Extra on Regular Languages and Non-Regular Languages

CS 351

Decision Properties of Regular Languages

- Given a (representation, e.g., RE, FA, of a) regular language L, what can we tell about L?
- Since there are algorithms to convert between any two representations, we can choose the representation that makes whatever test we are interested in easiest.

Decision Properties

- Membership: Is string w in regular language L?
  - Choose DFA representation for L.
  - Simulate the DFA on input w.
- Emptiness: Is L = Ø?
  - Use DFA representation for L
  - Use a graph-reachability algorithm (e.g. Breadth-First-Search or Depth-First-Search or Shortest-Path) to test if at least one accepting state is reachable from the start state. If so, the language is not empty. If we can't reach an accepting state, then the language is empty.

Decision Properties

- Finiteness: Is L a finite language?
  - Note that every finite language is regular (why?), but a regular language is not necessarily finite.
  - DFA method:
    - Given a DFA for L, eliminate all states that are not reachable from the start state and all states that do not reach an accepting state.
    - Test if there are any cycles in the remaining DFA; if so, L is infinite, if not, then L is finite. To test for cycles, we can use a Depth-First-Search algorithm. If in the search we ever visit a state that we've already seen, then there is a cycle.

Decision Properties

- Equivalence: Do two descriptions of a language actually describe the same language? If so, the languages are called equivalent.
  - For example, we've seen many different (and sometimes complex) regular expressions that can describe the same language. How can we tell if two such expressions are actually equivalent?
- To test for equivalence our strategy will be as follows:
  - Use the DFA representation for the two languages
  - Minimize each DFA to the minimum number of needed states
  - If equivalent, the minimized DFA's will be structurally identical (i.e. there may be different names for the states, but all transitions will go to identical counterparts in each DFA).

Minimization

- To test for equivalence on the previous slide, we need some method to minimize a DFA (not in textbook)
- We say that two states p and q in a DFA are **equivalent** if:
  - For **all** input strings w, following the path for w from state q is an accepting state if and only if following the path for w from q is an accepting state.
  - i.e. if we have reached state p or q, then any other string we might get that will lead to an accepting state from p will also lead to an accepting state from q.
  - Also any string we get that does not lead to an accepting state from p also does not lead to an accepting state from q.
  - If this condition holds, then p and q are **equivalent**. If this condition does **not** hold, then p and q are **distinguishable**.
- The simplest case of a distinguishable pair is an accepting state p and a non-accepting state q. In this case, $\delta(p, \varepsilon)$ is an accepting state, but $\delta(q, \varepsilon)$ is not accepting. Therefore, any pair of (accepting, non-accepting) states are distinguishable.

Algorithm to Discover Distinguishable Pairs

1. Given an automaton A that has states $q_0 \ldots q_n$ make a diagonal table with labels 1 to n on the rows and 0 to n-1 on the columns.
2. Initialize the table by placing X's for each pair that we know is distinguishable. Initially, this is any pair of accepting and non-accepting states.
3. Let p and q be states such that for some input symbol a, $r = \delta(p, a)$ and $s = \delta(q, a)$. If the pair (r,s) is known to be distinguishable (i.e. they have an X in their table entry) then the pair p and q are also distinguishable. Place an X for (p,q) in the table.
4. Repeat step 3 until all pairs have been examined.
5. Repeat steps 3-4 again for any empty entries in the table. If no new X's can be placed, then the algorithm is complete. Any entries in the table without an X are pairs of states that are equivalent.

Once the equivalent states have been identified, they can be combined into a single state to make a new, minimized automaton. DFA's have unique minimum-state equivalents.

Simple Minimization Example

- Minimize the following DFA

(q,r) distinguishable?

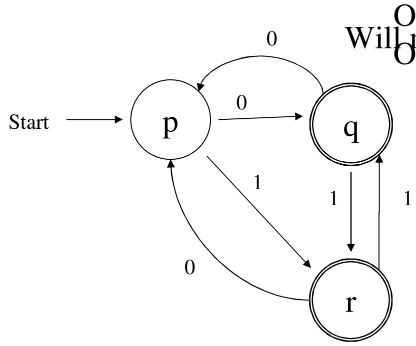
On 0: both to p, not distinguishable

On 1: to (q,r), no X, leave alone

"X": Known distinguishable states

Create Table:

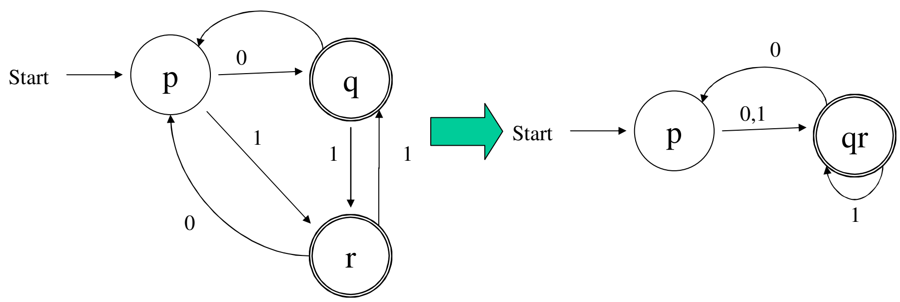
Will never distinguish (q,r), states equiv!



q	X	
r	X	
	p	q

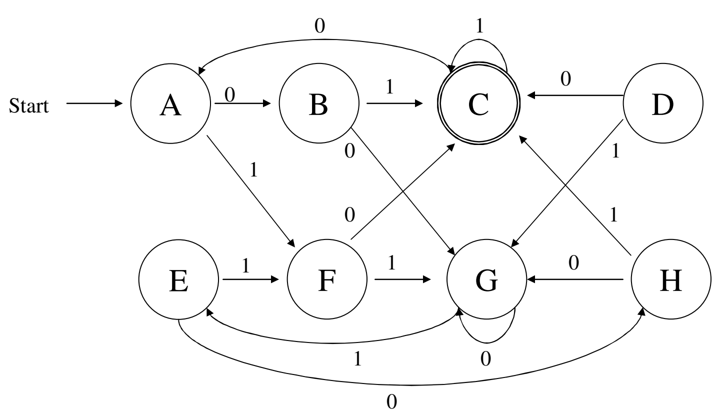
Simple Minimization Example

- Combine q,r:



Another Minimization Example

- Minimize the following:



Initial Table - Minimization

B							
C	X	X					
D			X				
E			X				
F			X				
G			X				
H			X				
	A	B	C	D	E	F	G

Table – After First Iteration

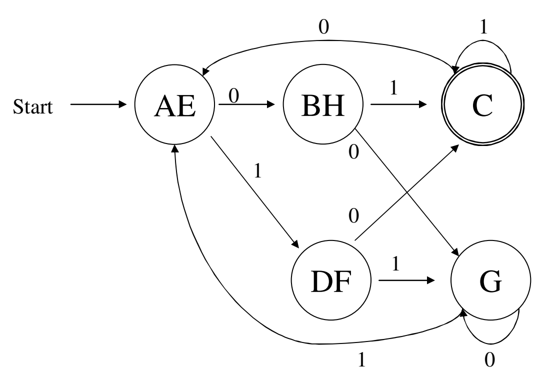
B	X						
C	X	X					
D	X	X	X				
E		X	X	X			
F	X	X	X		X		
G		X	X	X	X	X	
H	X		X	X	X	X	X
	A	B	C	D	E	F	G

Table – Last Iteration

B	X						
C	X	X					
D	X	X	X				
E		X	X	X			
F	X	X	X		X		
G	X	X	X	X	X	X	
H	X		X	X	X	X	X
	A	B	C	D	E	F	G

States (B,H), (A,E) and (D,E) are equivalent

Minimized DFA



Proving Languages Not Regular

- Before we show how languages can be proven not regular, first, how would we show a language is regular?
- Regular languages and automata seem powerful – after all they model everything we have seen so far!
  - But there are many simple examples that are not regular languages

The Pumping Lemma

- Our strategy for proving languages to be non-regular:
  - The Pumping Lemma states that all regular languages have a special property
    - If we can show that a language does not have this property, then the language cannot be regular.
    - The property states that all strings in a regular language can be "pumped" if they are at least as long as a special value, the **pumping length**.
    - This means that each such string contains a section that can be repeated any number of times with the resulting string remaining in the language.

The Pumping Lemma

- Let L be a regular language. Then there is a number p (the pumping length) where, if s is any string in L of length at least p, then s may be divided into three parts, s=xyz, satisfying the following conditions:
  1. $y \neq \varepsilon$ (but x and z may be ε)
  2. $|xy| \leq p$
  3. for each $i \geq 0$, $xy^iz \in L$

Pumping Lemma Explanation

- This theorem says:
  - when s is divided into xyz, either x or z may be empty, but y may not be empty.
  - x and y together must have length at most p.
  - we can always find a nonempty string y not too far from the beginning of s that can be “pumped”, i.e. it can repeat any number of times.
- Note that although $y \neq \varepsilon$, for the third condition, i may equal zero, giving us $y^0 = \varepsilon$, and then xz must be $\in L$.

Pumping Lemma Proof

- First, consider a simple case of a regular language L
  - In this language there are no strings of length at least p
  - In this case, the theorem becomes **vacuously** true.
    - The three conditions hold for all strings of length at least p if there aren't any such strings.
    - For example, if L is composed of simply the finite set { a }, then we could pick p=2 and the theorem is vacuously true because there are no strings of length at least 2.
      - This implies for any finite set of strings, the language is regular since we can pick a value p larger than the biggest string in L

Pumping Lemma

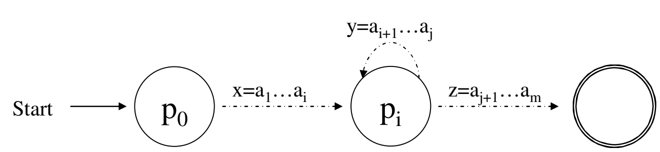
- More general case:
  - Suppose that L is regular. Then L=L(A) for some DFA A.
  - Suppose that A has n states.
  - Consider any string s of length n or more; say $s=a_1a_2\ldots a_m$ where $m \geq n$.
    - The DFA is in some state, defined as p_i, after reading the first i symbols of s.
    - By the pigeonhole principle, when we process input symbol i, since there are more (or equal) input symbols then there are states, we must repeat some state more than once!

More Pumping Lemma

- We must repeat some state more than once
  - Thus we can find two different integers i and j with $0 \leq i < j \leq n$, such that $p_i = p_j$. We can break s into s=xyz as follows:
    1. $x = a_1 a_2 \ldots a_i$
    2. $y = a_{i+1} a_{i+2} \ldots a_j$
    3. $z = a_{j+1} a_{j+2} \ldots a_m$
  - i.e., string x takes us to state p_i. Then we somehow repeat back to that state; at this point we are at input symbol j, and the corresponding state is p_j (where $p_i = p_j$). Finally we finish by moving to an accepting state.
  - Since i is less than j (not less than or equal to j) this means that we must have at least one symbol for y, so y may not be empty

Pumping Lemma Illustration

- The following automaton illustrates the pumping lemma



If this behavior is possible, then it means that xy*z must be in the language.

Using the Pumping Lemma

- To show that L is not regular, first assume that L is regular in order to obtain a contradiction.
- Then use the pumping lemma to guarantee the existence of a pumping length p such that all strings of length at least p can be pumped
- Find a string s in L that has length p or greater but cannot be pumped
  - This is demonstrated by considering all ways of dividing s into x,y, and z and showing that condition 3 is violated
- Since the existence of s contradicts the pumping lemma if L was regular means that L is not regular.

Pumping Lemma Proofs

- Tricky part - coming up with the string s.
  - This may take some creative thinking and trial and error, because for a non-regular language, some strings may fit the pumping lemma conditions, while others won't.
  - If a string works, you may need to pick another one until you find one that leads to the contradiction.

Example 1

- Let B be the language $\{0^n1^n \mid n \geq 0\}$. Use the pumping lemma to show that this is not regular.
  - Intuitively – not regular if you can't build a DFA for it
- Assume that B is regular.
  - Let p be the pumping length.
  - Choose s to be the string 0^p1^p. This string is clearly a member of B.
  - s has length at least p, so it is a valid choice.

For $s = xyz$, $|s| \geq p$
1. $y \neq \varepsilon$
2. $|xy| \leq p$
3. for each $i \geq 0$, $xy^iz \in L$

Example 1, Continued, 0^n1^n

- We have the following choices to split s into xyz, according to the constraints of the pumping lemma, where y may not be empty:
  - The string y consists only of 0's. Then we pick i=2. By condition 3, xy^2z should also be in B. But this results in the string xyyz. Since we added more 0's with the addition of another y, this is not in L.
  - The string y consists only of 1's. Then we pick i=2 and by the same logic as above, the string xyyz would have more 1's than 0's and this is also not in B.
  - The string y consists of 1's and 0's. Then we pick i=2 and xyyz may have the same number of 1's and 0's but now the 0's and 1's are not in the desired order (we needed to have all the 0's come before the 1's). Therefore, this string is not in B either.
- Contradiction no matter how s is chosen! B cannot be regular.

Example 2

- Let C = { w | w has an equal number of 0's and 1's }. Use the pumping lemma to show that this language is not regular.
- Assume that C is regular and let p be the pumping length.
- Choose s to be the string $(01)^p$. This is clearly of length at least p and is also in the language.
  - We split the string into an x, y, and z subject to the pumping lemma constraints.
  - Let's say we split it into $x=\varepsilon$, $y=01$, and $z=(01)^{p-1}$.
  - Can we find a value i such that xy^iz is not in C?
    - If i=0 then we just get the string xz, which is the string z. z has an equal number of 0's and 1's so it is in C.
    - If i=1 then we get the string xyz, which is also in C.
    - If we pick i=2 then we get the string xyyz, which is also in C.
    - No matter what value of i we pick, each resulting string is still in the language.
- This means that we didn't pick the right string to contradict the pumping lemma (or that the language actually is regular).

Example 2, cont.

- Try again by picking $s=0^p1^p$. This string is also clearly of length at least p and is also in the language.
- We break up the string s into xyz.
  - But we know that since $|xy| \leq p$, then x and y must consist entirely of 0's.
  - Based on condition 3, if we let i=2, then xyyz will have more 0's then there are 1's so this is not in C. Similarly, if we let i=0, then xz will have fewer 0's than 1's so this is also not in C (we can pick i equal to any value except 1).
  - Therefore, the language is not regular

Example 2 extra

- Given our selection of $s = 0^p1^p$
- What if we picked $x=\varepsilon$ and $z= \varepsilon$?
- That is, the string y contains the entire string. Then it would seem that xy^iz will still be in C, since y will contain an equal number of 0's and 1's.
- But since $|xy|$ must be $\leq p$ this selection is not valid since it would leave only y, and for y to equal 0^p1^p, violates the constraint of $|xy| \leq p$

Example 3

- Let $D = \{ww \mid w \in \{0,1\}^*\}$. In other words, pick w equal to any finite sequence of 0's and 1's. Then allow only those strings that have this word in it back to back. Show that D is non-regular using the pumping lemma.
- Assume that D is regular and let p be the pumping length. Choose s to be the string 0^p0^p. This is clearly of length at least p and is also in the language.
- Split the string into an x, y, and z. Let's say that it is split into $x=\varepsilon$, $y=00$, and $z=0^{2p-2}$. Can we find a value i such that xy^iz is not in D?
  - No, for any value of i the resulting string is still in D. Obviously this was not a good choice of a string s. Let's pick another one.

Example 3, cont.

- Choose s to be the string 0^p10^p1. This is clearly a member of D and has length of at least p.
- We split the string into an x,y, and z. Once again, since $|xy| \leq p$, we must have the case that x and y consist entirely of zeros.
  - If we pump y by letting i=2, then we now have more zeros in the first half of the string then in the second half, so the resulting string is no longer in D. Therefore, the language is not regular.

Example 4

- Let $E = \{0^i1^j \mid i > j\}$. Show this is non-regular using the pumping lemma.
- Assume that E is regular and let p be the pumping length. Choose s to be the string $0^{p+1}1^p$. This is clearly of length at least p and is also in the language.
- Split the string into an x, y, and z.
  - Since $|xy| \leq p$, both x and y must consist entirely of zeros.
  - If we pump y up, (i>0) then we end up with strings that are still in L.
  - If we pump y down by allowing i=0, then we get the string xz.
    - Removing the y decreases the number of 0's in s. But we constructed s so that there is exactly one more zero than one. So by removing a zero, we now have the same or fewer zero's than ones, so the resulting string is no longer in E.
- Therefore, the language is not regular.

Example 5

- Let F = { 1^{n^2} | n >=0 }. That is, each string consists of 1's and is of length that is a perfect square:
  - ε, 1, 1111, 111111111 , 1111111111111111, etc. (0, 1, 4, 9, 16, 25, 36, …)
- Notice that the gap between the length of the string grows in the sequence.
  - Large members of this sequence cannot be near each other.
  - If we subtract off the difference in length between successive elements, we get 1, 3, 5, 7, 9, 11, 13, etc. For position j where j >0, we get the difference from position j and j-1 as 2*j -1.

Example 5, cont.

- Assume that F is regular and let p be the pumping length.
- Let m = p^2. Choose s to be the string 1^m. This string is clearly in F and is at least of length p.
- Split the string into strings x,y, and z.
  - Consider the two strings xy^iz and $xy^{i+1}z$. Both of these strings must be in F.
  - These two strings differ only by a single repetition of y, or |y| which we know must be ≤ p.
  - But we saw that the length of the strings accepted by the language grows in length by 2j-1, not by some fixed amount |y|. Each time we pump the string we increase the value of j, so we can always find a value of j to make 2j-1 larger than |y|.
  - Consequently, we can always pick a large enough i such that assuming xy^iz is in the language, $xy^{i+1}z$ cannot be in the language because the length differential will be too small to equal to 2j-1.