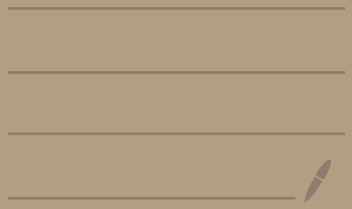
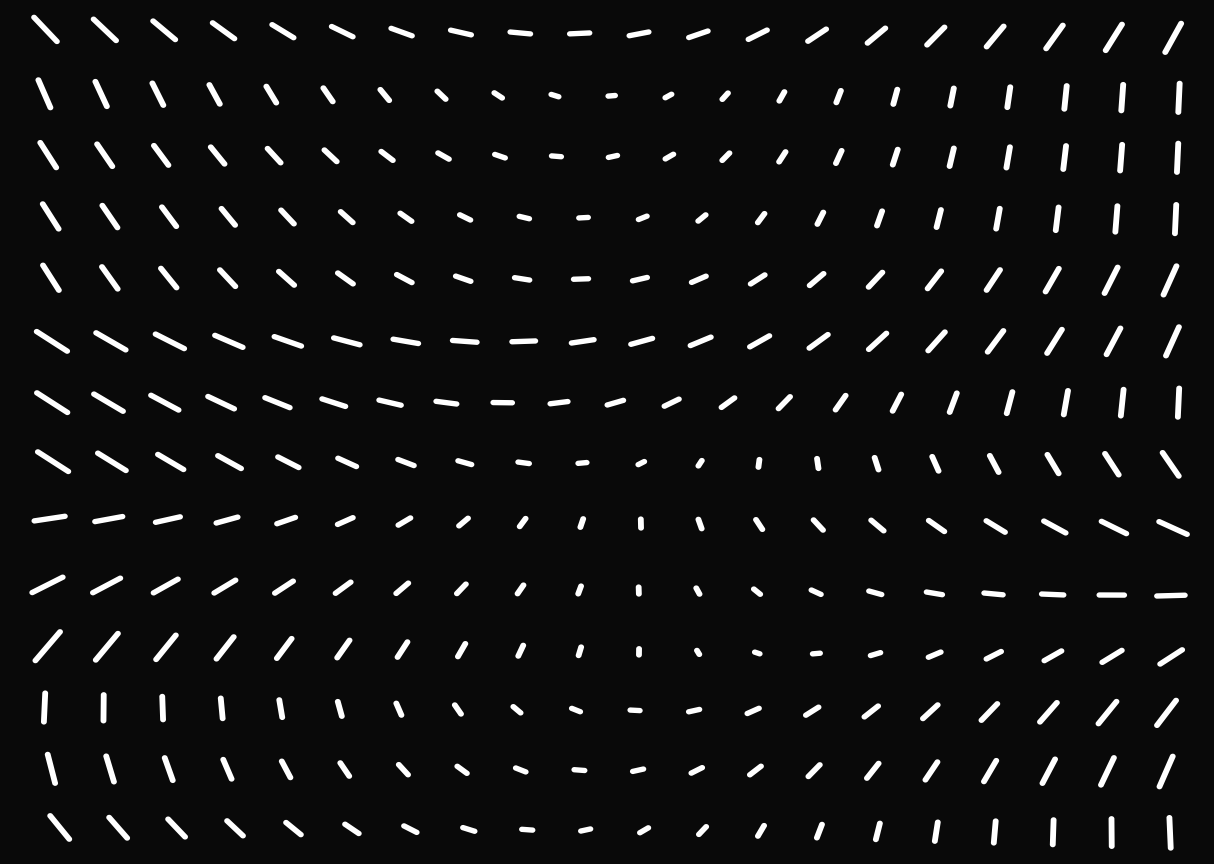
CISC 223



Lecture notes



Week 1 - Introduction Background

Overview of Theory of Computation

- Automata Theory - models of computation and their properties.
- Computability Theory - Problems that can/can't be solved with computers
- Complexity Theory - How efficiently problems can be solved.

Complexity Theory

- easy problems vs hard
- Sorting numbers → easy for large datasets
- Classification: problems are categorized by difficulty.
- Real world application: Cryptography, hard to compute

Computability Theory

- Early studies showed, some problems are impossible to solve with any computer
- Led to theoretical models
- Computability vs. Complexity
  computability → Focus on if problem has solution
  complexity → Focuses on how hard it is.

Automata Theory

- Automata are abstract models of computation used in:
  - Text Processing
  - Compilers
  - Hardware
- Finite Automata: simple models useful for search engines
- Introduces formal definitions and logic, used in other areas.

Mathematical Notation - sets

- A set is a collection of objects, $S = \{7, 21, 57\}$
- $\subseteq$ - subset
- $\cup$ - union
- $\cap$ - intersect
- A' - complement

Sequences and Tuples

- Sequence: ordered list of elements $(7, 21, 57)$
- Tuple: Finite sequence $(1, 2)$
- Cross product: $A = \{1,2\}, B = \{x, y\}$

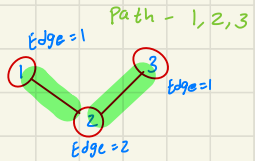
$$A \times B = (1,x), (1,y), (2,x), (2,y)$$

Functions / Relations

- Functions maps inputs to outputs $f(x) = x^2$ $3 \rightarrow 9$
- Unary (1 input)
- Binary (2 inputs)
- onto function: uses all values in range

Graphs

- A graph is a set of nodes connected by edges
- **Degree**: Number of edges at a node
- **Path**: Sequence of nodes connected by edges
- **Cycle**: A path that starts and ends at same node
- **Undirected Graph**: Edges have no direction
- **Directed Graph**: Arrows show direction



n - Vertices = n - 1 edges
For a Tree

Strings / Language

- A string is a sequence of characters
- Concatenation: "ab" + "cd" = "abcd"
- Languages: Set of strings over an alphabet

Alphabet: $\{0, 1\}$

Language: $\{$"0", "1", "10", "11", "101"$\}$

Boolean Logic

Operations:

- Not, negates value ($\neg$)
- And, True if both 1
- or, True if atleast 1 is true
- xor - True if inputs are
- Implication $\rightarrow$

Week 2/3 Lecture Notes

Formal Language and Automata

- Automation is an abstract machine that processes input and changes states based on rules.
- Automata helps us define formal languages, which are sets of valid strings

Example
Gate: has 2 states, locked / unlocked

Limited Memory: dishwashers, electronic doors, thermostats

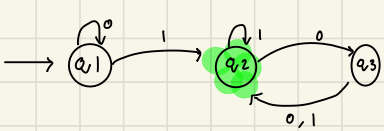
3 types of Automata

1.) Finite Automata (FA) - Models fixed number of states
- Used in regular expressions / pattern matching
- Can't remember past information

2.) Pushdown Automata (PDA) - Finite Automata with a Stack
- More powerfull than finite automata
- Used for parsing

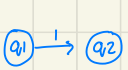
3.) Turing Machines - Most powerfull model
- used to define what can be computed

Simple Example M1



- q_2 is accepted State
- Must end in 1, or even amount of 0's
- Starts at q_1, lines are called transitions

Transition function, denoted δ, is the rule for moving between States.



we say $\delta(q_1, 1) = q_2$

Formal Definition for finite automata

- 5 tuples $(Q, \Sigma, \delta, q_0, F)$, where

1.) Q is a finitite set of States

2.) Σ is finite set of Alphabet

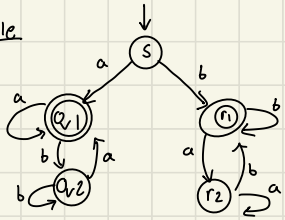
3.) δ is transition function

4.) q_0 is the start State

5.) F is the set of accepted States

- If machine M accepts a set of strings, A. Then A is the language of machine M. $L(M) = A$
- A machine can accept many strings, but only recognize one language
- If a machine accepts no strings, the empty language is correct

Example



DFA is already an NFA (with single transitions)

Accepts: a, b, aa, bb

Rejects: ab, ba, bbba

Rule: Accepts all strings that start and end with the same symbol

Regular Languages

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automation.
  Let $w = w_1 w_2 \dots w_n$, where w_i is the member of Σ

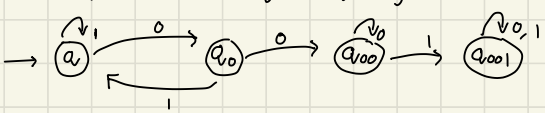
M accepts w if $r_0, r_1 \dots r_n$ exists with 3 conditions

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0 \dots n-1$
3. $r_n \in F$

- M recognizes language A if $A = \{w \mid M \text{ accepts } w\}$
- Language is regular language, if finite automata recognizes it

Example

M_1 only accepts strings containing 001

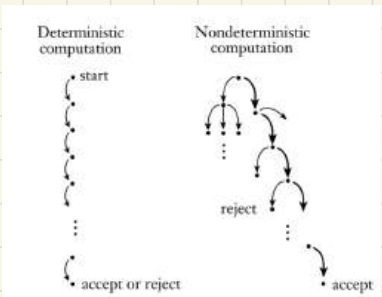


Regular Operations

- Union: $A \cup B$, if L_1 and L_2 are regular, so is their union
- Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
- Star $A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$ (if L_1 is regular, L^* is regular)

Deterministic vs Nondeterministic Finite Automata (DFA vs. NFA)

- DFA, Exactly one transition per input
- NFA, Can have multiple transitions, and ε moves
- Every determinism finite automata is an NFA
- For NFA, when you have a transition with multiple choices, it creates multiple branches and continous to check proceeding inputs



Epsilon (ε) Transitions: Machine can move between states without reading next character.

Having an input alphabet, with single symbol would be unary alphabet.

Non-deterministic Finite Automata

1.) q is finite set of states
2.) Σ is a finite alphabet
3.) $\delta : q \times \Sigma_\varepsilon \to P(q)$ is the transition function
4.) q_0 is the start state
5.) $F \subseteq q$ is the set of accepted states

Equivalence

- 2 machines are equivalent, if they recognize the same language
- DFA's and NFA's recognize same class of languages (regular)
- Every NFA has an equivalent DFA
- A language is regular iff some NFA recognizes it.

*pg 26

Converting NFA's to DFA's

1) Define DFA States

- DFA states correspond to all subsets of the NFA's states
- If NFA has n states, DFA may have up to 2^n states

2) Handle ϵ transitions

- If the NFA has ϵ moves, we first compute $E(R)$, reachable states from ϵ.
- $E(R) = \{ q \mid q$ can be reached from R, by traveling along 0 or more ϵ arrows$\}$

Ex

If state 1 in NFA has ϵ move to state 3, then $E\{1\} = \{1, 3\}$

3) Construct Transition Function

- Each DFA has 1 transition per input symbol
- New DFA moves to all possible NFA states

$$\delta(R, a) = \{ q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R \}$$

4) Find Start and Final States

- DFA starts from $E\{q_0\}$, set of states from ϵ moves
- Any NFA state that's final should be final for DFA.

NFA - 3 states

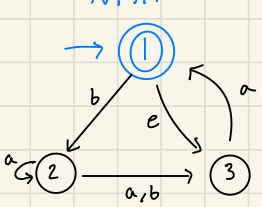
D - $2^3 = 8$ states

$E(\{1\}) = \{1, 3\}$

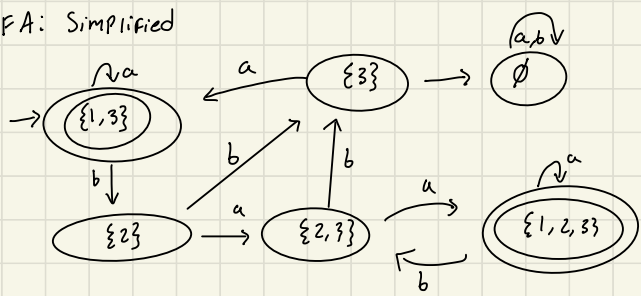
Revisit ★

Example

NFA:



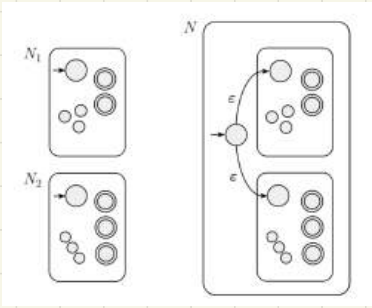
DFA: Simplified



Proofs: How regular languages are closed

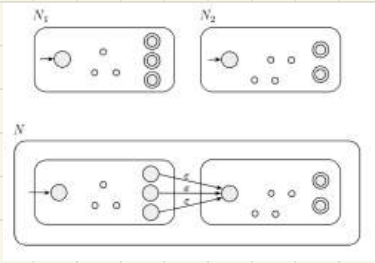
1) closed Under Union ($L_1 \cup L_2$)

- Key Idea: If NFA N_1 recognizes L_1 and NFA N_2 recognizes L_2, we can construct a new NFA that accepts $L_1 \cup L_2$.
- Create a new start state q_0
- Add ε moves to both starting states
- Accept if either N_1 or N_2 are accepted



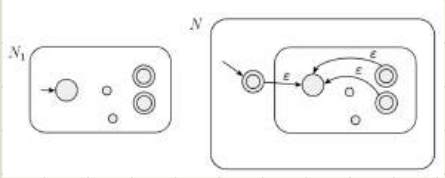
2) Closure Under Concatenation ($L_1 L_2$)

- If L_1 = strings in q_1, L_2 = strings in q_2 a string must go through both machines in order
- Connect all accepting states of N_1, to the start state of N_2, ε moves
- ε allows switch without consomption input



3) Closure under Star

- A language L^* means any number of repititions of L
- Add new start state (accepting state)
- ε moves from q_0 to start state N
- Add ε moves from each final state of N back to the start



1.3) Regular Expressions

- way to describe a set of strings using pattern matching rules
- regular expressions define regular languages

Ex $(0 \cup 1)0^*$: Set of all expressions that start with 0/1, followed by any number of 0's.

Regular Operations

- Union ($\cup$) represents choice (OR)
- Concatenation ($\circ$, implicit) - joins strings seaveniially
- Kleenes Star ($*$) - represent repetition

Precedence order -> Star, Concat, Union

Formal Definition of Regular Expressions

- A single symbol ($a, b, 0, 1$)
- Empty string (ε)
- Empty set ($\emptyset$)

Note: Difference between $\varepsilon, \emptyset$.
ε is empty string
$\emptyset$ is empty set

- Union of two regular expressions: $R_1 \cup R_2$
- Concatenation of two regular expressions: $R_1 R_2$
- Kleene Star of a regular expression: R_1^*

Regex	Description	Example Strings
`0*10*`	A single `1` surrounded by any number of `0`s.	`1`, `01`, `00100`
`Σ*1Σ*`	Any string with at least one `1`.	`101`, `110`, `11`
`Σ*001Σ*`	Any string containing `001` as a **substring**.	`001`, `1001`, `000101`
`1*(01*)*`	Every `0` must be followed by at least one `1`.	`1`, `1101`, `1011`
`(ΣΣ)*`	All strings of **even length**.	`00`, `11`, `1010`
`(ΣΣΣ)*`	All strings of **length multiple of 3**.	`000`, `110`, `101101`
`0 ∪ 1 ∪ 10`	The set `{0, 1, 10}`.	`0`, `1`, `10`
`Σ*0 ∪ 1Σ*`	Strings that start with `0` OR end with `1`.	`01`, `1101`, `011`
`(0 ∪ ε)1*`	Adds either `0` or `ε` before `1*`.	`1`, `01`, `111`

Regular Expressions in Programming

- Elemento objects in programming languages, tokens
- Automatic systems generate lexical analyzer (input compiler)

Equivalences

- Regular expressions and Finite automata have same power to recognize language
- Regular expression → Finite automation

Theorem 1.54 - A language is regular iff some regular expression describes it.

- First way:
  - Take a RE, R and construct NFA
  - Because NFA's can be converted to DFA's, language is regular
- Second way:
  - Convert DFA to equivalent regular expression
  - Every regular language can be expressed using union, concatenation, star
  - DFA → GNFA (generalized nondeterministic FA)
  - GNFA allows transitions labeled with regex instead of single symbols

What is GNFA

- A GNFA is a variation of an NFA, but with transitions labeled with regular expressions
- GNFA (5 Tople)
  1) Q is finite set of states
  2) Σ is input alphabet
  3) δ is transition, maps state pairs to regular expressions
  4) q - start is uniave start state
  5) q accept is accept state

- GNFA processes input string, by reading substrings that match regular expressions
- GNFA has single start state / single accept state

Converting a DFA to a GNFA.

1) Converting DFA to GNFA

- add new start state and accept states with ε transitions from original accepting states
- replace multiple transitions between same states with regular expressions using union ($\cup$).

Ex

DFA has

$q_1 \rightarrow q_2$ on a

$q_1 \rightarrow q_2$ on b

GNFA transition is $a \cup b$

2) Remove States Systematically

- remove states one by one, updating transition label.
- When we remove q_{rip} we need to make sure we dont lose any paths that went through it.

Ex

$q_1 \rightarrow q_{rip}$ with a

$q_{rip} \rightarrow q_{rip}$ with b

$q_{rip} \rightarrow q_2$ with c

$q_1 \rightarrow q_2$ with d

To replace the paths, the transition from $q_1 \rightarrow q_2$, would be $d \cup (ab^*c)$

d was direct path, a, b to get in, c to leave

3) Continue Until only two States remain

- repeat State elimination untill only q_{start} and q_{accept} remain

Full example

DFA: {1, 2}

Transitions

1 -> 1 on a

1 -> 2 on b

2 -> 2 on a, b

- add new start/accep state with an ε move from 1 and 2
- Update transition formula, $b(a \cup b)^*$

Then Final regular expression: $a^*b(a \cup b)^*$

accept stat a, any number of a followed by 1 b, Then any a, b after.

Week 4/5 Lecture Notes

What is Context-Free Language

- Used to describe more complex languages
- CFG's are more powerfull, allowing recursive rules
- Used for Parsing / NLP

Definition of a CFG

- A CFG is a 4 tuple (V, Σ, R, S)
  - V is a finite set of variables (non-terminals)
  - Σ finite set of terminals (input symbols)
  - R finite set of rules (productions) in form $A \rightarrow w$
    - A is a variable
    - w is a string of terminals
  - S is a start symbol which is variable in V.

Ex

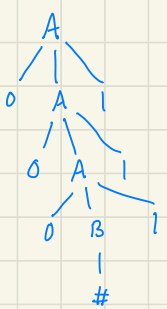
$A \rightarrow 0A1$
$A \rightarrow B$
$B \rightarrow \#$

S (start symbol): A
V: {A, B}
Σ: {0, 1, #}

- Derivation is getting a String from a CFG

1) $A \rightarrow 0A1$
2) $0A1 \rightarrow 00A11$
3) $00A11 \rightarrow 000A111$
4) $000B111$
5) $000\#111$

Parse tree for given derivation



- Root is Start node
  - leaf nodes are terminals
  - branches are rules

A CFG describes a language, which is a set of all possible valid Strings. meaning depending on when you end recursion, you can come up with more valid Strings.

Ex 2

$S \rightarrow (S)$
$S \rightarrow SS$
$S \rightarrow \epsilon$

starting: S
Variables: S
terminals: { ( ) }

(S) (SS) ( )

* remember epsilon means empty string

well balanced parenthesis

$S \rightarrow 0S1$
$S \rightarrow 1S0$
$S \rightarrow \epsilon$

Start: S
terminals: {0, 1}
Variables: {S}

0S1
0 1S0 1
" "

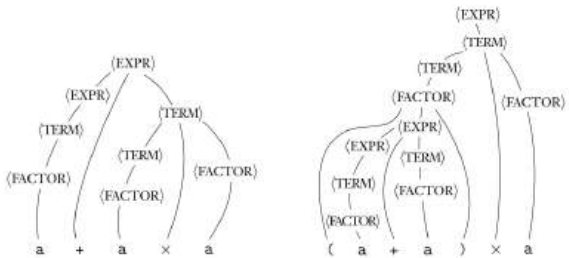
w has equal number of 0's, 1's

EXAMPLE 2.4

Consider grammar $G_4 = (V, \Sigma, R, \langle EXPR \rangle)$.
V is $\{\langle EXPR \rangle, \langle TERM \rangle, \langle FACTOR \rangle\}$ and Σ is $\{a, +, \times, (,)\}$. The rules are

$$\langle EXPR \rangle \rightarrow \langle EXPR \rangle + \langle TERM \rangle \mid \langle TERM \rangle$$
$$\langle TERM \rangle \rightarrow \langle TERM \rangle \times \langle FACTOR \rangle \mid \langle FACTOR \rangle$$
$$\langle FACTOR \rangle \rightarrow (\langle EXPR \rangle) \mid a$$

The two strings a+axa and (a+a)xa can be generated with grammar G_4. The parse trees are shown in the following figure.



terminal

variables

Example using words

- CFG, defines arithmetic expressions groups based on precedence

Note: If any language is regular, you can convert the DFA to CFG

Assign a variable to each state in the DFA and create CFG rules based on DFA transitions

Creating CFG's

- Language can be split up into easier parts

Ex $\{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$

$S \rightarrow 0 S_1 1 \mid \epsilon$

$S_2 \rightarrow 1 S_2 0 \mid \epsilon$

Grammar:

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow 0 S_1 1 \mid \epsilon$

$S_2 \rightarrow 1 S_2 0 \mid \epsilon$

Ambiguity

- A CFG is ambiguous if a string has two different parse trees
- "a + a x a" is ambiguous, can group addition or multiplication first
- Compilers need unambiguous grammar
- To fix ambiguity use new grammar that enforces precedence

Chomsky Normal Form

- More structured way of writing CFG
- Every CGF has a context free language as a CNF

Form: $A \rightarrow BC$ (B, C, A are variables)
$A \rightarrow a$ (a is terminal)

CFG to CNF Conversion

1) Remove ϵ-rules, if $A \to \epsilon$ exists, remove it and adjust grammar
2) Remove unit production, rules like $A \to B$, where B is var. Replace A directly with the right hand side of B's rules
3) Remove long productions, convert $A \to BCD$ to $A \to BX$, $X \to CD$
4) Remove mixed terminals, in long productions, $A \to Bc$ into $A \to Bx$, $x \to c$

Ex

$S \to AB \mid a$
$A \to BC \mid b$
$B \to AC \mid C$
$C \to AB \mid \epsilon$

1) $S \to AB \mid a$
$A \to BC \mid b$
$B \to A \mid AC \mid a$

2) $S \to AB \mid a$
$A \to BC \mid b$
$B \to BC \mid b \mid AC \mid a$

$S_0 \to S$
$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \epsilon$

3) $S \to AB \mid a$
$A \to BC \mid b$
$B \to Bx \mid b \mid a$
$x \to AC$

Pushdown Automata's (PDA's)

- is a finite state machine + a stack
- more powerfull than finite automata
- stack allows count to 0^n, 1^n, to match

PDA Components: 6 tuple

Q: Set of states
Σ: input alphabet
Γ: Stack alphabet
δ: Transition Function
q_0: Start State
F: set of accept states

Non Deterministic PDA's

- some languages can only be recognized by nondeterministic PDA's
- Deterministic PDA's $\neq$ Power of the NPDA's

Operations

- Push, Pop, read

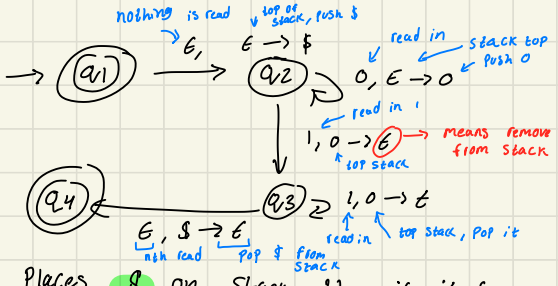
Acceptence Conditions

- Accepts String, if reaches an empty state
- The stack is empty at the end

Example of PDA's

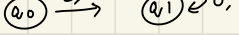
PDA : $\{0^n 1^n \mid n \geq 0\}$

- Can recognize when n is equal by having an empty stack

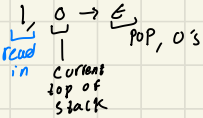


Places $ on Stack, then if it sees $ again it knows its empty

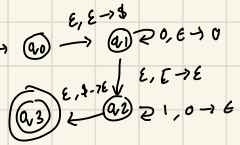
$L = \{0^n 1^n \mid n \geq 0\}$



Transition form:



- Way to think is one element must be pushed/ one must be popped
- Matching problem, stack starts empty



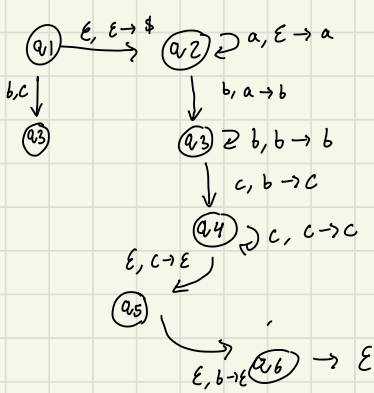
nondeterminism → at any given state, the PDA can have multiple possible moves for same input.

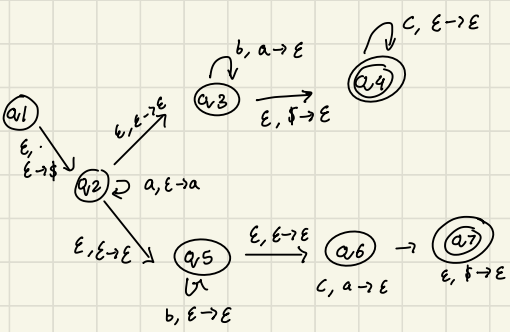
PDA guesses which transition to take

Deterministic can not recognize all Context-free language

Practise:

1) Given : $\{a^i b^j c^k \mid i,j,k \geq 0 \text{ and } i=j \text{ or } i=k\}$





Week 6-8 Lecture Notes

Church Turing Thesis

- Turing machines introduced by Alan Turing in 1936, more powerful model of computation
- Turing machine can compute anything that a real-world computer can compute.

What is a Turing Machine

- A turing machine (TM) is an abstract machine that reads, writes and moves on an infinite tape.

Key Parts

- infinite tape that stores input and intermediate data
- tape head that moves left (L) or right (R), reading and writing symbol.
- set of states including start, accept and reject states
- A transition function, which action to take based on current state and tape symbol.

Turing Machines vs. Finite Automata:

Feature	Finite Automaton (FA)	Turing Machine (TM)
Memory	Limited	Infinite tape
Head Movement	One-way read	Read & write, move left/right
Computational Power	Limited	More powerful
Halting	Always halts	May loop forever

Example Turing Machine

$\{w\#w \mid w \in \{0,1\}^*\}$

- This checks two halves of an input are identical seperated by #
- moves back and forth across tape to compare characters
- uses crossing out marks to track checked symbol

Formal Definition of Turing Machine

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

Where:

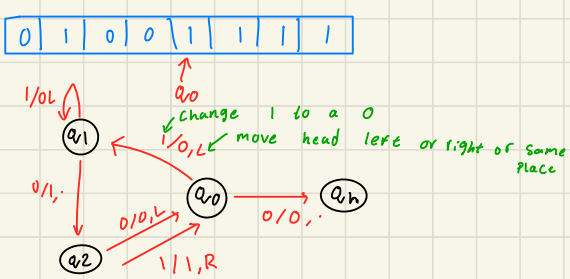
1. Q = Finite Set of States
2. Σ = Input alphabet (does not contain $\sqcup$)
3. Γ = Tape alphabet (includes $\sqcup$ and all input symbols)
4. δ (Transition Table): $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
5. q_0 = Start State
6. q_{accept} = accept State
7. q_{reject} = reject State

Configurations and Computation

- Configuration: Snapshot of a TM's State, tape and head position
- Example: $10\|q_7 0\|111$ means State q_7 is reading the second 0.
- Computation Steps

1) Initial Config: Starts at q_0 with input on tape
2) Transitions applied based on $\delta(q, a) = (r, b, L/R)$
3) Halting Conditions:
   - Accept State (q_{accept})
   - Reject State (q_{reject})
   - Loops forever: never reaches either State.

Ex

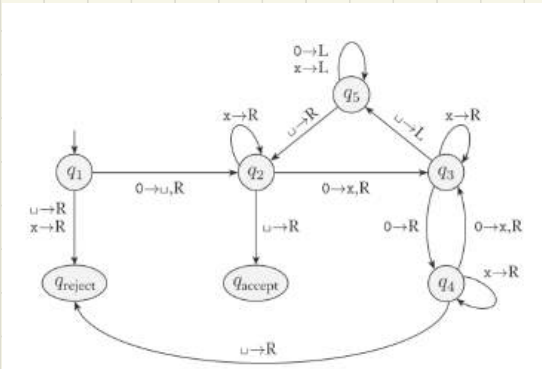


Turing Recognizable vs. Decidable Language

- Turing Recognizable Language: Turing machine can recognize it, may accept but may loop forever if it is not.
- Decidable Language: A turing machine always halts (accepts / rejects inputs)
- Decider is a TM that always halts and decides a language

EX Language: $A = \{0^{2^n} \mid n \geq 0\}$
(String with 0's whose length is a power of 2)

Algorithm:
1.) Cross off ever other 0 from left to right
2) If only one 0 remains, accept
3.) If number of 0's is odd, reject
4.) move head back to start and repeat



Variants of Turing Machines

- Turing machines can have different variations, all are computationally equivalent.

Multitape Turing Machines

- uses multiple tapes instead of one
- Each tape has it's own head for reading/writing
- Transition Function:

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$$

- Equivalent to single tape TM

Nondeterminsic TM's

- Can branch into multiple computations at once
- If any branch accepts, NTM accepts
- Equivalent to Deterministic TM, but can be exponentially slower

Enumerators

- TM that prints String in a language
- A language is Turing recognizable iff some enumerator prints its Strings.
- Basically: turing machine ⟶ printer attached
  - Starts with blank tape → prints strings one by one
  - If your string is printed its in the language

2 ways to prove: a language is Turing Recognizable if some enumerator enumerates it

1) Enemurator E, prints out strings: a, b, ab...
   ↳ every time E outputs something compare with your string w, if matching accept

2) If Turing machine M exists that recognizes language, we build enumerator.

How E works:
- List all possible strings in alphabet
- for step i, run M for i steps
- if M accepts String, print it

Trying all strings slowly

Key Terms:
Enumerator - generator of strings in language
Recognizer - checker if a String is in language

Equivalent for turing recognizable languages

Definition / History of Algorithms

- Algorithms are step-by-step instructions for solving a problem
- To Prove no algorithm exists, need formal definition
- Any intuitive idea of an algorithm → can be done by turing machine
- Recognizer: accepts correct strings, but may run forever on incorrect ones
- Decider: always give clear yes or no, in finite time

Describing Turing Machines

- Formal Description
- Implementation Level Description
- High-Level description

Note: Turing machines only take Strings as input
∴ if you want to add graphs, formulas, must first encode it to a string
Can encode backwards too

Week 6-8 Exercises

1) $A = \{0^{2^n} \mid n \geq 0\}$

accepts 0's if the length is Power of 2

Start with Single $0^{2^0} \rightarrow 2^0 = 1$

$\therefore$ Accept

Input = 00

First pass: ~~0~~ cross off first 0

then skip one → 1 zero remains

$2^1 \rightarrow$ accept

Input = 000, cross off 1st, skip 2nd, cross off third

↳ Left with one unmarked.

Original number of 000 was 3, 3 is odd and not 1

$\therefore$ Reject

Input 000 000

First pass: cross every other → left with 3 zeroes

↳ 3 is odd and more than 1, reject

Week 8-9 - Decidability

- Some problems can not be solved by any algorithm, these are unsolvable or undecideable
- Decideable languages are problems where there is an algorithm, that gives a correct yes/no in finite time.
- DFA accepting string is decideable, NFA's are decidable too but more work
- Undecideable language: may be only partially decidable, there exists no Turing Machine for that language

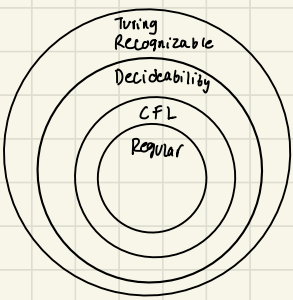
Theorem 4.2

- $A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts } w \}$

  is decidable

- Proof Idea: converts NFA B into equivalent DFA C
  - use TM M to simulate C on w
    - If M accepts, then accept, reject otherwise.

Note: DFA's, NFA's and regular expressions are equivalent in expressive power
Turing machines can convert between them



Theorem:

- Given CFG and string w, does G generate w.
- When there are infinitely possible derivations
  1) Convert to CNF,
  2) list all possible derivations using $2n-1$ steps

Undecideable

- means there is no turing machine, that gives us a yes or no
- Goal: in Software, we want to automatically verify if our program is correct.
  - ↳ Impossible: automatic reasoning is limited, real world softwares messy
- Given a TM M, and input w, does M accept w?
  - ↳ undecideable, can say yes if m accepts, but may also loop forever

Function Types

- One-to-One: Every input has unique output (Injective)
- Onto - Every output has at least one input mapped to it (Surjective)
- Bijective = both onto and one to one

- A Set S, is countable if either it is finite or has same size as natural numbers.

Uncountable Sets

- Set is uncountable if it is not possible to list all its elements in a seauence (no correspond with the natural numbers $\mathbb{N}$)
- all finite and countable infinite sets ($\mathbb{N}, \mathbb{Z}, \mathbb{Q}$) are countable
  - ↳ uncountable sets include: set of all Real numbers, Power set of $\mathbb{N}$, set of all languages

Correspondance - need to prove all members of $\mathbb{N}$ is paired with all members of $\mathbb{R}$ (bijection)

Theorem: Some languages are not Turing Recognizable

- there are uncountably many languages yet only countably many Turing machines
- There are more languages than Turing machines
  - ↳ Every Turing machine can be represented as a string ⇒ set of all TM's is countable
  - ↳ Set of all languages is uncountable

Proof: $A_{TM} = \{\langle w, M\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

- Assume decider H, which solves A_{TM}
- Build Machine D that uses H, but does opposite.
  - ↳ Run H on $\langle M, M\rangle$
    - If H says M accepts → M → D rejects
    - If H says M rejects → M → D accepts
- If D accepts / Rejects → Contradiction. ∴ no decider H exists, ∴ A_{TM} is undecideable

A Turing-Unrecognizable Language

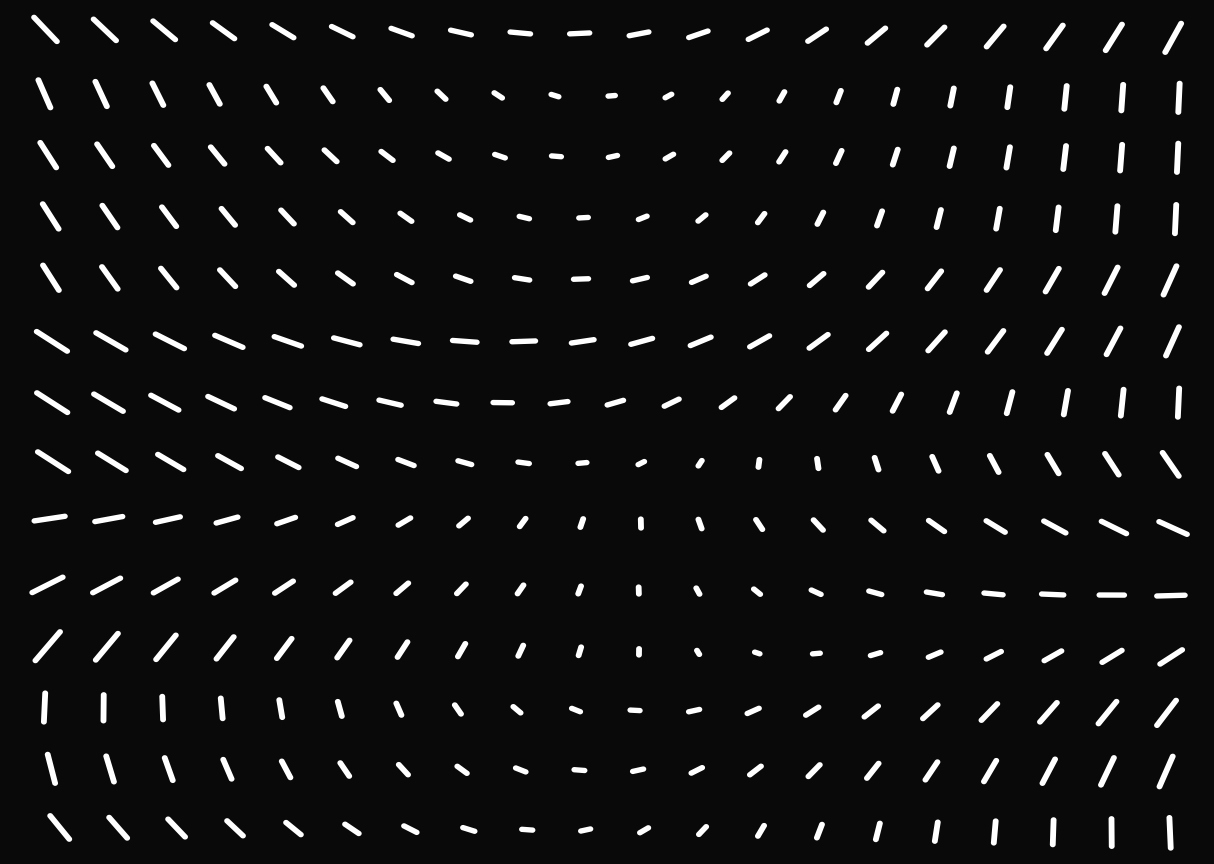
- A_{TM} is undecideable, but it is Turing recognizable. TM can accept input, may run forever tho.
  - ↳ Some languages are not even Turing recognizable. (No TM can recognize ownership)

Turing Recognizable

- Language A is turing recognizable if a TM exists S.t accepts all strings in A, may loop forever on strings not in A
- Language is Decideable ⟺ it is turing and co turing recognizable

- Languages could be Turing Recognizable - but not decideable

Midterm



Review For Midterm

- given a set, do set operations, power set of A, B
- less proofs
- given function definition, finding domain/range
- Basic definition of graphs / trees.
  defined by vertices and edges. Draw graph
- given a tree, finding leaves.

10-15%

- Formal Language, NFA/DFA
- Given a DFA definition, draw the DFA and vice versa
- Given a NFA definition, draw the NFA and vice versa
- Conversion NFA to DFA, wont be too many states
- Given regular expression, convert to NFA (can explain in words)
- Given a language, creating NFA/DFA

40-50%

Context-free language

- Given language, identify Start, set of variables, find terminals
- Give 3 strings member of the given language, or rejected examples
- Parse String / Derivation at terminal
- Can you design a push down automata for given

30-35%

223 Midterm Practise

$S \to aSb \mid A$ aSb

$A \to \epsilon \mid AA$ aaSbb

aaAbb

1.) S: {S}
2.) V: {S, A}
3.) Σ: {a, b}
4.) aabb, AA, ϵ
5.) BBB

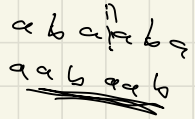
2) aSb
aABb
aaAbb
aaabb is a valid string

3) q: 0, 1, 2
Σ: 0, 1

0
0
0

When a one is seen, the 0 gets popped.
Stack alphabet is what you see in the stack and what signals its empty.

PDA = Finite state machine + Stack



Basic Operations, Definitions Stuff

A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

2) b, c
3) a, d
4) b, e

$A \cup B$ = {1, 2, 3, 4, 5, 6}
$A \cap B$ = {3, 4}

$\cup$ - means all elements from both
$\cap$ - means common elements
$A - B$ = {1, 2} elements in A not B
A' {5, 6, 7, 8}

The power set is set of all subsets of A.

P(A) : A: {1, 2}
P(A) = {{1}, {2}, {1, 2}, $\emptyset$}

If A has n elements, theres 2^n possible subsets of those elements.

Domain and Range Stuff

f: A → B maps, A to B

x^2, D: all real numbers
R: only $[0, \infty]$

$f: \sqrt{x}$, Domain: $\{x \in R \mid x \geq 0\}$
Range: $y \in R$

$g(x) = \frac{1}{x}$, Domain: $\{x \in R \mid x \neq 0\}$
Range: $y \in R$

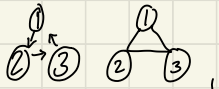
$h(x) = |x| - 2$ Domain: $x \in R$
Range: $y \geq -2$,

Note: Cross product of two sets are all combinations of two

Language: Set of strings over an alphabet.

Key Terms:

- Degree of vertex, number of edges connected to it
- Undirected graph, Edges have no direction
- Directed, Arrows have directions



$(A \wedge B) \vee (\neg A \wedge B)$
a and b or not a and b
$B \wedge (A$ or not $a)$
so B

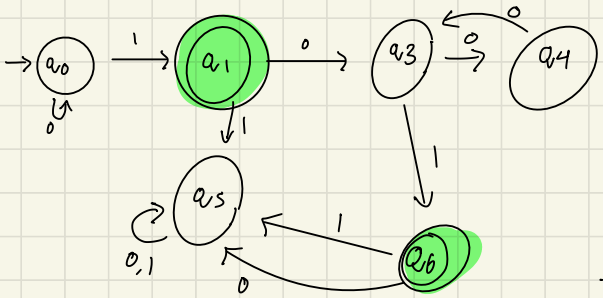
Truth table:
XOR:

A	B	A xor B
0	0	0
1	0	1
1	1	0
0	1	1

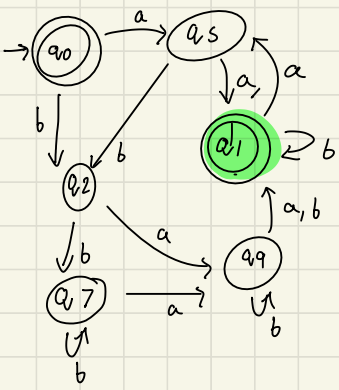
223 - Midterm Practise - Week 2/3 Review

- Automata of changing states, based on inputs
- DFA, has one transition per symbol, no ε transitions
- NFA, can have multiple transitions for same input.
- Finite automata 5 tuple $(Q, \Sigma, \delta, q_0, F)$
- Machine can accept many strings, but recognize only 1 language

Ex from AI
Draw M that recognizes: starts w 1, has odd amt of 0, then ends in 1



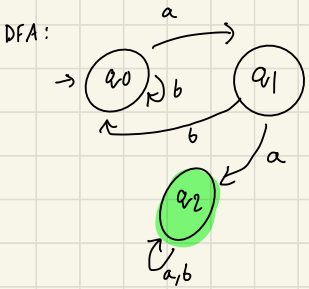
DFA diagram for even number of a's atleast 2



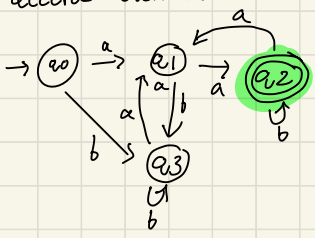
Given the following DFA:

$Q = \{q0, q1, q2\}$
$\Sigma = \{a, b\}$
$q_0 = q0$
$F = q2$
$\delta = \delta$

DFA:



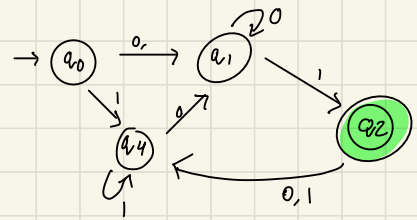
accepts even num of a's



b's are independent.

NFA Diagram For $L = \{w \in \{0,1\}^* \mid w \text{ ends in } 01\}$

Accepted strings: 01, 0001, 1101, 1001



If ε is allowed, starting state must be accepting

More midterm practise - week 2/3 stuff

Drawing NFA/DFA patterns

- Even/odd - have 2 states, flip a between even/odd, loop over irrelevant ones
- Ends in " ", tracking states, final states remember characters
- Same amt of x, y. PDA is required

Regex practise

1) $(aa)^* bcd \ldots z$
2) $((0(011)^*0 \mid (1(011)^*1))$
3) $(101)^*$
4)
5) $(01)^* (10)^*$

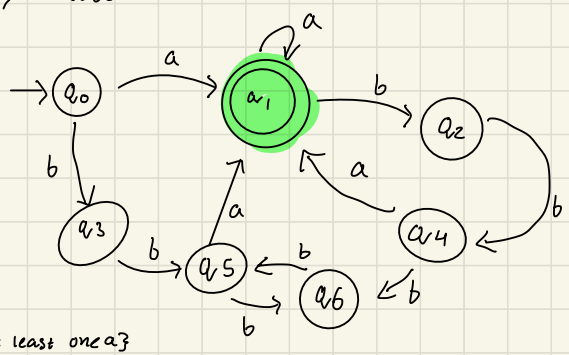
Convert the regular expression to DFA's

$\Sigma = \{a, b, c\}$ $M = \{Q, \Sigma, \delta, q_0, F\}$

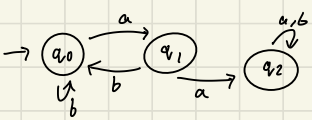
$a^*(bb)^*a$

Key Idea: any number of a's. any number of two b's and ends in a.

bba, abbbba aaabba

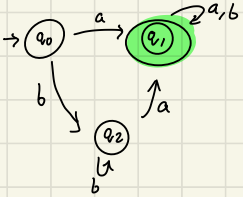


Draw the DFA, given def



Language of $(aa)(a^*b^*)$
starts with 2a's.

Draw DFA for $L = \{w \in \{a,b\}^* \mid w$ contains at least one a$\}$



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

$Q: \{q_0, q_1, q_2\}$

δ:

	a	b
q_0	q_1	q_2
q_1	q_1	q_1
q_2	q_1	q_1

$F: q_1$

$q_0: q_0$

$(01 \mid 10)^*$

Can accept strings with any amount of 01, or 10. or if 01 and 10 are grouped together.

$Q: \{q_0, q_1, q_2, q_3\}$

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

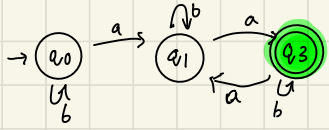
$q_0: \{q_0\}$

$F: \{q_1\}$

δ:

	a	b
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_1	q_2
q_3	q_1	q_3

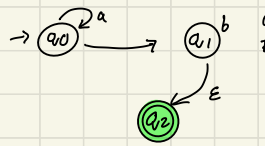
$L = \{w \in \{a,b\}^* \mid w$ has an even number of a's$\}$



Probably not most efficient, but works

More midterm practise

2)

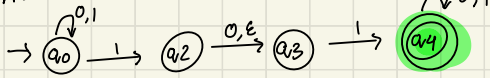


Q: {q0, q1, q2}
Σ: {a, b}
δ:

	a	b	ε
q0	q0	q1	∅
q1	∅	∅	q2
q2	∅	∅	∅

q0: q0
F: q2

NFA:



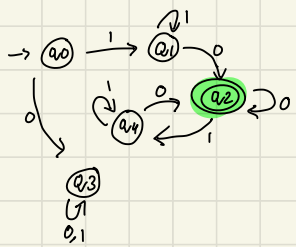
δ =

	0	1	ε
q0	q0	{q0, q2}	∅
q2	q3	∅	q3
q3	∅	q4	∅
q4	q4	q4	∅

4) (01|10)* - To construct NFA:

any number of 01 or 10's or a combination of them together ie 0110.

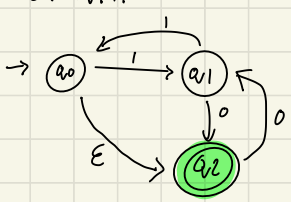
6) L = {w ∈ {0,1}* | w starts with 1, ends with 0}



Converting from NFA to DFA

* Epsilon Closure: find all states reachable only via ε-transitions

Given NFA:



3 states means $2^3 = 8$, DFA states

① ε closure:

q0 ⟼ {q0, q2}
q1 ⟼ {q1}
q2 ⟼ {q2}

② Potential DFA states: {q0}, {q1}, {q2}, {q0, q1}, {q1, q2}, {q0, q2}, {q0, q1, q2}, ∅

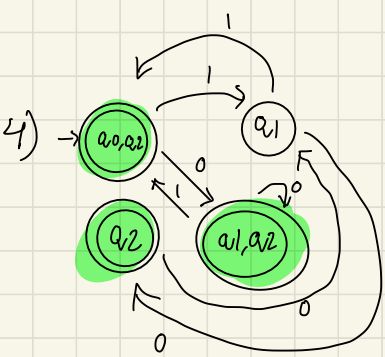
DFA states

D0 →	{q0, q2}
D1	{q1}
D2	{q2}
D3	{q1, q2}

3) DFA Transition Table

	0	1
{q0, q2}	{q1, q2}	{q1}
{q1}	{q2}	{q0, q2}
{q2}	{q1}	∅
{q1, q2}	{q1, q2}	{q0, q2}

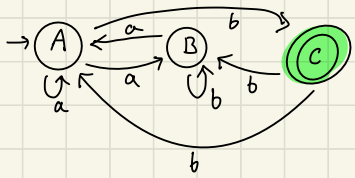
4)



If q2 is an accepting state in NFA, then {q0, q2}, {q2}, {q1, q2} are accepted here.

More Converting Examples

1) NFA:



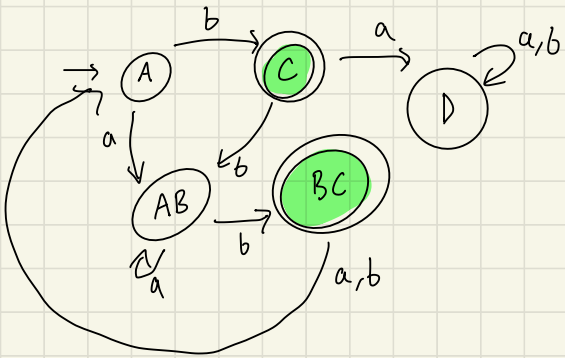
To DFA, no epsilon. C is accepted ones

$2^3 = 8$, possible outcomes, accepted states are {C}, {AC}, {BC}, {A,B,C}

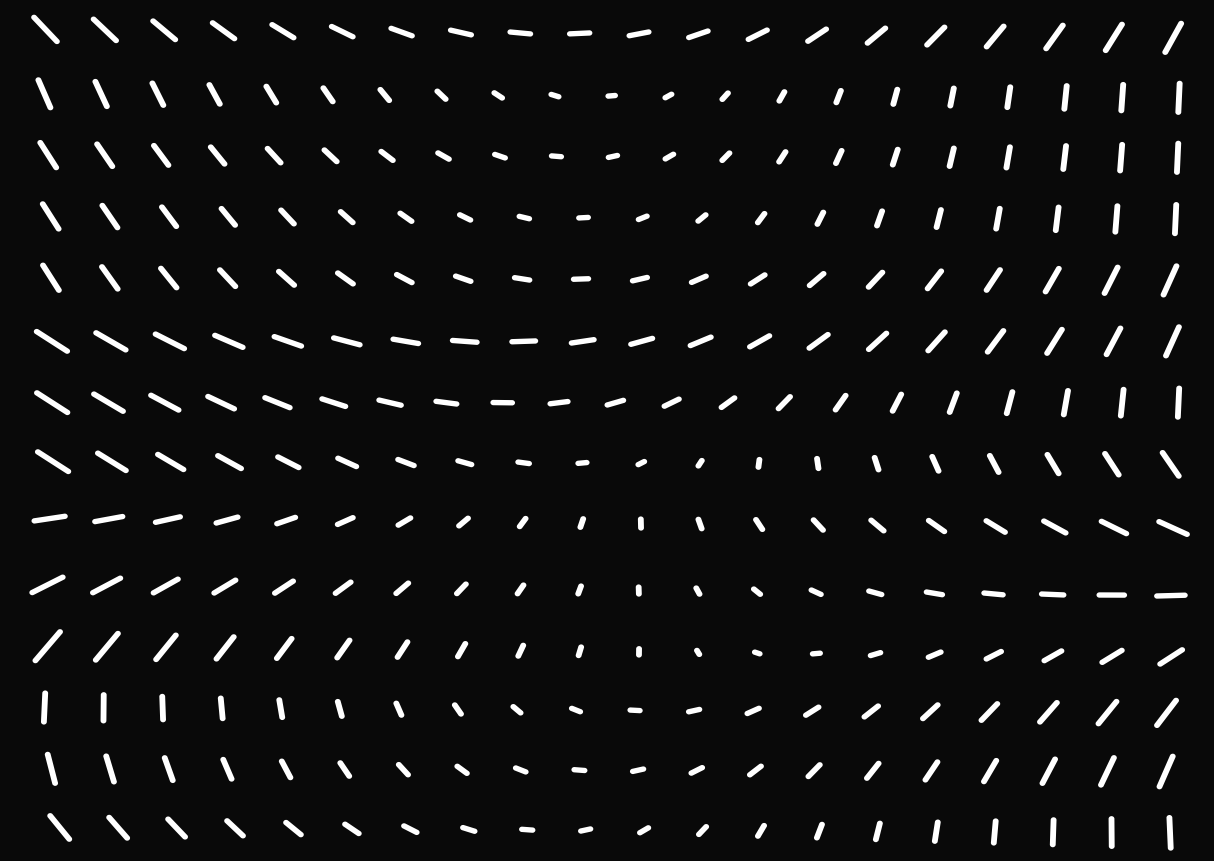
	a	b
~~{∅}~~	∅	∅
{A}	{AB}	{C}
~~{B}~~	{A}	{B}
{C}	{D}	{AB}
{AB}	{AB}	{B,C}
{A,C}	{B}	{A,C}
{B,C}	{A}	{A}
~~{A,B,C}~~	{A}	{A,BC}
{D}	{D}	{D}

After eliminating states that only have pointers, that point to himself,

we left w: {A}, {C}, {AB}, {B,C}, {D}



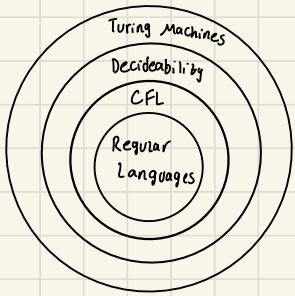
Final Exam Prep



Final Exam Notes

Format of Exam

- will need all 3 hours
- 7/8 questions
- 40% of Final grade
- will be harder than midterm



Regular Language

15% { - Regular language (NFA, DFA, converting between them)

15% {
- CFG, design PDA, → might need state transition diagram
- If regular language, give expression otherwise give the CFG

Turing Machines and Decideability

40% {
- Turing machine, one design question, implementation level description
- Decideability
- closed under star, union, concat operations
- know the diagram
- Simple question: given language, decide if its dicidcable or not
- Correspondence, sets, countable, simple function, show if its countable or not
  ↳ find map to natural numbers

30% {

Algorithm Specification

- write simple function, design pre/post, assertion (8-9 lines of code)
- will contain loops, looping variant/invariant assertions, if else assertions

Exam Prep - Regular Languages Stuff (15%)

Functions/Relations

- A set is a collection of objects, $S = \{7, 21, 57\}$
- $\subseteq$ - subset
- $\cup$ - union
- $\cap$ - interselt
- A' - complement
- cross product: $A = \{1, 2\}$, $B = \{x, y\}$

  $A \times B = \{(1,x), (1,y), (2,x), (2,y)\}$

Languages: set of strings over an alphabet.

Language is regular if a finite automata recognizes it

3 types of Automata

1.) Finite Automata (FA) - Models fixed number of states
- Used in regular expressions/pattern matching
- Can't remember past information

2.) Pushdown Automata (PDA) - Finite Automata with a Stack memory structure
- More powerfull than finite automata
- Used for parsing, context free languages ($a^n b^n c^n$)

3.) Turing Machines - most powerfull model
- Used to define what can be computed
- Recrsively enumerable languages, halting and such
- has infinite tape for reading/writing can move left/right

Regular Operations

- Union: $A \cup B$, if L_1 and L_2 are regular, so is their union
- Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
- Star $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$ (if L_1 is regular, L^* is regular)

Exam Prep - Regular Languages Stuff (15%)

- DFA, Exactly one transition per input
- NFA, Can have multiple transitions, and ε moves
- Every determinism finite automata is an NFA
- For NFA, when you have a transition with multiple choices, it creates multiple branches and continous to check procceeding inputs

Epsilon (ε) Transitions: Machine can move between states without reading next character.

5 Tuple: $\{Q, \Sigma, \delta, q_0, F\}$

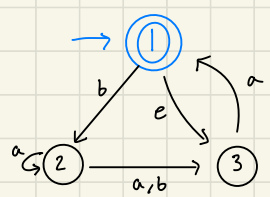
Equivalence

- 2 machines are equivalent, if they recognize the same language
- DFA's and NFA's recognize same class of languages (regular)
- Every NFA has an equivalent DFA

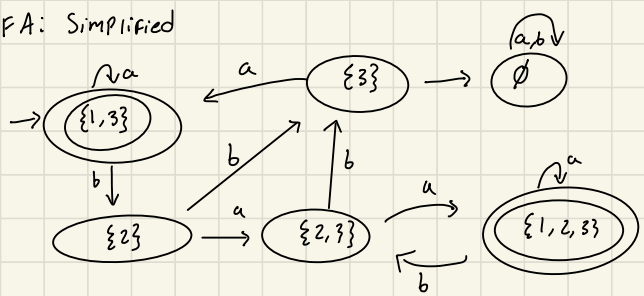
NFA ⟶ DFA

Example

NFA:



DFA: Simplified



Regular Expressions

- way to describe a set of strings using pattern matching rules
- regular expressions define regular languages

Ex $(0 \cup 1)0^*$: Set of all expressions that start with 0/1, followed by any number of 0's.

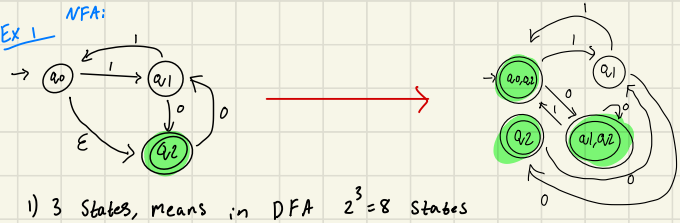
Regular Operations

- Union ($\cup$) represents choice (OR)
- Concatenation ($\circ$, impicit) - joins strings seaven tially
- Kleenes Star (*) - represent repetition

Converting NFA to DFA

EX 1

NFA:



1) 3 States, means in DFA $2^3 = 8$ States

E Closure: the set of states you can reach from a by taking 0 or more ϵ-transitions

$q_0 \rightarrow \{q_0, q_2\}$
$q_1 \rightarrow \{q_1\}$
$q_2 \rightarrow \{q_2\}$

2) List DFA States:

D0 → {q0, q2}
D1 → {q1}
D2 → {q2}
D3 → {q1, q2}

If q2 is an accepting state in NFA, then {q0, q2}, {q2}, {q1, q2} are accepted here.

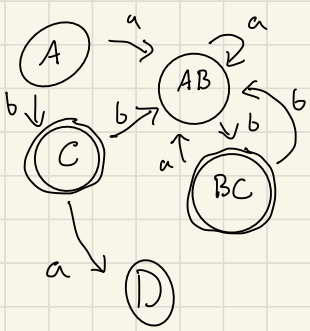
3) DFA Transition Table

	0	1
{q0,q2}	{q1,q2}	{q1}
{q1}	{q2}	{q0,q2}
{q2}	{q1}	∅
{q1,q2}	{q1,q2}	{q0,q2}

---

1) epsilon transitions if there are one
2) look at 2^n possible states, find any states, with atleast one accepted state
3) make transition diagram for DFA, eliminate any unreachable states.

	a	b
∅	∅	∅
A	AB	C
~~B~~	A	B
C	D	AB
AB	AB	BC
~~AC~~	A	BC
BC	A	BA
~~ABC~~	AB	ABC
D	D	D



Keeping States {A} {C} {AB} {BC}

More Converting Examples

1) NFA:

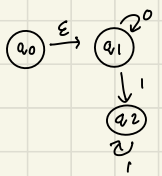
Q: { q_0, q_1, q_2 }

Σ: {0, 1}

δ:

q_0: {q_0}

F: {q_2}



3 States: $2^3 = 8$ Possible States

1) epsilon transitions

q_0: {q_0, q_1} → must be in

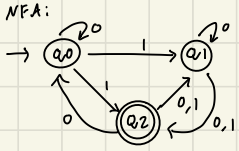
q_1: q_1

q_2: q_2

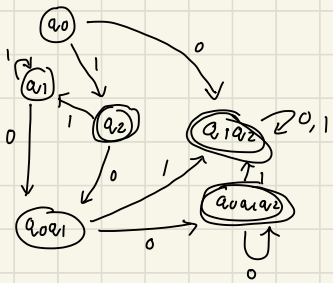
2)

	0	1	ε
∅	∅	∅	∅
q_0	?	?	q_1
q_1	q_1	?	∅
q_2	?	q_2	∅
q_0q_1	q_1	q_2	∅
q_0q_2	q_1	q_2	∅
q_1q_2	q_1	q_2	∅
$q_0q_1q_2$	q_1	q_2	q_1

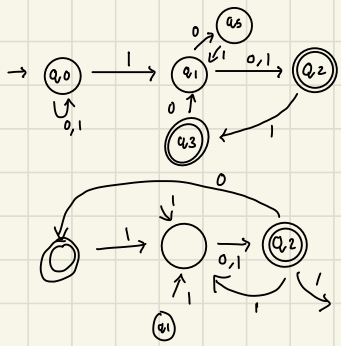
2) NFA:



	0	1
∅	∅	∅
q_0	q_1q_2	q_2
q_1	q_0q_1	q_1
q_2	q_0q_1	q_1
q_0q_1	$q_0q_1q_2$	q_1q_2
q_0q_2	q_0	q_1q_2
q_1q_2	q_1q_2	q_2q_1
$q_0q_1q_2$	$q_0q_1q_2$	q_1q_2



3)



Exam Prep - Context Free Grammar (15%)

Context - Free Language / Grammar

- way to describe more complex languages, (may use recursion)
- Context free grammars are a way to describe a context free language

Definition of a CFG

- A CFG is a 4 tuple (V, Σ, R, S)
  - V is a finite set of variables (non-terminals)
  - Σ Finite set of terminals (input symbols)
  - R finite set of rules (productions) in form $A \to w$
    - A is a variable
    - w is a string of terminals
  - S is a start symbol which is variable in V.

$A \to 0A1$
$A \to B$
$B \to \#$

S (start symbol): A
V: {A, B}
Σ: {0, 1, #}

- terminals: actual character/tokens of language
- non terminals: symbols that can be replaced using production rules
- start symbol: usually S
- Production Rules: different choice of expressions

Example

- For language: $\{a^n b^n \mid n \geq 0\}$, describe CFG
- CFG: $S \to aSb \mid \varepsilon$
- Idea: For every a, you add a b at the end → perfectly matched

2) $\{a^n b^n \mid n \geq 0\}$

any number of a's, followed by any number of b's

$S \to aS \mid B$
$B \to bB \mid \varepsilon$

3) $\{w \in \{a,b\}^* \mid w \text{ is palindrome}\}$

$S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$

4) $\{w \mid w$ has more a's than b's$\}$

- not context free, because CFL's cant compare direct counts unless strictly paired

5) $\{a^n b^m c^n \mid n \geq 0, m \geq 0\}$

$S \to aSc \mid B$
$B \to bB \mid \varepsilon$

Tips

- Design CFG for all strings over $\{a,b\}$
  $S \to aS \mid bS \mid \varepsilon$
- Palindrome: $S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$
- Exactly two 1's: $S \to A1A1A$
  $A \to 0A \mid \varepsilon$

Exam Prep - Context Free Grammar (15%)

Pushdown Automata Stuff

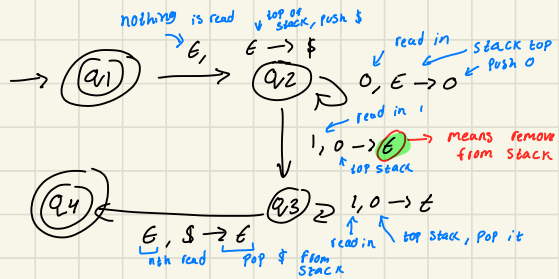
- finite state machine with a stack
- stack-like memory structure
- allows $0^n, 1^n$ to match elements
- For context free languages

Acceptance Conditions

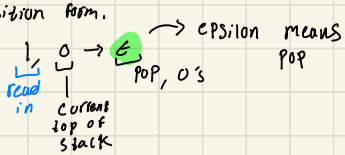
- Accepts string, if reaches an empty state
- The stack is empty at the end

Example

PDA: $\{0^n 1^n \mid n \geq 0\}$



Transition form.



PDA Components: 6 tuple

- Q: Set of states
- Σ: input alphabet
- Γ: Stack alphabet
- δ: Transition Function
- q_0: Start State
- F: set of accept states

Key Ideas:

- Start by pushing $ to stack to know, if it is empty.
- Can recognize when two values n are the same when the stack is empty

nondeterminism → at any given state, the PDA can have multiple possible moves for same input.

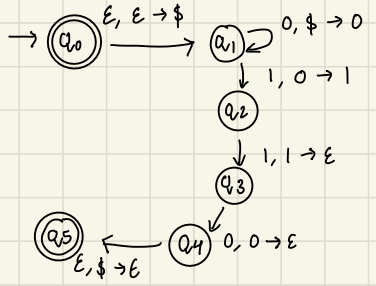
PDA guesses which transition to take

Deterministic can not recognize all context-free language

Rough work / PDA questions

PDA for $\{0^n 1^m 0^n \mid n \geq 1, m \geq 0\}$

Same number of 0's to start and end it, with any number of 1's in the middle



input		0			1			ε	
Stack	0	1	ε	0	1	ε	0	1	ε
q_0	∅	∅	{q1, $}	∅	∅	∅	∅	∅	∅
q_1	{q2, 0}	{q2, 1}	∅	∅	∅	∅	∅	∅	∅
q_2									
q_3									

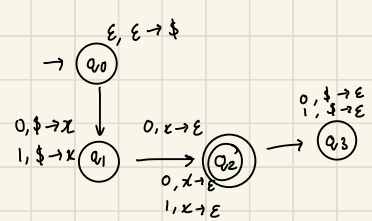
$M = \{Q, \Sigma, \Gamma, q_0, F, \delta\}$ → transition
- Q: all states
- Σ: language
- Γ: stack alphabet
- q_0: start
- F: accept

Problem: $\{w \in \{a,b\} \mid w \text{ is palindrome}\}$

- Push $ to stack
- Read a/b and push to stack
- Guess middle of string PDA non deterministically, read ε
- Pop the stack matching the pushed stuff
- accept if $ is read again

Assignment 2 - q1

- Language is where second half has at-least one 0.
- Idea: String is divided into 2 parts, second part should have a 0
  - ↳ must track halfway point
- Push x for every input in first half
- When 0 read, pop till end of string
- reject if $ is seen or before input ends
  - └ If theres still x in Stack, we know its second half



For Transition Table

- For each combination of input symbol and stack top, write the state you go to and what is added/removed from the stack.

CFG: For seeing a 0 in second half of word

$x: \{ \varepsilon, V, R, S \}$

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

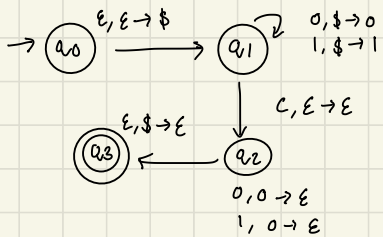
$V = \{A, B\}$

$R =$ $A \rightarrow 1A1 \mid 1B0 \mid 0A1 \mid 0B0$

$B \rightarrow 1B1 \mid 1B0 \mid 0B1 \mid 0B0 \mid 011 \mid \varepsilon$

B ensures at least one 0 was seen in second half

A2 Q2 Find w then split using C then reverse w



Exam Prep - Redoing Midterm

1) i) No, A is not a subset of B, not all of A is in B.

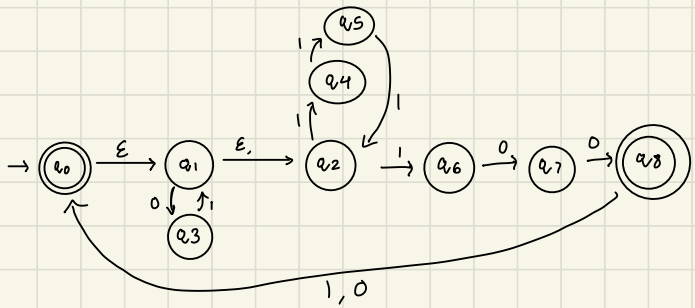
ii) B, is not a subset of A

iii) $A \cap B$: $\{a, b\}$

iv) $A \times B$: $\{aa, bb, c1, ab, ca, cb, a1, b1, ba\}$ → Cartesian product should have 3^2 elements in it

v) Power set of B: $\{a, b, 1, ab, a1, b1, ab1, \emptyset\}$ → power set should include $2^3 = 8$ elements

2) give NFA recognizing $((01)^*(111)^*100)^*$



Language is broken into 3 parts: $(01)^*$, $(111)^*$, 100

Formal 5 Tuple: $M = \{Q, \Sigma, \delta, q_0, F\}$

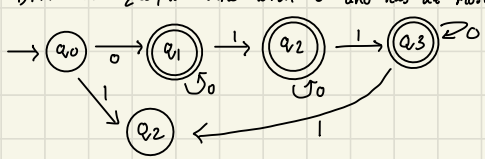
Q: $\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$

Σ: $\{0, 1\}$ F: $\{q_0, q_8\}$

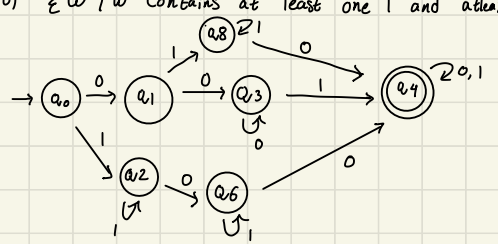
δ: Transition function

	0	1	ε
q_1			
q_2			
⋮			

3) a) DFA for $\{w \mid w$ starts with 0 and has at most two 1's$\}$



b) $\{w \mid w$ contains at least one 1 and atleast two 0's$\}$



F: $M\{Q, \Sigma, \delta, q_0, F\}$

Q: $\{q_1 - q_7\}$

Σ: $\{0, 1\}$

δ:

q_0: q_0

F: q_4

Q5) Regular expressions

i) at least two 0's at most one 1

All options: 000^*, 00^*100^*, 1000^*, 000^*1

then union all options $000^* \cup 00^*100^* \cup 1000^* \cup 000^*1$

ii) Strings with 011 as prefix

$011(0 \cup 1)^*$

iii) all strings with 101 as substring

$(0 \cup 1)^*(101)(0 \cup 1)^*$

6) Variables: $\{S\}$
terminals: $\{0, 1, \varepsilon\}$
Start variable: $\{S\}$

iv): 110, 101, 011, ε

v). 1111, 0000, 0, 1, 0110

vi): set of all strings over $\{0, 1\}$ where number of 1's is exactly twice number of 0's.

Example, 4.1 Exercise

a) $\langle M, 0100 \rangle \in A_{DFA}$ $0^*(11)^*0$

Accepted

b) $\langle M, 011 \rangle \in A_{DFA}$

Rejected

c) $\langle M \rangle \in A_{DFA}$

? Is there some string M accepts?

d) $\langle M, 0100 \rangle \in A_{Rex}$

- All DFA's recognize regular languages
- Every DFA language can be written as expression

e) Is $\langle M \rangle \in E_{DFA}$

- Is this language empty, no

f) $\langle M, M \rangle \in EQ_{DFA}$

- Does the model accept same strings, yes

Exam Prep - Turing Machines / Decideability (40%)

Turing Machines Overview

- Powerful model, abstract machine, reads, writes and moves on an infinite tape

Key Parts

- infinite tape that stores input and intermediate data
- tape head that moves left (L) or right (R), reading and writing symbol.
- Set of states including start, accept and reject states
- A transition function, which action to take based on current state and tape symbol
- May loop forever, follow mechanical procedures

Formal Definition of Turing Machine

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

Transitions

$a \to Y, R$

Reads a, change it to y, move right

$Y, \sqcup \to R$

Process more

Example 1

$L = \{ w \in \{0,1\}^* \mid w$ has equal number of 0's and 1's$\}$

Idea: match up 0's with 1's by marking them off one by one.

1) Find unmarked 0, mark it with X, scan right till 1 is found mark it
2) Return to start of tape, all symbols should be marked or matched

- As long as it is computable, there exists a turing machine that always halts with yes or no for any inputs. (Decideable)

Example 2

$w \in \{0,1\}^*$, such that w has twice as many 0's as 1's

Idea: Start at head, scan right, Find a 1, then mark it then find 2 zeroes to mark off, then find the next 1.

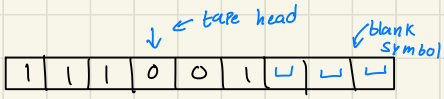
- Replace 1 with Y and every 0 with X.

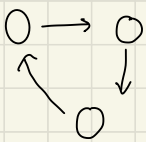
- Control Portion of your Turing machine is deterministic
- Nondeterministic TM's are equivalent to deterministic and give no - advantage.. In fact, makes it slower.

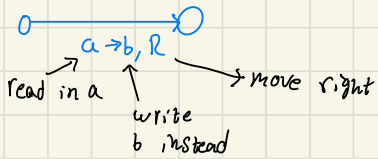
Exam Prep - Turing Machines / Decideability (40%)

Visual into Turing Machines

control portion is still deterministic







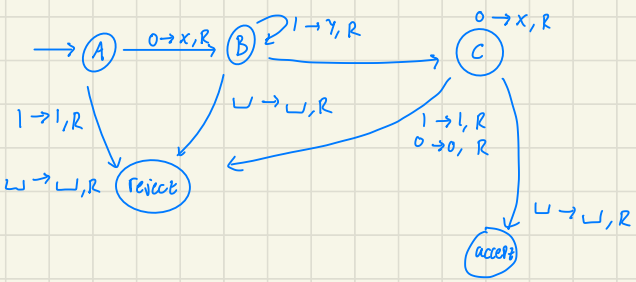
AT Each Step:

- → Read current symbol
- → Replace and write
- → move one cell Left/Right

In a turing machine, computation can:

1) Halt and Accept
2) Halt and Reject
3) Loop

Example 1) $L = 01^*0$



Idea:

- In State A, must start with 0, if a 1 is read we reject it.
- Then we read a 1, make it Y, go on to C then move right, if we see blank accept
- we send to reject state if blank is read prematurely, starts with 1, or ends in 1.

A2 a4)

$L = \{ w x w^R \mid w \in \{a,b\}^*, x \in \{a,b\} \}$

- Palindromic Structure
- x marks middle Character

$M = \{Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\}$

Idea: we start at q_0 and mark a/b with x untill we find an x. This is the halfway point. When we see this x, Switch modes into checking symmetry

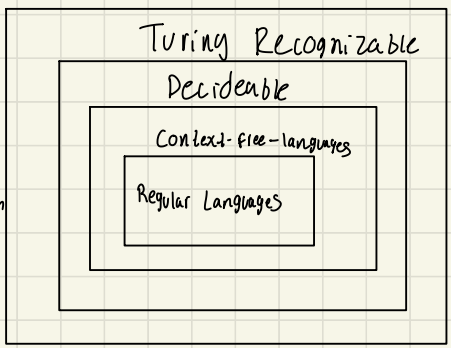
Then, we Keep Scanning to find last unmarked a/b.

Then, we scan left from x to compare it with the elements on the left. If at any point theres mismatch we reject.

Exam Prep - Turing Machine / Decideability (40%)

Turing Machines and Decideability

- **Decideable**: TM always halts and gives yes/no in finite time
- **Turing Recognizable**: TM accepts all strings in the language, may loop forever if input is not in the language
- **Co-Turing-Recognizable**: The complement is Turing recognizable
- **Undecideable**: NO TM always Halts and gives correct answer
- **Unrecognizable**: NO TM can even recognize it



Decideable Languages Closure Properties

- Decideable languages are closed under: Union, Concat, Star, intersection, Complement
- Turing Recognizable are closed under: union, concat, star
  - Turing complement is not always recognizable

How to Prove language is Decideable

- Must describe Turing Machine that halts on all inputs
  - ↳ Must show simple algorithm (counting/matching/simulating)

Example: Given (M, w), simulate M on w for all steps. If M halts, accepts, rejects otherwise. This language can be turing recognizable, but not decideable if it loops forever

To Prove a language is Turing-Recognizable not Decideable

- Show that a TM accepts when input is in language but may loop forever otherwise.
- Show halting problem is standard
- all decideable languages are recognizable.

Closure Properties

- Applying that operation to languages in the class produce a new language, also in that class

Ex/ Given L_1 is decideable, L_2 is turing recognizable, is $L_1 \cup L_2$ Turing recognizable.

Yes, both are closed under union, decideable & Turing recognizable

all decideable languages are Turing recognizable

Exam Prep - Turing Machines / Decideability (40%)

Counting Question

- A set is countable if elements can be put in one-to-one correspondence with the natural numbers $\mathbb{N}$.

To prove a set is countable:

1) Build correspondance a function mapping
   ↳ Design rule that assigns each element a unique number

2) List the elements in some clear order
   ↳ This is a countable list, naturally maps to $\mathbb{N}$.

Example 1) Show the set of all finite binary strings is countable?

$S = \{"", "0", "1", "01" \ldots\}$

Because we can order these by length (lexicographic order) this gives us a listable order, mapping, ∴ Countable

EX 2) Set of all TM's

Each turing machine can be encoded into a finite string over $\{0,1\}$

- The set of all TM's $\subseteq$ set of finite binary strings

∴ Countable

1) Set of all even natural numbers ✓ Countable (order direct mapping $f(n)=2n$)
2) Set of all Python Programs ✓ Countable (can be encoded as binary, then countable)
3) Set of real numbers between 0-1 ✗ Not Countable (real numbers, will miss some)
4) Set of all infinite binary strings ✗ not Countable (infinite means infinite positions, many more than $\mathbb{N}$)
5) Set of rational numbers $\mathbb{Q}$, ✓ Countable (Superset of integers, can be listed)

Cantors Diagonal Argument

- Diagonalizes list to find number not in list
- Builds counter example
- After listing all subsets, there will always be one not in list?

Exam Prep - Specifying Algorithms (30%)

What is Specifying an Algorithm

- logic based annotation
- Pre condition → Should be true before the function starts
- Post Condition → Should be true after it finishes
- loop invariant → Always true during loop
- Assertion → Boolean condition, checked at specific program
- loop invariant proves termination

* Loop invariant is pre condition and Post condition combined and related to n.

* Loop invariant should be in the loop

Declarative Interface

- all Static properties, Variables, properties of list.
  - ↳ Const → wont change (# of entries, target)

Code Example

```
bool present = false;
for (int i = 0; i<n; i++){
    if (A[i] == x){
        present = true;
        break;
    }
}
```

Pre Condition

$0 \leq n \leq max$

loop invariant

we haven't found x yet

loop variant

Proving it will terminate

Example 2

```
int search (int[] A, int n, int x) {
    for (int i = 0; i<n; i++) {
        if (A[i] == x) return i
    }
    return -1;
}
```

Pre Condition: $0 \leq n \leq max$

A is an array of length $\geq n$

Post Conditions:

Return = -1 for all indexes 0 to n-1, that is not x

Return = i such that i is found in array

Loop invariant:

Havent found x yet.

Sample Declarative interface

```
const int max;          /* maximum number of entries */
typename Entry;         /* type of entries, use == for equality */
const int n;            /* number of entries */
const Entry x;          /* search target */
Entry A[max];           /* A[0:n-1] are the entries to search */
bool present;           /* search result */
```

- will look Similiar for Search functions

∀ - For all

∃ - there exists

- Assertions used to provide useful documentation
- Entry is placeholder for some datatype

Complete Specification

- Declarative Intaface
- P, Q Pre/Post Condition

Rough Notes For Specifying Algorithms

Chapter 1

- Specifications are like designing specific instructions for algorithms
- Entry is placeholder type, == to compute
- Not allowed to use ands between quantifiers

a) all elements in array equals x
$\forall (i = 0; i < n)\ \ A[i] == x$

b) $\forall (i=0; i<n)\ \ \forall (j=0; j<n)\ \ A[i] == A[j]$

c) $\forall (i=0; i<n)\ \ \ A[i-1] <= A[i]$

d) $\exists (i=0; i<n)\ \ \ A[i] == x$ and
$\forall (j=0; j<n)\ (i \neq i \Rightarrow A[j] \neq x)$

1.12) x in A [a : b - 1] mean when $b \leq a$

when $b \leq a$, the range [a : b-1] is empty
theres no elements.

So:
x in A[a:b-1] means x is in empty set

Vacuous truth: universal statement over empty set is considered true
because nothing contradicts it

$x == A[a:b-1]$ vs x in $A[a:b-1]$
- $x == A[a:b-1]$: x is equal to whole segment
- x in $A[a:b-1]$: x is in segment

1.24) Find Declarative interface for max finder

```
Const int n;
Const int max;
Const Entry A[max];
  Entry largest;
```

Precondition
$0 \leq n \leq max$

Post
$\forall (i=0; i<n)\ A[i] \leq$ largest and
$\exists (j=0; j<n)\ A[j] ==$ largest

for all elements in the 1.12 largest is larger and there exists an element at index j which is the largest.

Post conditions

Present iff $\exists (x=0; x<n)\ A[x] == x$

Present $\Longleftrightarrow$ x in A [0 : n-1]

Present is true only if x is in array

Present iff x in A[0:n-1] when n=0?
x in an empty set which is false

Pre conditions

$0 \leq n \leq max$

- no conditions on x and present
- ghost variable A_0

Rough Notes for Specifying Algorithms

Chapter 2)

- Pre condition: $0 \leq n <= max$ and $A = A_0$
- Post condition: present iff x in A[0:n-1] and for all $(i=0, i<n)$, $A[i] == A_0[i]$
- risky logic `A[i] != x and i<n;`
  - should switch them
- for (;;) → infinite loop body
- {P} C {Q}
  - ↳ Hoare-style correctedness statements

to find Pre condition sub x into Post condition

P(x = x+1) x==1;

Replace → x+1 == 1

x == 0

Pre condition

Formal proof

- Seaweed of logical statements that prove correctness
  - Axiom
  - Math fact
  - rule of inference

Proof Tableau - annotated version of program

Shows: Asserts before/after every statement

```
Assert (n < 0)
  Fact (n<0 => n^2>0)
  n = n * n
Assert (n>0)
```

```
Assert (n != 0)
Fact (n<0 => n^2 > 0)
  n = n * n
Assert (n>0)
```

Practise Problems

Declarative interface

```
const int n;
typename Entry A[n];
const int counter;
```

```
int i = 0;
int count = 0;
While (i < n-1):
    if A[i] > 0;
        count t = 1;
    return count t+=1;
```

Pre Condition $0 \leq i < n$ → $0 \leq n \leq max$

invariant: $\forall (i \mid 0 \leq i < n)$ and count t=1 if x in A[0:i-1]

Post condition: count ≥ 0 → $[\{ K \in [0:n-1] \mid A(K) > 0]$

loop invariant: n-1

Declarative interface

```
Entry Min;
const int n;
typename Entry A[min];
  Entry A[i]
```

```
min = A[0]
For (i=1 : n-1)
    if ( A[i] < min)
        min = A[i]
```

```
Assert (0 < i < n)
while ( i < n-1)   ← loop invar (0<i<n and A[min] < A[0:i+1])
    if A[i] < A[i+1];
        min = A[i]
```

Post condition: $min \leq a \mid\mid A[i]$

$\forall (i<n;\ A(min] \leq x (j<n;$ $A[min] \leq A[j]$

$\forall i \in [0 \ldots n-1]$, $min \in A[j]$

$\exists K \in [0 \ldots n-1]$, such that $A[K] = min$