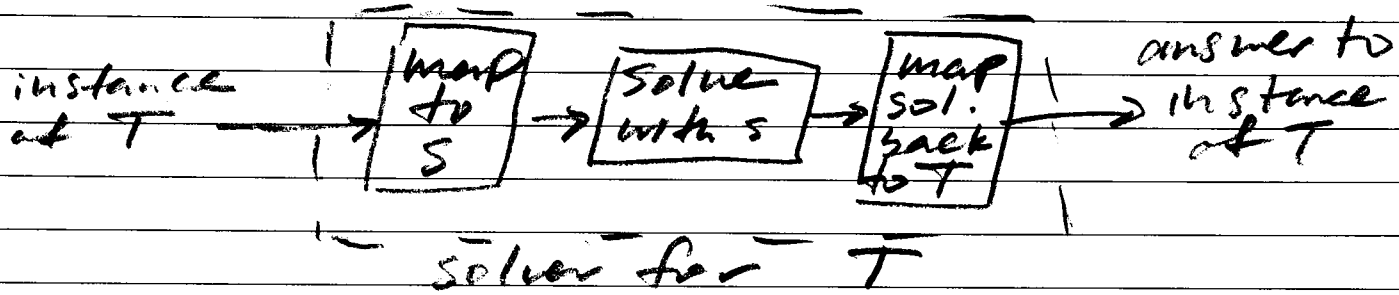
Theory of Computing

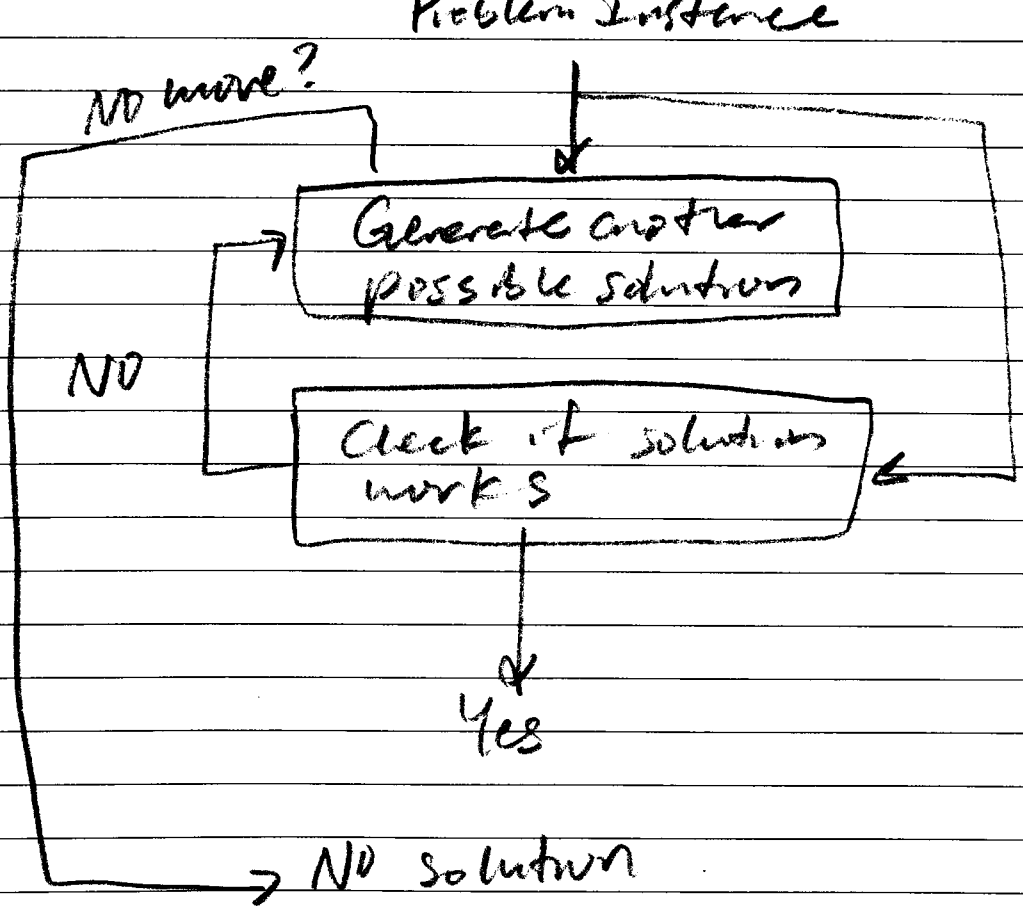
Date. 8.28.25
No. 2

Suppose you have 2 problem classes S & T with solver for S & T. If S can be used to solve T, what goes where?



called a Reduction: reducing problem T to S

Verifier: don't know how to solve problem, but can check if an answer is right?



& can enumerate all possible solutions

Date.
No.

Objects: can be "compared"

- Numbers: $\mathbb{N}, \mathbb{W}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$
  - → $\mathbb{W} = \{0, 1, 2, 3, \ldots\}$
  - → $\mathbb{N} = \{1, 2, 3, \ldots\}$

- Sets: unordered collection of objects
  - → $P(A)$ = "power set" - set of all subsets
  - → written $\{a, b, c, \ldots\}$ or $\{x \mid x \text{ has property}\}$
  - → $\{\}$ or $\emptyset$ is an object
  - → operators $\|, \subseteq, \subset, \cup, \cap$

ex. $A = \{0, 1, 2\}$
$P(A) = \{\{\}, \{0\}, \{1\}, \{2\}, \{0,1\}, \{0,2\}, \{1,2\}, \{0,1,2\}\}$

$|A| = n$, $P(A) = 2^n$

- Tuples: ordered collection of objects
  - → k-tuple: $(a_1, a_2, \ldots, a_k)$
  - → cartesian product: $A_1 \times A_2 \times \ldots \times A_k$
    set of all k-tuples $(a_1, a_2, \ldots, a_k)$ $\forall a_i \in A$

ex. $A_1 = \{0, 1, 2\}$, $A_2 = \{a, b\}$

$((0,a), (1,a), (2,a), (0,b), (1,b), (2,b))$

$|A_i| = n_i$

$\prod_{i=1}^{k} |A_i|$

Date.
No.

- **Strings**: linear sequence of characters

→ Alphabet: Σ = set of chars in the string
→ Concatenation: join 2 strings
→ Strng projection: $\pi_\Sigma(s)$: s without Σ
→ Kleene Closure: A^* = set of all strngs from A

ex. $A = \{0, 1\}$
$A^* = \{0, 00, 000, \ldots$
$01, 011, \ldots$
$101, \ldots$
$\ldots\}$ i.e. infinite

Graphs: defined by a tuple (V, E)

V: set of vertices
E: set of edges $= \{(u, v) \mid u, v$ are vertices$\}$

weighted graph may have numbers on edges
defined by (V, E, W) s.t. $W(E) \in \mathbb{Z}$
W = function

can be represented in an **adjacency matrix**

also has:
- subgraphs
- paths
- cycles
- directed / undirected
- tree

Date.
No.

Properties of Graphs:

- Degree: # of edges that start/end with v
- Path: sequence of vertexes connected by edges
- Circuit: path that starts/ends on same vertex
- Hamiltonian Path: Circuit that goes through each vertex once
- Clique: subset of vertices that is fully connected
- Spanning Tree: subset of edges that touch all vertices w/o cycle
- K-coloring problem: K-link needs $\geq k$ colors

Relations:

- K-place relation over sets $A_1, \ldots, A_k$ is a subset of $\underbrace{A_1 \times A_2 \times \ldots \times A_k}_{\text{set of k-tuples}}$
- Homogeneous relation: all A_i are the same
- Heterogeneous relation: not
- Predicate: use of relation R to test if a tuple is in it

ex. $ADD = \{(x,y,z) \mid x,y,z \in \mathbb{Z} \text{ and } z = x+y\}$

- Binary Relations: relation formed from 2 sets $A \times B$

Date.
No.

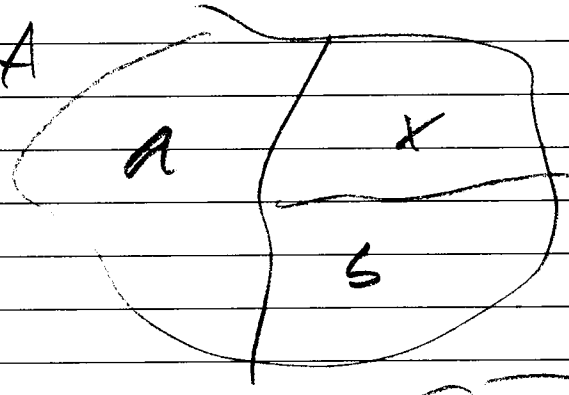
Properties of binary homogeneous relation R over $A \times A$

- Reflexive: $(a,a) \in R \quad \forall a \in A$
- Symmetric: $(a,b) \in R \rightarrow (b,a) \in R$
- Transitive $(a,b) \in R$ and $(b,c) \in R$ $\rightarrow (a,c) \in R$

Equivalence Relation: R has all 3 properties

- Partitions A into disjoint **equivalence classes**

$$V_a = \{ a \mid (a,x) \in R \}$$



no overlap!

ex. $\{0, 1, \ldots\}$ (mod 3) relation

$\{(0,0), (3,0), (6,0) \ldots$
$(1,1), (4,1), (7,1) \ldots$
$(2,2), (5,2), (8,2) \ldots$
$\}$

nice disjoint EC

Date.
No.

Functions: binary relation f over $A \times B$
written $f : A \to B$

- if $(a,b) \in f$ and $(a,c) \in f$ then $b = c$
- A is domain (may be cartesian product)
- B is range

Properties of some $f : A \to B$

- one-to-one: if $f(x_1) = f(x_2) \to x_1 = x_2$
- onto: $\forall b \exists a$ s.t. $f(a) = b$

note. f in of $k \geq 1$ arguments domain
$A = A_1 \times A_2 \times \ldots \times A_k$

Ex. $\mathbb{Z}^+ = \{((a,b),c) \mid a,b,c \in \mathbb{Z}, c = a-b\}$

Binary Functions: $k = 2$

IE, 1 object $\to$ 1 object

DQ. Assume you have 0 and a f that adds 1 to a number, how can you make 3?

$3 = f(f(f(0)))$

DQ. Assume you have $\mathbb{W}$, how can you build any $z \in \mathbb{Z}$?

$(a,b) \equiv a-b$ $\quad (0,1) = -1$, $(1,2) = -1$

DQ. Assume you have $\mathbb{W}$, how can you make any $r \in \mathbb{Q}$?

$(a,b) \equiv \frac{a}{b}$ s.t. $b \neq 0$

Proof Techniques

Date. 9.2.25
No. 03

Proof. a defensible argument that a statement is true

- Proof by Construction
- Proof by Contradiction
- Proof by Induction
  - base case: true for a_0
  - induction hypothesis: assume T up to a_k
  - induction step: prove for a_{k+1}

Construction: provide a method/program to create an object with a sought property

Contradiction: Assume thm is false and show that creates a contradiction with something known as true

Induction: Show all elements of an enumerable infinite set with a "starting value" have some property

Ex. k-regular graph - each vertex has degree k

Prove. $\forall$ even $n \geq 2$, there is a 3-regular graph with n vertices

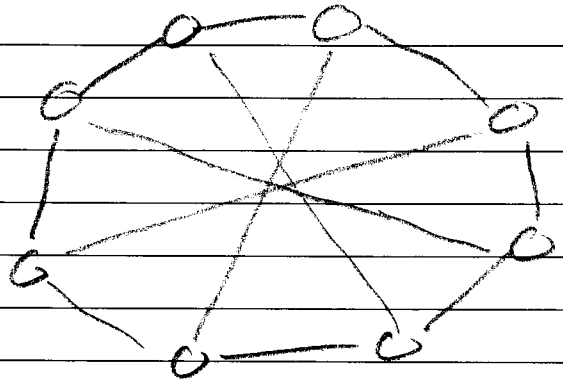
Proof by construction. $\forall n$, build a graph

- with n vertices $\{0, 1, \ldots, n-1\}$ arranged in a circle (n even ≥ 2)
- edges $= \{(i, i-1) \mid 0 \leq i \leq n-2\} \cup \{(n-1, 0)\}$ — these connect each vertex w/ its 2 neighbors
  $\cup \{(i, i+n/2) \mid 0 \leq i \leq n/2-1\}$ — this adds one more edge

ie. graph w/ n vertices with degree 3 $\forall$ vertices

Date.
No.

$n = 8$



all vertices have degree 3

Ex. $a, b \in \mathbb{R}$. one of them $\neq 0$. then
$\exists (c+di) \in \mathbb{C}$ s.t. $(a+bi)(c+di) = 1$

Proof by Construction - $\forall a, b \in \mathbb{R}$, construct $c+di$

1. $a+bi = \frac{1}{c+di}$

2. But $\frac{1}{c+di} = \frac{1}{c+di} * \frac{c-di}{c-di} = \frac{c-di}{c^2+d^2} =$

3. Solve for c and d in terms

$$a = \frac{c}{c^2+d^2} \qquad b = \frac{-d}{c^2+d^2}$$

Proof by Contradiction:

$\forall x$ with property P, $f(x)$ has property Q

Assume the opposite:

$\exists x$ with property p where $f(x)$ does not have property q

Show this leads to a contradiction

Date.
No.

Ex. The sum of two even integers is always even

Assume. $\exists a, b$ not even st $a+b=c$ is not even

$a+b=c$

$2w+2z=2x+1$

$w+z=x+\frac{1}{2} \notin \mathbb{Z}$

contradiction.

Ex. The negative of any irrational is irrational

Assume. an irrational r where $-r$ is rational

Ex. $\sqrt{2}$ is irrational

Assume. $\sqrt{2}$ is rational

1. $\sqrt{2}=\frac{m}{n}$ $\forall m, n \in \mathbb{Z}$, $n \neq 0$,
2. m and n are fully reduced and have no common divisors
3. one must be odd (otherwise both divisible by 2)
4. $n\sqrt{2}=m$
5. $n^2 2=m^2$
7. m^2 is even
8. m is even, must $= 2k$ $\forall k \in \mathbb{Z}$
9. $n^2 2=m^2=4k^2$
10. $n^2=2k^2$
11. contradiction, both must be even

Date.
No.

Ex. Proof by Induction

Prove $\forall n,\ 1+2+\ldots+n = \frac{n(n+1)}{2}$

Basis Step. $n=1$

$1 = \frac{1(1+1)}{2} = 1$

Hypothesis. true up to $n=k$

ie. $1+2+\ldots+k = \frac{k(k+1)}{2}$

Induction Step. show true for $n=k+1$

$1+2+\ldots+k+k+1 = (1+\ldots+k)+(k+1)$

$= \frac{k(k+1)}{2} + (k+1)$

$= (k+1)\left(\frac{k}{2}+1\right)$

$= \frac{(k+1)(k+2)}{2}$ done

SAT: Boolean Satisfiability

wff: well-formed-formula constructed from
- a set of Boolean variables
- Boolean operations

Satisfiability: ∃ a substitution of 0s and 1s to variables to make the wff true

Unsatisfiability: no substitution makes all clauses true at the same time

n variables → 2^n combinations

only need 1 case that's true!

Date.
No.

CNF: Clausal Normal Form

- wff restricted as AND of a set of clauses
  - each clause an OR of a set of literals
  - each literal a variable or its negation

For a wff in CNF
- all clauses must be true
- for any clause to be true, at least one literal must be true

Ex. $(\neg x \vee y) \wedge (x \vee y) \wedge (x \vee \neg y)$
- $x=1, y=1$ makes expression true

Ex. $(\neg x \vee y) \wedge (x \vee y) \wedge (x \wedge \neg y) \wedge (\neg x \vee \neg y)$
- no assignment of vars makes this true
- unsatisfiable

note. can always convert to CNF

Applications:

- all computable functions can be converted to SAT problems
- if we can solve SAT quickly, we can solve all computable problem quickly
- this solution does not exist

Date. 9.4.25
No. 04

States & DFAs

SAT. Given expression using boolean variables and operators...

Is there an assignment to the values to the vars to make the expression TRUE

Satisfiable: at least **one** assignment gives TRUE

Unsatisfiable: **no** assignment gives TRUE

Tautology: **every** assignment gives TRUE

ex. $(A \vee B)$ – satisfiable $\{(1,0),(1,1),(0,1)\}$

note. N vars → 2^N assignments

ex. $(A \vee A)$ – tautology

ex. $A \wedge \bar{A}$ – Unsatisfiable

CNF: Clausal Normal Form

iff restructured as an AND of a set of clauses
- clause: OR of literals
- literal: var or its negation

1-SAT: all clauses have **exactly** 1 literal
ex. $x_1 \wedge x_2 \wedge x_3$

2-SAT: all clauses have **at-most** 2 literals

⋮

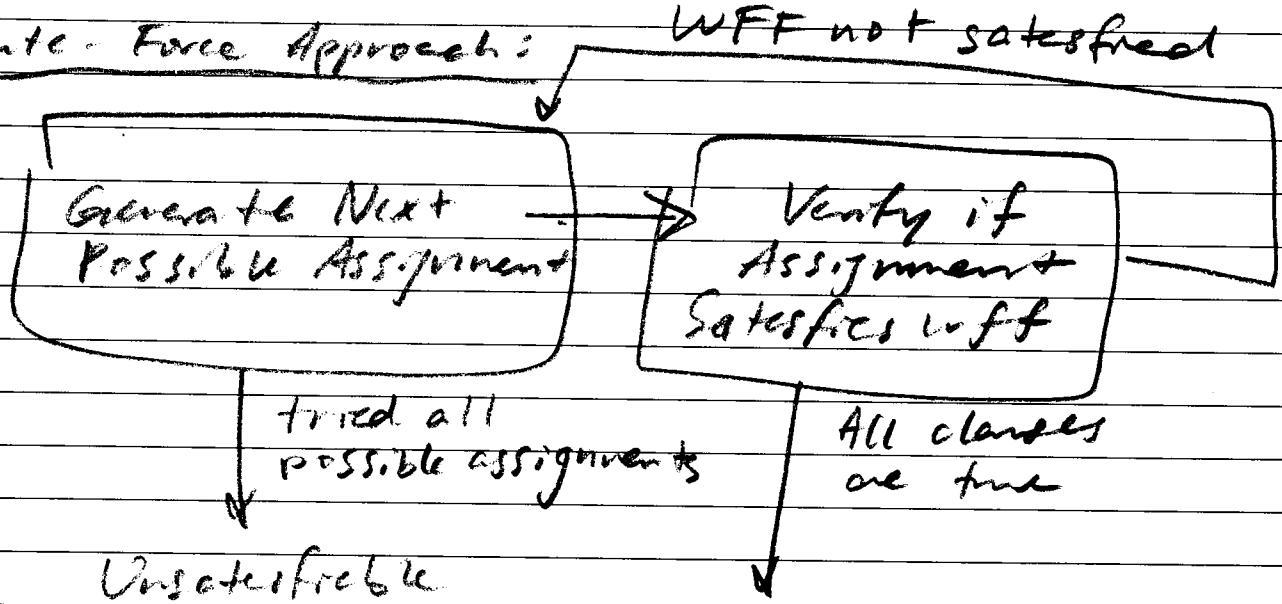
N-SAT: " " at-most N literals
- at least one-literal in each clause must be true

Date.
No.

World's Simplest SAT Solver: Truth Table!

but really slow, $O(2^N)$!

Brute Force Approach:



$2^{|V|} \cdot |C| \cdot |L|$

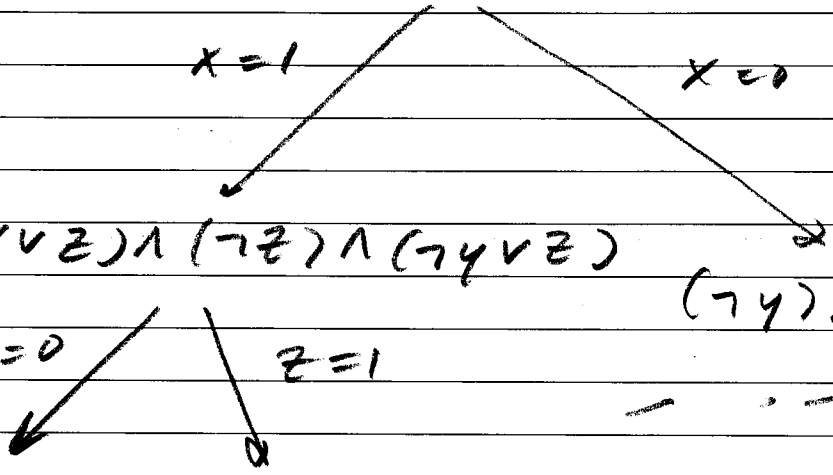
- $|V|$ = # variables
- $|C|$ = # clauses
- $|L|$ = # literals per clause

Backtracking: improved SAT solver

- see AI notes!

Date.
No.

Ex. $(x \vee \neg y) \wedge (y \vee z) \wedge (\neg x \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$

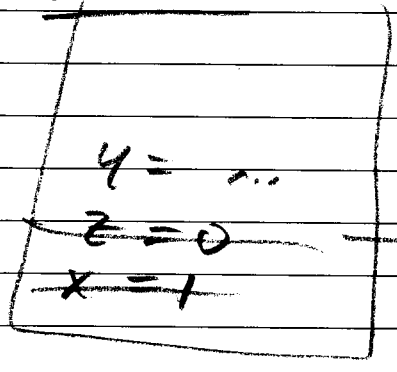


$(y) \wedge (\neg y)$ Unsat.

Unsat.

i.e. tree traversal!

Stack.



The Unit Clause Rule:

When a clause only has one undetermined literal complexity greatly reduced

2-SAT can be linear!

k-SAT still worst case 2^k

Date.
No.

State and Finite State Automata:

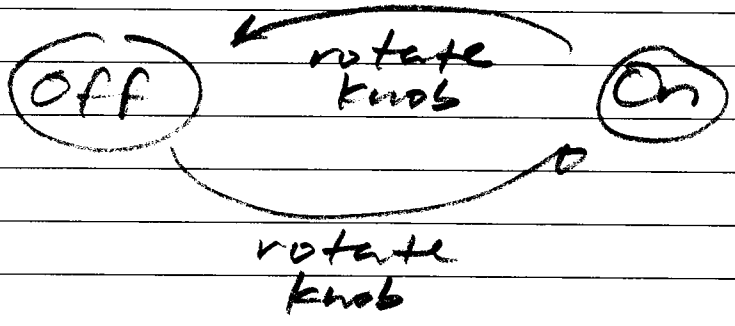
Goal. discussion of classical model, state, and computing w/ finite amount of state information

State. The condition (value) something is in at a specific time

Types of Systems:

- Stateless System: combinational logic
- Continuous System: state of water level in a lake
- Sequential System: changes in state at specific times at only discrete values

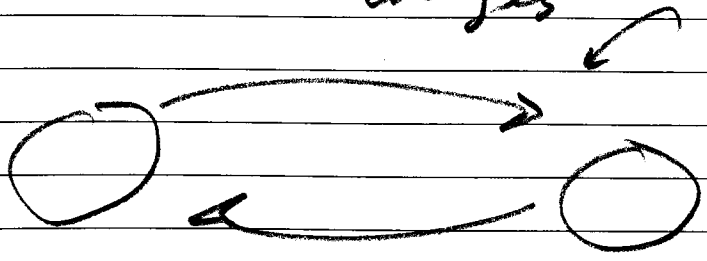
"Clockless" Finite State Machines (FSM):



Ex. Lamp

What we will study in this class

Clocked version: every "sword," the state changes



Date.
No.

State Transition Table

two ways :

Current State	Next Input	New State
...	...	...

or

current state	input	...
state	new state	...
...		

Initial Deterministic Finite Automata (DFA)

5-tuple : $(Q, \Sigma, \delta, q_0, F)$

object

Q : set of states
Σ : set of input symbols
$\delta : Q \times \Sigma \to Q$: transition function*
q_0 : initial state
F : set of final (or accept) states
- halt when we reach these states
- double circle on state diagram

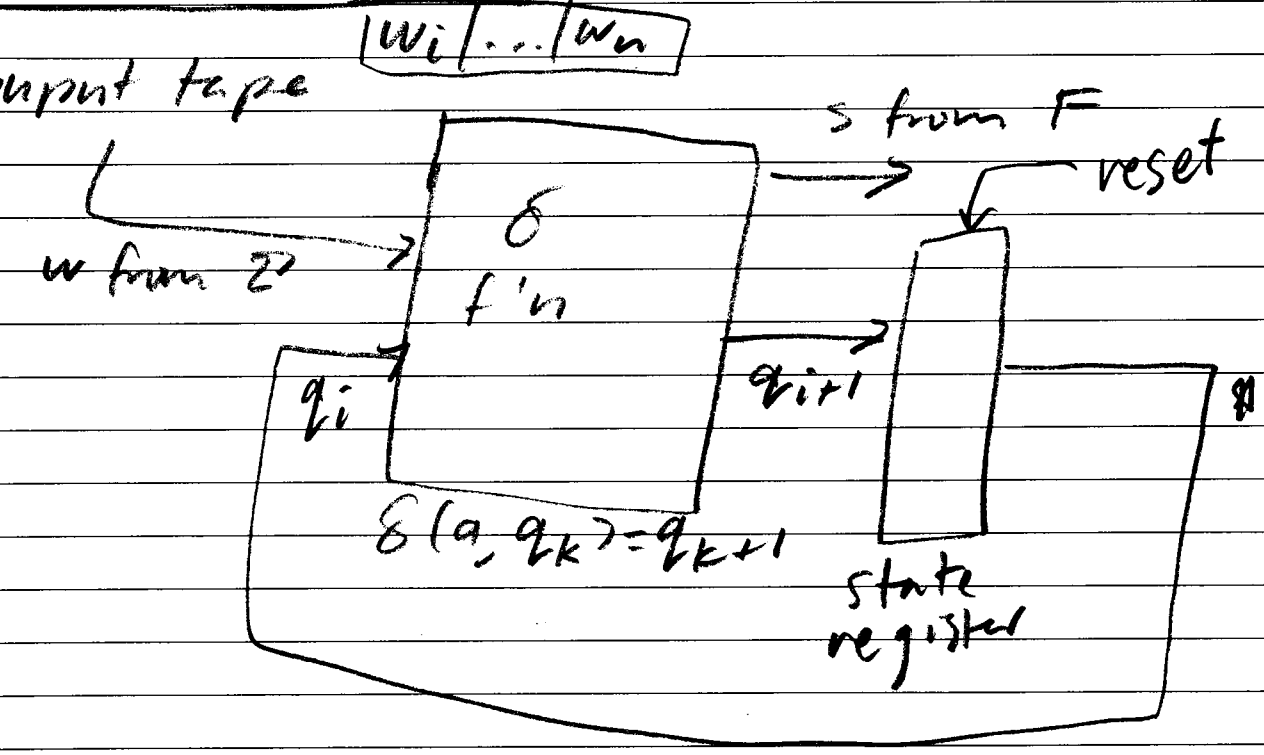
* given pair (q_i, w_i) return a new q_{i+1}

FSTs and Regular Language

Date. 9.9.25
No. 05

Aside. $\forall n$ if $P(n)$ then $Q(n)$
By contradiction
Assume $\exists n$ $P(n)$ is true,
and prove $Q(n)$ is false

Formal DFA Model:



signal if q_i is in F

Recall. 5 tuple $(Q, \Sigma, \delta, q_0, F)$

note. if state in F when last symbol from input read, automaton has accepted input

ie. machine stops at end of input

Date.
No.

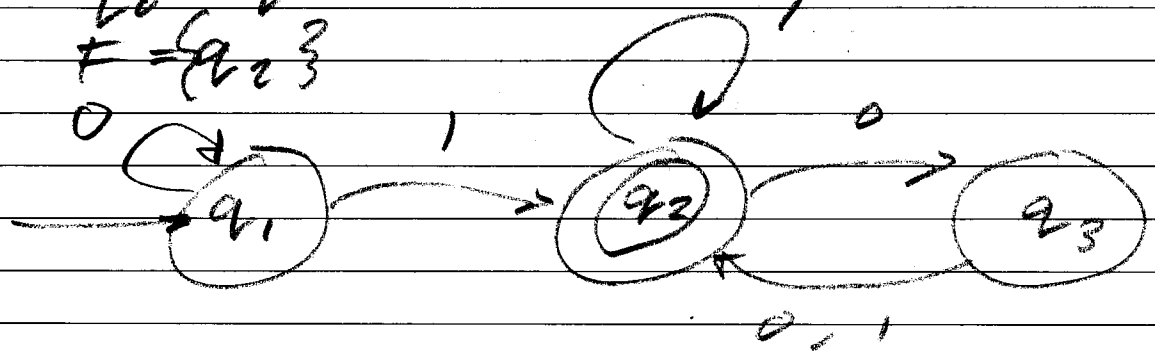
Ex. Sipser 1.4

$\Sigma = \{0, 1\}$

$\delta(q, q_k) = q_{k+1}$

$q_0 = q_1$

$F = \{q_2\}$



current state	next input	next state
q_1	0	q_1
q_1	1	q_2
q_2	0	q_3
q_2	1	q_2
q_3	0	q_2
q_3	1	q_2

Under what conditions does it end in a final state?

strings to get to $F = \{1, 11, (1^+), 0^*1, 0^*1^+, (0\{0,1\})^*\}$

i.e. $F = \{0^*1(1^*U1^*0\{0,1\})1^*\}$

note. $n^+ \rightarrow$ 1 or more n's
$n^* \rightarrow$ 0 or more n's

note. can always convert these into code!

Date.
No.

Finite Automata and Languages

- Language = set of strings
- FA M accepts a string if after last symbol, M is in F
- FA M rejects " " " not in F
- FA M recognizes language L if
  - → M accepts all strings in L
  - → M rejects all strings not in L

Notation: L(M) = languages M accepts (a set of strings)

{w} [w] tape → final state ∈ F

Accepting a String:

M has states $Q = \{q_0, q_1, \ldots, q_n\}$
Assume string $w = w_1, w_2, \ldots w_m$
Assume sequence of states $r_0, r_1, \ldots r_m$
where:

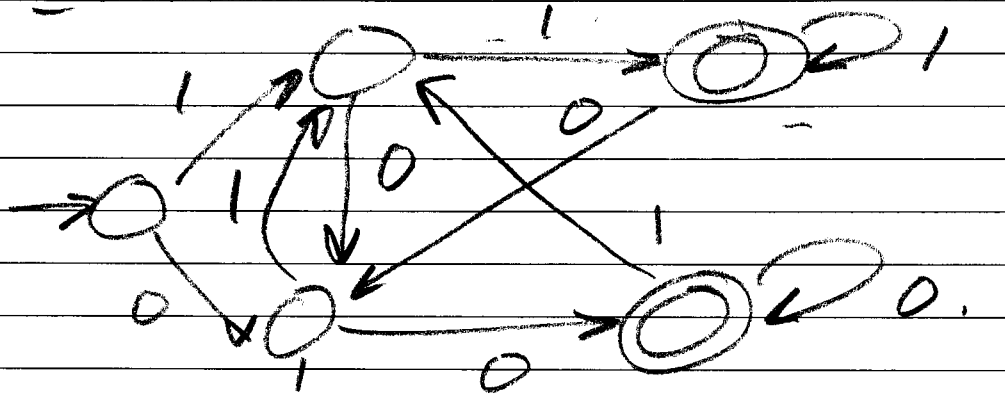
$r_i \in Q$
$r_0 = q_0$
$r_{i+1} = \delta(r_i, w_{i+1})$
$r_m \in F$

M accepts w

Date.
No.

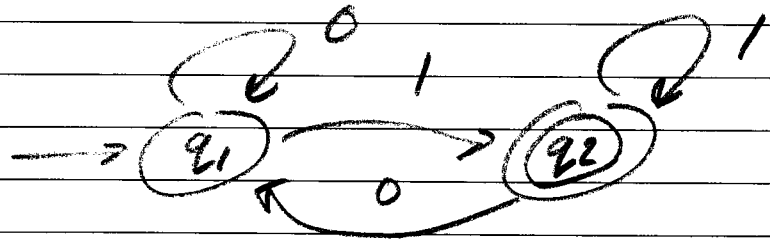
Ex. Draw a DFA which accepts when there is 00 or 11 at the end of a string from $\{0, 1\}$

ie. 01010100 but not 0001110



$L(M) = \{0,1\}^*(00 \cup 11)$

Ex. What's the Language?

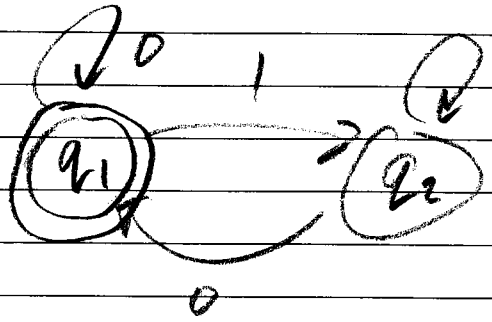


$L(M) = \{0,1\}^*1$ last char must be 1

Ex. 1.12 in worksheet

Ex. Sipser 1.10 - whats the Language?

$\Sigma = \{0,1\}$



any string that ends in 0 or no string (empty string)

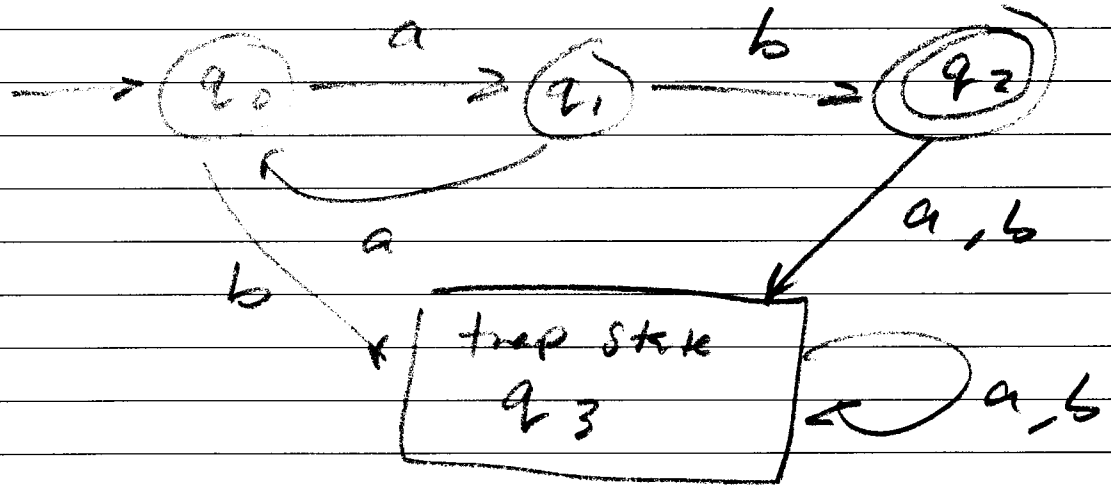
Date.
No.

Trap States:

δ should include all combos of state from Q and symbols Σ

... but may don't care

a trap state is where the don't care's go, and once there all symbols loop back to there



language: odd # of a's followed by one b

note. unless asked for, unused state transitions go to a trap state by default

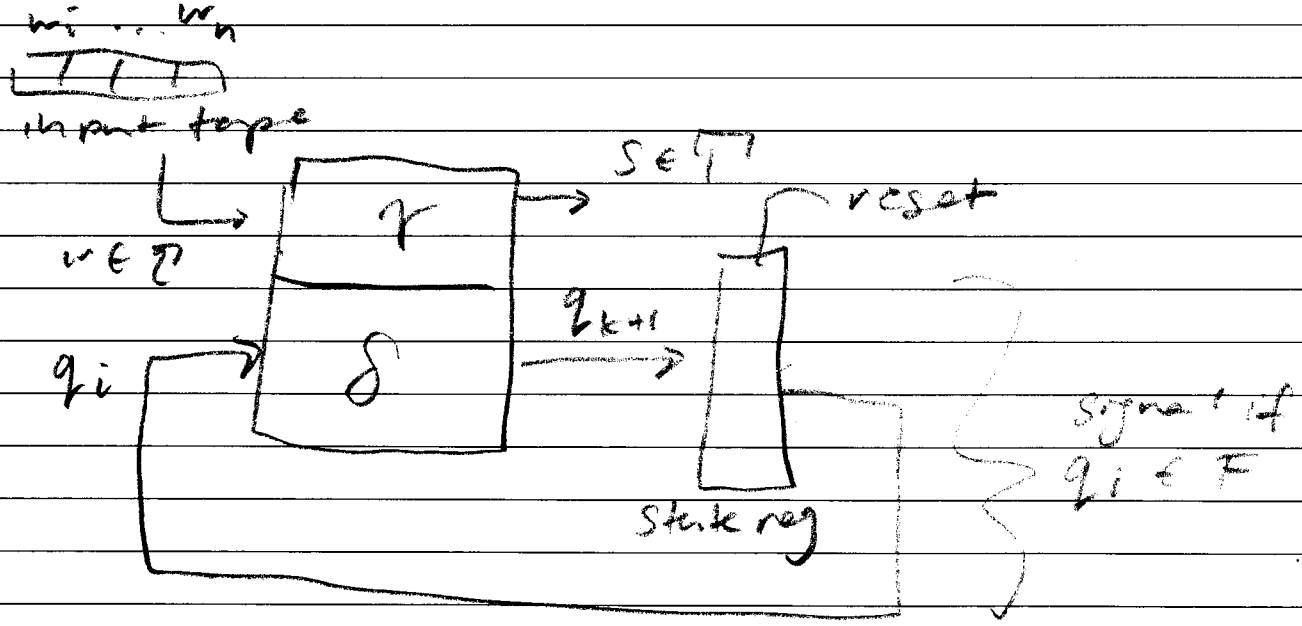
Date. 9.11.25
No. 06

FSTs and Regular Language

Finite State Transducer (FST):

FA with separate "output"

7-tuple: $(Q, \Sigma, \delta, q_0, F, \Gamma, \gamma)$
- $Q, \Sigma, \delta, q_0, F$ from FA
- Γ: set of output symbols
- $\delta: Q \times \Sigma \to Q \times \Gamma$
  - → argument of δ is a pair
  - → δ now returns a pair too



Function γ determines what symbol from Γ is output

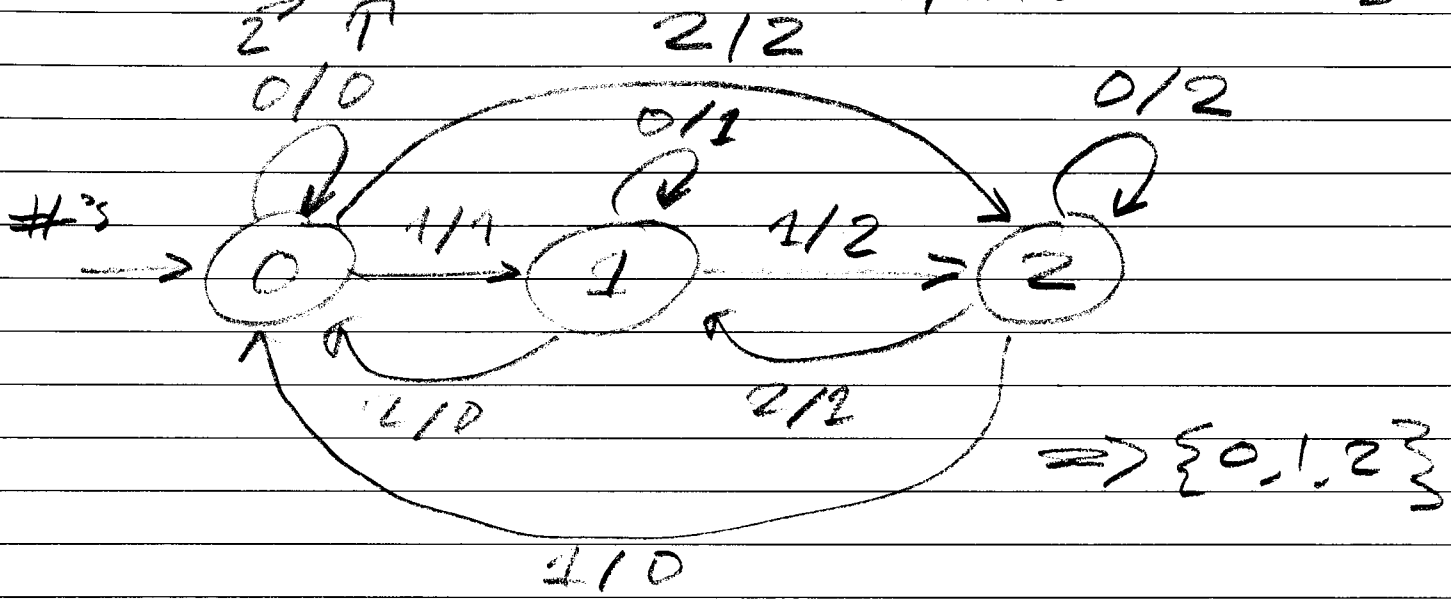
Moore Machine: $\gamma: Q \to \Gamma$
- output depends only on state
- $\delta: Q \times \Sigma \to Q \times (\gamma: Q \to \Gamma)$

Mealy Machine: $\gamma: Q \times \Sigma \to \Gamma$
- output depends on state AND input

Date.
No.

A Simple Mealy "Summing FST"

$\Sigma = \{0, 1, 2\}$, output = sum of all inputs modulo 3



$\Rightarrow \{0, 1, 2\}$

note, a/b on transition: a → input symbol
b → output symbol

state	input	new state	output
			mealy machine

Sipser 1.13 - Adding a Reset:

$\Sigma = \{0, 1, 2, \langle reset \rangle\}$

move machine + symbol

Date.
No.

Regular Languages:

understand "computation" and relationship between simple languages and finite automata

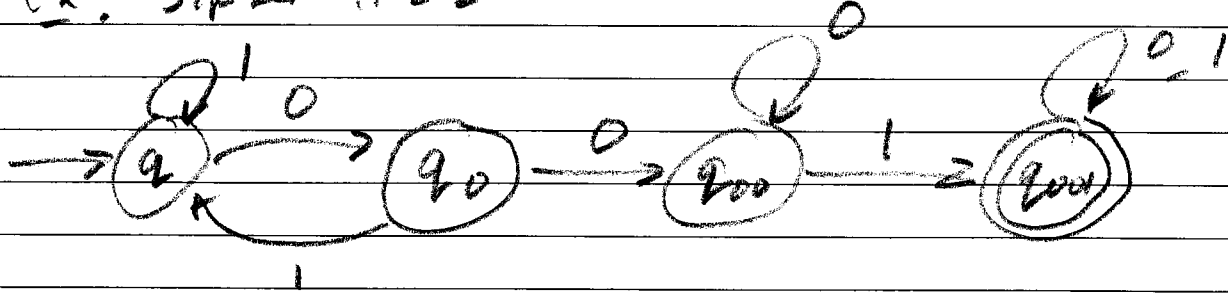
automaton M accepts language L if:
- when given a string in L, M stops in a state in F
- when given a string **not** in L, M stops in a state, but **not** in F

Regular Language: set of all languages for which a DFA can be designed as a recognizer (Σ, G)

Regular Expressions: notation that allows a concise description of a grammar for any regular language
- any regular language can be expressed as a regular expression

Th The language recognized by any DFA is regular, and can be expressed as a regular expression ∅

Ex. Sipser 1.22



Accepts strings containing 001

- this is a DFA
  - no trap states

Date.
No.

Languages as Objects:

Language = set() which implies language = object
- can perform computations over languages

Regular Operations over regular languages
- Assume A, B are languages ie sets

Union: $A \cup B = \{x \mid x \in A \vee x \in B\}$

Concatenation: $A \circ B = \{xy \mid x \in A, y \in B\}$

Kleene Star: $A^* = \{x_1 x_2 \ldots x_i \ldots x_k \mid k \geq 0, \forall x_i \in A\}$

A set of objects is closed under some operator if:
- the result of combining any two elements of the set
- using the operator
- creates a new object that is also in the set

note. $\emptyset$, ε (empty set) $\in A^*$

$|A^*| = \infty$

Properties:
- L is regular iff it is accepted by some FA M_L
- set of regular languages are closed under
  - union
  - concatenation
  - kleene star

Date.
No.

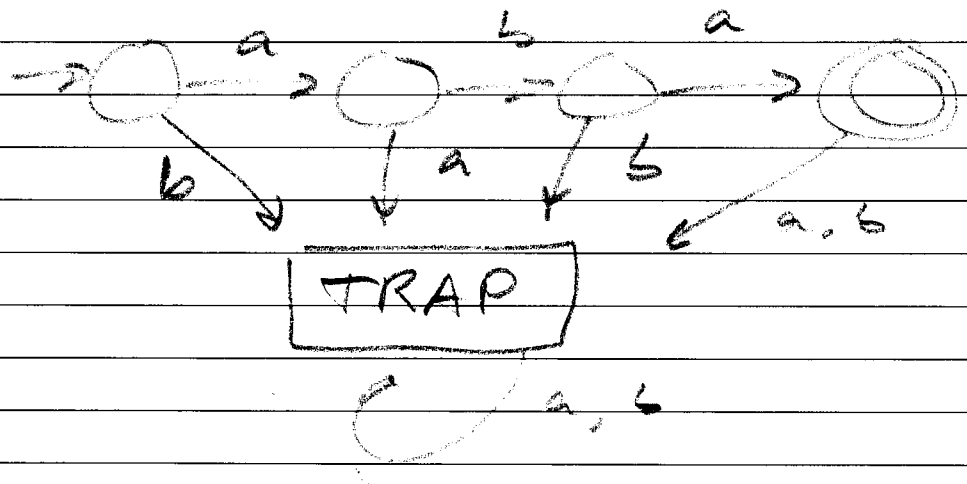
Regular Language Theorem:

Regular expressions are the grammer for RLs

Has these rules:

- any specific string $\in \Sigma^*$, $\{\}$, ε
- union, concat, or Kleene on R_1, R_2 s.t. $R_1, R_2 \in RE$

Ex. DFA that accepts $\{aba\}$



* don't need the TRAP state

NFA

Date. 9.16.25
No. 107

Properties of Regular Languages:

- L is regular iff it is accepted by some DFA M_L
- Set of reg langs are closed under
  - Union
  - Concatenation
  - Kleene Star
- How to prove?
  - take any two reg langs A and B
  - form their accepting FAs M_A and M_B
  - construct an FA that accepts the combined language

Proof that Union of 2 RLs is RL

- $M_A = (Q_A, \Sigma_A, \delta_A, q_A, F)$, $M_B = \dots$
  Assume Q_A and Q_B have different names for all states
  Assume $\Sigma_A = \Sigma_B = \Sigma$
- Construct $M_{A \cup B} = (Q_{A \cup B}, \Sigma_{A \cup B}, \delta_{A \cup B}, q_{A \cup B}, F_{A \cup B})$
  $Q_{A \cup B} = Q_A \times Q_B$
  $q_{A \cup B} = (q_A, q_B)$
  $F_{A \cup B} = F_A \times F_B$
  $\delta_{A \cup B} = ((q_x, q_y), a) = (\delta_A(q_x, a), \delta_B(q_y, a))$
- Run both machines "in parallel"

ex. in worksheet

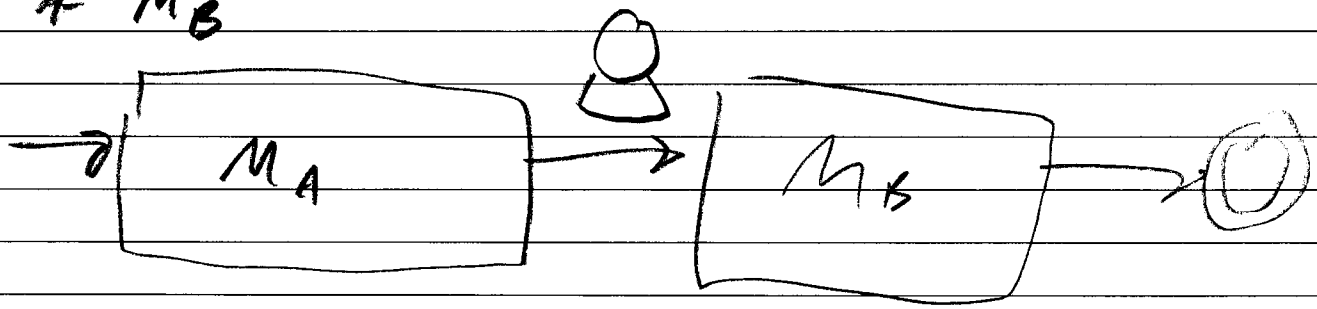
Date.
No.

Proof that Concatenation is Closed over Reg Langs:

$M_A = (Q_A, \Sigma_A, \delta_A, q_A, F_A)$, $M_B =$ " "

- Assume Q_A and Q_B have different names for all states
- For now assume F_A only has one state q_{AF}

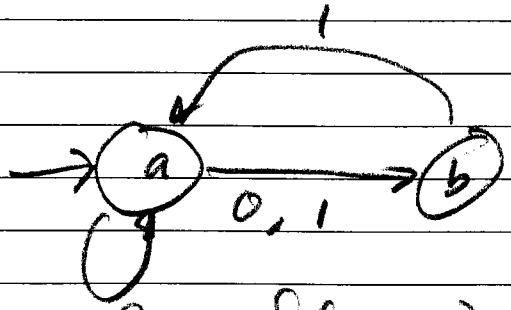
- Run M_A first, then M_B
  - naturally, make the final state(s) of M_A the "same" as the initial state of M_B



Determinism:

Deterministic FA (DFA): $\delta : Q \times \Sigma \to Q$
- δ has only one rule for any q, a

Nondeterministic FA (NFA): $\delta : Q \times \Sigma \to P(Q)$
- multiple choices of leaving q for some a



set of all subsets of Q
each subset is a possible state

$\delta(a, 1) = \{b\}$

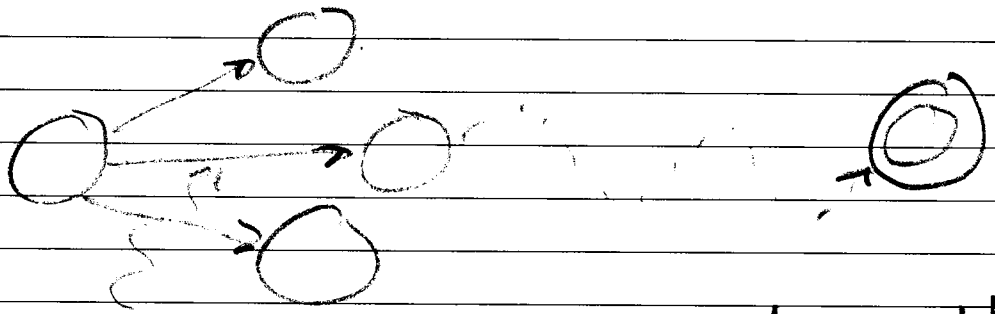
$\delta(a, 0) = \{a, b\}$

decision to be made

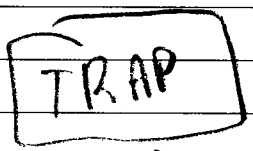
Date.
No.

NFA :

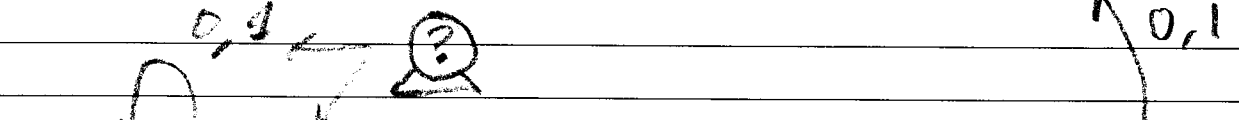
- Assume we have a "choice" of transitions
- Assume also a "crystal ball" to look into the future
  - → given inputs we know
  - → follow the choices that lead to a final state!



* tape must have right length



Ex. Sipser 1.30



current state	input = 0	input = 1
q_1	$\{q_1\}$	$\{q_1, q_2\}$
...		

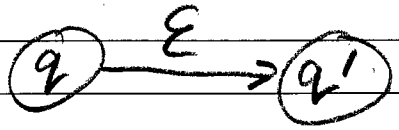
when are we 3rd from the end?

* try all possibilities to eliminate 

Date.
No.

NFA Feature: ε transitions

- Assume transitions include $\delta(q, \varepsilon) \to q'$



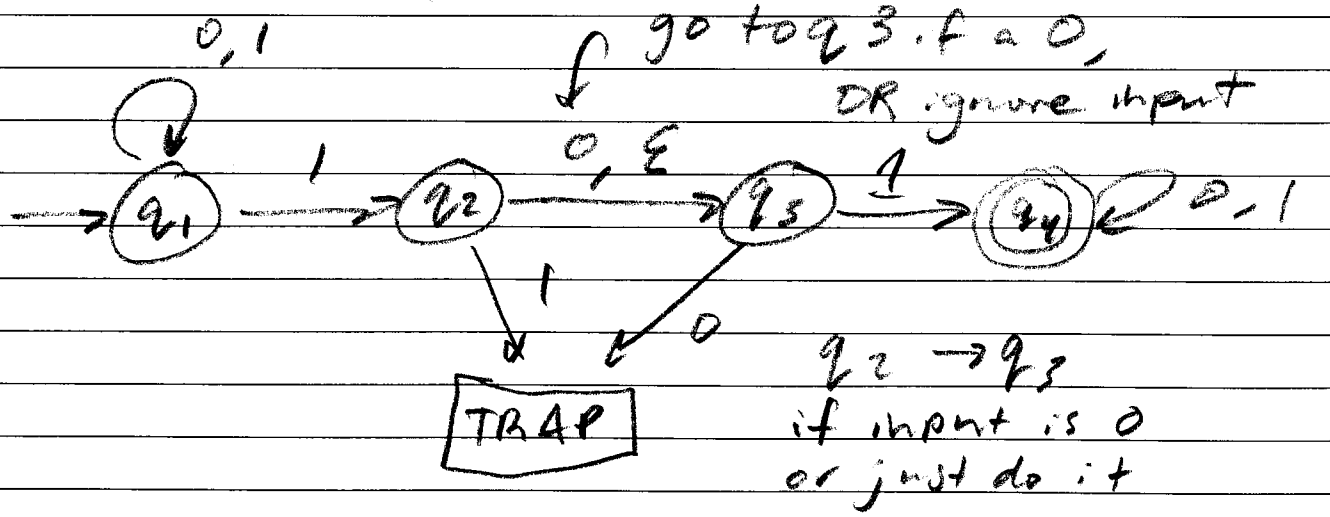
- When moving into state q, NFA can keep moving to state q' without looking at next input
  → leave input for next step

- only one path "non-deterministically"

- Formal Definition: $\delta : Q \times \Sigma_\varepsilon \to P(Q)$

Σ or ε

note. You do NOT have to take ε all non-deterministic ... decided by ?



Regular Expressions

Date. 9.18.25
No. 08

Regular Expressions: Set of strings R with rules:

- Any specific string from Σ^*, or $\{\}$ / ϵ
- Union of two regexes $R_1 \cup R_2$
- Concatenation of two regexes $R_1 R_2$
- Kleene Closure of regexes R^*

↳ L is regular iff it can be described as a regex

Shorthand:

- can use "()" to clarity
- R^+ means one or more strings from R
- R^k means k-copies of strings from R

Exs.

1. $\Sigma^* 001 \Sigma^*$: all strings containing 001
2. $\Sigma^2 \Sigma^*$: strings of length 2
3. $0\Sigma^* 0 \cup 1\Sigma^* 1 \cup 0 \cup 1$: start and stop with same 0 or 1

Grep: grep [options] [patterns] [files]

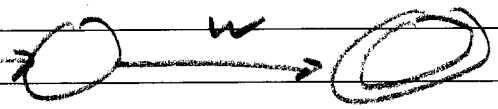
↳ patterns and examples in slides #

Date.
No.

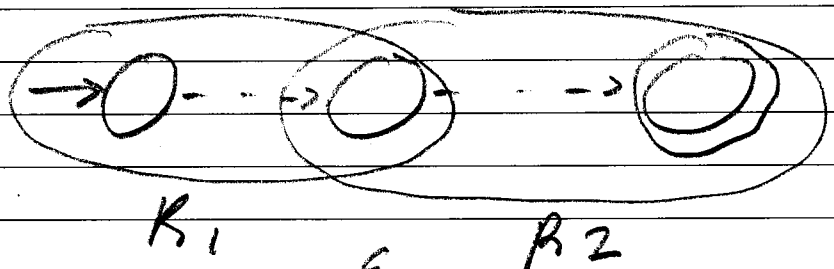
Theorem: Any RE can be converted into an NFA and thus to a DFA and thus is a regular language

Proof by construction: Assume R is a RE

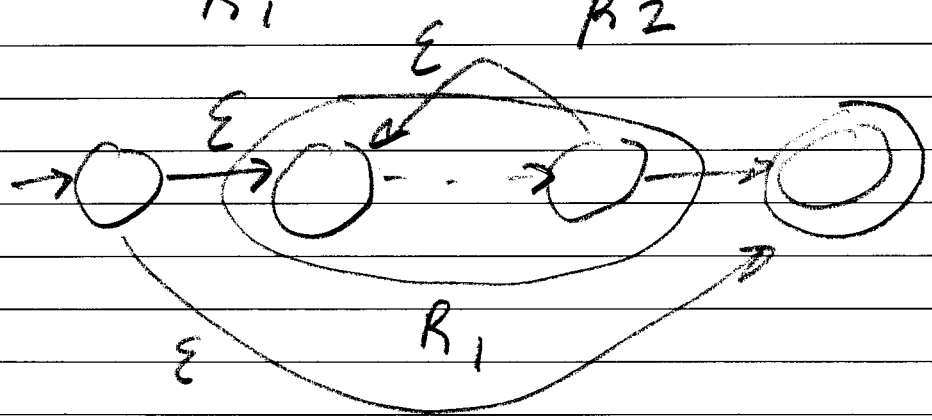
if $R = w, w \in \Sigma \cup \varepsilon$



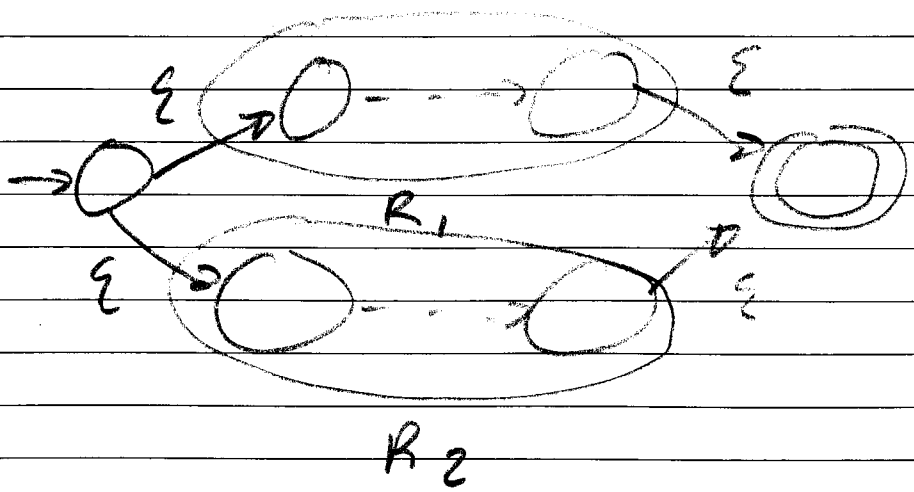
if $R = R_1 R_2$



if $R = R_1^*$



if $R = R_1 \cup R_2$

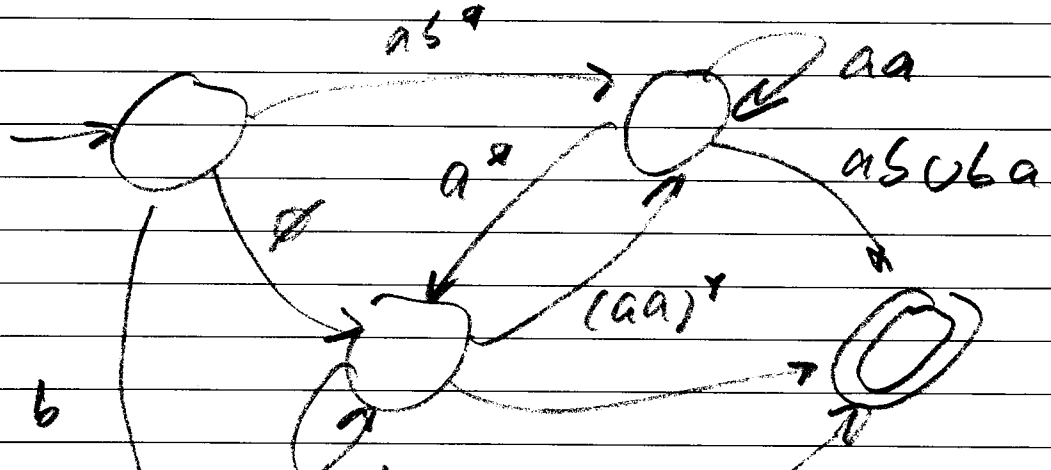


Date.
No.

Closing the Circle: Given a DFA, what is the Regex?

GNFA: generalized NFA

- NFA w/ regexes on edges
- start state has arrows to every other state, but no incoming
- single accept state with arrows from every other state, but no outgoing
- all other states fully connected



DFA to GNFA:

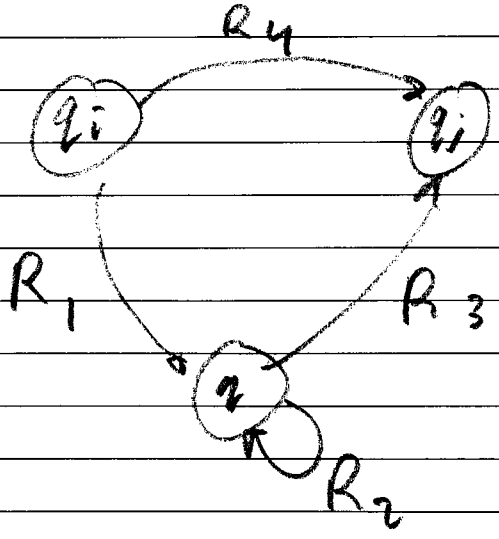
- Add new start state
  → ε to old start state
- Add new final state
  → ε from old final states
- If multiple arcs b/w any two states
  → replace with single arc with "∪" of edges
- B/w any states w/o arc
  → add edge with ∅ label

Date.
No.

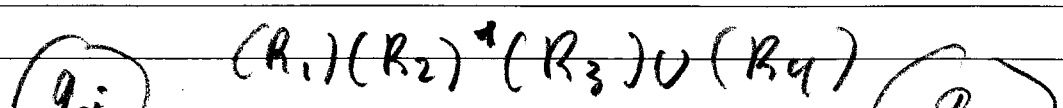
GNFA to Regex:

1. Initial GNFA has k states, $k > 2$
   → each edge labeled w/ regex r_{ij}
2. Pick some state q to be removed
3. Concatenate regexes between edges going in and out of q
4. Add in Kleene star for any closed loop
5. ∪ any arcs between same states
6. Repeat until only 2 states
   → arc btw states is regex

Ex.



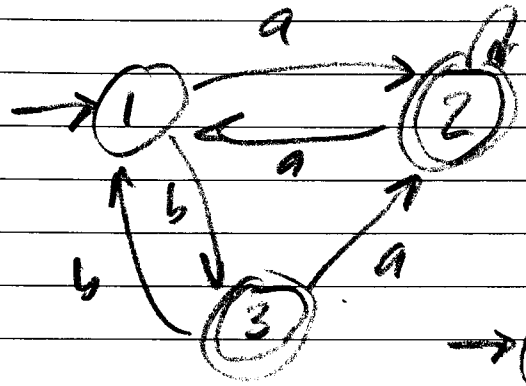
before



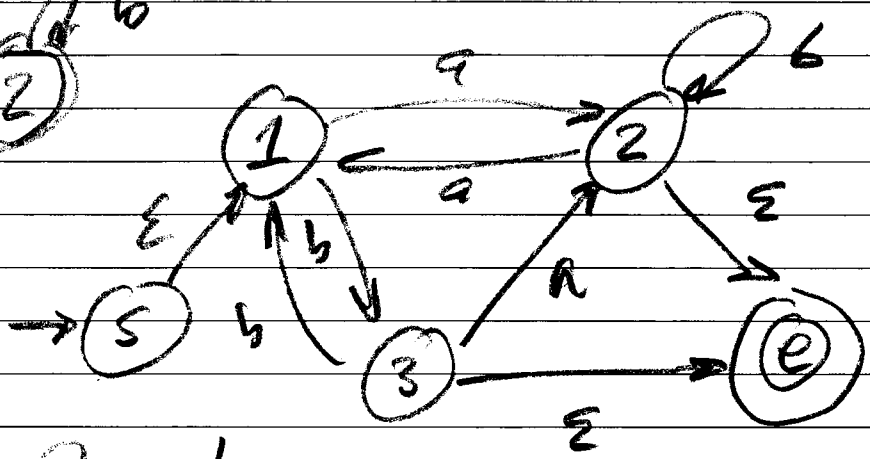
after

Date.
No.

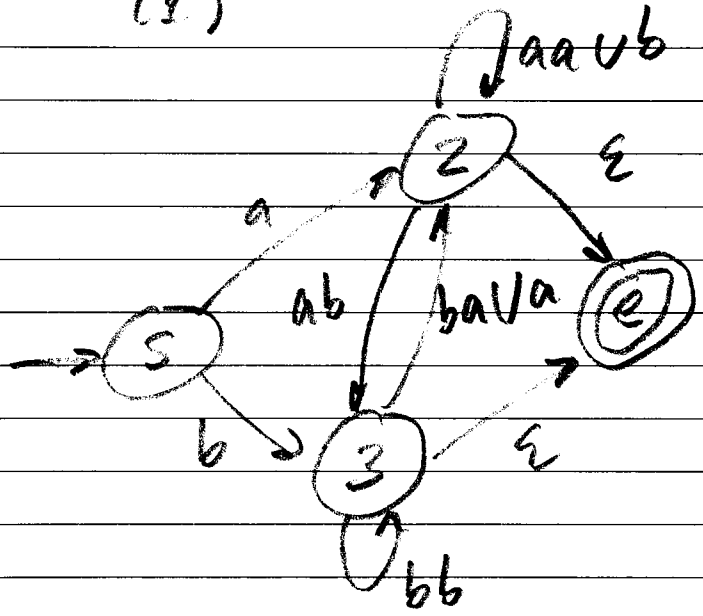
Sipser 1.67 DFA to Regex: steps



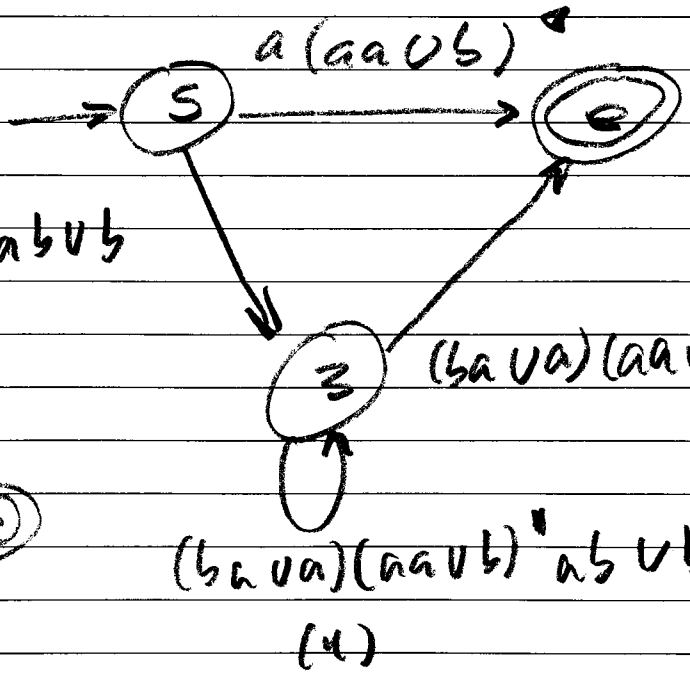
(1)



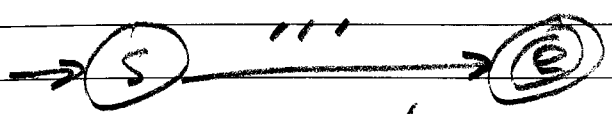
(2)



(3)



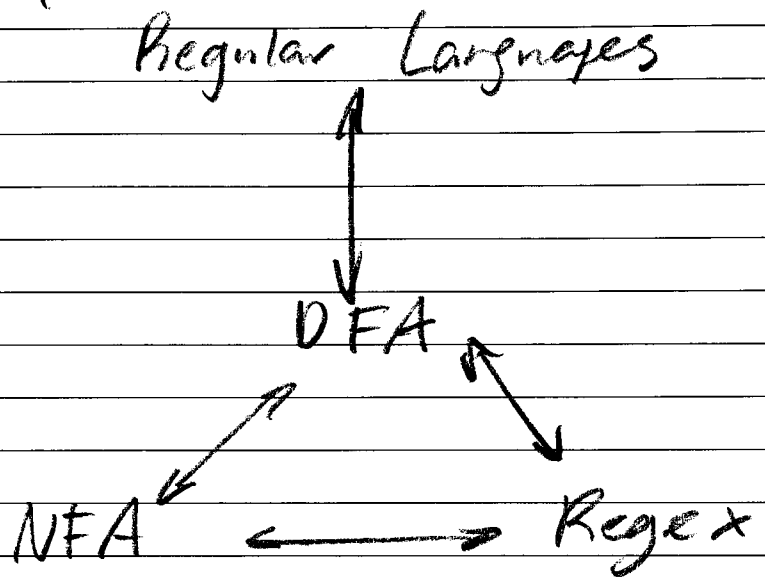
(4)



(5) ✓

Date.
No.

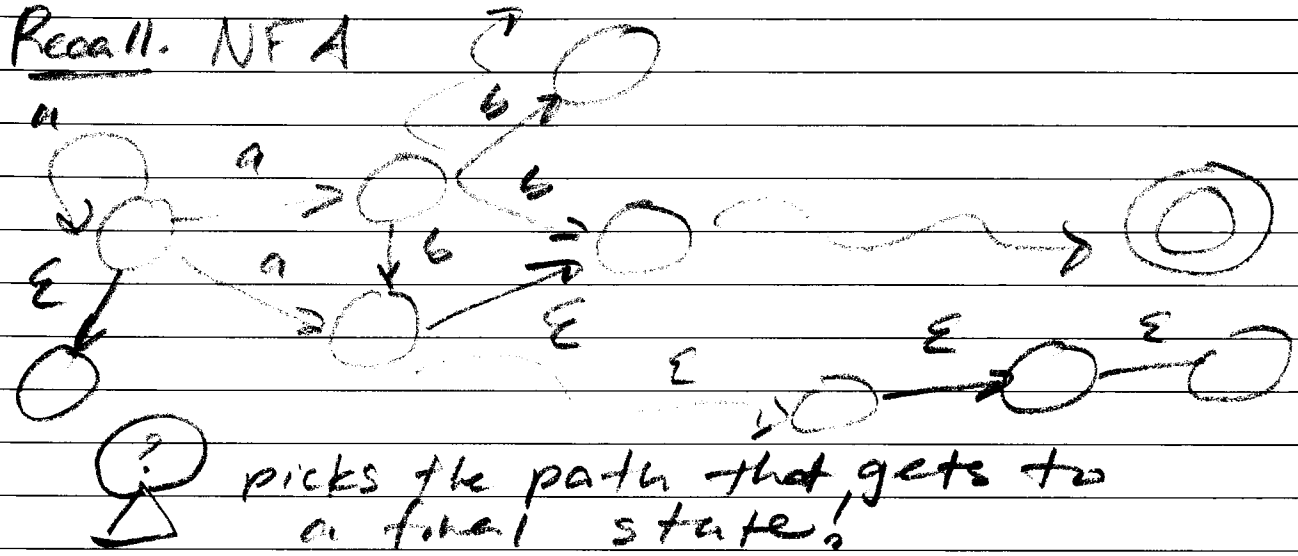
Summary :



NFA to DFA

Date. 9.23.25
No. 09

Recall. NFA



picks the path that gets to a final state!

ε is a "free" transition you can take or not take after any transition

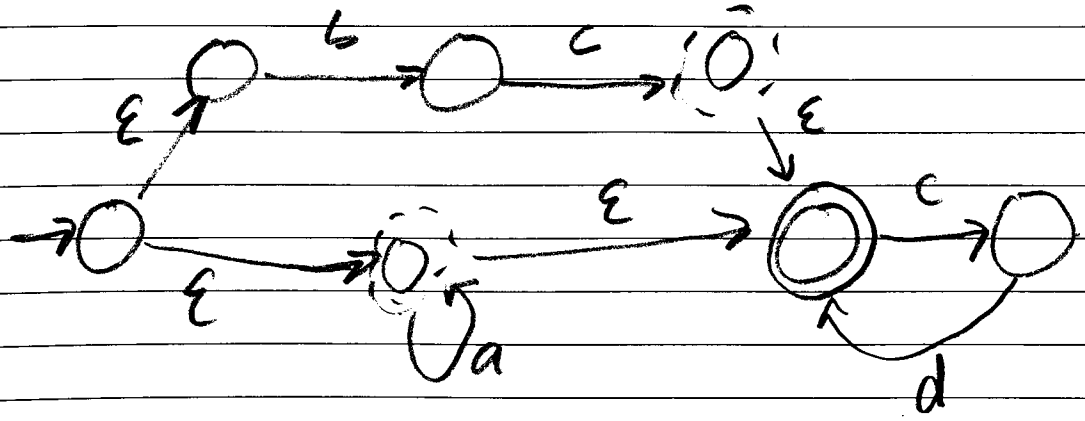
i.e. keep-going option

$\delta(q_0, c) = \{q_0, q_1, q_2, q_3, \ldots\}$

non-deterministic outcome

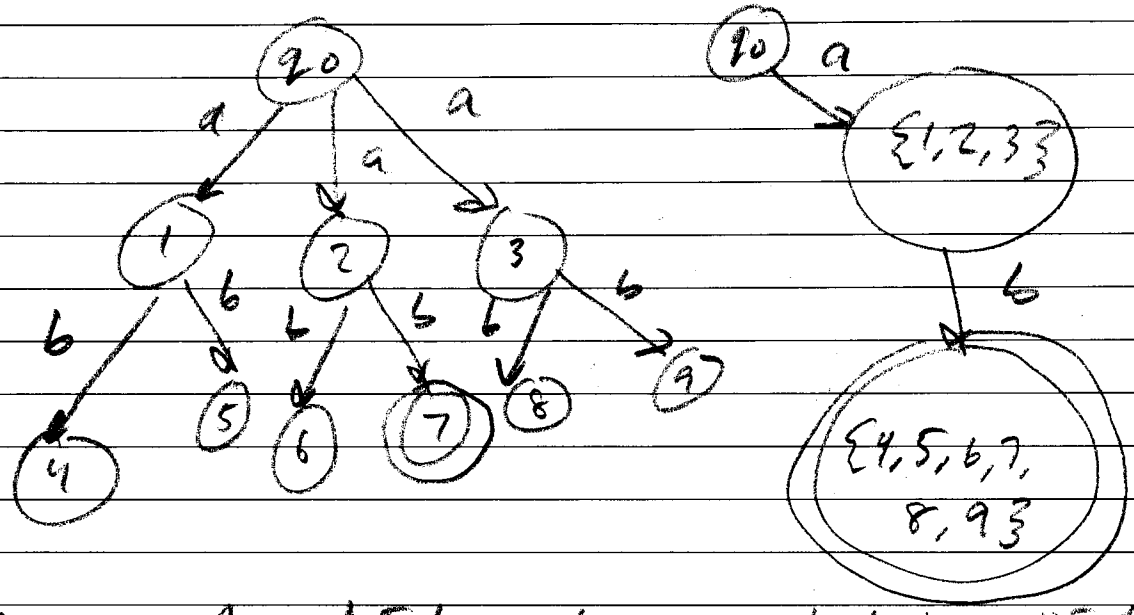
Goal. NFAs are no more powerful than DFAs, just are easier to design

Ex. Regex to NFA, $R = (bc \cup a^*)(cd)^*$



Date.
No.

General Idea:



Theorem. Any NFA can be converted to a DFA

NFA: $(Q, \Sigma, \delta, q_0, F)$

DFA: $(Q', \Sigma, \delta', q_0', F')$

- $Q' = P(Q)$ power set
- $q_0' = \{q_0\}$
- $\delta'(q', a) = \{q \mid \delta(q'', a) = q\}$

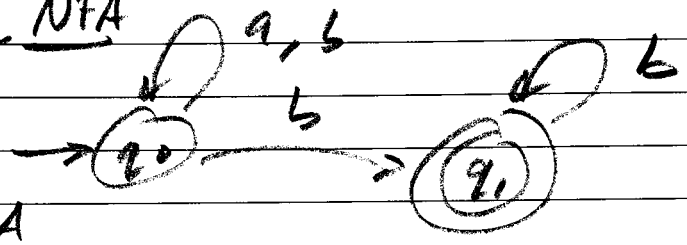
  a single state in the DFA

  → for some $q'' \in q'$
- $F' = \{q' \mid q' \in Q'\}$

  some member of F is in q'

Date.
No.

Ex. NFA



NFA

$Q = \{q_0, q_1\}$

$\Sigma = \{a, b\}$

$L = (a+b)^*$

DFA

$Q' = \{\{\}, \{q_0\}, \{q_1\}, \{q_0, q_1\}\}$

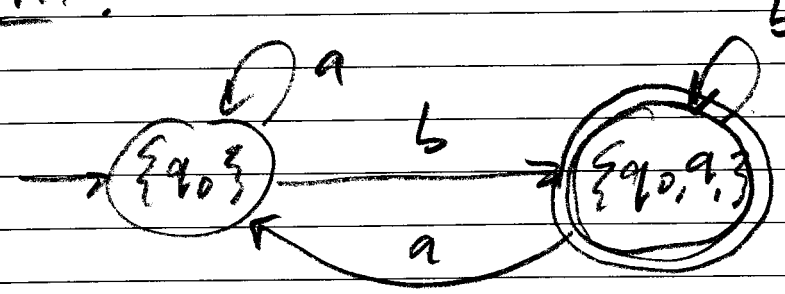
$q_0' = \{q_0\}$

$F' =$ states with $q_1 = \{\{q_1\}, \{q_0, q_1\}\}$

δ' :

State	Char	New State
$\{q_0\}$	a	$\{q_0\}$
$\{q_1\}$	b	$\{q_0, q_1\}$
$\{q_0, q_1\}$	a	$\{q_0\}$
$\{q_0, q_1\}$	b	$\{q_0, q_1\}$

DFA.



Dealing with Σ in NFAs:

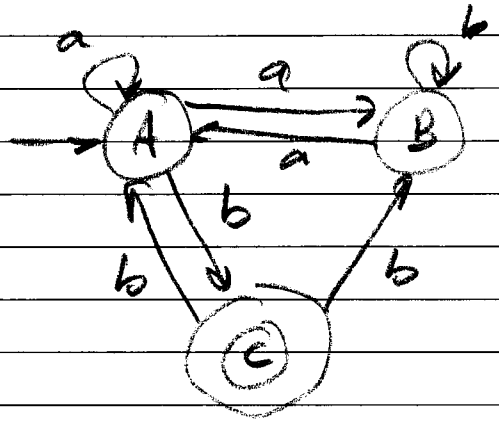
- Assume transitions include $\delta(q, \Sigma) \rightarrow q'$
- $E(q) = \{q, \ldots\}$
  $\subset$ represents all Σ transitions off of q

* more on slides *

Date. 9.25.25
No. 10

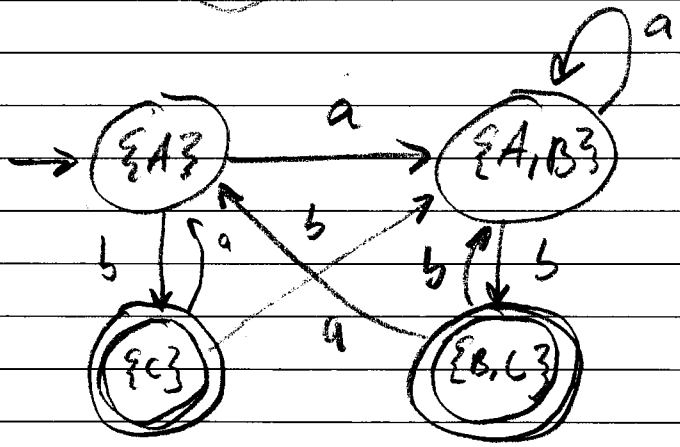
NFA to DFA cont.

NFA to DFA Ex.



δ for DFA'

State	a	b
{A}	{A,B}	{C}
{A,B}	{A,B}	{B,C}
{C}	TRAP	{A,B}
{B,C}	{A}	{A,B}

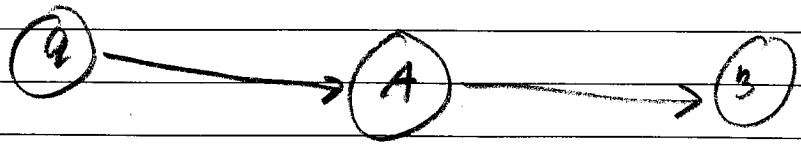


NFA to DFA with ε

- Assume transition includes $\delta(q, \varepsilon) \to q'$
- when moving into a state q, NFA can keep moving to state q' without looking at next input

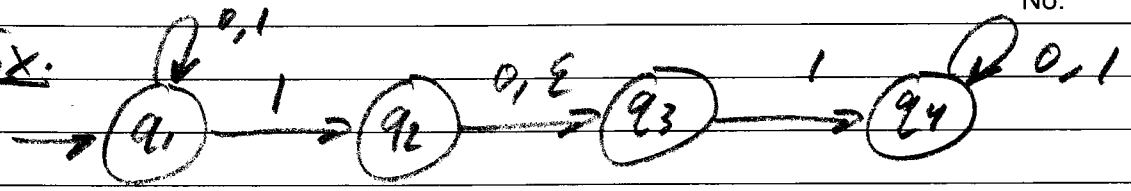
$E(q) = \{$ reachable by t if I enter by q

↑ q state in NFA



Date.
No.

Ex.



Epsilon readable states:

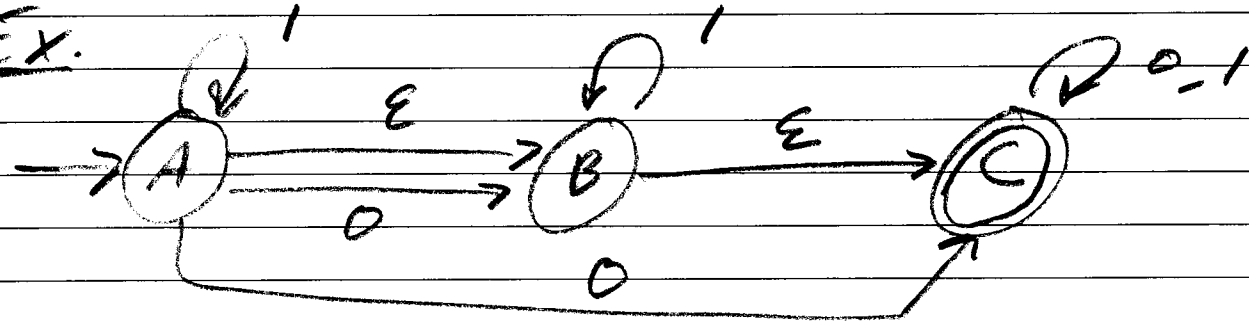
$t(q_1) = q_1$
$t(q_2) = \{q_2, q_3\}$
$t(q_3) = q_4$
$t(q_4) = q_4$

i. $\{t(q_1), t(q_2)\}$

State	0	1
$\{q_1\}$	$\{q_1\}$	$\{q_1, q_2, q_3\}$

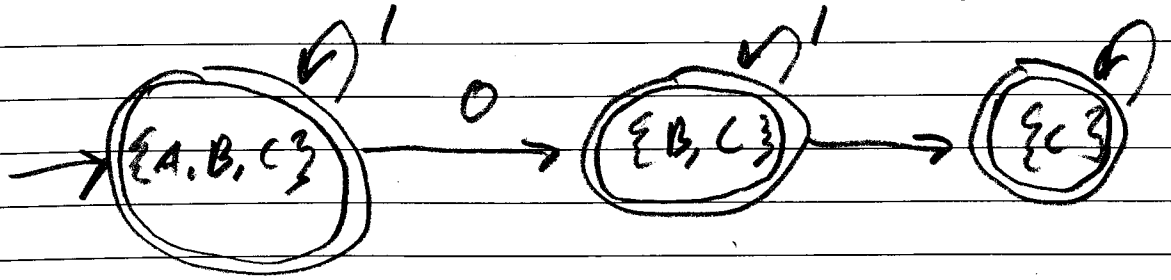
...

Ex.



$E(A) = \{A, B, C\}$
$E(B) = \{B, C\}$
$E(C) = \{C\}$

State	0	1
$\{A, B, C\}$	$\{B, C\}$	$\{A, B, C\}$
$\{B, C\}$	$\{C\}$	$\{B, C\}$
$\{C\}$	$\{C\}$	$\{C\}$



Date. 9.30.25
No. 11

The Pumping Lemma

Regular Languages

Language: Alphabet + Grammar

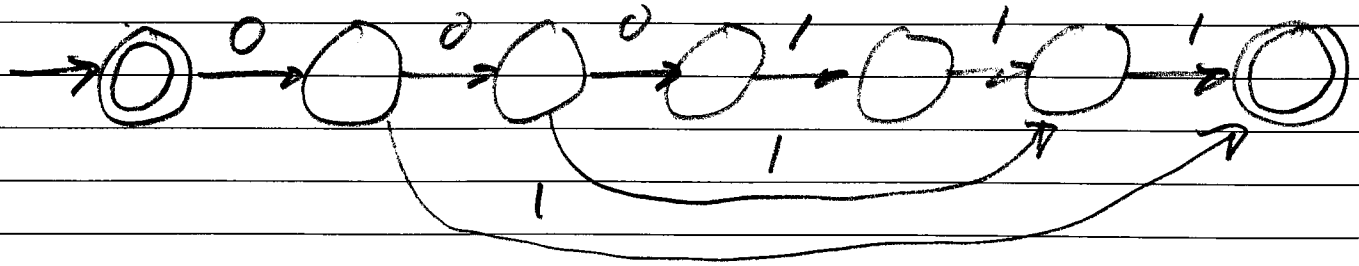
Regular Language:
- describable by a regex (its grammar)
- decidable by a NFA/DFA

Grammar Rules:
- finite # of symbols
- union, concatenation, or Kleene closure of 2 regular languages

Ex:
- $\{(01)^n \mid n \geq 0\}$
- $\{0^n 1^m \mid n \geq 0, m \geq 0\}$
- $\{0^k 1^k \mid 0 \leq k \leq n$, for some fixed $n\}$
- $\{w \mid w$ has equal # of 01 and 10 substrings$\}$

Non Ex.
- $\{0^n 1^n \mid \forall n \geq 0\}$

Let's Implement $\{0^k 1^k \mid 0 \leq k \leq n = 3\}$
$= \{\varepsilon, 01, 0011, 000111\}$



7 states
How many states for arbitrary n? $2n+1$
(fixed)

Date.
No.

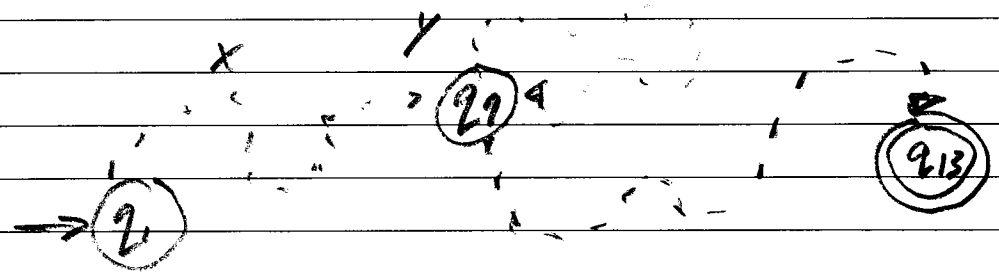
The Limits of Regular Languages:

- DFAs can only "count" up to their number of states
  → no "memory" or "registers"

- There are languages that require "counting" or similarly
  → cannot be accepted by DFA/NFA
  → cannot be regular

Ex's.
- $\{w \mid w$ has n a's and b's, n arbitrary$\}$
- $\{a^n b^n \mid n \geq 0\}$

"Pumping" in Regular Languages



- string xyz follows dashed path, thus in L
- substring y loops around q_9
- Thus any string of form $xy^k z$ $\forall k$ is in L

ie we have **pumped** substring y

- $k > 0$ ⇒ "pump up"
- $k = 0$ ⇒ "pump down"

Date.
No.

The Pumping Lemma:

if language A is regular, then there is a number p (called pumping length), where:

$\forall$ strings $s \in A$ s.t. $|s| \geq p$

$\exists x, y, z$ where

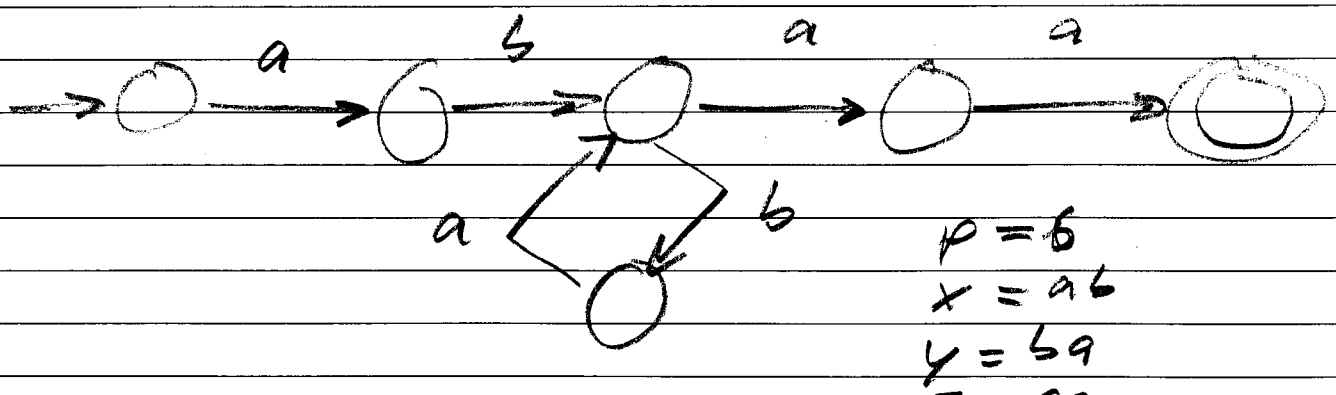
if $s = xyz$, $|y| > 0$, $|xy| \leq p$

then $xy^i z$ must also in A $\forall i \geq 0$

we say string has been pumped

$\exists$ $p \cong$ number of states in smallest DFA that accepts A

Ex. $ab(ba)^*aa$ regular



$p = 6$

$x = ab$

$y = ba$

$z = aa$

Showing Some Language B is NOT Regular:

Proof by Contradiction: Assume B is regular

- Find a string that is in B and show it cannot be pumped
  - → look at all ways to divide string w/ p
  - → Show that pumping each of them is not in B
- Conclusion: assumption must be FALSE
  - → B not regular

Date.
No.

Ex. 1.73: $B = \{0^n 1^n \mid n \geq 0\}$

Assume B is regular with pumping length p

$|0^p 1^p| \geq 2p$

$0^p 1^p$ is in B and has length $> p$, so choose it as s

- since $|xy|$ must be $\leq p$, and also it must so that $1 \leq |y| \leq p$
- you can only be some number of 0s (k of them)

using pump up: $xy^i z = 0^{p+(i-1)k} 1^p$ not in B
- we added 0s but not 1s

using pump down $xy^i z = 0^{p-k} 1^p$ not in B

since both not in B by contradiction it's not regular

Ex 1.74. $C = \{w \mid w$ has same # of 0s and 1s$\}$

- Assume C is regular with pumping length p
  - if any string $> p$ must have a loop
- $0^p 1^p \in C$ and has length $> p$, so choose it as s
  - note. theorem works for any string $> p$

...

4 more exs worked out in slides

Date. 10.2.25
No. 12

Pumping Lemma cont.

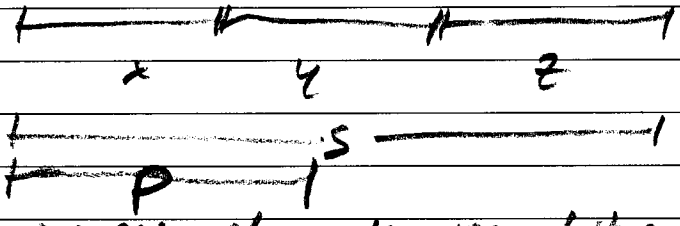
How to show L is not Regular?

Proof by Contradiction.

1. Assume L is Regular

$\forall s \in L$ st $|s| \geq p$ must be pumpable

Pick one s st s divisible into xyz



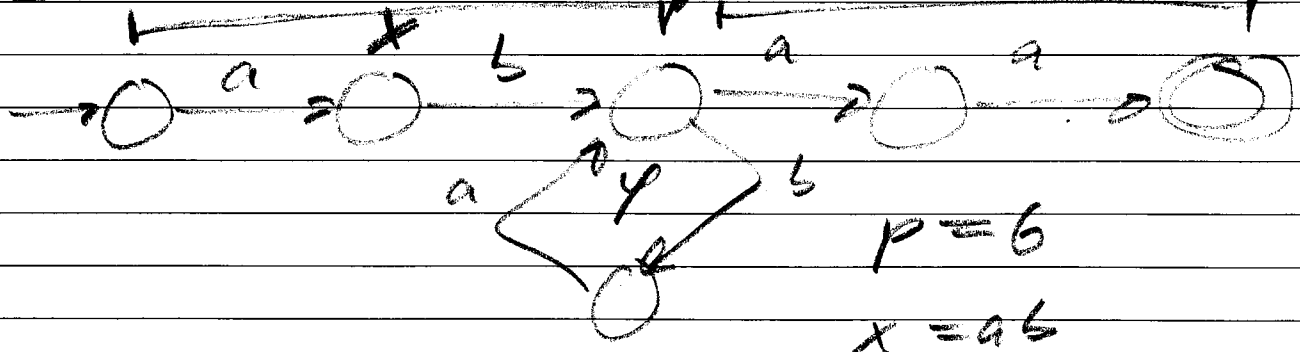
$|s| \geq p$

Proof process, show no partition exists:

show ~~$\forall$~~ $xy^i z \in L$

$|xy| \leq p$, $|y| \geq 1$

Ex. $L = aL(ba)^* aa$ regular?



$p = 6$

$x = ab$

$y = ba$

$z = aa$

$\forall s \in L$ st $|s| \geq p$

$\exists x, y, z$ st $s = xyz$,

$|y| > 0$, $|xy| \leq p$

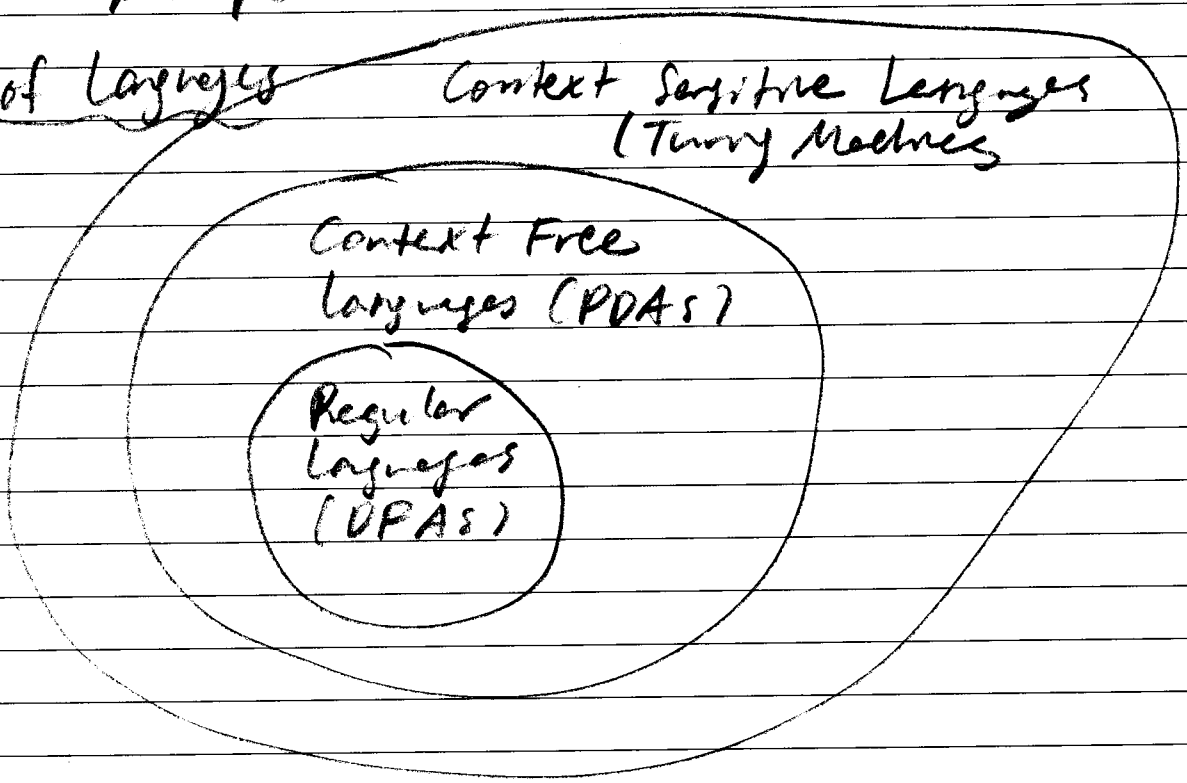
AND $xy^i z \in A$ $\forall i > 0$

Date.
No.

Showing L is NOT Regular:

1. Assume L is regular
2. Find one string in L and show it can't be pumped

Types of Languages



Ex. $L = \{0^i 1^j \mid i > j\}$

Assume L is regular with pumping length p
$|0^{p+1}1^p| = p$ so choose $s = 0^{p+1}1^p$
Therefor xy must be all 0s since $|xy| \leq p$
Therefor y must be all 0s, since $|y| > 0$

Now Pump down: try xy^0z
This removes at least 1 0 from the string
Now only have p 0s, and i NOT > j anymore

Date.
No.

Context Free Grammars:

- Collection of substitution rules of form

  $A \rightarrow w_1, w_2 \ldots w_n$

  ↑ concatenation of strings

- LHS: a variable / nonterminal
  → name of a subset of strings relevant to the language

- RHS: strings of either
  → symbols (aka terminals) from Σ (alphabet)
  → or nonterminals

- Start Variable: nonterminal representing any string in the language

Interpretation: LHS is a set of strings using RHS

Formal CFG: (V, Σ, R, S)

V = set of nonterminals (vocabulary)
Σ = alphabet
R = set of rules
S = start nonterminal

Ex. $V = \{A, B\}$
$\Sigma = \{0, 1, \#\}$
start nonterminal = A
Rules:

$A \rightarrow 0A1$ $\quad B = \{\#\}$
$A \rightarrow B$ $\quad A = \{0\#1, 00\#11, \ldots\}$
$B \rightarrow \#$ $\quad$ recursive!

Push Down Automatas

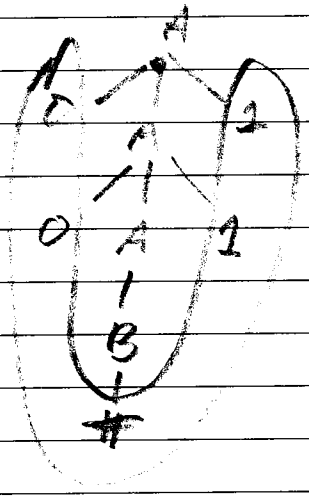
Date. 10.14.25
No. 15

Notation.

$A \to bc$
$A \to ad$
can become $A \to bc \mid ad$ (↑ OR)

Parsing:

ex. 000#11 from last ex
can use a Parse Tree:



the leaves of the parse tree gives a string accepted by the language

Derivation: sequence from a variable to a string

Assume u, v, w are strings of terminals & nonterminals

- uAv yields uwv (written $uAv \Rightarrow uwv$) if there is a rule $A \to w$ from the grammar
- u derives v (written $u \overset{*}{\Rightarrow} v$) if either $u = v$ or $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_k$ and $u_k = v$ $\forall k \geq 0$

ex. $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111$
$\Rightarrow 000\#111$

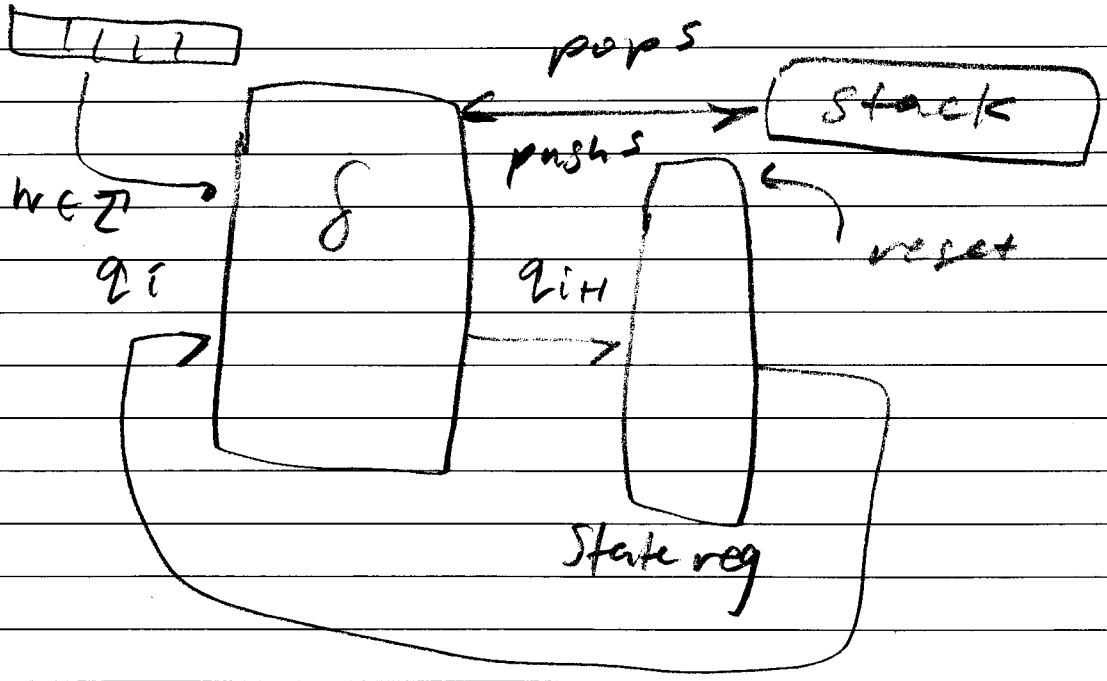
Date.
No.

note. leftmost derivation first (convention)

→ solves nondeterminism in CFGs
i.e. it's possible to have 2 valid parse tree/derivations for the same string

Ends of completely CFGs in slides and worksheet

Push Down Automata: computing with infinite stack



assume stack can grow to unbounded depth

6-tuple $(Q, \Sigma, \delta, q_0, F, \Gamma)$
Q: set of states
Σ: set of input symbols
$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to P(Q \times \Gamma_\epsilon)$
q_0: initial state
F: set of final states
Γ: stack alphabet

power set

$\Sigma_\epsilon = \Sigma \cup \epsilon$
$\Gamma_\epsilon = \Gamma \cup \epsilon$

Date.
No.

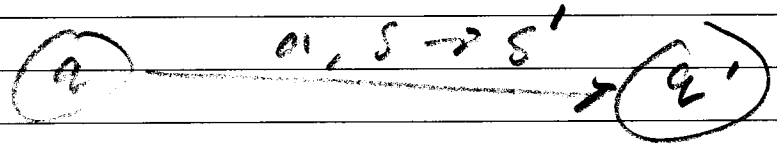
Normal PDA Transitions: (no ε)

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$$

if in state q, with input w, and top of stack s:

- pop s from stack
- choose one $(q', s') \in \delta$
- change state to q'
- push s' onto stack

note, nondeterminism
multiple valid (q, s) possible



$a, s \to s'$
$\delta(q, a, s) \to (q', s')$

a: tape character
s: current top of stack
s': new top of stack

Special PDA Transitions: allows us to not touch the stack

if $\delta(q, w, \varepsilon)$:
- don't need to match to stack
- don't pop anything from stack

if $\delta(q, \varepsilon, s)$: NFA transition
- don't need to match with input
- but match s with top of stack

if $\delta(q, \varepsilon, \varepsilon)$: DFA transition

Date. 10.16.25
No. 16

CFGs to PDAs

$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$

if $\delta(q, w, \varepsilon)$: stack doesn't matter
if $\delta(q, \varepsilon, s)$: input doesn't matter
if $\delta(q, \varepsilon, \varepsilon)$: like NFA

if pair chosen from δ is (q', ε) :
just change state not stack

Summary : 4 combinations

① $\delta(q, a, x) \to (q', y)$
1. a must be input
2. x must be top of the stack
3. pop the x
4. push the y

② $\delta(q, a, x) \to (q', \varepsilon)$
pop x no push

③ $\delta(q, a, \varepsilon) \to (q', y)$
don't pop, push y

④ $\delta(q, \varepsilon, \varepsilon) \to (q', \varepsilon)$
ignore the stack

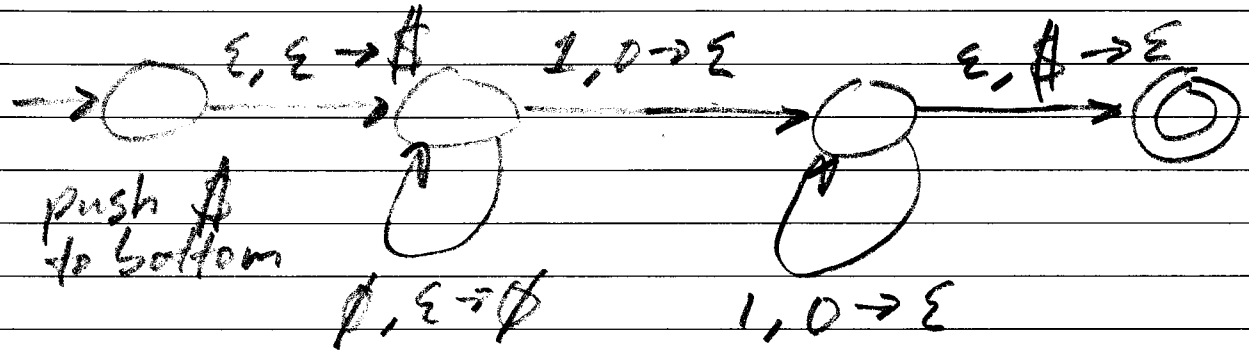
8 possibilities with output in mind

⑤ $\delta(q, \varepsilon, x) \to (q', y)$
⑥ $\delta(q, \varepsilon, x) \to (q', \varepsilon)$
⑦ $\delta(q, \varepsilon, \varepsilon) \to (q', y)$
⑧ $\delta(q, \varepsilon, \varepsilon) \to (q', \varepsilon)$

Date.
No.

Ex. $L = \{0^n 1^n \mid n \geq 0\}$ is NOT regular

Build a PDA to accept it:



* most PDA's follow this structure * i.e. with the $

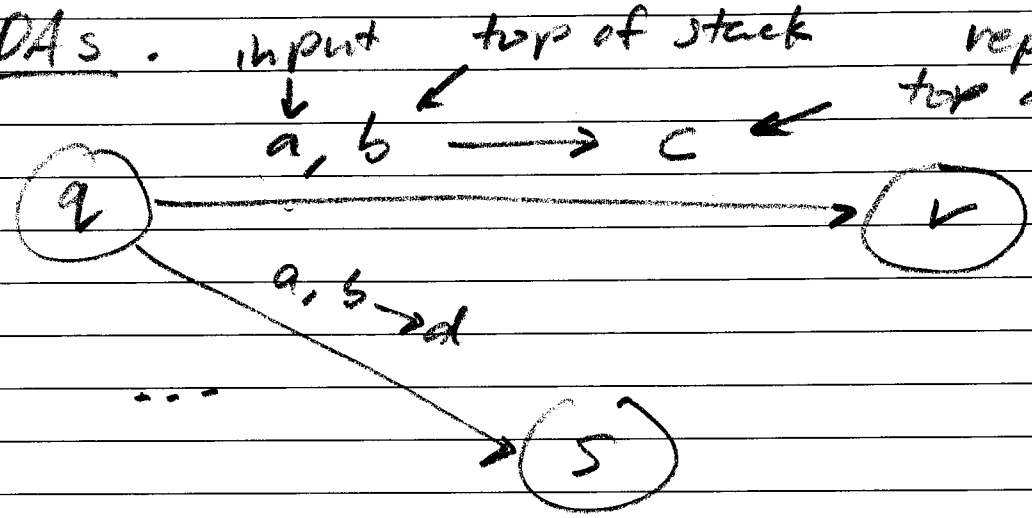
note. this doesn't work for $0^N 1^N 2^N$!

note. input, stack_pop → stack_push

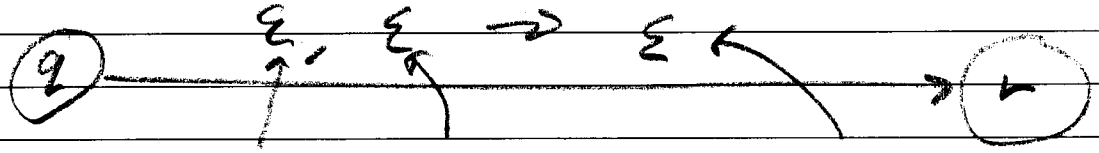
Date. 10.28.25
No. 17

GFG to PDA cont

PDAs. input, top of stack, replace top of stack



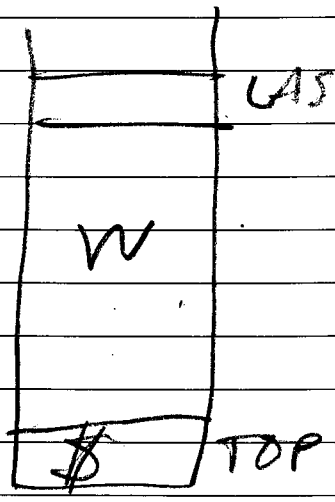
can be nondeterministic!



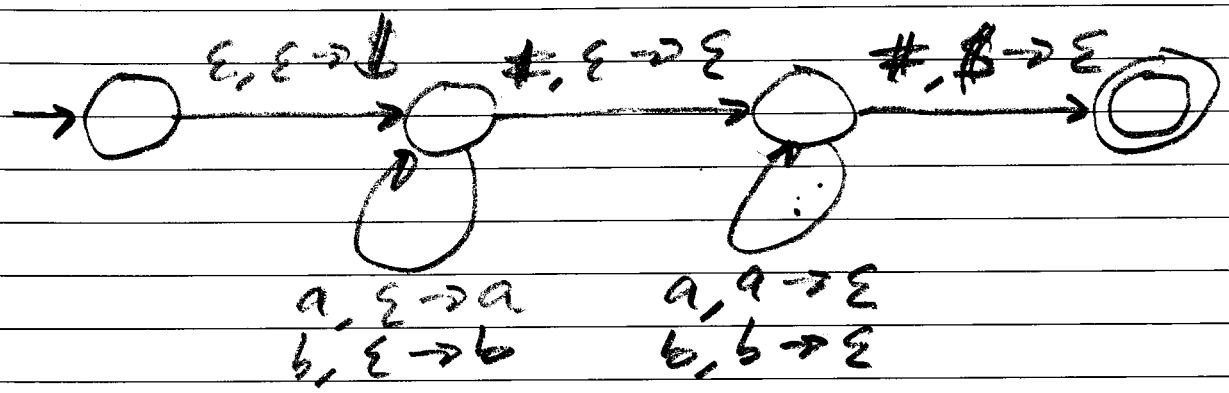
ignore tape! ignore top of stack. don't push anything

Ex. design a PDA to accept:
$\{ w\#w^R\# \mid w \in \{a,b\}^* \text{ and } w^R \text{ is reverse} \}$

Strategy:
1. push tape to stack until #
2. pop while tape is top of stack and then pop top of stack
0. push $ to stack
3. if at end of tape we see $ on TOS, DONE



Date.
No.

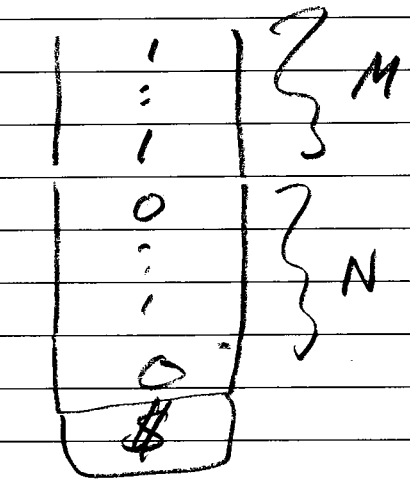


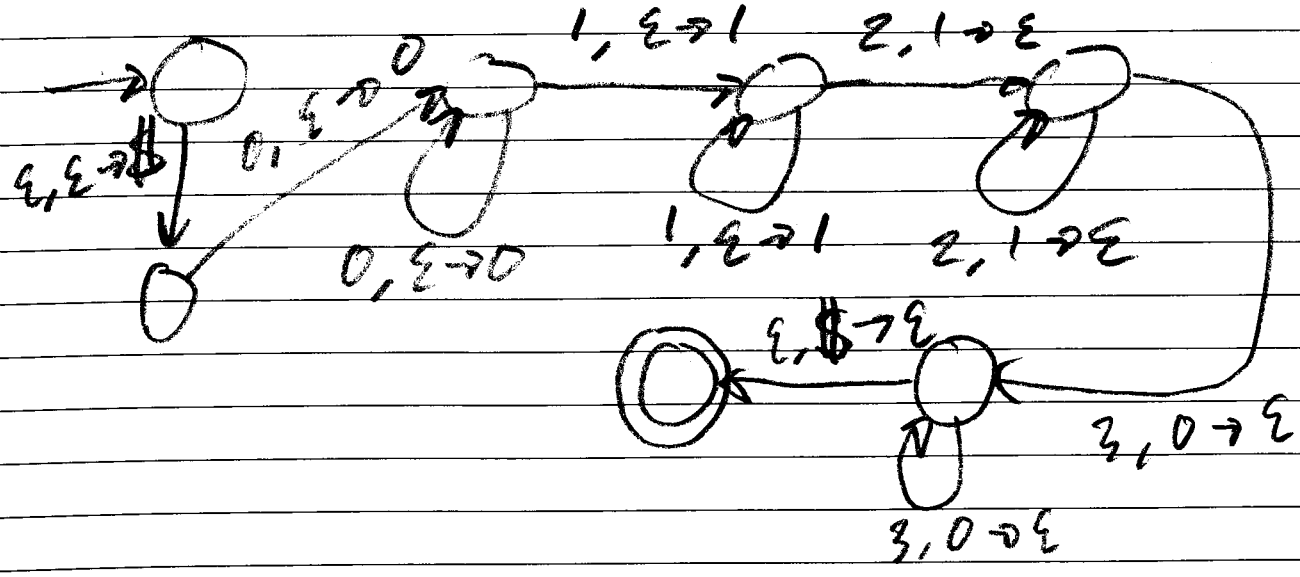
Design a PDA for language L:

$L = \{ 0^n 1^m 2^m 3^n \mid n \geq 1, m \geq 1 \}$

Strategy:

- push # to stack
- push 0s to stack
- push 1s to stack
- if a 2 comes with a 1 on the stack, pop the stack
- if a 3 comes with a 0 on the stack, pop the stack





Date.
No.

Are there Limits to PDAs?

Consider $L = \{a^i b^i c^i \mid i \leq k\}$
if k is fixed, can build NFA at most $3k+1$ states

yes! can't match these using one stack
i.e. can't remember a binary # twice

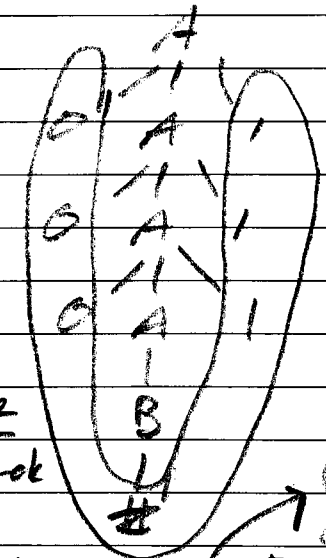
Lemma 2.21: Every CFL decidable by some PDA

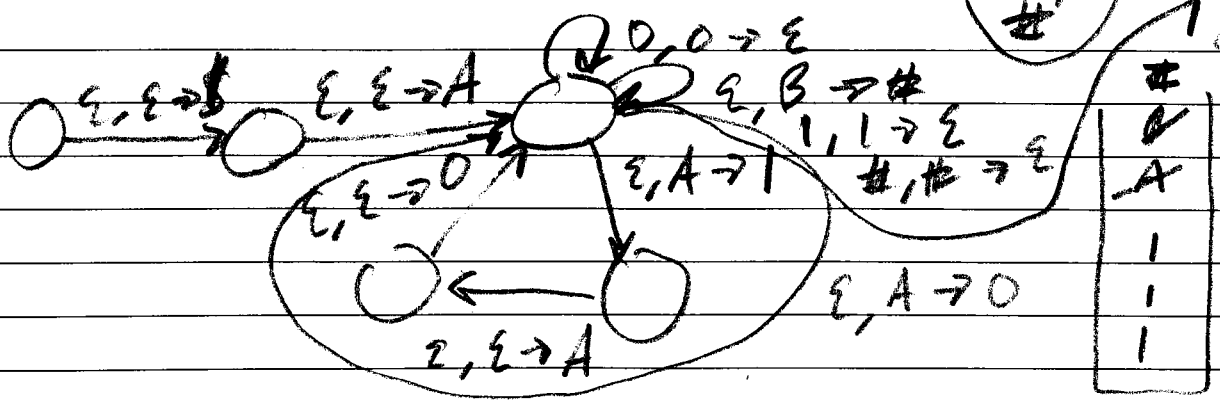
Proof by construction: build PDA from CFG
- keep next expected symbol on top of stack
  - i.e. the terminal or nonterminal
- if expect terminal, TOS is that char
- if expect nonterminal, nondeterministically select correct rule
- replace top of stack by rule's RHS

Ex. $A \to 0A1$ $\quad B \to \#$
$A \to B$
$B \to \#$

$\Sigma = \{0, 1, \#\}$
$V = A, B$

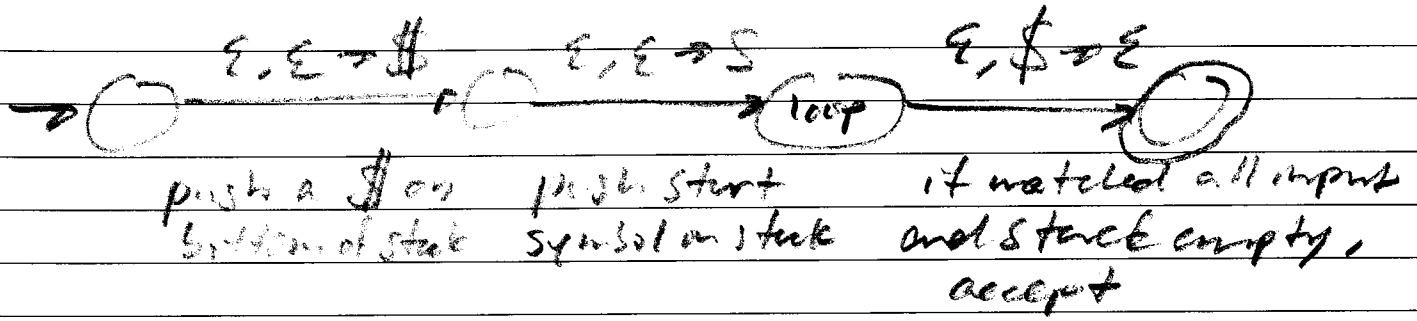
Strategy:
- push $
- push start nonterminal
- if nonterminal choose right rule and push to stack





Date.
No.

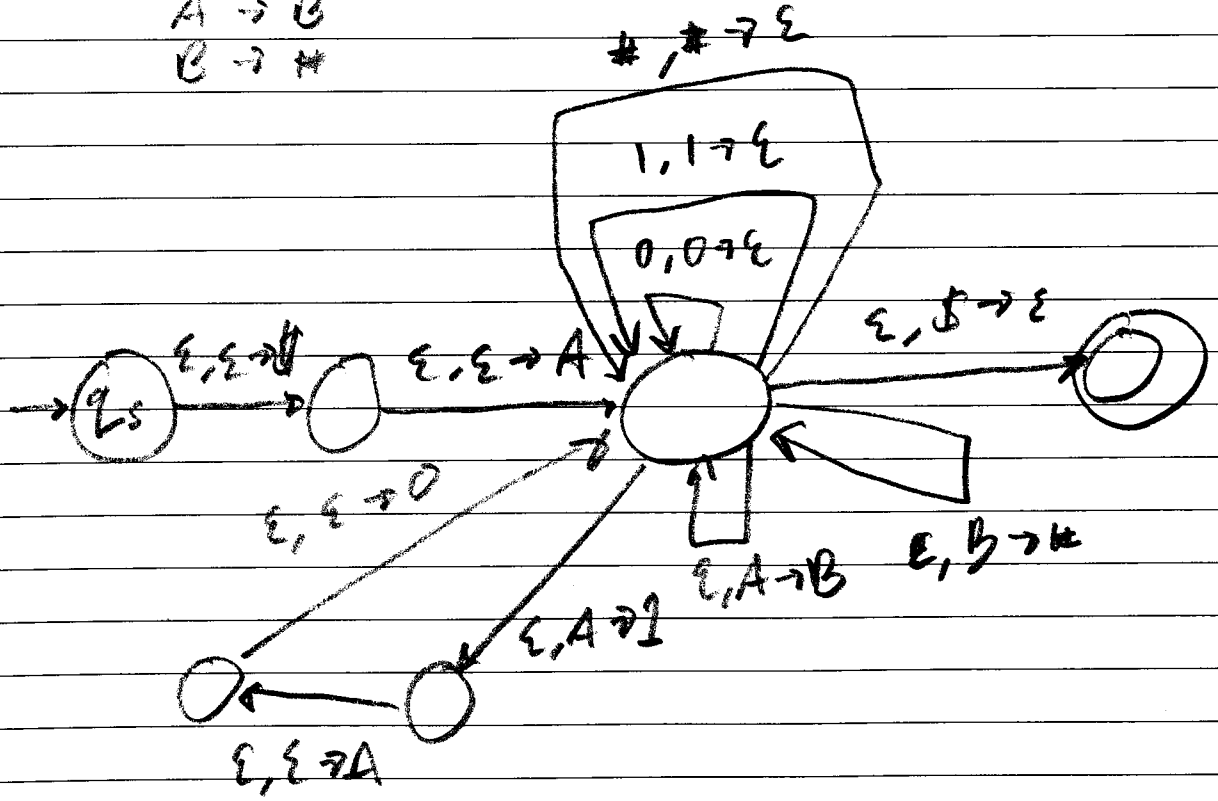
Basic CFG → PDA structure:



Now add two types of loops around "loop":

1. Loops that match a terminal from Σ
2. Loops that replace a nonterminal like A by one of the rules A → w x y z

Ex. A → 0A1
A → B
B → #



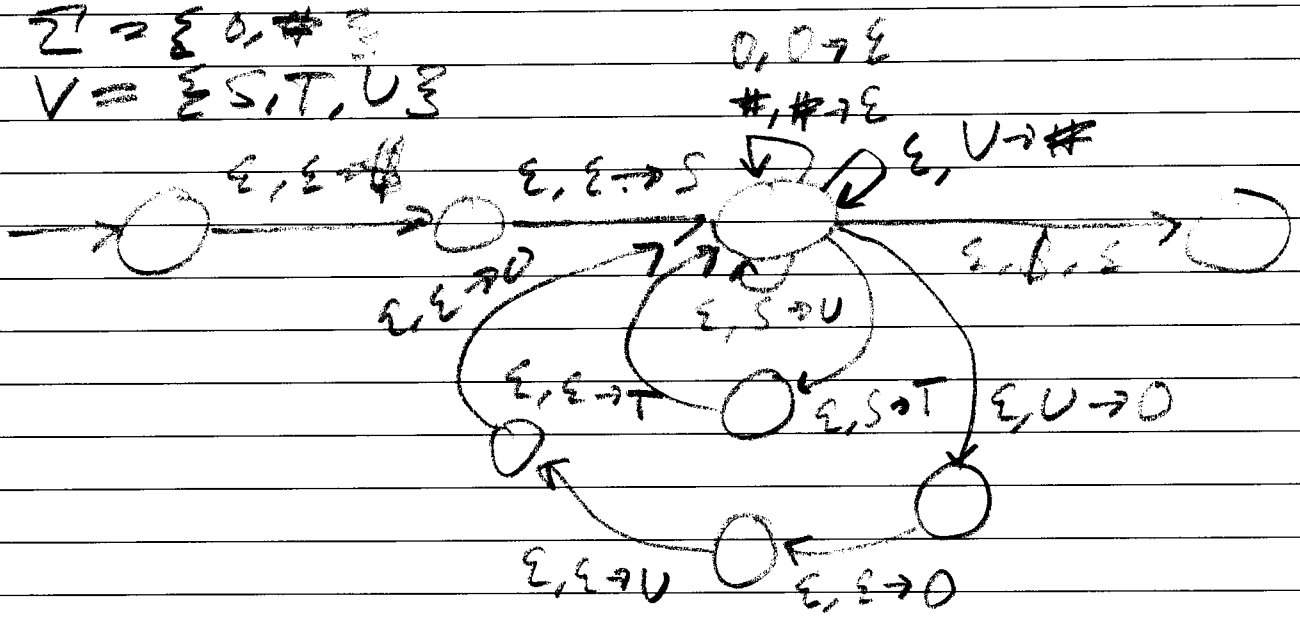
Date. 10.30.25
No. 18

Pumping Lemma for CFLs

Ex. Design a PDA for

$S \to TT \mid U$
$T \to 0T1 \mid T01 \mid \#$
$U \to 0U00 \mid \#$

$\Sigma = \{0, \#\}$
$V = \{S, T, U\}$



Convert CFG to CNF:

Ex. Convert ~~$S \to AaBb$~~, ~~$A \to aB$~~, ~~$A \to aB$~~, $B \to b$, $B \to c$, S the start symbol

$A \to b$
$A \to c$
$U_1 \to a$
$U_2 \to b$

$S \to AA_1$
~~$A_1 \to aA_2$~~
~~$A_2 \to Bb$~~

$A \to U_1 B$
$A_1 \to U_1 A_2$
$A_2 \to B U_2$

All rules of this in worksheet 4

END of exam 2 material

Date.
No.

Are there limits to PDAs?

Consider input $a^k b^k c^k$ for some k
if k is fixed, we can build a PDA with
at most $3k+1$ states

but if k is arbitrary? the stack can't
remember an arbitrary k twice.

Recall.
every CFL generated by CFG
every CFG has a PDA that recognizes it
given a string in a CFL, it can be parsed into
its CFG
every PDA can be mapped to a CFG and
is equivalent to a CFL

Pumping Lemma for CFL:

if L is a CFL $\exists p$ where
for any string s where $|s| \geq p$
s can be divided into 5 substrings $s = uvxyz$
where:
$|vy| > 0$ (i.e either at least v or y not empty)
$|vxy| \leq p$ (ignore leading and trailing z's)

* so that for all $i \geq 0$, $uv^i xy^i z$ is in L

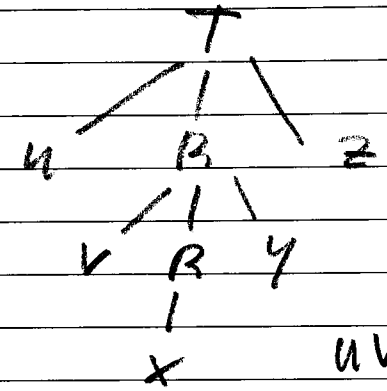
note. u, x, z may be empty strings

Date.
No.

Consider any CFL L: any string in L can be parsed

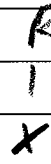
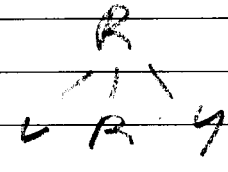
Ex.

$T \to uRz$
$R \to vRy$
$R \to x$

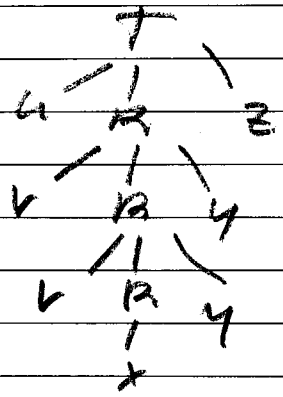


$|V| = |\{T, R\}| = 2$ $\quad uvxyz$

Assume depth of parse tree exceeds $|V|$
Thus some nonterminal must be reused

pull  out of the tree and label 

becomes

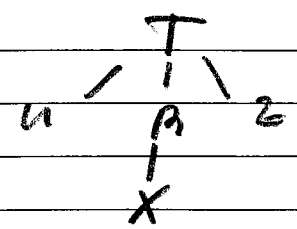


$uvvxyyz$

is pumped vRy

can do this again and again . . uv^ixy^iz

Or if we don't invoke R : uxz



Pumping CFL Exs

Date. 11.4.25
No. 19

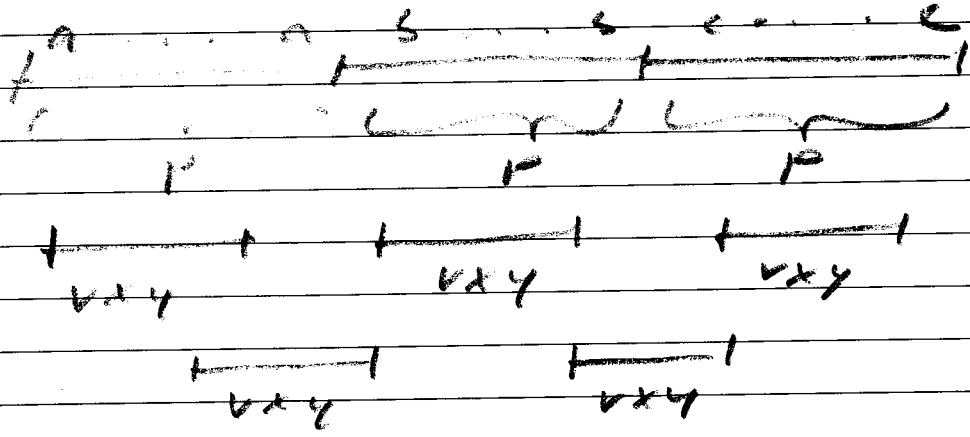
Ex. $B = \{a^n b^n c^n \mid n \geq 0\}$ is not a CFL

Assume B is a CFL

$\exists p$ s.t. $\forall s \in B$, $|s| \geq p$ and $s = uvxyz$ where:

1. $|vy| > 0$
2. $|vxy| \leq p$
3. $\forall i \geq 0$, $uv^i x y^i z \in B$

Consider $s = a^p b^p c^p$. By (1) and (2), vxy must be either:



i.e. all are char or on the boundary of a char

Case A: vxy in a single block. pumping up will give more a's / b's / c's than the other two letters. no longer in B. contradiction

Case B: vxy on boundary. Pumping up will add more a's/b's OR b's/c's than the other third letter. s no longer in B. contradiction

A more exs of this in slides today

Date.
No.

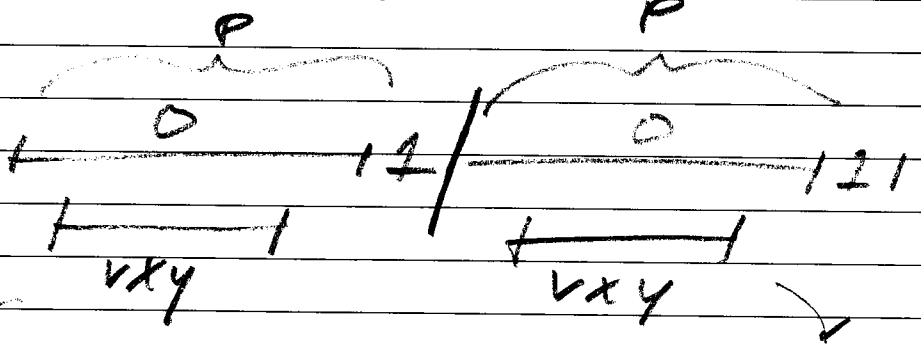
Ex. $D = \{ww \mid w \in \{0,1\}^*\}$ is not CFL

Assume D is a CFL:

$\exists p$ st $\forall s \in D$, $|s| \geq p$ and $s = uvxyz$ where:

1. $|vy| > 0$
2. $|vxy| \leq p$
3. $\forall i \geq 0$, $uv^i x y^i z \in D$

Consider $s = 0^p 1 0^p 1$. This divides into $0^{p-1} 0 1 0 0^{p-1} 1$



if pump down, less zeros in first half than second
vice versa

Turing Machines

Date. 11/18/25
No. 22

Estimating p in CFGs

Assume G is a CFG where b = max # symbols on R+LS

$A \to a_1 a_2 \ldots a_b$ where a_i is terminal or nonterminal

Thus no vertex can have more than b children

Thus if height $= h$, no more than b^h leaves

Thus if $|s| \geq b^h + 1$ leaves, must have at least $h+1$ height

With $|V|$ nonterminals, height of $|V|+1$ has at least 2 repeated nonterminal so has up to $b^{|V|+1}+1$ leaves

So p must be no bigger than $b^{|V|}+1$

Turing Machines: computing with Read/Write tapes

Are there limits to PDAs? YES

the stack can't remember an arbitrary n twice!

Date.
No.

Turing Machine (TM) Model:

7-tuple: $(Q, \Sigma, \delta, q_0, q_{accept}, q_{reject}, \Gamma)$

Q, Σ, q_0 as before

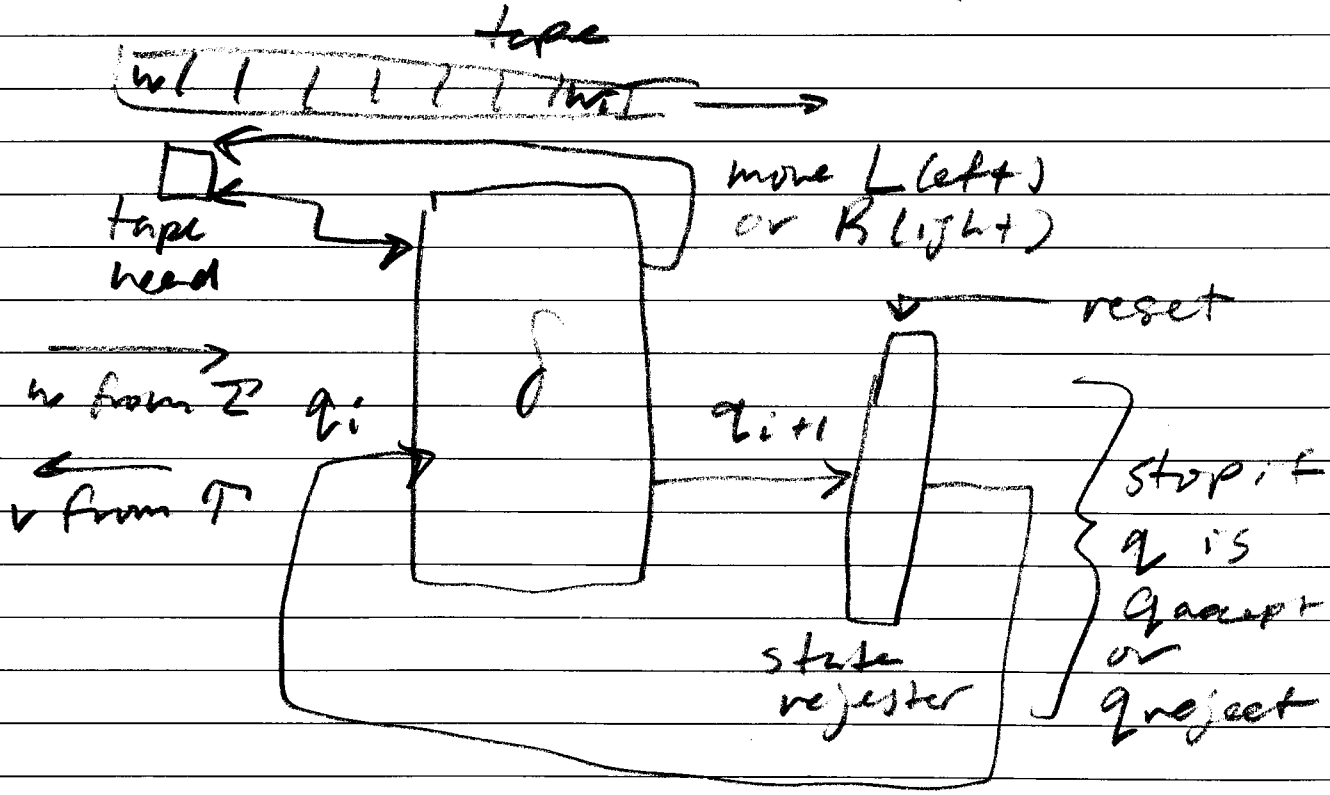
Γ: symbols that may be written

$\Sigma \subseteq \Gamma$

q_{accept}: stop and accept state

q_{reject}: stop and reject state

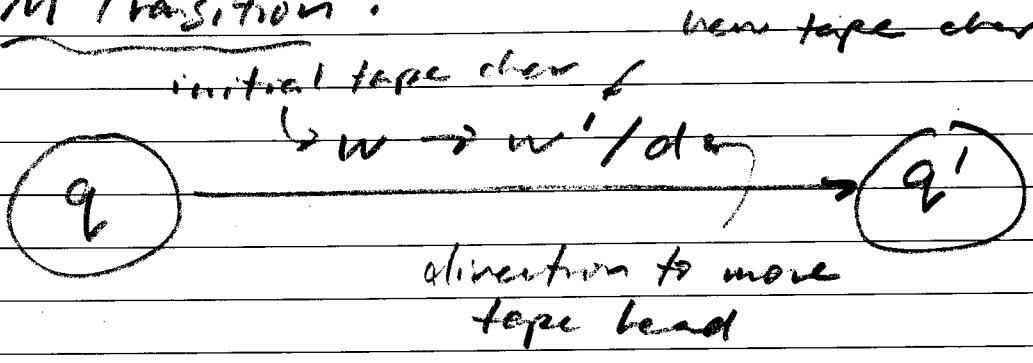
$\delta: Q \times \Gamma \to Q \times \Gamma \times \{Left, Right\}$



∴ no nondeterminism (for now) ∎

Date.
No.

TM Transition:



read symbol under tape head: w
given current state q, find $\delta(w, q) = (w', q', dir)$

TM differences from PDA:

1. tape is infinite size
   - only leftmost holds initial data
   - rest holds □ (blank)
   - □ ∈ Γ (always)
2. can write to tape
3. can move left on tape
4. halts as soon as enters either
   - accept
   - reject
5. BUT can loop forever!

Date.
No.

DTMs and Languages:

Assume M is a DTM and L is a language

M recognizes L if given any string w

- if $w \in L$, $M(w)$ halts in accept
- if $w \notin L$, $M(w)$ *never* halts in accept
  - but may halt in reject or loop forever

M decides L if given any string w

- if $w \in L$, $M(w)$ halts in accept
- if $w \notin L$, $M(w)$ halts in reject

Language of L: set of all strings $w \in L$ that is accepted in $M(L)$

Recognizing $a^n b^n c^n$

Turing Machine Variations

Date. 11/18/25
No. 23

A configuration C:
- current state
- current tape contents
- current head location

left ← state → right

current head

TM transition:
- change from one configuration C_i to another C_{i+1} that obeys TM transition rules

$C_i = uaq_i bv$ more formally
(u: left, b: right)

TM Variations:

- stay option: "S" direction to not move head
- multiple tapes: each independent of each other

$$\delta : Q \times \Gamma_1 \times \dots \times \Gamma_k \to Q \times \Gamma_1 \times \dots \times \Gamma_k \times \{L, R, S\}^k$$

- bidirectional infinite tape:
- TM with a stack
- Queue Automata: replace tape w/ infinite queue
- Non-deterministic transitions
- Two stack machine
- Probabilistic Turing Machine

Date.
No.

Enumerators: useful type of DTM

- DTM + printer
- programmed with grammar of some language L
- starts with blank tape
- generates and prints out all strings in L
  - possibly random order
  - may go on forever

"enumerates L"

Theorem. L is Turing-recognizable if some enumerator enumerates it

Pf. Assume enumerator E
given string w, is it in L
compare output of E
if in L, eventually halts
if not in L, and L infinite, E never halts

given recognizer M, how to build E
generate all possible strings in Σ^*
run each through M
if M accepts it, print it out

Result. None of these variations are more powerful than basic TM

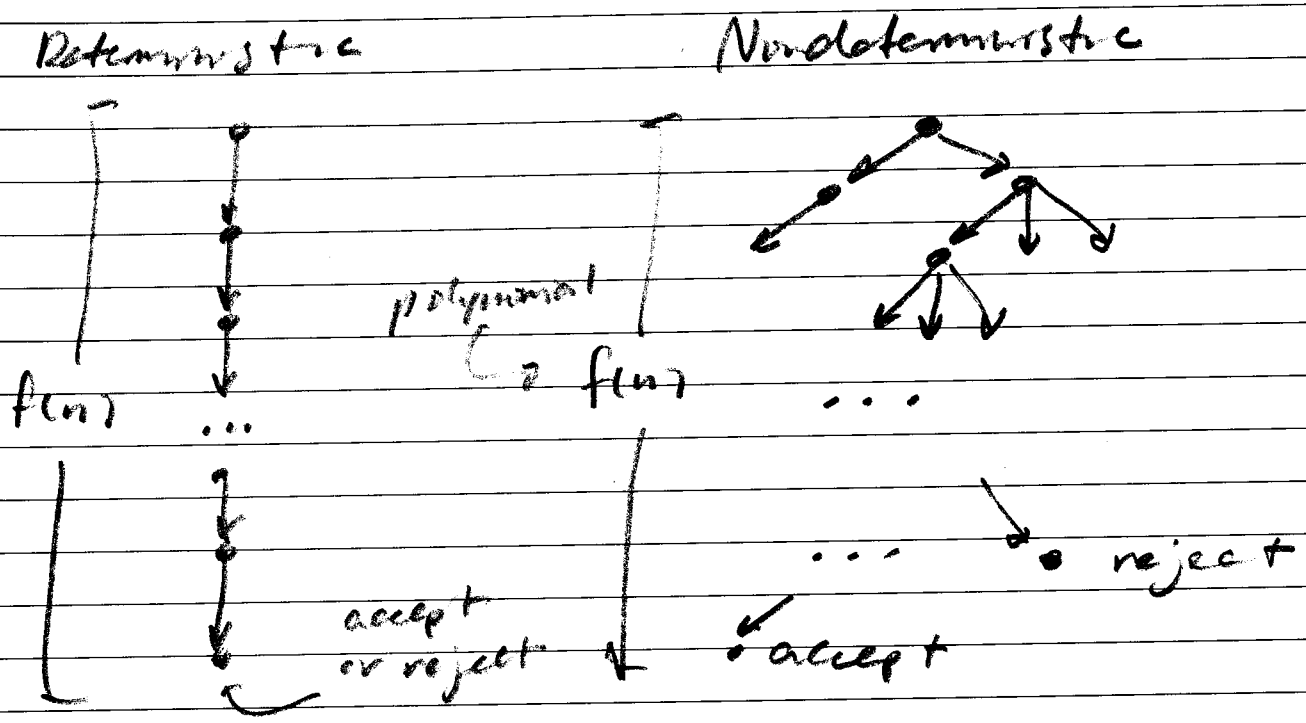
Eg. simulating k-tape TM on 1-tape

single tape = #1 ... tape1 ... # tape2 ... # ... tapek ... #

if time on k-tape is $t(n)$, 1-tape runs in $t^2(n)$ time

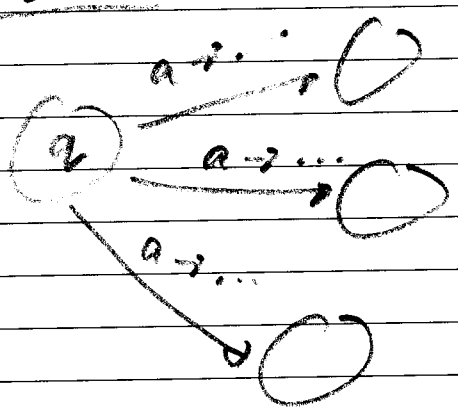
Nondeterministic TMs

Date. 11/20/25
No. 24



if $|Q|$ states, in $f(n)$ time at most $|Q|^{f(n)}$ branch points

transitions:



in state q, if "a" on tape, multiple options available

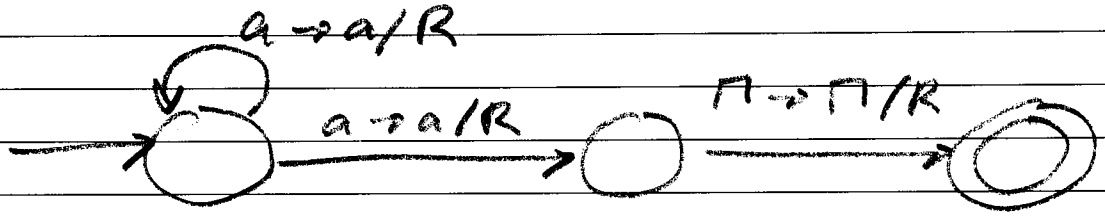
If a NTM can solve a problem in $O(f(n))$, then a DTM can solve it in $O(|Q|^{f(n)})$

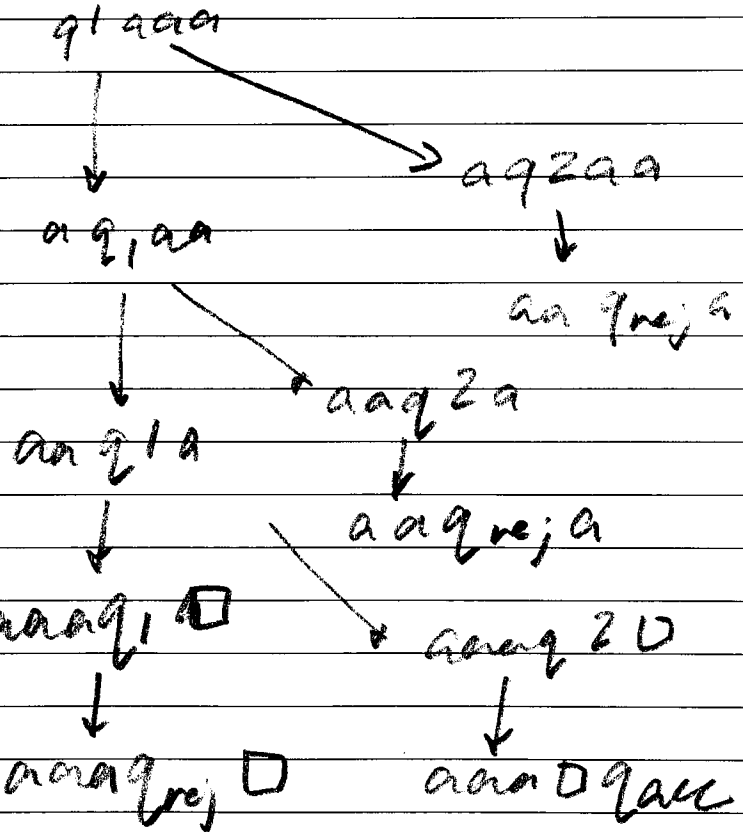
Date.
No.

Nondeterministic Algorithms

1. Generate (nondeterministically) a certificate
2. Run certificate and problem through DTM verifier and return answer

Ex. Accept a^+

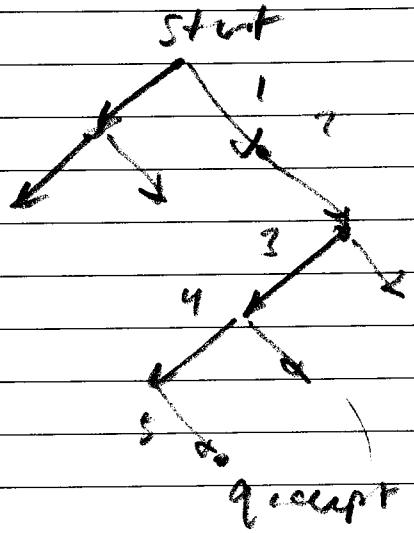




Ex: given string 1^n, is n composite?
in slides!

Date.
No.

Timing an NTM :



shortest path from start to q accept

5

Language Recognition : for language L

recognizable :
- always accepts $\forall S \in L$
- never accepts $\forall S \notin L$

decidable : a turing machine recognizes it and always halts!

turing-recognizable : iff NTM recognizes it

turing decidable : iff NTM decides it

Date. 11/25/25
No. 25

Algorithms

Definitions.

Computability: ability to answer a question via a mechanical process

Automaton: performs a function according to a predetermined set of coded instructions

Function: $f: D \to R$

Effectively Computable: computable by a well-defined mechanical process

Church-Turing Thesis: Any function over $\mathbb{N}$ is computable iff computable by a TM

Gödel and Herbrand: General Recursive Functions can make $f: \mathbb{N} \to \mathbb{N}$ using just constant, projection, successor

Peano's Axioms: all you need for arithmetic
- $0 \in \mathbb{N}$
- $\forall x, y$, if $x = y$ then $y = x$ ...
- $\exists$ successor function $S(x)$ ...

details in slides

Kurt Gödel's Incompleteness Theorem:
if a system is consistent, it cannot be complete vice versa
details in slides

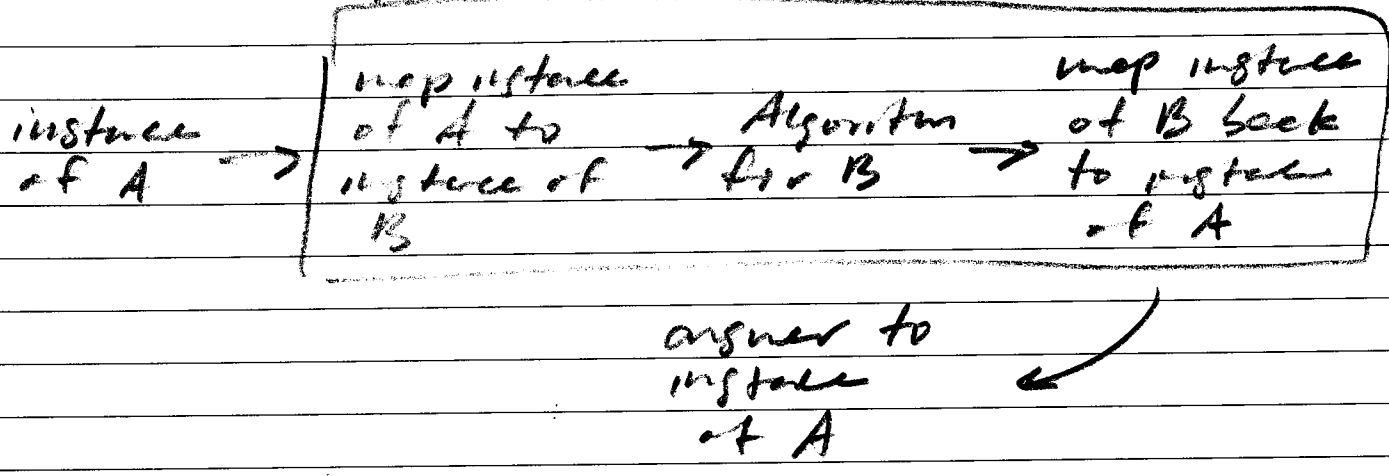
Big O:
if $f(n) =$ # steps in TM
$g(n): \mathbb{N} \to \mathbb{R}$

$f(n) = O(g(n))$ if $\exists n_0 \in \mathbb{N}$ and $c \in \mathbb{R}$
$\forall n \geq n_0, f(n) \leq c\,g(n)$

Reducibility, Decidability

Date. 12.2.25
No. 26

Reducibility: showing problem A is as simple as problem B



Complexity of A = O(Map A→B) + O(B) + O(Map B→A)

More formally,

$f : D_f \to R_f$ is reducible to $g : D_g \to R_g$ if

$\exists\, h : D_f \to D_g$ and $\exists\, q : R_g \to R_f$ s.t.

$\forall\, x \in D_f$, $f(x) = q(g(h(x)))$

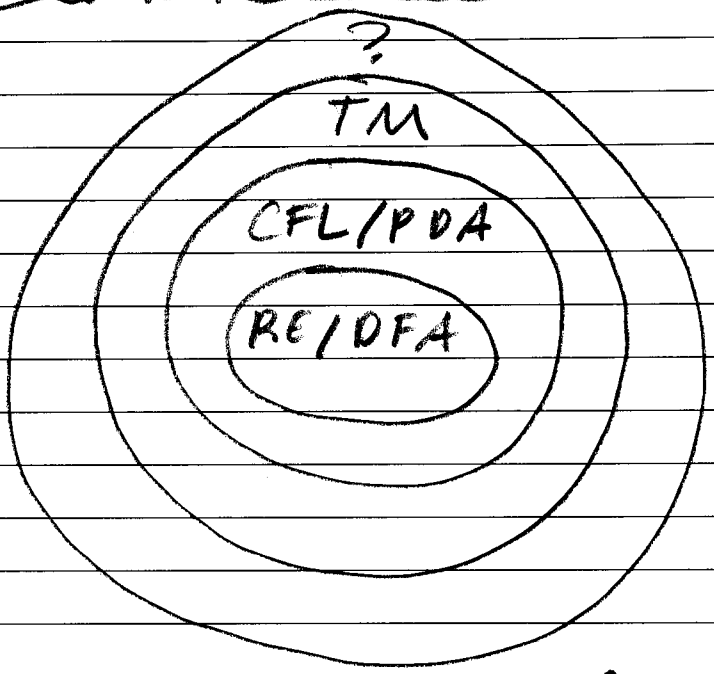
$x \in A \to y = h(x) \to z = g(y) \to \ell = p(z) \to \ell \in A$

Complexity of A = $O(h(x)) + O(g(y)) + O(p(z))$

A ex's of this in slides 4

Date.
No.

Decidability, Languages, Automata:



language classes based on type of automaton needed for decider

Decidability, Complexity

Date. 12.4.25
No. 27

Ex. from Quiz 12

Assume problem class A can be reduced to problem class B

i.e. $A \leq B$, meaning B is at least as hard as A

a) if there exists a polynomial solver for B, then A is polynomial!

b) if there exists an exponential solver for B, then A solver for A is known to exist, but of unknown complexity

Intuition, $A \leq B$ means:

1. You can turn an instance of A to an instance of B in polynomial time
2. If you can solve B, then you can solve A
3. The complexity of the solver for B is an upper bound for the solver for A

c) if no solver for B can exist, then:

we cannot tell if A has a solver

d) if no solver for A can exist, then:

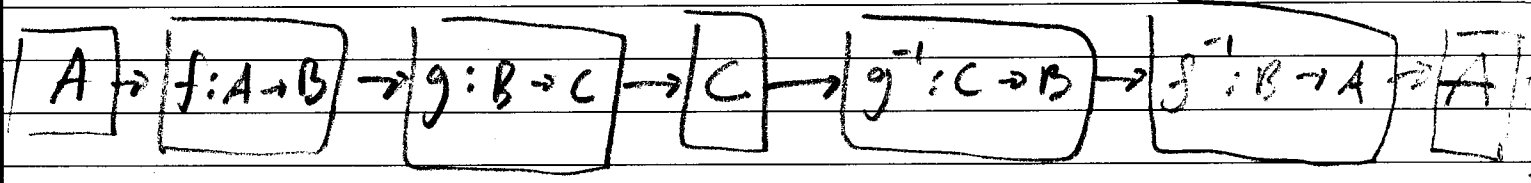
B cannot have a solver

Date.
No.

Another Ex. Three classes of problems A, B, C

A reduces to B, and B reduces to C
ie. $A \leq B \leq C$

Draw a picture of this chain, starting with A:



a) if C has polynomial decider? so does A, B

b) if B has polynomial decider? A is poly, C unknown

c) if A has poly decider? tells us nothing

d) if C has exponential decider? A, B are decidable, unknown complexity

e) if B has exp decider? A is decidable, C unknown

f) if A has exp decider? tells us nothing

g) if C cannot be solved? tells us nothing

h) if B cannot be solved? C cannot be solved, so into a sort A

i) if A cannot be solved? Both B and C cannot be solved either

j) if $C \leq A$, and is polynomial? all 3 polynomial

k) if $C \leq A$, and is undecidable? all 3 undecidable

Date.
No.

Undecidability:

Decision problem: problem poses a yes/no question

Decidable: decision problem for which an algorithm exists that always leads to a correct answer to a decision problem

Undecidable: decision problem for which it's proved to be impossible to construct an algorithm that always leads to the correct yes/no answer

Non-Decision Problems and Verifiers:

desired result more than a yes/no

Verifier: V for language L: function f that for any string w:

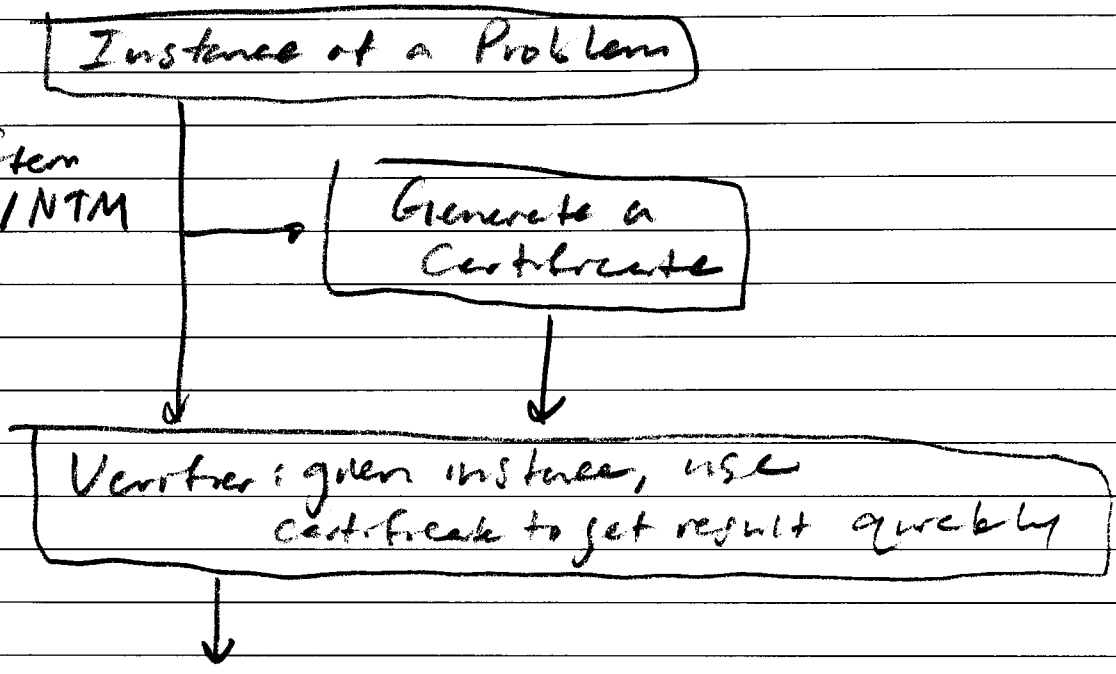
- $w \in L$, $\exists f(w) = c$ where V accepts (w, c)
- $w \notin L$, no such c exists

Certificate: String c (key, solution)

ex. nonprime: w is a number, c is a divisor
hamiltonian: w is a graph, c is a sequence of vertices

Date.
No.

Solution via Verification:



intuition: many "tough" problems can be "solved" quickly with a "hint"

Verification often doable w/ DTM in poly time

Generating hint often possible with NTM

Date.
No.

Languages over Automata:

Consider $L = \{\langle B, w\rangle \mid \langle B, w\rangle$ has some property $\}$

B: "Encoding" description of some automata B
as a string
i.e. express each element of formal model tuple
as a string

w: some possible input string for that TM

Property: String $\langle B, w\rangle \in L$ if B does something with w

Properties of such Languages:

Turing Decidable: TM decider that always answers
Yes/No

Turing Recognizable: TM recognizer exists where

- Always stops and answers yes if $\langle B, w\rangle$ has property
- Never answers yes if $\langle B, w\rangle$ does not have property
- And might loop forever if not yes

Turing Undecidable: may be recognizable but not
decidable

Co-Turing Recognizable: recognizer for complement of L

* list of classes of languages in slides / worksheet

Date.
No.

The Halting Problem:

Consider $A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM which accepts } w\}$

Recognizer U emulates M on w and stops when M accepts

Assume decider H exists:
- stops in q_{accept} if M halts
- stops in q_{reject} if M loops and never halts

Define D: takes $\langle M\rangle$, runs H on $\langle M, M\rangle$, outputs opposite of H

Run D on itself: on $\langle D, D\rangle$
- D halts and accepts if its simulated D rejects itself
- D halts and rejects if its simulated D accepts itself

Contradiction: Thus H cannot exist!

we can never decide yes/no if a TM will go into an infinite loop

i.e. A_{TM} is undecidable!

Further Theorems:

Theorem 4.22: Language is decidable exactly when both it and its complement are Turing recognizable

Theorem 4.33: Complement of A_{TM} is not Turing Recognizable (if it were, then A_{TM} would be Turing-decidable)

Date.
No.

Complexity Classes of Problems:

divides problems based on TM solver time complexity as a function of length of input tape

Class P (Polynomial): some 1-tape DTM can solve it in $O(n^k)$ time for some k

Class NP (Non-deterministic Polynomial): some NTM can solve it in $O(n^k)$ for some k

Class NP-Complete: any problem in NP can be solved by reduction to a solver in this class

If any NP-complete problem is in P, then all of NP is

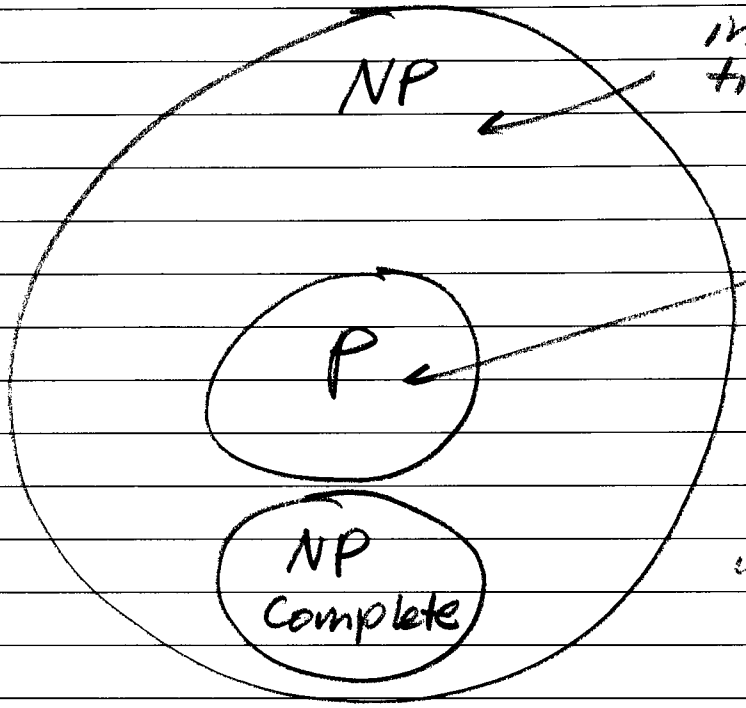
note, more details and exs of these in slides

Does P = NP? Is there any NP complete problem that is in P?

NP-Hard: a class of problems at least as hard as NP but bigger than NP

Date. 12.9.25
No. 28

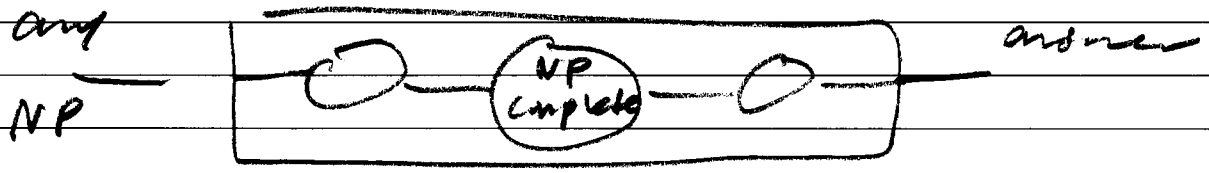
Final Exam Review



in polynomial time on NTM

in polynomial time on DTM

if in NP but not P
- poly on NTM
- unknown on DTM



universal solvers, can be reduced for any NP problem

Date.
No.