F\})$$

   where $\delta'([p], x) = [\delta(p, x)]$.

8. As in the proof of the Myhill-Nerode theorem, we should be careful to verify that δ' is well defined, behaves as desired, and that the set of final states is appropriate. We won't.

9. Instead, we will consider an algorithm that determines which states are equivalent to one another (by actually determining which states are not equivalent to one another).

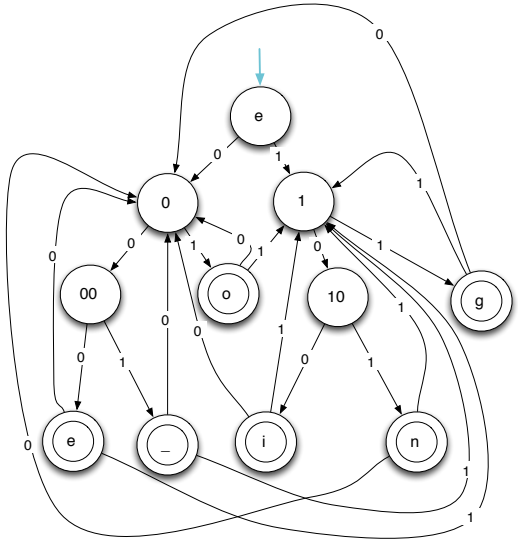
10. The basis of the algorithm is a somewhat recursive definition of not being equivalent. The base case is basically

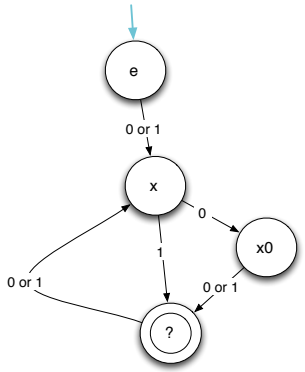
$$p \not\approx q \text{ if } p \in F \iff q \in F$$

and the recursive clause is

$$p \not\approx q \text{ if } \exists w \in \Sigma^*, \hat{\delta}(p, w) \not\approx \hat{\delta}(q, w)$$

11. In class this semester, I used a different machine as my example for how we can algorithmically compute the $\not\approx$ relationship. I didn't have time (or energy) to turn those slides into LaTeX. So, here I will present an old example based on the following DFA and it equivalent mimimal DFA:





12. The mechanics of the algorithm use a table in which we record all pairs of states we can identify as non equivalent. Each entry in the table reflects our knowledge of the relationship between the states at the top of its column and the right end of its row. For our example machine, the table starts out like this (with names like oh! and eee used to make it easy to distinguish the states for empty and zero from those for the letters O and E).

e										
	0									
		1								
			00							
				10						
					oh!					
						gee				
							eee			
								_		
									i	
										n

13. The first step is to use the basis step described above to realize that all final states are not equivalent to all non-final states. We record this by putting big X's in all of the cells in the table for such pairs..

e										
	0									
		1								
			00							
				10						
X	X	X	X	X	oh!					
X	X	X	X	X		gee				
X	X	X	X	X			eee			
X	X	X	X	X				_		
X	X	X	X	X					i	
X	X	X	X	X						n

14. Next, we use the recursive step over and over again for different pairs of states that still appear to be equivalent restricting our attention to strings w of length 1. For example:

- At this point in our table, the entry for the pair of states e, 0 is empty because these state might still be equivalent:

e										
?	0									
		1								
			00							
				10						
X	X	X	X	X	oh!					
X	X	X	X	X		gee				
X	X	X	X	X			eee			
X	X	X	X	X				_		
X	X	X	X	X					i	
X	X	X	X	X						n

- Looking back at the state diagram, we can see that on input 0, $\delta(e, 0) = 0$ and $\delta(0, 0) = 00$. Since the entry in our table for this pair of destinations (0, 00) is still empty, these states might be equivalent, so it would still appear that e and 0 might be equivalent.

- On the other hand, on input 1, $\delta(e, 1) = 1$ and $\delta(0, 1) = oh!$. The entry for the pair of states (1, oh!) in our table already has an X in it indicating we know these states are not equivalent. Therefore, we can conclude that e and 0 are not equivalent and record this fact with a new X in our table.

e										
X	0									
		1								
			00							
				10						
X	X	X	X	X	oh!					
X	X	X	X	X		gee				
X	X	X	X	X			eee			
X	X	X	X	X				_		
X	X	X	X	X					i	
X	X	X	X	X						n

15. We then continue methodically (we will go left to right and top to bottom) through the table considering all of the unmarked pairs:

(e,1) Since $\delta(e, 0) = 0$ and $\delta(1, 0) = 10$ and the pair (0, 10) is still unmarked in our table, we make no changes.

However, $\delta(e, 1) = 1$ and $\delta(1, 1) = gee$ and the states 1 and gee are known not to be equivalent, so we get to put another X in for e, 1:

e										
X	0									
X		1								
			00							
				10						
X	X	X	X	X	oh!					
X	X	X	X	X		gee				
X	X	X	X	X			eee			
X	X	X	X	X				_		
X	X	X	X	X					i	
X	X	X	X	X						n

(0,1) Since $\delta(0, 0) = 00$ and $\delta(1, 0) = 10$ and the pair (0, 10) is still unmarked in our table, we make no changes.

Similarly, $\delta(0, 1) = oh!$ and $\delta(1, 1) = gee$ and the states oh! and gee are still unmarked so we make no changes.

It is important to note, however, that in both cases, we are deciding whether the two states the machine would move into are not equivalent by checking to see if their entry in our table contains an X before we have even gotten to that entry. If, when we eventually process those entries we discover they should have X's, we will need to reconsider the pair (0,1). We won't do this by specially reconsidering (0,1). Instead, we will make an additional pass over all table entries that are still blank after the first pass.

(e,00) Since $\delta(e, 0) = 0$ and $\delta(00, 0) = eee$ and the pair (0, ee) is marked as non-equivalent, we get to mark (e,00)

Similarly, $\delta(e, 1) = oh!$ and $\delta(00, 1) = gee$ and the states oh! and gee are still unmarked so we make no changes.

e										
X	0									
X		1								
X			00							
				10						
X	X	X	X	X	oh!					
X	X	X	X	X		gee				
X	X	X	X	X			eee			
X	X	X	X	X				_		
X	X	X	X	X					i	
X	X	X	X	X						n

16. Continuing to consider every empty cell in the table in the same way until we reach (i,n), we eventually get the following:

e										
X	0									
X		1								
X	X	X	00							
X	X	X		10						
X	X	X	X	X	oh!					
X	X	X	X	X		gee				
X	X	X	X	X			eee			
X	X	X	X	X				_		
X	X	X	X	X					i	
X	X	X	X	X						n

17. At this point, as mentioned above, we need to reconsider all of the blank cells because when we considered them on the first pass we might have based our decision not to mark them on cells that we had not yet processed. In this case, on the second pass, we will discover that nothing actually changes. In general, we would keep making passes until nothing changes during one complete pass.

18. The information in the table justifies many simplifications of the original machine. It indicates that states 0 and 1 can be merged as can 00 and 10. It also says that all of the final states are equivalent and can be merged. Thus, the reduced machine will look like: