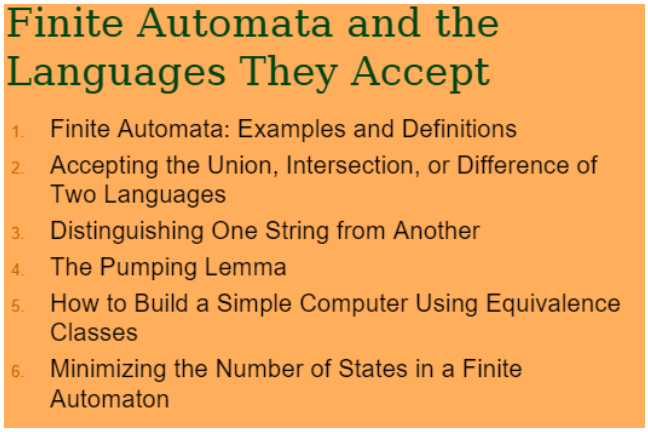
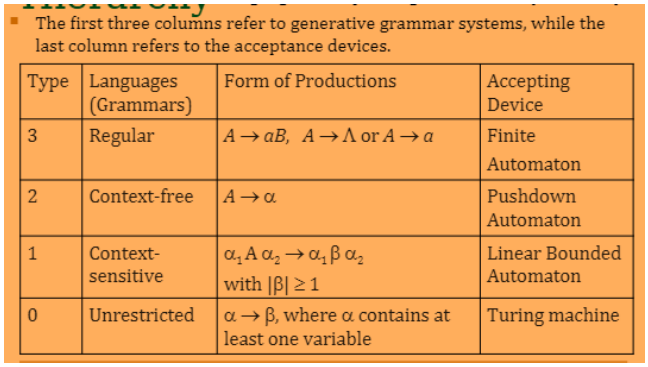
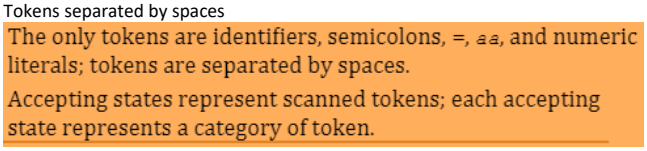
Lecture 2

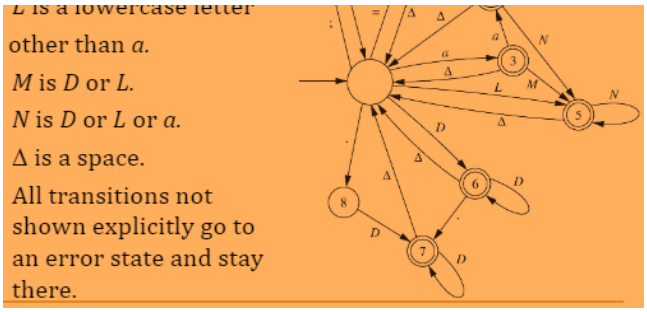
Tuesday, January 31, 2023 12:11 PM

- Finite automata and the lang they accept (overview)
  - 

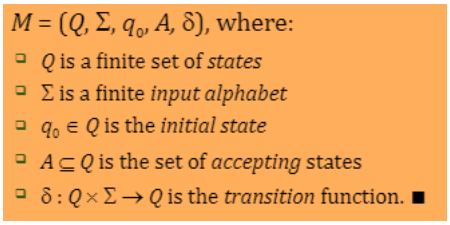
- Destination: Chomsky hierarchy
  - 4 levels of lang, chart, first 3 columns refer to generative grammar sys, last is acceptance devices
  - 

- Finite automata
  - It's a simple type of pc, output is yes/no, any pc that answers y/n acts as lang acceptor
    - For this ch, input in form of string of individual input symbols
  - Has finite number of states, move depends on what state currently in
  - States either accepting/nonaccepting
  - Initial state, accepting state if & only if lang FA accepts includes A(the symbol for the alphabet)
  - Diff arc diff labels
  - Deterministic finite automata
  - State based
  - States either accepting or nonaccepting
  - FA can be described by set of states, input alphabet, the initial state, the set of accepting states
    - Plates, parallel
    - Inner circle is smaller
    - Slide 9 is example
    - If a comes to me (being q0)
    - if forced to do once, its +, if repeated to itself, then *
    - + means option
  - Dead state is when it repeats itself, no matter what input
  - This is fun
  - Token
  - Lexical analysis
    - Takes string of chars and gives a string of tokens (have simple structure, like "41.3", "main")
    - Tokens separated by spaces
    - 

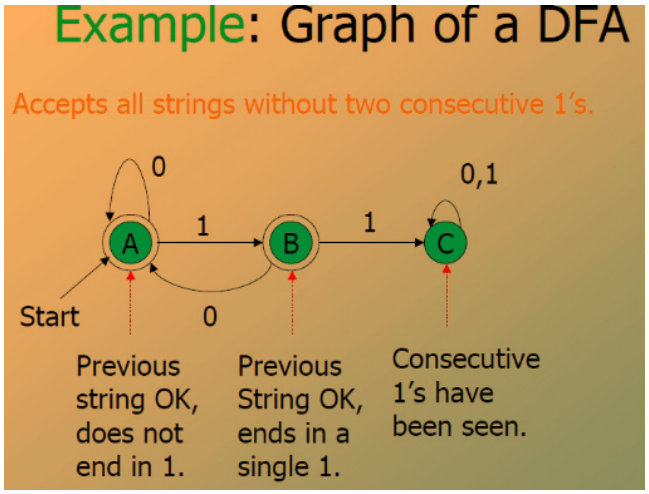
    - 



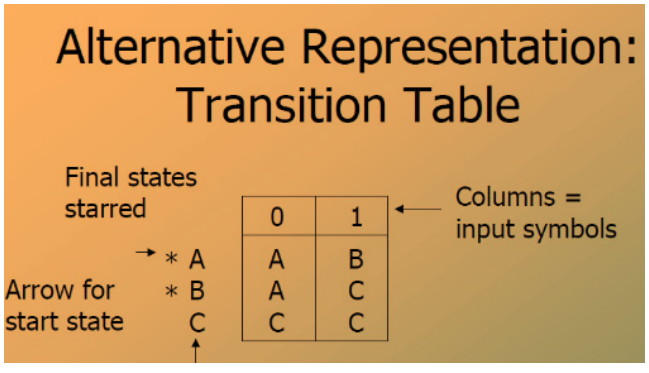
- Finite automaton has 5 elements, must be finite
  - Formula:



  - Q is finite set of states, the set of states
  - Sigma is finite input alphabet
  - Q0 is an element of Q is the initial state
  - A is a subset of Q is the set of accepting states
  - delta: Q x sigma -> Q Is the transition function
  - From state q, the machine will move to state delta(q,roe) if it gets input symbol roe
  - EX:
    - Start @ A, accepting state
    - Accepts all strings w/out 2 consec 1's
    - Dead end/dead state is like end, up till the end of the word entered
    - If 11 is not accepting, look like:
    - 11 not element of L(A)
    - (0* + 0*10 + 0*1)*
    - Is language for A and B (where + means or)
    - Transition table:
    - Left to top, read "from _ going on _" like from A* going on 0 will go to A, where * is accepting state
    - In matrix, list
    -

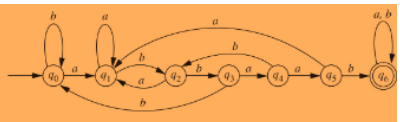


    -



Rows = states

- Definition
- **Finite Automata: Examples and Definitions (cont)**

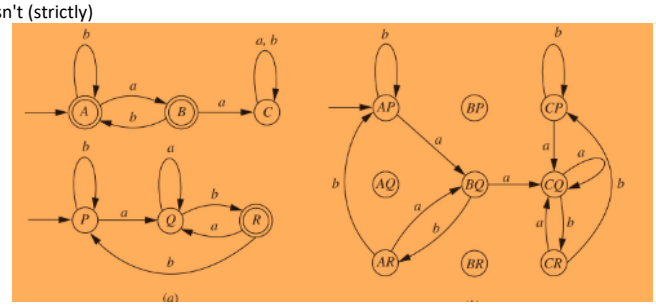


  - Suppose we wanted to evaluate $\delta^*(q_0, baa)$ in the example above.
  - This is easy using the diagram (follow the arrows), but let's use the recursive formula on the previous slide, to see how it works.
  - $\delta^*(q_0, baa) = \delta(\delta^*(q_0, ba), a) = \delta(\delta(\delta^*(q_0, b), a), a)$
    $= \delta(\delta(\delta^*(q_0, \Lambda b), a), a)$
    $= \delta(\delta(\delta(\delta^*(q_0, \Lambda), b), a), a)$
    $= \delta(\delta(\delta(q_0, b), a), a) = \delta(\delta(q_0, a), a)$
    $= \delta(q_1, a) = q_1$
  - We had to look at the diagram only in the last 3 steps, to get the values of δ.
- No one likes this type of recursion, have to wait till center
- Lang accepted by M is
- **Definition:**
  - Let $M=(Q, \Sigma, q_0, A, \delta)$ be an FA, and let $x \in \Sigma^*$. Then x is *accepted* by M if $\delta^*(q_0, x) \in A$ and *rejected* otherwise.

  The *language* accepted by M is
  $L(M) = \{x \in \Sigma^* \mid x \text{ is accepted by } M\}$. ■
- Slide 21 is membership problem help

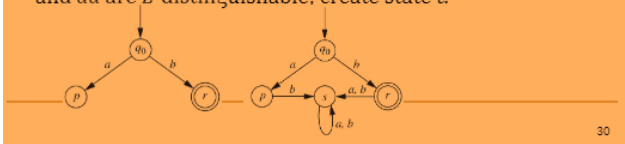
- Accepting the Union, Intersection, or Diff btwn 2 lang
  - Theorem:
  - **Two Languages (cont'd.)**
    - **Theorem:** Suppose $M_1=(Q_1, \Sigma, q_1, A_1, \delta_1)$ and $M_2=(Q_2, \Sigma, q_2, A_2, \delta_2)$ are FAs accepting L_1 and L_2.

      Let $M=(Q, \Sigma, q_0, A, \delta)$ be defined as follows:
      - $Q = Q_1 \times Q_2$
      - $q_0 = (q_1, q_2)$
      - $\delta((p, q), \sigma) = (\delta_1(p, \sigma), \delta_2(q, \sigma))$
    - Then, if:
      - $A = \{(p, q) \mid p \in A_1 \text{ or } q \in A_2\}$, M accepts $L_1 \cup L_2$
      - $A = \{(p, q) \mid p \in A_1 \text{ and } q \in A_2\}$, M accepts $L_1 \cap L_2$
      - $A = \{(p, q) \mid p \in A_1 \text{ and } q \notin A_2\}$, M accepts $L_1 - L_2$ ■
  - So U is either accepts, (upside down U) is both accepts, and – is first accepts and second one doesn't (strictly)
    -



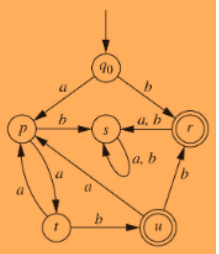
      - So for this, the U would be AP, AR, BQ, and CR (the other ones that are not connected are unreachable, so cannot be part of union)
      - (upside down U) would be just AR
      - - would be AP and BQ
  - Can simply the charts, do if no accepting states and no leaving
- Distinguishing One String from Another
  - Ex would be car sharing (carpooling)- have own car, one car for all ppl
  - Any 3 state FA (like one accepting the strings ending in aa) ignores/forgets lot of info
  - 2 strings or chars are distinguishable if end on different states,
    - Distinguishability- end up in accepting states and another end up in nonaccepting/diff state, it is distinguishable IF when you add 'z' and they are still on different states

- / means cutting strings, so opp of concatenation
- X & y are L-distinguishable if L/x not equal L/y, where L/x = {x element of sigma star | xz element of L}
- L=[4091234567], L/409 = [1234567]
- ' * ' Means goes to accepting state, ',' means distributive, or 'or' (comma means 'or'
- To create an FA to accept $L = \{aa, aab\}^*\{b\}$, we notice first that Λ is not in L, b is in L, a is not, and Λ and a are L-distinguishable (for example, $\Lambda b \in L$, $ab \notin L$).
  - We need at least the states in the first diagram.
  - L contains b but nothing else that begins with b, so we add a state s to take care of illegal prefixes.
  - If the input starts with aa we, need to leave state p because a and aa are L-distinguishable; create state t.



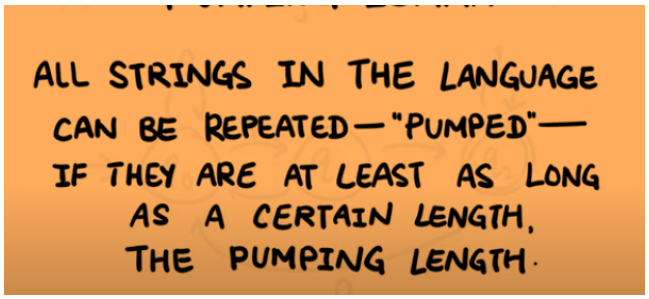
30

  - $\delta(t, b)$ must be accepting, because $aab \in L$; call that new state u.
  - Let $\delta(u, b)$ be r, because $aabb$ is in L but not a prefix of any other string in L.
  - States t and u can be thought of as representing the end of an occurrence of aa or aab; if the next symbol is a it's the start of a new occurrence, so go back to p.
  - The result is shown here.



- So here, no matter if you choose aa or aab, the b gets added to the end, and these are the only combo's that get you to an accepting state

- Pumping Lemma
  - Pigeon house principle, at one point will have loop
  - Pumping Lemma- can traverse loop any number of times
  - https://www.ling.upenn.edu/courses/ling106c/PumpingLemma#:~:text=What%20does%20Pumping%20Lemma%20say%3F&text=The%20Pumping%20Lemma%20says%20that,which%20is%20the%20pumping%20length).
  - Common application for it is showing that lang cannot be accepted by a FA, bc doesn't have the properties that the pumping lemma says are req for every lang that can be
    - Proof by contradiction- let n be the int in the pumping lemma, suppose lang can be accepted by an FA
    - Choose string x w/ |x| >= n which can apply the lemma to get a contradiction
    - Assume L is accepted by an FA
    - Choose x = a(^n)b(^n); then x as element of L, and |x|>=n
    - By pumping lemma, there are strings u,v,and w such that x=uvw and 3 conditions hold
    - Bc |uv| <= n, and x start w/ n # of a's, all the symbols in u and v are a's, so v=a(^k) for some k>0
    - Uvvw is an element of L, so a(^n+k)b(^n) is an element of L-------this is our contradiction, conclude that L cannot be accepted by an FA
    - EX:
    - The Pumping Lemma (cont'd.)
      - Let us show $L = \{a^{i^2} \mid i \geq 0\}$ is not accepted by an FA.
        - Suppose L is accepted by an FA, and let n be the integer in the pumping lemma.
        - Choose $x = a^{n^2}$.
        - $x = uvw$, where $0 < |v| \leq n$.
        - Then $n^2 = |uvw| < |uv^2w| = n^2 + |v| \leq n^2 + n < (n+1)^2$.
        - This is a contradiction, because $|uv^2w|$ must be i^2 for some integer i (because $uv^2w \in L$), but there is no integer i whose square is strictly between n^2 and $(n+1)^2$. ■
    - To check if language empty, just check is there is any accepting states
    - PUMPING LEMMA:

- 

  - https://www.youtube.com/watch?v=qtnNyUlO6vU
- How to Build a Simple Computer Using Equivalence Classes
  - M (an FA accepting the lang of strings ending in aa), (get a*) shown that 3 states needed, why is that enough?
    - 3 states are for the sets of strings:
    - Either not ending in a
    - End in a, but not aa
    - Those ending in aa
  - The initial state should be the empty language, bc when start we have received no input
  - Accepting state should be equivalence class of strings ending in aa, since that's lang we wanna get
  - Transitions defined in a natural way;
    - Take any element x of one class, and consider xa or xb
    - New string is in some equivalence class
    - The a-transition or b-transition from the set w/ x simply goes to that class
  - Theorem (Myhill-Nerode): L □ □* can be accepted by an FA if and only if the set Q_L of equivalence classes of the relation I_L (that is, □*/I_L) on □* is finite.
  - Easier to make an FA directly than to determine the set of equivalence classes, theorem serves to answer question of how much comp accepting a lang L needs to remember about the current string x: only its equivalence class
  - Id the equivalence classes, if we already have an FA accepting L, not too hard
    - 1. For each state q, we define $L_q = \{x \in \Sigma^* \mid \delta^*(q_0, x) = q\}$.
    - 2. Every L_q is a subset of some equivalence class of I_L (is the whole class if the FA has as few states as possible).
  - The theorem gives way to show lang cant be accepted by an FA
    - Consider the equivalence classes of I_L for $L = AnBn$:
      - for $i \neq j$, a^i and a^j are L-distinguishable, because $a^ib^i \in L$ and $a^jb^i \notin L$.
      - This implies that there are an infinite number of equivalence classes, and thus there can be no FA accepting L.
- Minimize the Numb of States in Finite Automaton
  - Suppose $M = (Q, \Sigma, q_0, A, \delta)$ accepts $L \subseteq \Sigma^*$.
    Define $L_q = \{ x \in \Sigma^* \mid \delta^*(q_0, x) = q\}$.
  - First step is to remove every state q for which Lq is null, along with transitions from these states (wont affect lang)
  - Now define ≡ on Q: $p \equiv q$ means that strings in L_p are L-indistinguishable from strings in L_q.
    This is the same as saying L_p and L_q are subsets of the same equivalence class of I_L.
  - 2 strings (x & y) are L-distinguishable if, for some string z, exactly one of xz, yz is in L
  - Therefore, $p \not\equiv q$ if, for some string z, exactly one of the states $\delta^*(p, z)$, $\delta^*(q, z)$ is in A.
  - Sm is set of unordered pairs (p,q) of distinct states satisfying p (not in relation) q
    - Systematic way to find Sm is
    - If exactly 1 p,q is in A, then (p,q) is an element of Sm
    - For every pair of states r and s, and every symbol roe, if
      - For every pair of states r and s, and every symbol σ, if $(\delta(r, \sigma), \delta(s, \sigma)) \in S_M$, then $(r, s) \in S_M$.
  - An algorithm to identify the pairs (p, q) with p ≡ q:
    - 1. List all unordered pairs (p, q) of distinct states.
    - 2. Make a sequence of passes through these pairs:
    - 3. On the first pass, mark each pair (p, q) so that exactly one of the two states is in A.

4. On each subsequent pass, and each unmarked pair (r, s), if $\delta(r, \sigma) = p$ and $\delta(s, \sigma) = q$ for some $\sigma \in \Sigma$, and (p, q) is marked, then mark (r, s).
5. After a pass in which no pairs are marked, stop.
6. The marked pairs are the pairs (p, q) for which $p \not\equiv q$.

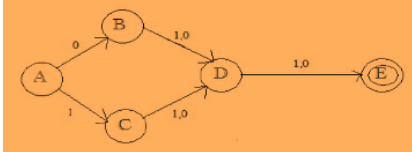
- When algorithm ends, unmarked pairs (p,q) rep 2 states that can be combo into 1
- Make 1 final pass thru the states
- First state rep state in the new min FA
- Every subsequent state q finds a new state only if (p,q) is marked for every p considered previously

- Algorithm for Marking states follows the inductive
  - Make table called DISTINCT w/ an entry for each pair of states- table cells initially blank
  - For every pair of states (p,q) - if p is final and q is not (or opposite), set DISTINCT (p,q) to be x
  - Loop until there no change in table contect
    - For each pair of states (p,q) and each char a in the alphabet
    - If DISTINCT(p,q) is empty and DISTINCT
    - Set DISTINCT (p, q) to be x

(2) Loop until there is no change in the table contents
- For each pair of states (p, q) and each character a in the alphabet
- if DISTINCT(p, q) is empty and DISTINCT (δ(p, a), δ (q, a) is not empty
- Set DISTINCT(p, q) to be x.

(3) Two states p and q are distinct iff DISTINCT (p, q) is not empty.

Let us consider the following FA with five states (A – initial, E –accepting):



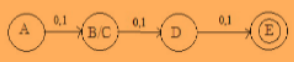
The table DISTINCT after executing step (1) and one iteration of step (2) is:

B				
C				
D	×	×	×	
E	×	×	×	×
	A	B	C	D

- The table DISTINCT after executing another iteration of step (2) is:

B	×			
C	×			
D	×	×	×	
E	×	×	×	×
	A	B	C	D

- The minimized automata is:



Summary

- Finite Automata and the Languages They Accept

1. Finite Automata: Examples and Definitions
2. Accepting the Union, Intersection, or Difference of Two Languages
3. Distinguishing One String from Another
4. The Pumping Lemma
5. How to Build a Simple Computer Using Equivalence Classes
6. Minimizing the Number of States in a Finite Automaton

- Look at delta star as future
- Set of unordered pairs (p,q) of distinct states satisfying p (not in relation with) q
- An algorithm to id pairs (p,q) w/ p (not related to) q
  - List all unordered pairs (p,q) of distinct states
  - Make sequence of passes thru these pairs
  - On the first pass, mark each pair (p,q) so that exactly one of the 2 states is in A
  - On each subsequent pass, and each unmarked pair (r,s), if delta(r,roe) = q and delta(s, roe) = q for some roe is an element of sigma, and (p,q) is marked, then mark (r,s)
  - After a pass in which no pairs are marked, stop
  - The marked pairs are the pairs (p,q) for which p (is not related to) q
- Algorithm for marking distinct states follows this inductive definition
- Create a table distinct w/ an entry for each pair of states, table cells initially blank