δ: Transition function defined as

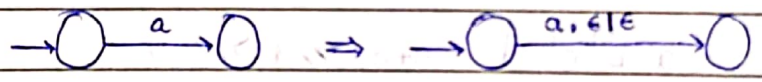
$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \longrightarrow 2^{Q \times (\Gamma \cup \{\epsilon\})^*}$$

where $2^{Q \times (\Gamma \cup \{\epsilon\})^*}$ is power set

& $(\Gamma \cup \{\epsilon\})^*$ is set of all strings over stack alphabet.

- Prove that every Reg. Lang. is CFL.

Proof: Convert the DFA of the RL to PDA.



This represents that we are just not using the stack.

- For every CFL there exist some PDA.
The language of a PDA M, L(M) is a CFL.

- Acceptance of string

We have already seen the main def^n of string by PDA:

- "At the end of string, the PDA should be able to go to some final state." (stack content doesn't matter).
- There's one other def^n (there's no final state):
"At the end of string, the stack must be empty"
- There can be other def^{n}s. and all accept equivalently for any language.

- We have these four equivalent acceptance conditions for PDA:
After completely consuming the input string there exist a path such that:
(1) The path ends on some final state.
(2) The stack becomes empty.
(3) The path ends on some final state <u>or</u> stack becomes empty.
(4) The path ends on some final state <u>and</u> stack becomes empty.

- Cond 1 can be represented as: (Final State Acceptance)

Init Config.

$$(q_0, w, z_0) \xrightarrow{\exists \text{ path}} (q', \epsilon, xyz_0)$$

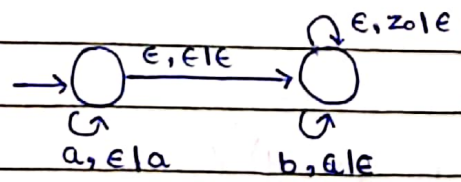
q_0: Start State; w: Remaining String; z_0: Current Stack Content

& $q' \in F$

- Cond 2: (Empty Stack Acceptance)

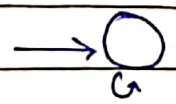
$$(q_0, w, z_0) \xrightarrow{\exists \text{ path}} (q', \epsilon, \epsilon)$$

Ex. $L = \{a^n b^n \mid n \geq 0\}$.



PDA with Empty Stack Acceptance.

Ex. L = All balanced parenthesis strings. $w = \{(,)\}$



(, z0|(z0

(, (|((

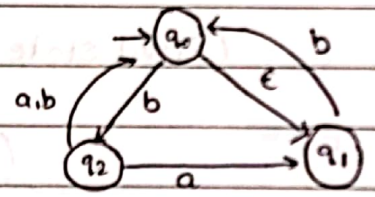
), (|ε

ε, z0|ε

* Deterministic PDA & Deterministic CFL

- **Non-Determinism:** Non-determinism for a ~~state~~ machine happens when at a certain configuration it has more than one transitions possible.

Ex. Count number of Non-Determinism



There are two non-determinism:

(1) On a q_2 can go to q_0 or q_1 Config: (q_2, aw)

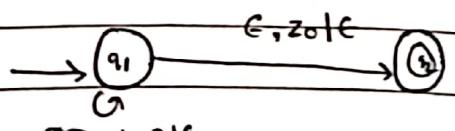
(2) On b q_0 can go to q_2 or q_1 (because of ε).
Config: (q_0, bw)

- DPDA is a PDA with no non-determinism.

Ex. Non-determinism in PDA

$$\delta(q, a, x) = \{(q_1, a), (q_2, y)\}$$

Ex. Is there any non-determinism



0, 0|00

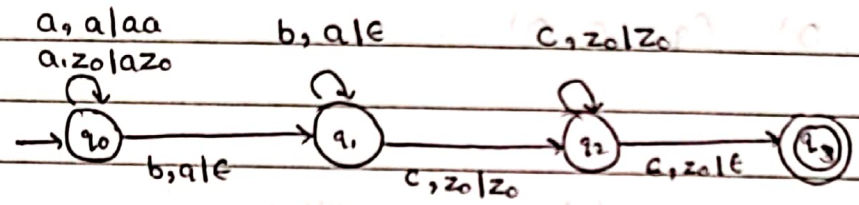
0, z0|0z0

Yes. From q_0, on the following config:

$(q_1, 0w, z_0) \longrightarrow (q_1, w, 0z_0)$ $\{0, z_0 | 0z_0\}$

$\longrightarrow (q_1, 0w, \epsilon)$ $\{\epsilon, z_0 | \epsilon\}$

Ex.



Yes. From q_2, on following config:

$(q_2, cw, z_0) \longrightarrow (q_2, w, z_0)$ $\{c, z_0 | z_0\}$

$\longrightarrow (q_3, cw, \epsilon)$ $\{\epsilon, z_0 | \epsilon\}$

$\therefore$ Non-Deterministic PDA.

Say, $w = aabbcc$.

State Remaining string

$(q_0, aabbcc, z_0) \rightarrow (q_0, abbcc, az_0) \rightarrow (q_0, bbcc, aaz_0) \rightarrow (q_1, bcc, az_0)$

Current Stack Content

$\downarrow$

$\times\ (q_3, c, \epsilon) \leftarrow (q_2, c, z_0) \leftarrow (q_1, cc, z_0)$

$\checkmark\ (q_3, \epsilon, \epsilon) \leftarrow (q_2, \epsilon, z_0) \leftarrow$

$\therefore$ Accepted.

- DPDA can still have null-transitions and no-transitions, but it can't have non-determinism.

Ex. Which transition will have non-determinism with $(q_1, \epsilon, \epsilon)$?

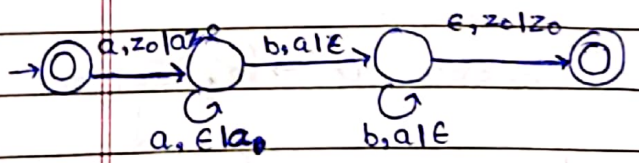
All transitions of the form: $(q_1, __, __)$

Ex. ~~[illegible]~~ will they cause non-determinism:

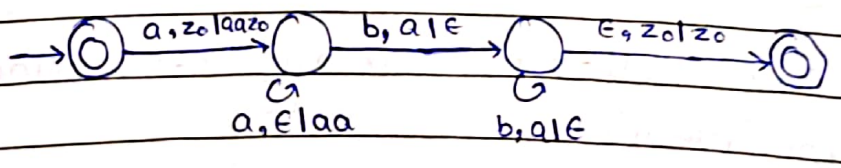
(a) (q_1, a, ϵ) $(q_1, \epsilon, 0)$ ✓
(b) (q_1, a, ϵ) (q_1, b, ϵ) ✗
(c) $(q_1, a, 1)$ $(q_1, \epsilon, 0)$ ✗
(d) $(q_1, a, 0)$ $(q_1, \epsilon, 0)$ ✓

Ex. what causes non-determinism with:
(a) $(q_1, a, 0)$: $(q_1, a, 0)$, $(q_1, \epsilon, 0)$, $(q_1, \epsilon, \epsilon)$, (q_1, a, ϵ)
(b) (q_1, a, ϵ): (q_1, a, Γ), (q_1, ϵ, Γ)
(c) $(q_1, \epsilon, 0)$: $(q_1, \epsilon, \epsilon)$, $(q_1, \Sigma, 0)$, $(q_1, \epsilon, \epsilon)$
(d) $(q_1, \epsilon, \epsilon)$: (q_1, Σ, Γ)

Ex. Create DPDA for $L = \{a^n b^n \mid n \geq 0\}$



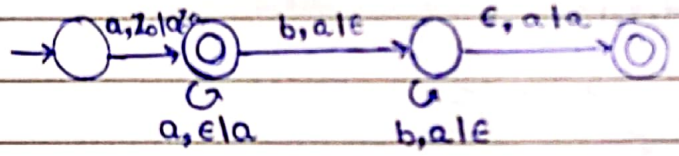
Ex. $L = \{a^n b^{2n} \mid n \geq 0\}$



- DCFL : A language L is DCFL if there exists some DPDA which recognizes L.

- We can show that every Regular Language is DCFL, then we take DFA of that regular lang. & convert it to DPDA.

Ex. $L = \{a^n b^m \mid n > m\}$. Create DPDA.



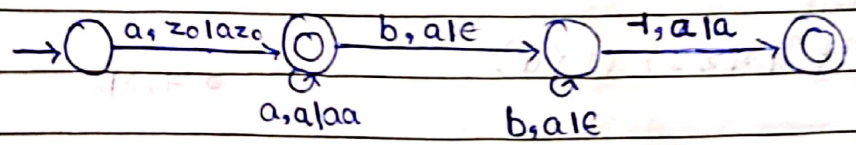
But this is NPDA not DPDA.

Note: There exists a very complicated DPDA for this language & this is indeed a DCFL.

Note: A defn of DPDA allows for endmarker symbol (⊣) which isn't in alphabet. This defn makes this DPDA easy.

Note: There exists a DPDA without using endmarker, but it is very complicated.

Note: DPDA with endmarker ≡ DPDA without it.



(using DPDA with endmarker defn.
DPDA accepts $w\dashv \longleftrightarrow w \in L$)

- DPDA only accepts by final state.

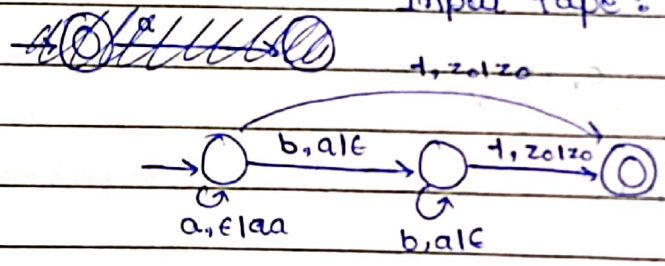
- Formal Def^n: A DPDA is an 8-tuple:

$$(Q, \Sigma, q_0, F, \delta, \Gamma, Z_0, \dashv)$$

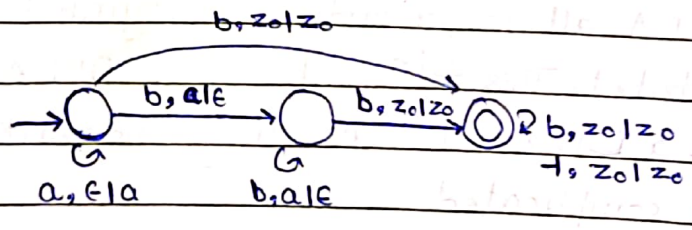
$$\delta : Q \times (\Sigma \cup \{\epsilon\} \cup \{\dashv\}) \times (\Gamma \cup \{\epsilon\}) \longrightarrow (Q \times (\Gamma \cup \{\epsilon\}))$$

Ex. $L = \{a^n b^{2n} \mid n \geq 0\}$

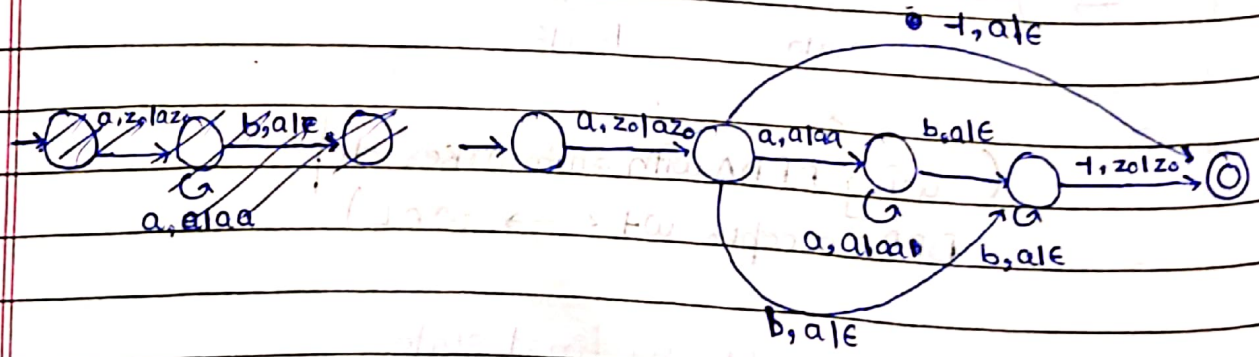
Input tape: $\omega \dashv$



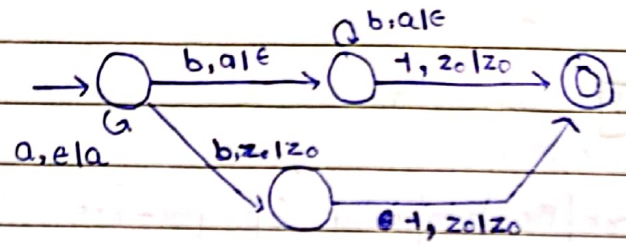
Ex. $L = \{a^n b^m \mid n < m\}$



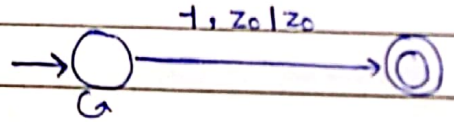
Ex. $L = \{a^n b^n \mid n \geq 1\} \cup \{a\}$



Ex. $L = \{a^n b^n \mid n \geq 1\} \cup \{b\}$

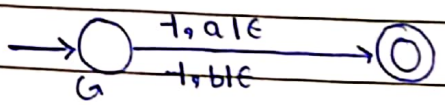


Ex. $L = \{w \mid \#a(w) = \#b(w)\}$



a, a|aa
a, z0|az0
a, b|ε
b, b|bb
b, z0|bz0
b, a|ε

Ex. $L = \{w \mid \#a(w) = \#b(w)\}$



a, a|aa
a, z0|az0
a, b|ε
b, b|bb
~~b, z0|bz0~~
b, a|ε

Ex. $L = \{ww^R \mid w \in \{a,b\}^*\}$

At any point in string we can't be sure what is ahead, so we'll have to use non-determinism.

So, this is CFL but not DCFL.

- So, DCFL $\not\subseteq$ CFL. DCFL $\subset$ CFL

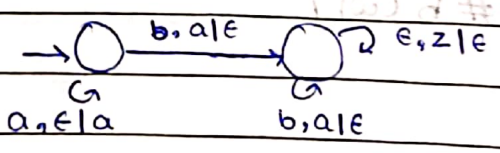
- ## Prefix Property

A language has prefix property (prefix-free) if there is no string in the language which is prefix of ~~any~~ any other string in the language.

Ex. $L = \{a, ba\}$ ✓
Ex. $L = \{a, ab\}$ ✗

- Although DPDA is only defined for final state acceptance, we can create DPDA with empty stack acceptance.

Ex. $L = \{a^n b^n \mid n \geq 1\}$



- Unlike PDA, where PDA with final state acceptance and PDA with empty state acceptance have equal power, in DPDA the ~~power~~ ~~rel~~ ~~same~~ relation of DPDA (FS Acceptance) & DPDA (ES Acc.) power isn't clearly defined and is author dependent. Generally it is:

DPDA (FS Acc.) > DPDA (ES Acc.) [Ullman]

or DPDDA (FS Acc.) = DPDA (ES Acc.)

- [Ullman] DPDA (ES Acc.) accepts a lang L iff it is DCFL and has prefix property.

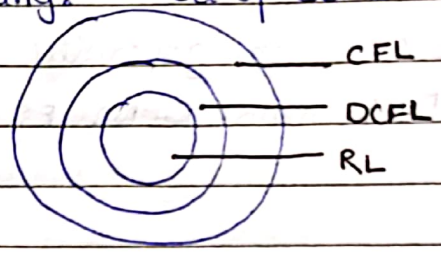
This DPDA (ES Acc.) isn't generally used defn.
By default DPDA means DPDA (FS Acc.).

- Application of DPDA: In Compilers we study DCFL only, so we ~~have~~ see simulations of DPDA only.

* ~~Def~~

* Detection of Language Class

- Set of Regular Lang. $\subset$ Set of DCFL $\subset$ Set of CFL over Σ



* Detection of Regular Language

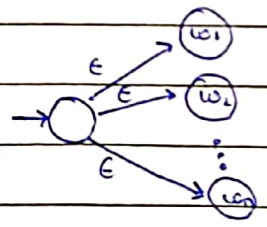
- Every finite language is Regular

$L = \{w_1, w_2, \ldots, w_n\}$ for some whole no. n.

$n \in W$

$RE = w_1 + w_2 + w_3 + \ldots w_n$

NFA:



RG: $S \rightarrow w_1 \mid w_2 \mid \ldots w_n$

- Some standard languages:

1. ϕ
2. $\{\epsilon\}$
3. Σ^*
4. Starting with
5. Ending with

⋮

- Use Myhill-Nerode theorem as it can show both Regularity or Non-Regularity of a language.

- [Informal] If the language requires comparision, or matching or counting, then try to show that there are infinite possibilities and hence language is non-regular.

Ex. Check Regular or Not Regular.

(a) $A = \{a^n b^n \mid n \geq 10\}$

(b) $B = \{a^m b^n \mid n \geq 10 \text{ or } m \leq 10\}$

(c) $C = \{a^m b^n \mid m \neq n\}$

(d) $D = \{a^m b^n \mid m \% 2 = 1 \wedge n \% 2 = 1\}$

(e) $E = \{a^n \mid n \text{ is prime}\}$

(a) [Informal] requires comparision. Possibilities: $a^{10}b^{10}, a^{11}b^{11}, \ldots$

$\therefore$ Non-regular.

[Formal]

$S = \{a^n b \mid n \geq 10\}$ is differentiable for all two strings by b^{n-1}, i.e.

$\forall m, n \geq 10, \quad a^n b \not\equiv a^m b$ by b^{n-1}

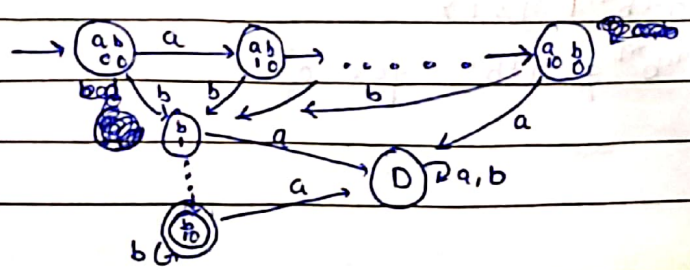
$\therefore$ Non-regular.

(b) [Informal] At any point of time, we just keep track if $n \geq 10$ and if $m \leq 10$.

$\therefore$ Regular.

[Formal]

$RE = (\epsilon + a + a^2 + a^3 + \ldots a^{10}) \cdot b^{10} b^*$



$\therefore$ Regular.

(c) [Informal] Comparision. At any point in time we need to remember exact count of a, so ② ③ ... ∞ states would be needed.

[Formal]

For $S = \{a^n \mid n \geq 01\}$

$\forall_{\substack{m,n \\ m \neq n}}$ $a^m \not\equiv a^n$ by b^m

$\therefore a^m b^m \notin L \wedge a^n b^m \in L$.

(d) Odd no. of a's followed by odd no. of b's.

RE = $a(aa)^* . b(bb)^*$

(e) For $S = \{a^p \mid p \text{ is prime}\}$

$\forall_{\substack{m,n \\ m \neq n}}$ $a^p \not\equiv a^q$ by a^4

(f) $F = \{ wcw \mid w \in \{a,b\}^* \}$

(g) $G = \{ w \mid w \text{ is either } a^m b^n \text{ or } b^m a^n, m,n \geq 0 \}$

(h) $H = \{ a^n b^n c^n \mid n \geq 0 \}$

(i) $I = \{ a^x b^y c^z \mid x,y,z \geq 1 \}$

(j) $J = \{ a^n b^m a^n \mid n \geq 10 \ \& \ m \leq 10 \}$

(f) For $S = \{ a^m c \mid m \geq 1 \}$ by

$\forall_{\substack{m,n \\ m \neq n}}$ $a^m c \not\equiv a^n c$ by a^m

(g) RE = $a^* b^* + b^* a^*$

(h) For $S = \{ a^n b^n \mid n \geq 1 \}$

$\forall_{\substack{m,n \\ m \neq n}}$ $a^m b^m \not\equiv a^n b^n$ for c^n

(i) $aa^*bb^*cc^*$

(j) $S = \{a^n b^m \mid n \geq 10, m \leq 10\}$.

$\forall_{\substack{m,n \\ m \neq n}}$ $a^m b^z \not\equiv a^n b^y$ by a^m

(k) $K = \{a^n b^m a^n \mid n \leq 10, m \geq 10\}$

$b^{10}b^* + ab^{10}b^*a + a^2b^{10}b^*a^2 + \ldots + a^{10}b^{10}b^*a^{10}$

Ex. $\Sigma = \{a,b\}$. $L = \{a^{2m}b^m \mid m \geq 0\}$. Prove L isn't reg. using Pumping Lemma.

Let's say L is Regular.

For ~~some~~ all $p \geq 1$. $w = a^{2p}b^p$ belongs to L.

$x = a^{p-t}$ $y = a^{t}$ $z = a^p b^p \Rightarrow w = a^{p-t}a^t a^p b^p$ $t > 0$.

So, $|xy| \leq p$. ✓

but for xy^2z i.e. $a^{p-t}a^t a^p b^p \notin L$.

$\therefore$ L isn't Regular.

Ex. Which are regular:

(a) $A = \{a^n a^n \mid m \geq 0\}$

(b) $B = \{1^m 0 1^n \mid m \geq 0, n \geq 0, m > n\}$

(c) $C = \{w \in \{0,1\}^* \mid w$ has same no. of 00 & 11$\}$

(a) $\equiv \{a^{2n} \mid n \geq 0\}$ $RE = (aa)^*$. Reg.

(b) $S = \{1^m 0 \mid m \geq 1\}$

$\forall_{\substack{m,n \\ m \neq n}}$ $1^m 0 \not\equiv 1^n 0$ by m, $m < n$

n, $n < m$.

(c) $S = \{000^n \mid n \geq 1\}$

$\forall_{\substack{m,n \\ m \neq n}}$ $000^n \not\equiv 000^m$ by 111^n

Ex. $L = \{a^n b^m c^l \mid 1 \leq m \leq 3,\ n \geq m,\ l \geq m\}$

RE: $a^* a b c c^* + a^* a^2 b^2 c^2 c^* + a^* a^3 b^3 c^3 c^*$

Ex. $L = \{w \mid w \in \{0,1\},\ n_0(w) > n_1(w)\}$

$L = \{0\}$ $\quad w \in \{0,1\}$ not $\{0,1\}^*$

$\therefore$ Regular

Ex. $L = \{0^n \mid n = 2^k$ for some $k > 1\}$

$L = \{0^4, 0^8, 0^{16}, 0^{32}, 0^{64}, \ldots\}$

$S = \{0^n \mid n = 2^k$ for some $k > 1\}$

$\forall_{m,n}$ $0^{2m} \not\equiv 0^{2n}$ by 0^{2m}

$0^{2m}, 0^{2m} \in L$

$0^{2n}, 0^{2m} \notin L$

Ex. $L = \{w \mid w \in \{0,1\}^*,\ w$ has at least as many (00) as 1111$\}$

$\#(000)_L \geq \#(1111)_L$

$S = \{00^n \mid n \geq 1\}$

$\forall_{\substack{m,n \\ m \neq n}}$ $000^m \not\equiv 000^n$ by 111^m $\quad m > n$

111^n $\quad n > m$

$\therefore$ Non-regular.

Ex. $L = \{w \mid \#01(L) = \#10(L),\ w \in \{0,1\}^*\}$.

we have comparison. But do we have ∞ possibilities?

L has, 0 ✓ 1 ✓ ϵ ✓ 0^* 1^*

- If we start our string with 0: 0~~, whenever we try to add 10, 01 would already be there.

  Ex. 000$\boxed{10}$, 000$\boxed{11}$$\boxed{10}$, 00$\boxed{11}$$\boxed{01}$$\boxed{01}$

  If we end 0~~ with 0, #01 = #10, else #01 > #10.

- Similarly for 1~~1, #10 = #01.

$\therefore$ RE $\Rightarrow$ $\epsilon + 0 + 1 + 0\Sigma^*0 + 1\Sigma^*1$

- Similarly for L $\Rightarrow$ #01 > #10

  RE $\Rightarrow$ $0\Sigma^*1$

- For L $\Rightarrow$ #10 $\geq$ #01

  RE $\Rightarrow$ $\underbrace{\epsilon + 0 + 1}_{0} + \underset{\text{equal}}{0\Sigma^*0} + \underset{\text{equal or greater}}{1\Sigma^*}$

Ex. $L = \{w \mid w \in \{0,1\}^*,\ \#110 \geq \#011\}$

- ~~If we start with 0: 0~~

  ~~Ex. 011, 0110, 00011, 000110, 001.~~

  ~~So, ① 0~~0 has #110 = #011 ③ 001 has #110 = #011~~

  ~~② 0~~1 has #011 > #110~~

- ~~If we start with 11, 110, 111011, 11011000011, 110011~~

  ~~11~~0 has #110 > #0011~~

  ~~11~~1 has #110 = #011~~

~~• If we have 10:~~

Informal

We already have comparision. Now we have to show ∞ possibilities.

• Is it possible to have infinitely many 110 without having 011?

... 110 ~~~ 110 ~~~~~~ 110 ~~ 110 ~~~~

No matter what we do. b/w every two consecutive 110, we'll have exactly one 011.

Similarly b/w every two 011, (consecutive)

011 ~~~~~~ 011

we'll have exactly one 110.

So, we can't show infinite possibilities. So, Regular.

Ex. $L = \{ w \in \{0,1\}^* \mid \#010 = \#101 \}$

$S = \{ (010)^n \mid n \geq 1 \}$

$\forall_{\substack{m,n \\ m \neq n}}$ $(010)^n \not\equiv (010)^m$ by $01(101)^n$

$\therefore$ Not-regular.

Ex. Figure out $L = \{ w \in \{0,1\}^* \mid \#110 \geq \#011 \}$

Ob.1: Starts with 0: 0 ~~~

If we have 110, there will also be 011 preceeding it

If we end on 0: $\#110 = \#011$.

If we end on 1: $\#110 < \#011$

Ob.2: Start with 11: 11 ~~~

If we have 011, there will be 110 preceeding it.

If we end with 11: $\#110 = \#011$

If we end with 0: $\#110 > \#011$

If we end with 01: $\#110 > \#011$

Ob.3: Start with 10: 10 ~~~

Behaves like Starts with 0.

$\therefore$ RE: $0+1+\epsilon + \underbrace{0\Sigma^+0 + 10\Sigma^+0 + 11\Sigma^+}_{\text{3 or more len.}}$

$\underbrace{(0+1)(0+1)}_{\text{2 len}}$

Ex. $L = \{ w \in \{0,1\}^* \mid \#001 = \#100 \}$

Ob1.: Starts with 1: 1 ~~~

If we have a 001, we'll have 100 preceeding it.

If we end on 1: $\#001 = \#100$

If we end on 00: $\#001 < \#100$

If we end on 10: $\#001 = \#100$.

Ob2.: Start 00: $\#001 \geq \#100$

end on 1: $\#001 > \#110$

end on 00: $\#001 = \#100$

end on 10: $\#001 > \#100$

Ob3: 01 Start: Same as ~~Starts~~ with 1.

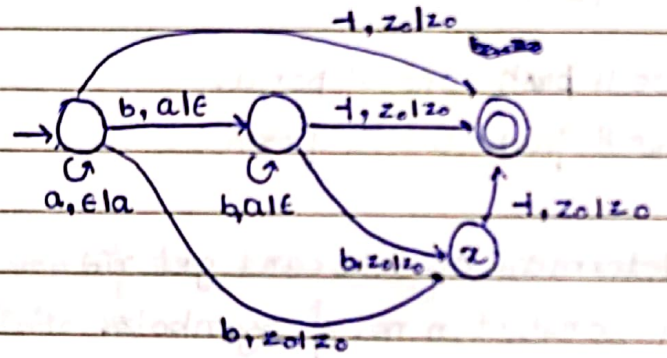
$\therefore$ RE: $\epsilon + 0 + 1 + (0+1)(0+1) + 1\Sigma^*(1+10) + 01\Sigma^*(1+10)$
$+ 00\Sigma^*00 + 000$

Informal: consecutive
Two 001 can't come without exactly one 100.

* Detection of DCFL & CFL

- At any point of time DPDA can only see atmost one symbol on the input tape and deterministically decide which move to take, what to push, what to pop.

Ex. $L = \{a^n b^m \mid n = m \text{ or } n+1 = m\}$



In state x, we are skipping exactly one 'b'.
Similarly we can have another state which can be used to skip 2 b's & so on.

So, we can have states in DPDA which can see/skip n no. of symbols where n would be constant.

- So, DPDA can deterministically look at some "constant" number of symbols.

- So, for a lang. L, if by seeing a "constant" no. of symbols (const., not finite) if you can decide about what to push, what to pop, when to push, when to pop, etc. deterministically, then the language is DCFL.

Ex. $L = \{a^n b^{2n} \mid n \geq 0\}$

Yes. On every a, we push 'aa'. On every b we pop one 'a'. On ⊣, if we pop z_0, we accept.

Ex. $L = \underset{(L_1)}{\{a^n b^n\}} \cup \underset{(L_2)}{\{a^n b^{2n}\}}$

If $w \in a^n b^n$, we'll push one 'a' per a.
If $w \in a^n b^{2n}$, we'll push two 'a' per a.
(or $a^n b^{n+n}$)
So, there is non-determinism. We can't get rid of it by seeing any constant n no. of symbols.

∴ Not DCFL. (Even though L_1 & L_2 are DCFL)

Ex. $L = \{a^n b^n\} \cup \{a^n b^{n+K}\}$ K's constant.

For each a, we'll push exactly one a.
For each b, we'll pop exactly one a.
If b, but z_0 on stack, we'll skip exactly K b's by states before accepting.
If ⊣ on z_0, accept.

Ex. $L = \{ww \mid w \in \{0\}^*\}$

$(00)^*$. Regular. Hence DCFL.

Ex. $L = \{ww \mid w \in \{0,1\}^*\}$

Even if we could ~~be~~ see K symbols at a time, we can't find middle deterministically.
Even if we were able to find middle m the $(m+1)^{th}$ symbol would have to compare with 1st symbol. But stack is LIFO.
So this isn't even PDA.

Ex. $L = \{ww^R \mid w \in \{0,1\}^*\}$

Even if we see K symbols, can't find middle, with some finite K.
$\therefore$ Non-determinism. So, not DCFL. But CFL.

Ex. $L = \{wcw^R \mid w \in \{a,b\}^*\}$

We can find middle with just one symbol. If c, middle.
So, DCFL.

Ex. $L = \{a^n b^n \mid n \geq 0\}$
Not Regular. DCFL. So, CFL.

Ex. $L = \{w \in \{a,b\}^* \mid \#a \neq \#b\}$
This is $L = \{w \in \{a,b\}^* \mid \#a > \#b\} \cup \{w \in \{a,b\}^* \mid \#a < \#b\}$

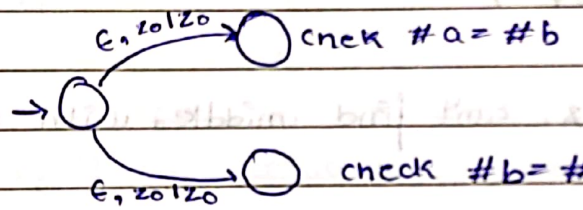
For a on tape, if a on stk, push a, if b on stack pop b.
For b vice-versa.
For $\dashv$, if a or b on stk, accept.

Ex. $L = \{a^n b^m c^k \mid n = m \text{ or } m = k\}$

Not Reg. $S = \{a^{n-1} b^n \mid n \geq 1\}$ Diff by c^n

There are two overlapping comparisions, i.e if we do comp. of a & b, we can't compare b & c, and vice-versa. So, non-determinism. ∴ Not DCFL.

CFL, since we can use non-determinism.



∴ CFL.

Ex. $L = \{a^i b^j c^k \mid i, j, k \geq 0,\ i \neq j \text{ or } j \neq k\}$

Again, two overlapping comparisions. So, not DCFL. But CFL.

Ex. $L = \{xw \mid x \neq w,\ x, w \in \{a,b\}^*\}$

Regular set $x = \epsilon$ & $w = \{a,b\}^+$

∴ $L = \{a,b\}^+$

∴ DCFL. CFL.

Ex. $L = \{xw \mid |x| = |w|,\ x, w \in \{a,b\}^*\}$

$L = ((a+b)^2)^*$ All even len strings.

$\therefore$ Regular. DCFL. CFL.

Note: $w \in \{0,1\}^*$

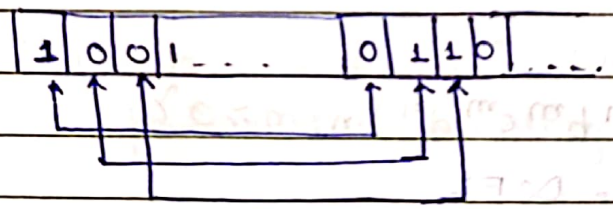
then we can define $\bar{w}$ or w' $\Rightarrow$ reverse every bit

Ex. $w = 101101$

$\bar{w} = 010010$

Ex. $L = \{ww' \mid w \in \{0,1\}^*\}$

Not CFL. Requires comparision of type



(This is forward matching & isn't possible in stack.)

Ex. $L = \{wcw \mid w \in \{0,1\}^*,\ |w| \leq 2^{100}\}$

This is finite language, since there are finite no. of w, $|w| \leq 2^{100}$.

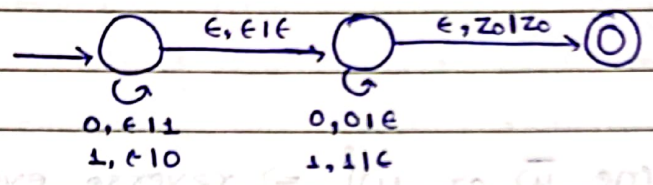
$\therefore$ Regular.

Ex. $L = \{a^n b^n c^n \mid n \geq 0\}$

Not CFL. Single comparision but b/w more than two.

Ex. $L = \{ w(\overline{w})^R \mid w \in \{0,1\}^* \}$

CFL.



Not DCFL. Can't find middle deterministically.

Ex. $L = \{ a^n b^n c^m d^m \mid n, m \geq 0 \}$

Not Reg.

DCFL.

Ex. $L = \{ a^n b^m c^m d^n \mid n, m \geq 0 \}$

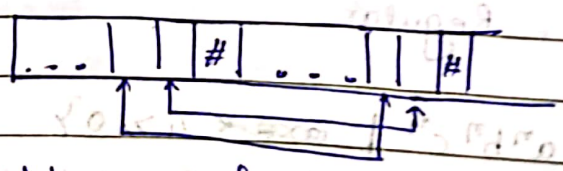
Not Reg. DCFL.

Ex. $L = \{ a^n b^m c^n d^m \mid n, m \geq 0 \}$

Not CFL. Overlapping comparision which can't be solved using Non-Det.

Ex. $L = \{ a\#b\#c \mid a, b, c \in \{0,1\}^*, \ bin(a) + bin(b) = bin(c) \}$

Not CFL.



Addition would req. fwd matching.

Ex. $L = \{ w \in \{a,b,c\}^* \mid \#a < \#b < \#c \}$

One comparision but more than two symbols.
$\therefore$ Not CFL.

Imp

Ex. $L = \{ ww \mid w \in \{a,b\}^* \}$
Find $\bar{L}$ & class of $\bar{L}$.

$\bar{L} = \{ w \mid len(w) \text{ is odd} \} \cup \underbrace{\{ wx \mid |w| = |x| \text{ but } w \neq x \}}$

This is CFL. Reg $\cup$ CFL = CFL.
(This might not look like CFL, but it is. Proof is complicated. But this is a famous language.)

Ex. $L = \{ wcx \mid |w| = |x|,\ w, x \in \{a,b\}^*,\ w \neq x \}$
Not DCFL. Comparision of not equality req. non-determinism
$\therefore$ CFL.

Note: Take care when unary alphabet. Every CFL over unary alphabet is regular, and vice-versa.
- For any lang L over unary alphabet, L^* is regular.

Ex. $L = \{ a^p \mid p \text{ is prime} \}$
L isn't reg. but L^* is regular.

Ex. $L = \{ a^{n^2} \mid n \in \mathbb{N} \}$
Not reg. since over unary alpha.
Reg $\longleftrightarrow$ CFL
$\neg$Reg $\longleftrightarrow$ $\neg$CFL $\therefore$ Not CFL.

Ex. $L = \{a^n \# a^n \mid n \geq 0\}$

CFL. DCFL. Not regular. (Not unary)

- Note: In CFL, we check through multiple paths but, CFL accepts only when one of the paths checks all cond^{n}s successfuly.

Ex. $L = \{a^n b^n c^n \mid n \geq 0\}$

We can create PDA which has three paths

(i) $\#a = \#b$ (ii) $\#b = \#c$ (iii) $\#a = \#c$.

but a single path can't check all 3 cond^n at once.

Closure Properties

- We have a set A & operation $*$. then we say set A is closed under op^n $*$ iff

$$\forall x,y \in A \quad x * y \in A$$

Ex. $(\Sigma^*, \cdot)$ $\cdot$ is concatenation.
Yes Σ^* is closed under $\cdot$ since all strings $\in \Sigma^*$.

Ex. $(\{w \mid |w| \% 2 = 0\}, \cdot)$ $\Sigma = \{a,b\}$
Closed $\forall w, x \in L$ $w \cdot x \in L$
since even + even = even length.

Ex. $(\{w \mid |w| \% 2 = 1\}, \cdot)$ $\Sigma = \{a,b\}$
Not closed. Counterexample $a^3 \cdot a^3 \notin L$.

Ex. (Set of all Reg. Lang. over Σ, complement)

Closed $\forall L \in RL \Rightarrow$ Create DFA $\Rightarrow$ Swap F & NF States $\Rightarrow$ DFA for $\bar{L}$
Since DFA for $\bar{L}$ exists for all L, so closed.

- Reg: Set of all Regular Languages over some Σ.

- For all set operations over Reg., like union, intersection, subtraction, symm. difference, we can create product DFA, therefore

$$\forall L_1, L_2 \in Reg \quad L_1 * L_2 \in Reg$$

$* \in \{\cup, \cap, -, \Delta$, or other set op$^n\}$

$\therefore$ (Reg, $*$) is closed over $*$.

$L_1 - D_1$
$L_2 - D_2$ $\rightarrow D_{1\times 2} - \therefore$ Reg.

- The set Reg is also closed under Kleene Star.

$L_1 \Rightarrow$ RE exists (r_1) $\Rightarrow$ For any r_1, r_1^* is also RE.
$\therefore$ L_1^* is Regular.

Similarly L_1^+ is also regular

- (Reg, .) . is concatenation is also closed
$\forall L_1, L_2 \in Reg$ L_1 has r_1, L_2 has r_2 & $r_1 . r_2$ is also RE

$\therefore$ (Reg, .) is closed.

{Can also show using NFA}

Ex. Reg = Set of all reg. Lang.
$S = Reg - \{\emptyset\}$ {except empty lang.}
then, S is closed under:

① Union ③ Set Diff ⑤ Kleene Star
② Intersection ④ Symm. Diff ⑥ Kleene Plus
⑦ Concatenation

① Closed. Union of two non-empty lang can't be empty.
② Not Closed. Counterex: $L_1 = \{a\}$ $L_2 = \{b\}$
③ Not Closed. Counterex: $L_1 = \{a\}$ $L_2 = \{a\}$
④ Not closed. ——— " ———
⑤ Closed. $L^* = L^0 \cup L^1 \cup L^2$ For any non-empty
i.e. L^i isn't empty & there union isn't empty.

⑥ $L^+ = L^1 \cup L^2 \cup L^3 \ldots$

i.e. L^1, isn't empty & this union isn't empty either.

$\therefore$ closed.

Ex. #states in mDFA of L & $\bar{L}$ are same. Prove.

$L \Rightarrow D$. Say D has m states as min DFA. $\Rightarrow$ Swap F & NF

$\Downarrow$

This is DFA for $\bar{L}$, say D'

Now let's say,

D' (with m state) isn't min DFA for $\bar{L}$.

Say D_2' is mDFA for $\bar{L}$ with n $(n < m)$ states.

So, $\bar{L} \to D_2' \Rightarrow$ Swap F & NF states $\Rightarrow D_2'$ is DFA for L with n $(n<m)$ states.

But we said m is min no. of states in mDFA for L.

$\therefore$ Contradiction.

$\therefore$ #states in mDFA of L & $\bar{L}$ is same.

Ex. Given a Reg. lang, is it closed under concatenation?

No. Counterex: $(\{a\}, .)$ $L = \{a\}$

$a.a \notin L$.

$\therefore$ $\{a\}$ is Reg but not closed under concatenation.

(Mind the "base set")

Ex. Is Reg closed under reversal.

for any $L \in Reg$, $L^R = \{ w^R \mid w \in L \}$

Basically is reversal of any reg. lang also reg?

Yes.

$\forall L \in Reg \; \exists RE \; r \quad r^R$ is also RE.

$\therefore L^R \in Reg.$

Ex. Given that Reg is closed under Compl. & Intersec.

Show Reg is closed under union.

$\forall L_1, L_2 \in Reg, \quad L_1 \cup L_2 = \overline{\overline{L_1 \cup L_2}}$

$\Rightarrow \overline{\overline{L_1} \cap \overline{L_2}}$ {Demorgan's}

So, $R \in Reg \rightarrow \bar{R} \in Reg.$

$\bar{L_1} \Rightarrow Reg \quad \bar{L_2} \in Reg.$

$\therefore \bar{L_1} \cap \bar{L_2} \in Reg$

& $\overline{\bar{L_1} \cap \bar{L_2}} \in Reg.$

$\therefore L_1 \cup L_2 \in Reg.$

Ex. Is Reg closed under $\odot$ (XNOR kinda op).

$L_1 \odot L_2 =$ (Either in both L_1 & L_2) or (not in L_2 and not in L_2)

$\Rightarrow \underbrace{(L_1 \cap L_2)}_{\in Reg} \cup \underbrace{(\bar{L_1} \cap \bar{L_2})}_{\in Reg}$

$\in Reg \quad \cup \quad \in Reg$

$\in Reg.$

$\therefore$ Closed.

Ex. Is Reg closed under $\rightarrow$

$L_1 \rightarrow L_2 = \neg L_1 \cup L_2$

Yes. $\forall_{L_1, L_2 \in Reg}$ L_1 is reg, L_2 is reg $\rightarrow$ $L_1 \cup L_2$ is reg

Ex. Which of these is/are regular if L is regular:

(A) $\{ww \mid w \in L\}$

(B) $\{wx \mid w, x \in L\}$

B is always regular. B is same as $L^2 = L.L$ & for any reg L, $L.L$ is reg.

A may/maynot be reg.

Say $L = \Sigma^*$ then $A = \{ww \mid w \in \Sigma^*\}$

We know this isn't even CFL.

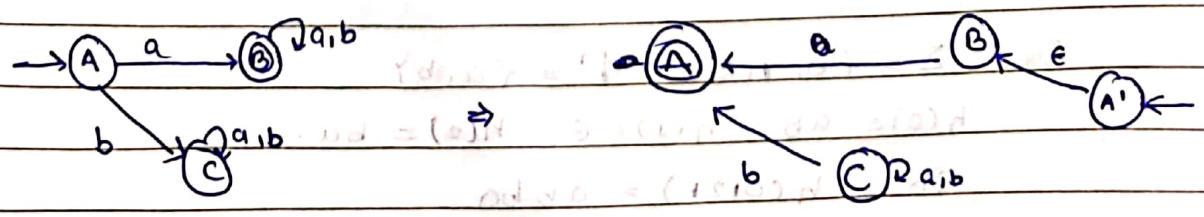
- To create reversal lang (L^R) of DFA D's lang: $D(Q, \Sigma, \delta, F, q_0)$

① Reverse all transition.

② Create a new state. Make it initial. (q_0'). From q_0' add ϵ-transition to all final states in F.

③ Make original initial state (q_0) as final & all current final states (in F) non-final.

Ex.



L(D) $\qquad$ $L(D_2) = L(D)^R$

Ex. Let for Reg. lang L, D be mDFA with m no. of states. Comment of no. of states in mDFA of L^R.

Let's say $L = \{ w \in \{a,b\}^* \mid K^{th}$ symbol from left is 'a'$\}$
w.k.t mDFA of L has $(K+2)$ states.

L^R would be $= \{ w \in \{a,b\}^* \mid K^{th}$ symbol from right is 'a'$\}$.
w.k.t mDFA of L^R has 2^K states.

$\therefore$ L $\downarrow$ m states | mDFA | L^R $\leq 2^n$ states.

(Here since consider K^{th} symbol from right as L_1 & K^{th} from left as L_1^R) $\downarrow$ $(K+2)$ $\downarrow$ 2^K

- Homomorphism: Given alphabet Σ & Γ, homomorphism is any func h, $h: \Sigma \rightarrow \Gamma^*$.

Ex. $\Sigma = \{a, b\}$ $\Gamma = \{0, 1\}$
$h: \Sigma^* \rightarrow \Gamma^*$
$h = \{ (a, 010), (b, 110) \}$

So, $h(abaa) = 010110010010$.

Ex. $\Sigma = \{0, 1, 2\}$ $\Gamma = \{a, b\}$
$h(0) = ab$ $h(1) = \epsilon$ $h(2) = ba$.
So, $h(0121) = abba$.

We can map a lang L over Σ, for $h: \Sigma \to \Gamma^*$

$$h(L) = \{ h(w) \mid w \in L \}.$$

Note: $h(w)$ where $w = \epsilon$ is ϵ by def^n.

Ex. $h(0) = acdb$ $\quad h(1) = ccc$.

$L = 0^* 1^*$

$h(L) = (acdb)^* (ccc)^*$

(Homomorphism means same structure).

Ex. Is Reg closed under homomorphism h?

Yes. $\forall L \in Reg$ L has some RE r. In r replace all symbol x with h(x). The result is still RE.

$\therefore$ h(L) is also RE.

* CFLs

- Let C be set of all CFLs over some alphabet Σ.

- C is closed under union.

$\forall L_1, L_2 \in C$, let G_1 & G_2 be grammar (CFG) resp. with S_1 & S_2 as start symbols. So,

$$S \rightarrow S_1 \mid S_2$$

is also CFG.

$\therefore$ $L_1 \cup L_2$ is also CFL.

$L_1 \cup L_2 \in C$.

- C is closed under concatenation.

$\forall L_1, L_2 \in C$ let G_1 & G_2 be CFG with start sym. S_1 & S_2 resp.

So, $S \rightarrow \underbrace{S_1 . S_2}_{\{(V+T)^*(V+T)^*\}}$

is also CFG.

$\therefore$ $L_1 . L_2 \in C$.

- C is closed under Kleene Star and Kleene Plus.

$\forall L_1 \in C$ .... S_1, then,

$S \rightarrow S_1 S \mid \epsilon$ is also CFG.	$S \rightarrow S_1 S \mid S_1$ is also CFG.

- C is closed under homomorphism.

(Complicated proof).

- C isn't closed under intersection.

Counterex: $L_1 = \{a^n b^n c^m \mid n, m \geq 0\}$

$L_2 = \{a^n b^m c^m \mid n, m \geq 0\}$

$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$

& this isn't CFL. (comparision of more than two symbols)

So, $\forall L_1, L_2 \in C$, $L_1 \cap L_2$ may/maynot $\in C$.

- C isn't closed under complementation

Counterex:

$L = \{a^m b^n c^k \mid m \neq n \text{ or } n \neq k\}$

$\bar{L} = \{w \mid ba \text{ or } ca \text{ or } cb \text{ is substring}\} \cup \{a^n b^n c^n \mid n \geq 0\}$
$\cup \{a, b, c\} \cup \{ab, ac, bc, aa, bb, bc, cc\}$

& $\{a^n b^n c^n \mid n \geq 0\}$ isn't CFL.

$\therefore \forall L \in C$ $\bar{L}$ may/maynot $\in C$.

- Some standard Non CFLs whose complement is CFL:

① $\{ww \mid w \in \{a,b\}^*\}$

- C isn't closed under set difference.

Counter ex:

$$\underset{\text{CFL}}{\Sigma^*} - \underbrace{(\{ww \mid w \in \Sigma^*\})}_{\text{CFL}} = \underset{\text{Not CFL.}}{\{ww \mid w \in \Sigma^*\}}$$

- C is closed under reversal. (Can be shown by reversing all RHS of production rules.

- Let L_R be arbitrary Reg. Lang & L_C be arbitrary CFL.

Ex. Are the following correct?

① $L_R - L_C \in C$

② $L_C - L_R \in C$

1 is false. Counterex. $\underset{\text{Reg}}{\Sigma^*} - \underset{\text{CFL}}{\overline{\{ww \mid w \in \Sigma^*\}}} = \underbrace{\{ww \mid w \in \Sigma^*\}}_{\text{Not CFL.}}$

(2) $L_C - L_R = L_C \cap \overline{L_R} = \underset{\text{CFL}}{L_C} \cap \underset{\text{Reg}}{L_R'} \in C.$

$\therefore$ Correct.

- $L_C \cap L_R \in C.$

- Note: when specific lang given, don't use closure property. For example, if CFL's set isn't closed under compl. means $\exists L \in$ CFL's $\overline{L} \notin$ CFL's.

But

$L = \{a^n b^n \mid n \geq 0\}$.

$\overline{L} = \{w \mid aba \text{ is substr.}\} \cup \{a^n b^m \mid n \neq m\}$

$\overline{L}$ is CFL.

They can be helpful in the case, some set is closed under some op^n.

Ex. Reg's is closed under compl.

So, we can just check if L is reg & it's compl. will be Reg as well.

imp

- DCFLs are only closed under Complement.

- L_R, L_C, L_{DC} are arbitrary RL, CFL & DCFL.

① $CFL \cap Reg = CFL$

② $DCFL \cap Reg = DCFL$

③ $CFL \cup Reg = CFL$

④ $DCFL \cup Reg = DCFL$

⑤ $CFL - Reg = CFL \cap \overline{Reg} = CFL$

⑥ $DCFL - Reg = DCFL \cap \overline{Reg} = DCFL$

⑦ $Reg - CFL$ may/maynot be CFL

⑧ $Reg - DCFL = Reg \cap \overline{DCFL} =$ ~~Reg~~ DCFL

* Closure Properties of Finite Languages

- Closed under union, intersection, set difference, reversal.
- Not closed under complement.

Counter: $L = \{a^n b^n \mid n \leq 2\}$ is finite,

$\bar{L} = \{a^n b^n \mid n > 2\} \cup \{w \mid w \in \{a,b\}^*, ba \text{ is substr}\}$

$\cup \{a,b\}$

- Not closed under Kleene Star & Kleene plus.

Counterex: $L = \{a,b\}$

- Closed under homomorphism.
- Closed under concatenation.

* Closure Properties of Non-Regular

(Base Set: $\overline{Reg}$ or $2^{\Sigma^*} - Reg$)

- Not closed under union.

  Counter ex: $L_1 = \{ww \mid w \in \Sigma^*\}$ $L_2 = \overline{L_1}$

  $L_1 \cup L_2 = \Sigma^* \notin$ NonReg

- Not closed under intersection.

  Counterex: $L_1 = \{ww \mid w \in \Sigma^*\}$ $L_2 = \overline{L_1}$

  $L_1 \cap L_2 = \emptyset \notin$ NonReg.

- ~~Not closed under complement.~~

  ~~Counterex. $L = \{a^n b^n \mid n \geq 0\}$~~

  ~~$\overline{L} = \{w \mid ba$ substr$\} \notin$ NonReg.~~
  ~~$\cup \{a,b\}$~~

- Not closed under concatenation.

  Counterex: let L_1 be some non-reg.

  $L_2 = \overline{L_1} \cup \{\epsilon\}$

  (Assuming $\lambda \in L_1$)

  So, $L_1 . L_2$ would be Σ^*

  {Since L_1 has ϵ $\epsilon . w = w$
  & L_2 has ϵ $w . \epsilon = w$}

  & $L_1 \cup L_2 = \Sigma^*$

- Closed under complementation.

  Proof: Let L be some non-reg. Let's say non-reg not closed under compl. So, $\overline{L} \in$ Reg.

  We know $L_R \in$ Reg, so $\overline{L}$ must be Reg.

  but $\overline{\overline{L}} = L \in$ NonReg.

  $\therefore$ Contradiction.

- Closed under reversal.
- Not closed under Kleene Star & Kleene Plus.

Counterex: $L = \{a^p \mid p \text{ prime}\}$

w.k.t any lang L over unary alphabet, L^* is reg.

$\therefore L^* \notin$ Non-Reg.

- Note: Non-Reg lang's set is only closed under
  1. Complement
  2. Reversal.

- Note: Any lang L belonging to ~~Stan~~ some standard class of language (RL, CFL, DCFL, CSL, RE, Rec), union with some set of finite strings doesn't change the class of the language.

- Note: ① Finite language L can be also defined as:

either [$\exists n \in \mathbb{N}$ such that there exist a bijection b/w L & $\{1, 2, \ldots, n\}$

i.e. $L \equiv \{1, 2, \ldots, n\}$]

or [L is empty]

② Finite Lang: L is finite iff $\exists n \in \mathbb{N}\ \forall w \in L\ |w| \leq n$.

Ex. Is the class of Reg. lang. closed under inf. union?

False.

Counter Ex:

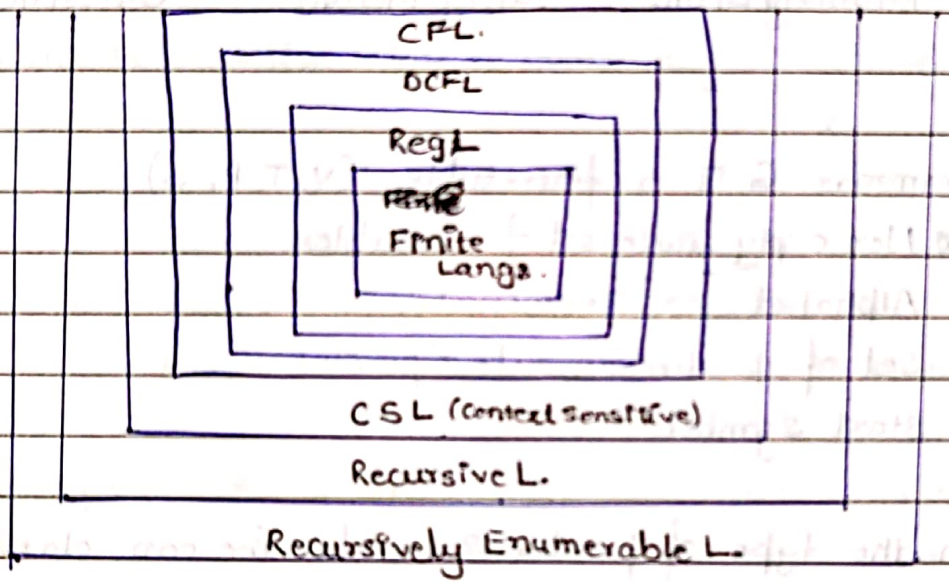
$L_1 = \{ab\}$ $L_2 = \{a^2 b^2\}$ ... $L_K = \{a^K b^K\}$ {All Reg}

$L = L_1 \cup L_2 \cup \ldots$

$L = \{a^n b^n \mid n \geq 1\}$ & $L \notin$ Reg.

Chomsky Hierarchy

- Let, L_R : Some arbitrary Reg. language
  R : Set of all Reg languages.
  L_C, C : CFL
  L_{DC}, DC : DCFL
  L_{NR}, NR : Non-Regular.

- 

So,

Recursively Enumerable	(Type 0)
Recursive Languages	
Context Sensitive	(Type 1)
Context Free	(Type 2)
Deterministic Context Free	
Regular	(Type 3)
Finite	

- For each type we have respective machine & grammar.

Type	Name	Machine	Grammar
3	Regular	DFA or NFA	Regular
2	Context Free	PDA	Context Free
1	Context Sensitive	Linear-Bounded Auto.	Context Sensitive
0	Recursively Enum.	Turing Machine.	Unrestricted.

- Any grammar G is a four-tuple (V, T, P, S)
  - V: Non-empty finite set of variables.
  - T: Alphabet or Terminals.
  - P: set of production rule (finite)
  - S: Start symbol.

Based on the type of production rules we can classify the grammars.

- Regular Grammar: Either all prods are of form:

$$V \rightarrow VT^* \mid T^* \quad \text{(LLG)}$$

or all prods are of form

$$V \rightarrow T^*V \mid T^* \quad \text{(RLG)}$$

- Context-Free Grammar: All prods are of the form

$$V \rightarrow (V+T)^*$$

- Unrestricted (Type 0) Grammar: All prods are of form:

$$(V+T)^* V (V+T)^* \longrightarrow (V+T)^*$$

(i.e. atleast one variable on left).

- Context Sensitive (Type 1) Grammar: All prods are of form

$$u \longrightarrow y$$

① $|u| \leq |y|$

② Start Symbol: S; $S \rightarrow \epsilon$ is allowed. $|u| = 1 < |v| = 0$ here. provided S doesn't appear on RHS of any prod.

(Funfact: ~~All~~ Most natural lang. like Hindi, English, etc. are CSL)

Ex. $S \rightarrow$ 'boy'V | 'boys'V

'boy'V $\rightarrow$ 'boy' is

'boys'V $\rightarrow$ 'boys' are.

So depending on context V is replaced.

- For:

DCFL —— LR(1) Grammar —— DPDA.

Recursive L. —— — —— Halting TM.

- For all Type-0, 1, 2, 3 languages, they are closed under:

① Union ③ Kleene Star

② Concatenation ④ Kleene Plus.

As can be shown using their grammar.

* Context Sensitive language's set is closed under:

(1) Union
(2) Intersection
(3) Complement
(4) Concatenation
(5) Kleene Closure
(6) Reversal.
(7) Intersection w/ Reg. Languages.

* Operations	Reg	DCFL	CFL	CSL	Rec.	REL
Union	✓		✓	✓	✓	✓
Intersec.	✓			✓	✓	✓
Complement	✓	✓		✓	✓	
Set Difference	✓			✓	✓	
Intersec. with Reg. L.	✓	✓	✓	✓	✓	✓
Kleene Star	✓		✓	✓	✓	✓
Kleene Plus	✓		✓	✓	✓	✓
Concatenation	✓		✓	✓	✓	✓
Reversal	✓		✓	✓	✓	✓

Ex. CFL's are closed under union. DCFL's aren't. Comment.

DCFL's are also CFL, so for $L_{DC1} \cup L_{DC2}$ it is guarenteed that it will be CFL, but may/maynot be DCFL.

Ex. For any lang. L, consider the Op^n Repeat.

$$\text{Repeat}(L) = \{ww \mid w \in L\}$$

Which of the following is/are closed under repeat?

Set of all

(A) Reg. Lang.
(B) DCFL
(C) CFL
(D) ~~CSL~~ Finite
~~(E) RECURSIVE~~
~~(F) REL~~

Say $L = \{a, ba\}$ $\quad$ $\text{Rep.}(L) = \{aa, baba\}$.

A. Not closed. Say $L = \Sigma^*$ $\quad$ $\text{Rep}(L) = \{ww \mid w \in \Sigma^*\}$.

B. Not Closed.

C. Not Closed.

D. Closed. If $|L| = n$ then $|\text{Rep}(N)| = n$ for $\exists n \in \mathbb{N}$.

Ex. For any lang L over Σ, ~~Eq-Con(L) = {w}~~

$$\text{Eq-Con}(L, M) = \{w \mid \exists x \in L, y \in M, |x| = |y|, w = xy\}$$

i.e concatenate strings of same length.

Ex. $L = \{a, bb\}$ $\quad$ $M = \{c, dbe\}$

$$\text{Eq-C}(L, M) = \{ac\}$$

Reg isn't closed. Counter ex. $L = \{a^n \mid n \geq 1\}$ $\quad$ $M = \{b^n, n \geq 1\}$

$$\text{Eq-C}(L, M) = \{a^n b^n \mid n \geq 1\}.$$

DCFL & CFL aren't either. Counter ex:

$L = \{a^n b^n \mid n \geq 0\}$ $\quad$ $M = \{b^n c^n \mid n \geq 0\}$

$$\text{Eq.C}(L, M) = \{a^n b^{2n} c^n \mid n \geq 0\}$$

$\therefore$ Only set of all finite lang ~~set~~ is closed under Eq-C.

* [CSL] Every lang. whose strings can be found by you is CSL.

Ex. $L = \{a^n b^n c^n a^n b^n c^n \mid n \geq 0\}$

Can you find it's strings?

$L = \{\epsilon, abcabc, aabbccaabbcc, \ldots\}$

So this is CSL.

Ex. $L = \{a^p \mid p \text{ is prime}\}$

$L = \{a^2, a^3, a^5, a^7, \ldots\}$

* No standart lang class (Reg, CFL, DCFL, CSL, Rec., RE) are closed under:

(1) Subset

(2) Superset

(3) Infinite union

(4) Infinite intersection.

Ex. Show Reg isn't closed under above op^n.

Subset: let $L = \Sigma^*$ Subset of L includes $\{a^p \mid p \text{ prime}\}$

Not even CSL

Superset: let $L = \{a\}$. Superset $\{a^p \mid p \text{ prime}\}$

Inf. Union: $L_1 = \{abc\}$ $L_2 = \{aabbcc\}$ $L_K = \{a^K b^K c^K\}$

$\bigcup_{i=1}^{\infty} L_i = \{a^n b^n c^n \mid n \geq 1\}$

Not even CFL.

Inf. Int: $L_1 = L_2 = L_K = \{a^p \mid p \text{ prime}\}$

$\bigcap_{i=1}^{\infty} L_i = \{a^p \mid p \text{ prime}\}$.

Ex. Comment on set of all Finite Lang for above opⁿ.

Subset : Closed

Superset : Not closed. $L = \{a^2\}$ $\quad$ $Sup(L) = \{a^p \mid p \text{ prime}\}$

Inf Union : $L_1 = \{a\}$ $L_2 = \{a^2\}$ ... $L_k = \{a^k\}$

$$\bigcup_{i=1}^{\infty} L_i = \{a^n \mid n \geq 0\}$$

$\therefore$ Not Closed.

Inf. Intersec : Closed.

Turing Machine, Decidability, Countability

* Turing Machine

- We've seen over some alphabet Σ, Σ^* is the set of all strings over Σ. Any subset L of Σ^*, $L \subseteq \Sigma^*$ is a language. Recognizing a language is equivalent to solving computational problem.

Ex. Say we want to sort n numbers. Let's try to devise some language.

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

Say, Inp: 3, 5, 2 Out: 2, 3, 5

011, 101, 010 010, 011, 101

So our string could be: 011#101#010 ## 010#011#101

So, lang. consists of such strings. ⇒ L

If we now have an automata M which can recognize such strings, then we can solve Sorting problem.

Let's say we have m/c M which recognizes L:

for w = 011#00 ## 00#01

M would reject this string, since for the inp: 11#00, the out: 00#01 isn't the correct sorted order.

Hence M solves the sorting-problem.

(We'll use language & problem interchangabily).

Ex. ① $a^* b^*$, ② $a^n b^n$

We can write algorithm to solve both these problems, but a FA can only solve ①.

① $a^n b^n$ ② ww

We can write algo. to solve both these problems but PDA can only solve ①

So we can say our algo. for 2 is more powerful than PDA computational model.

We can although write C code for ②.

So we can say that C is a more powerful computational model than PDA.

- We want to find mathematical equivalent to C prog. which is powerful enough to solve such computational problems.

- Turing Machine is the mathematical equivalent to a program.

OR

TM is as powerful as a program and vice-versa.

OR

For a problem:

We can create TM $\longleftrightarrow$ We can write program.

- Turing m/c has many def^n in literature. We study Stanford's def^n.
  TM's understanding helps us understand Decidability & Countability, but the def^n of Decidability doesn't depend on def^n of TM.

- Informally,

  TM = FA + ∞ Tape

  - Tape is a data struct. for TM. Not same to Input tape.
  - Just like FA, TM has finite states (finite control).
  - There's a unique initial state, unique final (accept) state and unique reject state.
  - TM can halt only on either Accept state or Reject state. On any other state, the TM can't halt, it keeps running.
  - So if TM never goes to either Accept or Reject state, it will run ∞, i.e., it never halts.
  - Since there is no input tape, you'll write the input strings on Tape. Say w = abab.

..	B	B	a	b	a	b	B	B	..

all other cells are filled with a special symbol, called Blank Symbol (B).

Input Tape	Tape
Contains only input string.	Contains string, but also acts as m/m.
It's read only.	Read / Write.
It's finite.	It's infinite.

FA	TM
① Set of finite states (Q)	① Q.
② $q_0 \in Q$	② $q_0 \in Q$.
③ Σ	③ Σ
④ No additional m/m	④ Tape $\equiv \infty$ m/m
⑤ Input tape is Read Only. Can only go left to right. One cell at a time.	⑤ Tape is R/W. Can go both L→R & R→L.
⑥ $F \subseteq Q$	⑥ $q_f \in Q$, $q_{rej.} \in Q$.

- Formal Def^n: A turing machine is defined as: 8-tuple:

$$M(Q, q_0, \Sigma, \Gamma, B, q_f, q_r, \delta)$$

Q: Finite set of states.
q_0: Initial state. $q_0 \in Q$.
Σ: Input alphabet. $\Sigma \subset \Gamma$
Γ: Tape Alphabet.
B: Blank Symbol. (Not part of Σ). $B \in \Gamma$
q_f: Unique accept state.
q_r: Unique reject state.
δ: Transition func.

- q_f & q_r are stuk (halting) states, i.e., there's no outgoing transitions. As soon as we go to q_f, computation stops and the string is immediately accepts. As soon as we go to q_r, computation stops and the string is immediately rejected.

- Suppose you are in the following current state:

...	B	B	a	b	B	B	...

q

What all can happen given current state & current tape symbol:

1. You can go to some state q'
2. You can write something on the tape.
3. You can move left or right.

So, transition func. is defined as:

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}.$$

Also, except for q_f and q_r, for all remaining states we define transition on every tape symbol.

(This def^n of TM is deterministic).

So, we can also say $\delta : (Q - \{q_f, q_r\}) \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$

Ex. How would you put $w = \epsilon$ on tape?

(A)

..				..

(B)

...	ϵ	ϵ	ϵ	..

✓(C)

..	B	B	B	..

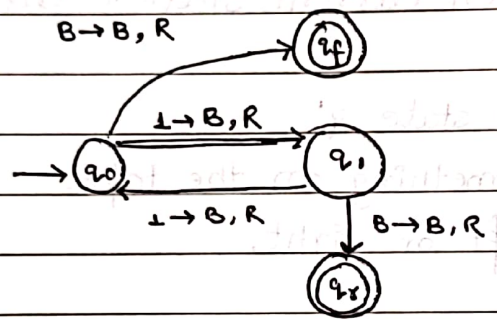
(D)

..	B	ϵ	B	...

Ex. $\delta(q_0, a) = (q_1, d, L)$

This means, when on state q_0, if you read a on ~~input~~ tape, ~~we~~ then you'll go to q_1, replace a with d on the tape and move left on tape.

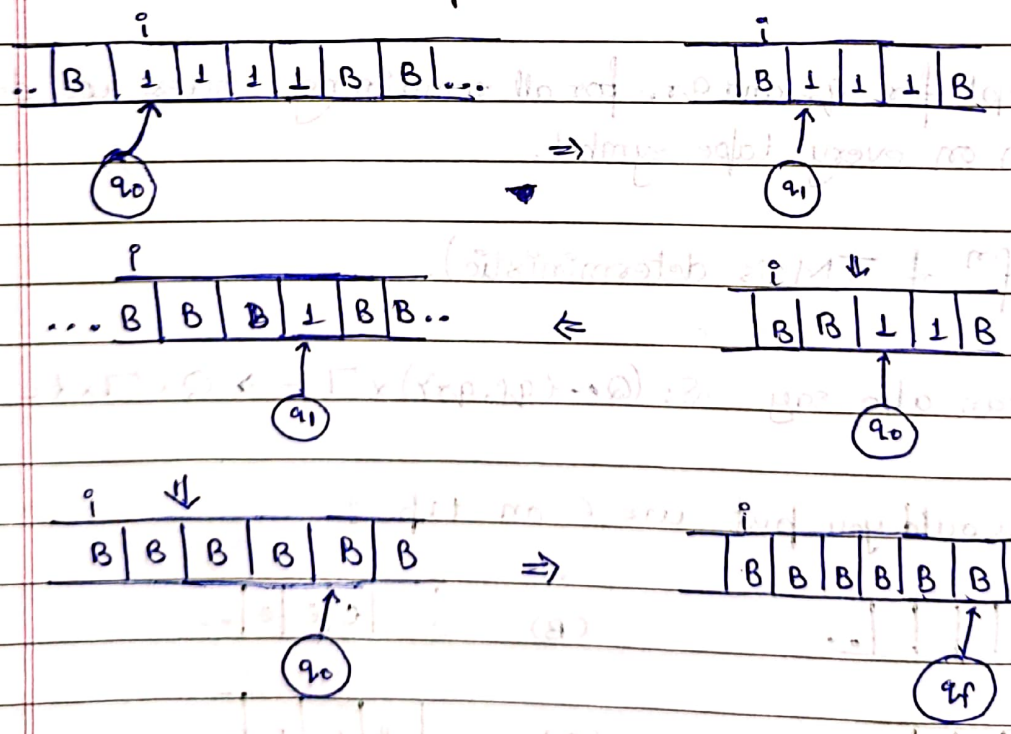
Ex.



$\Gamma = \{1, B\}$

$\Sigma = \{1\}$

Is "1111" accepted?



So, we are at q_f, so immediately computation halts and we accept $w =$ "1111".

- While writing FA's config we only care about what on the right, but in TM we may go to left or right. So, we'll write the following config

..	B	a	B	b	a	B ..

q_0

aBq_0ba {and q_0 is pointing to its rightnext sym}

Ex. Previous example but with just configs.

$$q_0 1111 \xrightarrow{1} q_1 111 \xrightarrow{1} q_0 11 \xrightarrow{1} q_1 1 \xrightarrow{1} q_0 \xrightarrow{B} q_f$$

Computation halts immediately. 1111 accepted.

Ex. Previous TM with $w = \epsilon$

Tape:

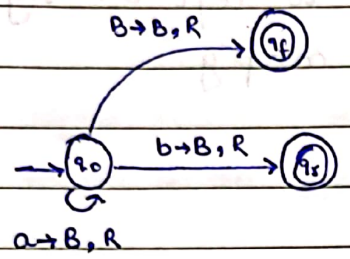
..	B	B	B	B	..

$$q_0 \xrightarrow{B} q_f$$

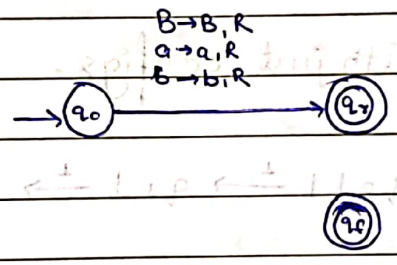
Ex. Previous TM with $w = B$.
Invalid. $B \notin \Sigma^*$.

Note: As soon as your TM goes to q_f or q_r, it immediately accepts or rejects respectively. The string may/may not be completely read. If not on q_f and not on q_r, the machine keeps processing, i.e. doesn't halt.

Ex. Design TM. $L = a^*$ $\Sigma = \{a, b\}$

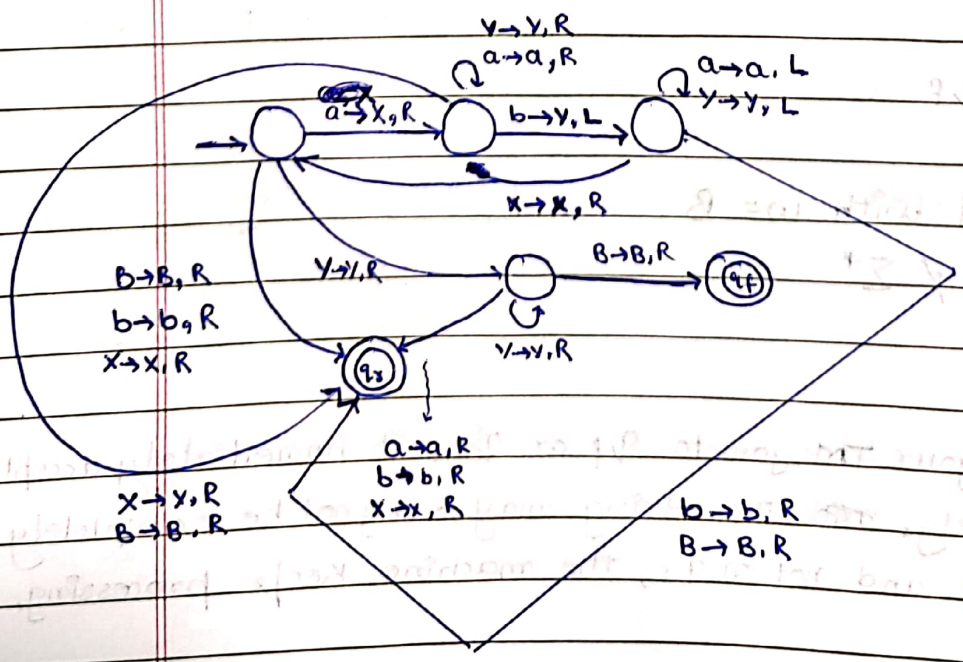


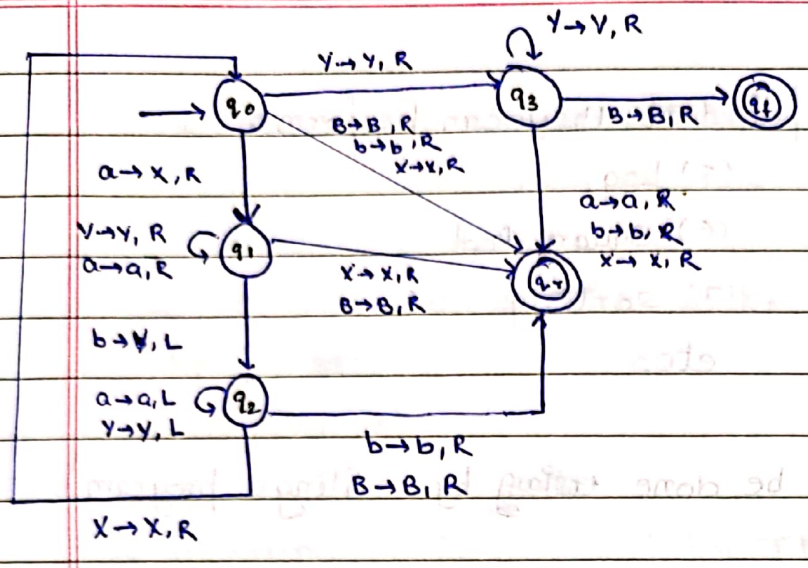
Ex. $L = \phi$ $\Sigma = \{a, b\}$



Here, no matter what's on the tape, we just read first symbol & goto reject state, and stop processing.

Ex. $L = \{a^n b^n \mid n \geq 1\}$ $\Sigma = \{a, b\}$ $\Gamma = \{X, Y, B, a, b\}$





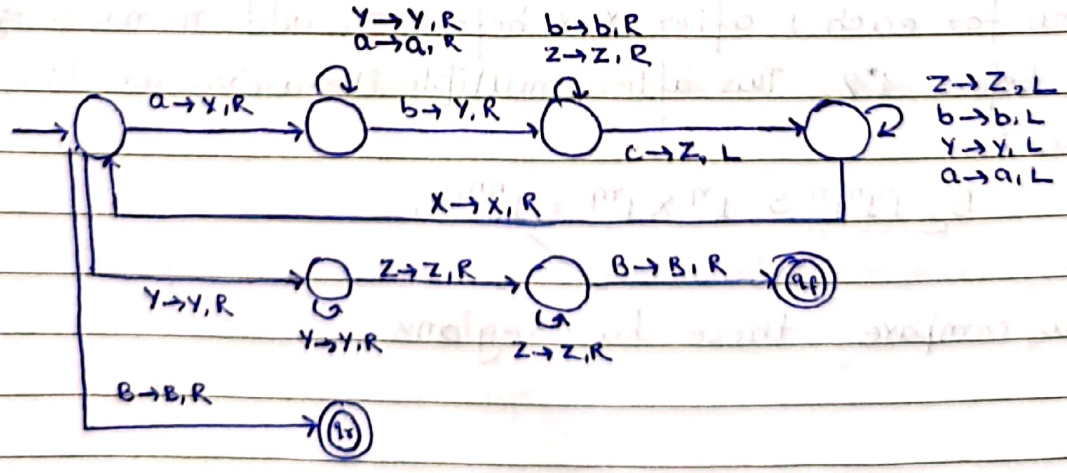
Let's say $w = aba$

$q_0 aba \longrightarrow Xq_1ba \longrightarrow q_2XYa \longrightarrow Xq_0Ya \longrightarrow XYq_3a \longrightarrow XYaq_r$

$\therefore$ string w rejected.

Ex. $L = \{a^n b^n c^n \mid n > 0\}$

Same idea as above. $\Gamma = \{X, Y, Z, a, b, c, B\}$



All remaining transitions go to q_r.

- TM are so powerful that they can perform:

(1) Addition
(2) Subtraction
(3) Multiplication
(4) Division
(5) Log
(6) Square Root
(7) sorting
etc.

(Anything that can be done ~~using~~ by writing a program).

Ex. For multiplication think how you will recognize using TM.

$$L = \{1^n \times 1^m = 1^{nm} \mid n, m \geq 0\}$$

First check if structure is correct.
We can mark a scratchpad area on Tape.

...	S	...1^n...	x	...1^m	=	..1^{nm}..

↑ scratchpad.

Now for each 1 after X & before =, add n no. of ~~symbol~~ 1s before 1*s. This after multiple iterations would look like:

$$B \underbrace{..1^n 1^n 1^n}\ S\ 1^n \times 1^m = \underbrace{1^{nm}}\ B$$

Now compare these two regions.

- We know that configuration of a TM looks like:

$$YX\, q\, _\, MN$$

The Halting configuration looks like:

$XY..\, q_p..\, MN$ (Accepting Halt) or $XY..\, q_r...\, MN$ (Rejecting Halt)

So a string w is accepted by TM iff

$$q_0 w \xrightarrow{*} \alpha q_f \beta$$

(init config) (Accepting Halt)

$\alpha, \beta \in \Gamma^*$

- Recursively Enumerable Language: Any lang. L over some Σ is REL iff there exists a TM M such that $L(M) = L$

- "Every Reg. is RE language".

Proof:

① For any Reg. lang. L, create DFA. Convert DFA to TM, by only going Right. And adding trans. for Blank symbol.

② Every LLG & RLG is Unrestricted Grammar (Type 0).

- "Every CFL is RE language".

Proof:

① Use tape as stack by restrictions.

② Every CFG is UG.

* Some Open Problems (Conjecture)

- Conjecture : Neither proven true nor disproven Mathematical statement.

① P = NP
② Hodge Conjecture
③ Riemann Hypothesis
④ Yang - Mills existence & mass gap,
etc.

- Goldbag's Conjecture : "Every even > 2 is the sum of two primes."

- 3x + 1 Conjecture : Take any $n \in N$, $n \geq 1$.

$n = 1 \longrightarrow$ stop

n is even $\longrightarrow n/2$

n is odd $\longrightarrow 3n+1$

Conjecture says " For any n, it will reach 1".

* Computability

- Intuitive idea of algorithm : Automated procedure i.e doesn't need to change as per i/p. Write once use for all inps. Finishes in finite amount of time. Has finite steps. Is unambiguous.

- Functions have clear mathematical def^n: A func. $f: A \to B$ is a mapping from set A to set B, such that

$$\forall a \in A \ \exists! \ b \in B \ \ni f(a) = b.$$

Ex. Say we define a func Isprime (f) as

$$f: \mathbb{N} \to \{Yes, No\}$$

$$\forall n \in \mathbb{N} \ f(n) = \begin{cases} Yes, & n \text{ is prime} \\ No, & \text{else.} \end{cases}$$

This is just the def^n of func. not the procedure to calculate it.

- Computability means, given x can we find $f(x)$ through some mechanism, maybe some algo, or calculator or washing machine, or anything.
Just that there's an automated procedure which for all possible x in domain gives output.

Ex. For previous IsPrime, we have an algo.:

① Find $\sqrt{n}$

② for all values 2 to $\lfloor \sqrt{n} \rfloor$

check if $n \% i == 0$

then not prime.

③ else prime.

This works for all n in domain.

So, this is Computable.

Ex. Define f: set of all mathm. statement $\rightarrow \{T, F\}$

Is this defined precisely?
Yes. Even for Conjectures like Goldbag's conjecture although we don't know the answer it will either be T or False.
Not neither, not both.
So yes the def^n is precise.
But there's no Algo. (automated procedure) which maps all statements to T or False
So, this is uncomputable.

Ex. $\Sigma = \{0, 1\}$
Say we define $L = \begin{cases} 0, & x \text{ is PM in 2050} \\ 1, & x \text{ isn't PM in 2050.} \end{cases}$

Is it regular?
Yes. Either way L is finite. L is either $\{0\}$ or $\{1\}$. So it is regular.

The func. f: set of all Math stmt $\rightarrow \{T, F\}$ is uncomputable not just because we don't have an automated procedure to compute it "today", but there would never exist such a procedure 10, 100, 100^{100} years from now.
(we'll prove)

- Church-Turing Hypothesis (Belief): Anything that can be done using any automated mechanical procedure, you can also do using Turing Machine.

So, since we don't have def^n of automated procedure, we've taken TM as the def^n. or model.

- So, we can now define "Computable".

Any func. or problem or language that can be calculated or solved or recognized by some Turing Machine is "Computable".

Any model of computation that has come since TM, is either equivalently powerful to TM (or strictly weaker than it.)

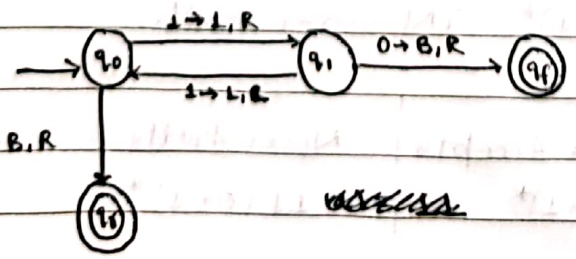
* Halting & Looping of TM.

- We know that as soon as the TM transitions to q_f or q_r, the computation halts immediately (irrespective of the fact that string is completely read or not) and the string is either accepted or rejected.

- On any other state the computation can't halt since transitions are defined on all tape symbols.

- So TM runs forever (infinite loop) if halting states: q_f, q_r aren't entered.

Ex.



$\Gamma = \{0, 1, B\}$

(All other trans. go to q_r).

$w = 10$

$q_0 10 \rightarrow 1q_1 0 \rightarrow 1Bq_f$ So accept.

$w = 01$

$q_0 01 \longrightarrow B q_r 1$ $\therefore$ Rejected.

$w = 00$

$q_0 00 \longrightarrow q_r 0$ $\therefore$ Rejected.

$w = 11$

$q_0 11 \longrightarrow 1 q_1 1 \longrightarrow q_0 11 \longrightarrow 1 q_1 1 \ldots$

So, the m/c never halts.

For all strs: $0(0+1)^*$ go to q_r after 1 transition, halts and rejects.

For 1: $q_0 1 \rightarrow 1 q_1 \rightarrow 1 B q_r$ $\therefore$ Rejects in 2 trans.

For all strs: $10(0+1)^*$ go to q_f after 2 of transitions & accepts.

For 11:

$\underline{q_0 11} \longrightarrow 1 q_1 1 \longrightarrow \underline{q_0 11}$

State repeats and since our TM is deterministic we know that transitions further would also repeat. So, the TM never halts.

So for $11(0+1)^*$, TM never halts.

So,

Halts & Rejects	Halts & Accepts	Never halts
$0(0+1)^*$, 1, ϵ	$10(0+1)^*$	$11(0+1)^*$

$\therefore L(M) = 10(0+1)^*$

Also,

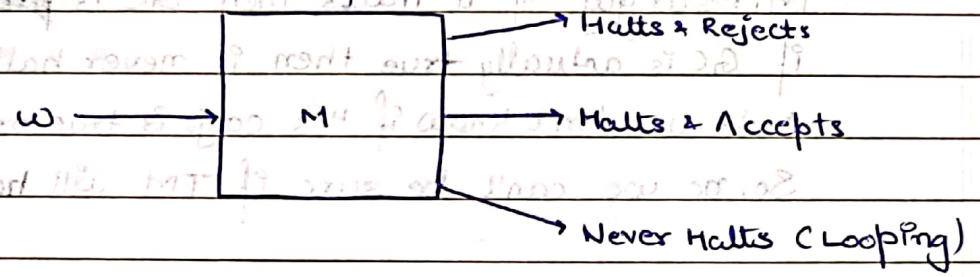
Initial Config: $q_0 w$

Rejecting Config: $u q_r v$

Accepting Config: $u q_f v$.

Halting Config: $u q_r v$, $u q_f v$.

- So on a string w the TM can do one of three things:



① $L(M) = \{ w \mid M \text{ halts \& accepts on } w \}$

② M doesn't accept means either "Halts & Rejects" or "Loops".

③ M doesn't reject means either "Halts & Accepts" or "Loops".

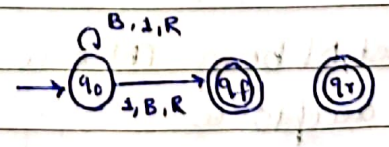
④ M doesn't loops means either "Halts & Accepts" or "Halts & Reject".

- Loop / Never Halting in a TM can be due to various reasons:

① Config repeats: $C_1 \longrightarrow C_2 \longrightarrow C_3 \ldots C_K$

② Keep moving right or Keep moving left:

Ex.

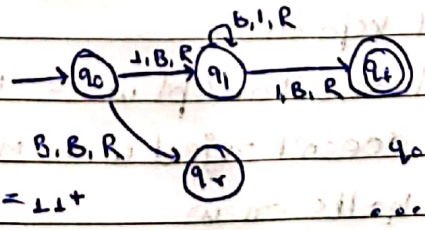


$L = 1^+$

on $w = \epsilon$

$q_0 \rightarrow 1q_0 \rightarrow 11q_0$ Config doesn't repeat.

Ex.



$L = 11^+$

on $w = 1$

$q_0 1 \rightarrow q_1 \rightarrow 1q_1 \rightarrow 11q_1$

$\ldots 1111q_1 \leftarrow 1111q_2$

Config doesn't repeat.

Note: For past few examples we could observe the ~~loop~~ looping, but we can not always observe looping.

- Say a TM running for 5×10^{11} years, can we be sure it'll halt or not? No.
- Say we create a TM to check Goldbach Conjecture. Can we be sure the TM will halt?
  Intrestingly, if it halts then GC is ~~proven~~ false, but if GC is actually true then it never halts.
  And we don't know if this conj. is true or not.
  So, no we can't be sure if TM will halt.

- For a w a TM M:

$w \in L(M)$	$w \notin L(M)$
$w \longrightarrow \boxed{M} \longrightarrow$ Halts & Accepts	$w \longrightarrow \boxed{M} \longrightarrow$ Halts & Rej / Never halts

- Hence Recursively Enumerable languages are also called Turing Recognizable or Recognizable or Turing Acceptable or Acceptable language.

Ex. Let M be a TM & $w \notin L(M)$, then which are/is not possible?

(A) M accepts w ✓

(B) M rejects w

(C) M loops on w

(D) M doesn't accept w $\equiv$ M halts & rej + M loops.

(E) M doesn't reject w $\equiv$ M halts & acc + M loops

(F) M halts on w

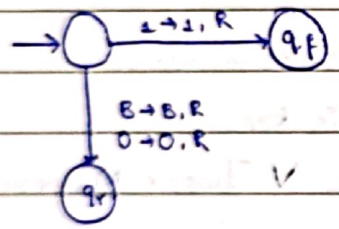
Ex. Previous. ~~B~~ $w \in L(M)$.

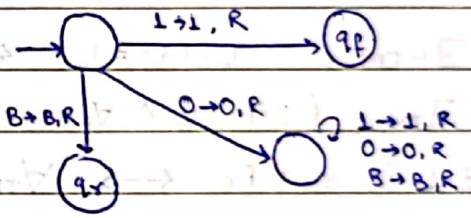
B, C, D,

* Halting TM: A TM which for all strings w, halts.

Ex. $L = 1(0+1)^*$

$\Gamma = \{0, 1, B\}$.





Halting TM
(Halts after only one trans. for all strings)

Doesn't halt : $0(0+1)^*$

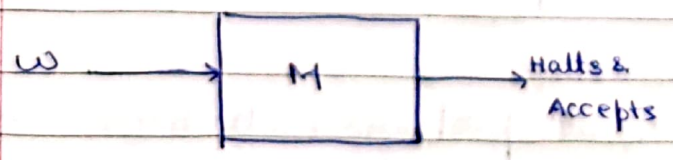
Halts & Rej. : ϵ

Halts & Acc. : $1(0+1)^*$

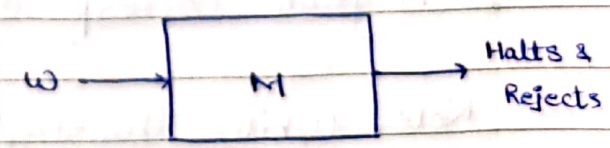
So we can say, for $1(0+1)^*$ there exists a HTM.

- HTM ≡ Decider ≡ Total TM.
- So, for any string w, a HTM

$w \in L(M)$



$w \notin L(M)$



- Every HTM is TM, but not every TM is HTM.

- **Recursive Language:** Any lang. L over some Σ, is called Rec. iff there exists some HTM (or Decider) which recognizes the language.

- Recursive Language $\equiv$ Decidable $\equiv$ Turing Decidable.

- So, every Decidable lang. is Recognizable, but not vice-versa.

- L is a lang. over Σ.

① $\exists_{HTM\,M}\ L(M) = L \longleftrightarrow$ L is Recursive lang.

$\equiv \exists_{TM\,M}\ (L(M) = L \wedge \forall_{w \in \Sigma^*}\ M \text{ Halts}) \leftrightarrow$ L is Rec.

② $\equiv$ L is not Rec. $\longleftrightarrow \forall_{TM\,M}\ (L(M) \neq L \vee \exists_{w \in \Sigma^*}\ M \text{ doesn't halt on } w)$

- So a lang L is not Rec (Decidable) iff either no TM exists for L or TM exists by it doesn't halt on all strings.

Note: Sometimes Rec. Enumerable lang. are called Semi-Decidable.

- "Algorithm" is similar to HTM, since it should always terminate in some finite amount of time for all inputs.

- "Program" corresponds to "TM".

Note: While thinking about problems we'll hence use "Algorithm" & "Program" instead of "HTM" & "TM".

Ex. $L = \{ wwwwwww \mid w \in \{0,1\}^* \}$. Is L decidable?

Yes. We can write an algo $\equiv \exists$ some HTM.

① Find length
② Check if div by 5.
③ Divide in 5 parts
④ Compare the parts.

This is algo, since for all strings it terminates in finite amount of time.

∴ Decidable.

Ex. $L = \{ a^{n!} \mid n \geq 0 \}$

Yes. For any str: a^K, $K = n!$ must be div by $1, 2, 3 \ldots n$
So, just check starting from 1 if K is div by all values upto some finite n.

For some nonmember a^x also, there would be some $i \in [2, x]$ which won't divide x first. So we can terminate in finite time.

* There are two types of ~~loops~~ infinite loops:
① Repeating Configuration Loop: This type of loop doesn't cause undecidability

$$q_0 \rightarrow q_1 \rightarrow q_2 \ldots q_K$$

② Continuous Meaningful Work: This type of loop may cause undecidability.

Ex.
```
n=0;
while (true){
    print n;
    n++;
}
```

Can you detect if it will run infinitely or not?
(Assume no m/m bounds)
(Repeated Config loop)
(Never causes undec.)

Yes. We can tell that this will never (under no circumstance) stop.

Ex. Give an example of a problem for which some algo. runs for long time but would guarenteed stop.
(Cont. meaningful work. Not causing undecidability)
Graph Isomorphism on 64 vertices.
We know it will definitely stop after very long time.

Ex. Which type out of above two is "Checking Goldbach Conjecture"?
Neither. First type is where TM runs infinitely long i.e. we know it will never stop.
Second type is where TM, even if runs for a very very long time, it can be guarenteed to stop.

Checking Goldbach conjecture, we can't be sure if it will run for upto $n = 10$, or $n = 10^{10}$ or $n = 10^{10^{10}}$, we don't know if will stop or not. We can't say with guarentee that it halts, and we can't guarentee that it won't halt.

(Cont. Meaningful work. Causing undecidability)

So, can we say that there's no HTM for Goldbach Conj.?
No.
Can we say there is?
No.
We can't guarentee anything.

- Two special type of algorithms:

(1) Always Yes Algorithm: Blindly says Yes
Ex. Given $n > 100$, is $n > 50$?
Since any inp: that we give to the algo. is > 100, the answer is always "Yes".

(2) Always No Algorithm: Blindly says No.
Ex. Given $n > 100$, is $n < 50$?
Again, any inp. will be > 100, so answer is always "No".

Both these types of algo. are decidable.

Ex. Checking Conjecture vs Conjecture.

Any one Conjecture by itself is decidable, since algorithm exists.
It is either "Always Yes" or "Always No" algo.
Say Goldbach Conj.: Every even no. > 2 is sum of two primes.
What would be possible answers:
① For all $n > 2$, Yes
② For all $n > 2$, No.
So one of the two algos work for GConj.

i.e. Some HTM exists to answer the statement for all inputs.

But, Checking the conjecture isn't decidable. The TM may halt or it may never halt & keep checking. Although, if it halts the answer to the check would be No.

So, "Is Goldbach Conjecture true" is undecidable, by GConj: "For all $n > 2$, n is sum of two primes" is decidable.

* Variants of TM

- Our TM :
  Deterministic, No Null moves, No dead transitions, except q_f & q_r. Special accept & reject states

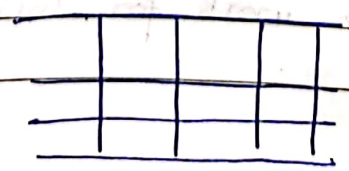
- TM itself is the most powerful, giving it any additional power won't make it more powerful.

- Variant 1 : + Ability to stay, along with move L or R.

- Variant 2 : + Multiple Tape (each w/ R/W head separate)
  $\delta : Q \times (\Gamma_1 \times \Gamma_2 \times \Gamma_3 \times .. \Gamma_n) \longrightarrow Q \times (\Gamma_1 \times \Gamma_2 \times \Gamma_3 .. \times \Gamma_n) \times \{L, R\}^n$

  Var 2 $\equiv$ DTM (Deterministic TM)

- Variant 3 : Single Tape Multitrack (single R/W head)



Var 3 ≡ DTM

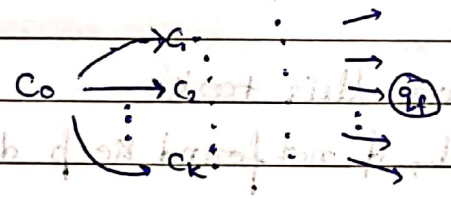
- Non-Deterministic: From a single configs, more than one move is possible.

OR

On a state it is possible to ~~make~~ have more than one transition ~~using~~ for a symbol, i.e,

$$\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma}$$

Acceptance: Starting from initial config, there exists a path which goes to q_f.



$C_0 : q_0 w$

Rejection: No path from initial config. goes to accept state & all paths go to q_{reject}.

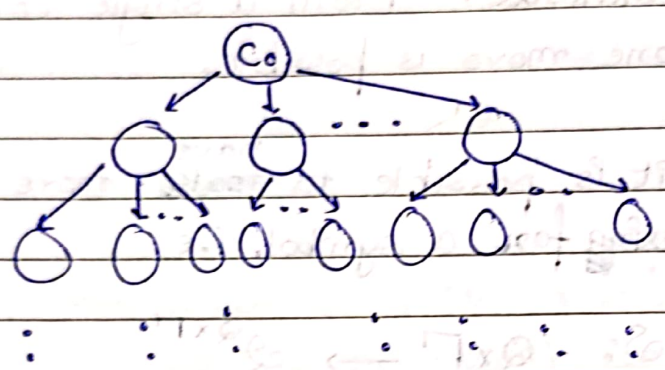
Looping: No path goes to q_{accept}, and at least one path never halts.

To analyze NTM we use tree computation method.

We already have DTM → NTM by def^n. We want to further show that every NTM is also DTM.

i.e. NTM ≡ DTM.

Proof Idea: Suppose NTM has following config path starting from initial config C_0.



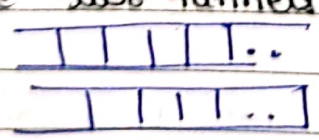
DTM performs BFS (not DFS since it may lead to inf. loop due to repeated config or cont. useful work)

Ex. Let's suppose you have this task:
"Find finite tunnel, if not found keep digging"

Scenario 1: You have single tunnel.

No matter finite or inf, you have no option but to dig this tunnel.

S2: You have two tunnels. One is finite one inf.



Say you go with previous strategy & dig one tunnel at a time. What if you dig ∞ tunnel first? This way you'd never find finite tunnel.

So, even though there's a finite tunnel you can't guarentee it will be found. (This is analogous to DFS)

Let's try doing this: Dig 1 hr T_1, then 1 hr T_2, then T_1, then T_2, and so on (interleaving). Known that there's a finite tunnel this guarentees it will be found. (BFS)

S3: Now say there are ∞ no. of tunnels.

First strategy definetly doesn't guarentee finding finite tunnel.

Even second strategy fails: Dig 1hr T_1 1hr T_2, T_3, T_4... Since ∞ no. of tunnels you won't even get back to T_1 ever.

Let's adopt new strategy:

1. Dig 1 hr T_1.
2. Dig 2hr T_1, 2hr T_2
3. Dig 3hr T_1, 3hrs T_2, 3hr T_3

⋮

T_1	1hr	2hr	3hr	4hr	5hr	...
T_2		2hr	3hr	4hr	5hr	
T_3			3hr	4hr	5hr	
T_4				4hr	5hr	
T_5					5hr	
⋮						

If there is a finite tunnel, this is guarenteed to find it.

This interleaving method is called "Dovetailing".

Ex. Suppose given a sequence of ∞ jobs, you can do a J_i at t_i.

This means we can complete all jobs, since for all ∞ jobs we have answer to "when did you complete J_i?" "At t_i".
When did you complete J_{2100}? at t_{2100}.

- S1: Suppose you have two seq. of ∞ jobs.
  The first strategy doesn't work.

  We can alternate the two sequence. $J_{11}, J_{21}, J_{12}, J_{22}, J_{13}, J_{23}, \ldots$
  Again we will complete every job, since we can report time slamp for all of them.

- S3: Suppose you have ∞ sequence of ∞ jobs.
  We can complete every job by Dovetailing.

J_{11}	J_{12}	J_{13}	J_{14}	J_{15}
J_{21}	J_{22}	J_{23}	J_{24}	J_{25}
J_{31}	J_{32}	J_{33}	J_{34}	J_{35}
J_{41}	J_{42}	J_{43}	J_{44}	J_{45}
J_{51}	J_{52}	J_{53}	J_{54}	J_{55}

OR

J_{11} ; $J_{12}\ J_{21}$; $J_{13}\ J_{22}\ J_{31}$; $J_{14}\ J_{23}\ J_{32}\ J_{41}$; ...

* Decidability

- For a problem P, iff we have an algorithm A, then we have a HTM. Problem P would hence be called decidable.

- Algorithm ≡ HTM ≡ Decider
- Program ≡ TM

- Some decidable problem:
  - Given a list, is it sorted? < Yes / No
  - Given a no., is it perfect square?
  - Given a no. is it prime?
  - Given a string, is is palindrome?
  - Given two graph, are they isomorphic?

Ex. Which of the following lang. are decidable?

(a) Σ^* (b) ϕ (c) $a(a+b)^*$

(d) $\{a^n b^n \mid n \geq 0\}$ (e) $\{www^R w \mid w \in \{a,b\}^*\}$

(f) $\{a^p \mid p \text{ is prime}\}$

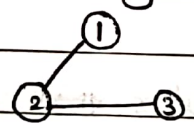
All are decidable since we have algorithms for all of them.

- Encoding: Every finite object can be encoded using any ~~binary~~ alphabet of choice.
  Ex. Computer encodes in binary. Huffman encoding. Movie encodings, etc.

- Why do we need encoding?
  To convert finite objects to strings so as to pass them to TM.

- Can we encode π ?
  No. π is infinite object. (In computers we approx. the value of π to some decimal places).

Ex. Try encoding the following graph using $\{0,1\}$ alphabet



No. of vert ## $\underbrace{V_1V_2 \# V_2V_3}_{\text{edges}}$ ##

Every val. is 4 bit. # = 0000

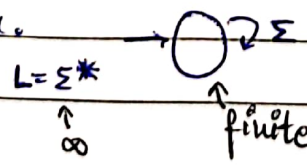
So, 0011 0000 0000 0001 0010 0000 0010 0011 0000 0000

- If we can encode, then there exists an algorithm to decode.
- Finite objects in TOC: DFA, NFA, Any Grammar, PDA, RE, TM, ~~etc~~, string, alphabet

Inf. objects in TOC: Language

- Although a language (set of strings) may/may not be finite, it can be finitely represented using some model, such as DFA, PDA, grammar, etc.

Ex.



- Finite representation of class of languages:

① Finite Lang: language itself or DFA or NFA or RE
② Regular Lang: DFA, RE, NFA, LLG, RLG.
So, the stmt: "Given a Regular Lang. L" means, "Given a DFA/NFA/RE/RG of L".
③ CFL: PDA, CFG.
④ Recursive Lang: HTM
⑤ CSL: CSG, LBA
⑥ REL: UG, TM.

So the stmt: "Given a lang. L" means "some finite representation of L is given".

Ex. How will you tell a computer ~~that~~ about Natural no.?
One way is to tell the set {1, 2, 3..} but this isn't finite repr.
We can tell: 1 is Natural No. and
if n is Natural No, then n+1 is Natural no.

- For the following interconversions we have some algorithm

①

$$\begin{array}{ccccc} DFA & \longleftrightarrow & NFA & \longleftrightarrow & RE \\ & & \updownarrow & & \\ & & RG & & \end{array}$$

② CFG $\longleftrightarrow$ PDA
③ CSG $\longleftrightarrow$ LBA
④ UG $\longleftrightarrow$ TM

- From now on, we'll use the following convention:
  "Given a finite object M"
  $\langle M \rangle$ is the encoding of M over some alphabet, using some encoding scheme. (we don't care about the alphabet or the encoding scheme.)

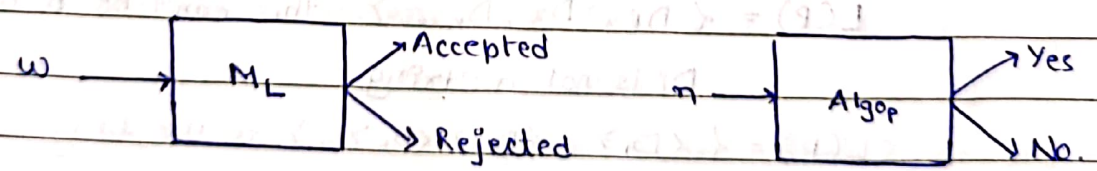
- **Language Vs Problem:** Every language is a "Decision Problem" and Every "Decision Problem" is a language.

- **Decision Problem:** A Yes/No problem.

Ex. "Given a list, is it sorted"? Decision Problem
"Given a list, sort it". Not.

Ex. $L = \{a^2, a^3, a^5, \ldots\} = \{a^p \mid p \text{ is prime}\}$

P = Given a number is it prime.



So, Accepted by M_L corresponds to Yes by $Algo_P$.
Rejected by M_L —"— No by $Algo_P$.

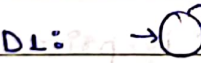
So, solving P corresponds to recognize L.

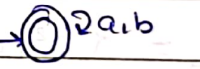
Ex. P: Given a string over $\{a,b\}$ is it in ww form?
$L = \{ww \mid w \in \{a,b\}^*\}$.

- So, for input I, if problem output is Yes then
  $I \in L$

- For a problem P:
  Domain: Set of all possibe inputs
  Instance: A particular input
  Yes Instance: Instance for which output is Yes.
  So
  $L(P) = \{ I \mid I \text{ is Yes Instance} \}$

Ex. "Given a DFA D, is L(D) finite?"
So, this is decision Problem.
Domain: Set of all possible DFAs.
L(P) = Set of all yes instances.

D_1: 

D_2: 

So, $D_1 \in L(P)$. $D_2 \notin L(P)$.

$L(P) = \{ D_1, D_3, D_K \ldots \}$ this can't be a lang.
D_i is not a string.

- $L(P) = \{ \langle D_1 \rangle, \langle D_3 \rangle, \langle D_K \rangle \ldots \}$ is the lang. of P.

$L(P) = \{ \langle D \rangle \mid D \text{ is DFA} \wedge L(D) \text{ is finite} \}$

- So, solving Problem P for instance I is same as accepting/rejecting $\langle I \rangle$ for L(P).

Ex. Find the corresponding Language :

① Given a DFA, does it accept a given string? (P_1)

$L_1 = \{ \langle D, w \rangle \mid D \text{ is DFA } \& \ w \text{ is word } \& \ w \in L(D) \}$

② Given a DFA does it accept any word? (P_2)

$L_2 = \{ \langle D \rangle \mid D \text{ is DFA } \& \ L(D) \neq \phi \}$

③ Given two DFAs, do they accept same language? (P_3)

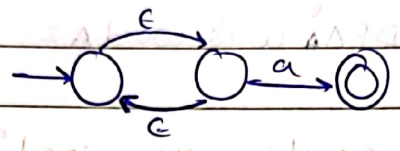
$L_3 = \{ \langle D_1, D_2 \rangle \mid D_1 \& D_2 \text{ are DFA } \& \ L(D_1) = L(D_2) \}$

Ex. Which of the problems above are decidable?

- P_1 is decidable. Algo takes $\Theta(n+1)$ time.
  So, L_1 is Recursive language.
- P_3 is decidable. Minimize D_1 & D_2 => D_{1m} D_{2m}. Then its just graph isomorphism (restricted, since we can start from start state & compare transitions)
- P_2 is decidable. Given DFA D:
  ① Minimize D.
  ② Compare it with DFA for $L = \phi$, i.e. $(\{q_0\}, \Sigma, q_0, \{\}, q_0 \xrightarrow{\Sigma} q_0)$
  $\rightarrow (q_0) \circlearrowleft \Sigma$ $\quad Q \quad \Sigma, q_0, F \quad \delta$
  ③ If isomorphic, return false.
  Else return true.

- The following machine can go to ∞ loop (No halt):
NFA, PDA, DPDA, LBA

Ex. NFA



But the only type of looping possible in these are of "Repeated Config" type.
Hence the loops in above machines can be algorithmically detected.

Loop in TM may/maynot be due to Repeated Config, so it can't be algorithmically detected.

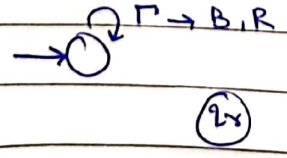
- DFA & HTM ~~[illegible]~~ are two m/c which always halt.

Ex. Let M is TM. M' is all similar to M except it's q_r & q_f states are swapped. Then which of the following is/are possible?

(A) $L(M) = L(M')$
(B) $L(M) \neq L(M')$
(C) $L(M) = \overline{L(M')}$

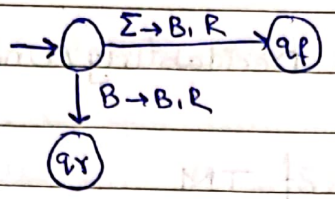
All three are possible.

for (A)



$L(M) = \phi$
$L(M') = \phi$

for B, C



$L(M) = \Sigma^* - \{\epsilon\}$ $\qquad$ $L(M') = \{\epsilon\}$

$= \Sigma^+$

One thing we can guarentee for a TM M if we swap M's q_f & q_r, and get M'.

$$L(M') \subseteq \overline{L(M)}$$

A is possible when all strings loop.
B is possible when some string loop, i.e., some don't loop.
C is possible when no string loops.

Ex. Which encoding helps recognize $L = \{2^n \mid n \geq 0\}$ by a DFA?

(a) Binary ✓
(b) Unary

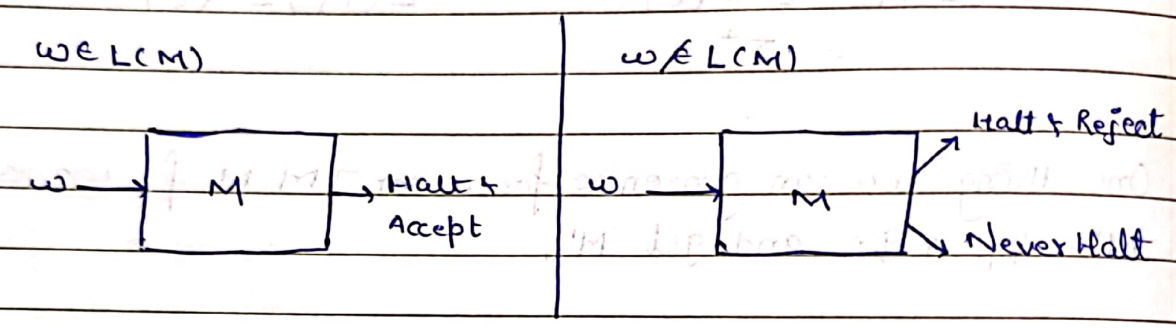
In binary 2^n is 10^n So, $L = 10^*$

Ex. $L = \{\langle D \rangle \mid D \text{ is DFA } \& \ L(D) = \phi\}$

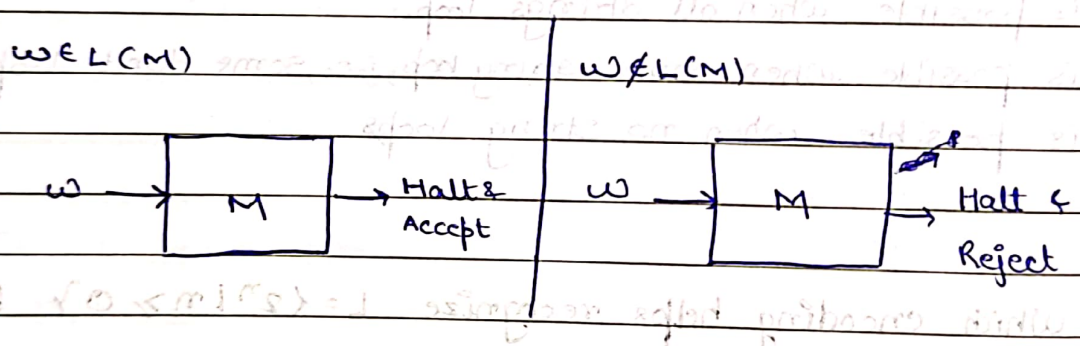
- D is a DFA
- $\langle D \rangle$ is encoding (string)
- L is set of strings (encodings)
- L is a language containing encoding of all such DFA's D, whose lang. L(D) is empty.

* Important Idea behind Decidability/Undecidability

- We know the behaviour of TM



- We know behaviour of HTM



- For a decision problem P, we first define its corresponding language L(P) containing all "Yes Instances" of P.

Ex. P: Given an int, determine if it is div. by 10.

$L = \{ \langle I \rangle \mid I \text{ is int } \& \ I \% 10 = 0 \}$

↑ encoding of I

Ex. P: Given two numbers x & y, are they coprime?

$L = \{ \langle X, Y \rangle \mid X \& Y \text{ are numbers}, X \& Y \text{ are coprime} \}$

- Next, we check decidability:
  - If for a lang. L, you can write an algorithm which definitely halts for all strings (member and non-member) then lang. L is decidable.
  - If you can write some procedure for L, which definitely halts for all members but may/maynot halt for non-members, then L isn't decidable but L is recognizable.
  - If for L, you can't even guarentee to hall for members then L isn't even recognizable.

Ex. For a TM M, if it is known that $w \in L(M)$ then can we verify this in finite amount of time?

Yes. Since $w \in L(M)$, M will definitely halt on w.

Ex. Same as above, $w \notin L(M)$

No. Since $w \notin L(M)$, M may never halt.

Ex. For a TM M, if it is known that out of two strings u & w, at least one belongs to L(M). Can you verify in finite amount of time?

Procedure 1:
- Verify for u
- Verify for w

This doesn't work. What if 'u' loops forever. We might not be able to detect the looping.

Procedure 2: (Dovetailing)

$n = 1$

Repeat:

Run M on u for n more steps.

Run M on w for n more steps

n++

This way for at least one of them halts. We can guarentee it.

Ex. For TM M, out of two strings, u & w, at least one doesn't belong to L(M). Can you verify this in finite amount of time?

No. This can't be verified in finite amount of time. Even with dovetailing, both may not halt.

Ex. For TM M... exactly one of them belongs to L(M). Can you...?

No. Suppose $u \in L(M)$ & $w \notin L(M)$, then it may happen that M on w never halts. We can't detect this.

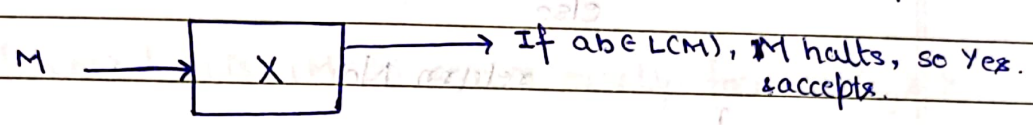
Ex. For TM M... atmost one of them belongs to L(M). Can you...?

No. If one of them loops forever we can't stop prematurely.

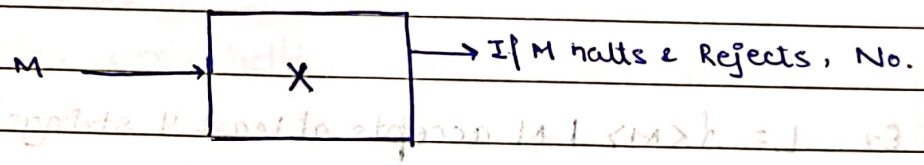
Note: The previous idea of checking Dec./Undec./Recog./Unrecog. of problems related to language of TM.

Ex. $L = \{ \langle M \rangle \mid M \text{ is TM } \& \text{ M accepts } ab \}$. Is L Decidable?
↳ (say we use TM X, $L(X) = L$)

No. For a given M, if $ab \in L(M)$, it will def. halt.



For a given M, if $ab \notin L(M)$, it may halt & rej or it may not halt. So,



So, X can guarentee that for any $\langle M \rangle \in L(X)$, X definitely halts, but for a non-member, $\langle M \rangle \notin L(X)$, X may halt if M halts & rejects on 'ab' or X may not halt if M doesn't halt.

∴ Recognizable, but not decidable.

Algo: Checker (M) {

```
    out = Run M on 'ab';   ← This may not halt for non-member M.
    if out Yes:
        return Yes;
    else
        return No;
}
```

Ex. $L = \{\langle M \rangle \mid M \text{ is TM}, ab \notin L(M)\}$

Algo : Checker (M) {

out = Run (M, ab); ← can't guarentee that it will halt for member M

if out == No:

return Yes;

else

return No;

}

∴ Not even Recognizable.

Ex. $L = \{\langle M \rangle \mid M \text{ accepts at least 4 strings}\}$

(Again a problem related to lang. of TM)

Procedure :

```
Checker (M) {
    S = Set of all possible strings over M.Σ
    n = 1 ; count = 0;
    for s = S[i] , i ∈ [1,n] {
    state = run (M, s)
    if state == M.qf :
        count ++;
    if count == 4 :
        break;
    }
    return "Accept M"
}
```

For members, the procedure would definitely halt.
For non-members, we can't say if it is non-member or it's just taking long to accept 4 strings.

∴ Recognizable but not decidable.

Ex. $L = \{ \langle M \rangle \mid M \text{ is TM, } M \text{ accepts exactly four strings} \}$

for member M, we can't stop at clucking at least 4. We'll have to check every string, which may / maynot halt.
Dovetailing won't help either.

So, unrecognizable.

Ex. $L = \{ \langle M \rangle \mid M \text{ is TM, } |L(M)| \leq 4 \}$

Unrecognizable.

* Decision Problems on Regular Languages

Note: ① "Given Regular Lang." means "Given some finite repr. of Reg. L."

② DFA $\longleftrightarrow$ NFA $\longleftrightarrow$ RE

$\updownarrow$

RG

- Membership / Acceptance Problem: "Given a language L, string w; does w belong to L?"

For Reg Lang: "Given a DFA/NFA/RE/RG, string w; does w belong to DFA/NFA/RE/RG?"

This Problem can be stated for any class of Language.

Ex. $L = \{ \langle D, w \rangle \mid D \text{ is DFA}, w \text{ string}; w \in L(D) \}$

Is L Decidable?

Yes.

Algo: Checker (D, w) {

Out = Run (D, w);

return out;

}

T.C. = $\Theta(|w|)$

$\therefore$ L is decidable.

Ex. $L = \{\langle M, w\rangle \mid M \text{ is NFA}, w \text{ is string}; w \in L(M)\}$

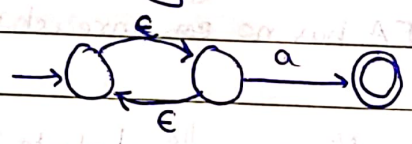
Decidable.

Algo:

```
Checker (M, w) {
    D = convert-NFA-to-DFA (M);
    out = Run (D, w);
    return out;
}
```

Without converting we can't run & guarentee halt.

Ex.



If NFA didn't have null moves, conversion wouldn't be needed.

Ex. $L = \{\langle R, w\rangle \mid R \text{ is RE}, w \text{ string}; R \text{ generates } w\}$

RE $\longrightarrow$ NFA $\longrightarrow$ DFA

Then run DFA on w.

This always halts.

$\therefore$ L is decidable.

Ex. $L = \{\langle G, w\rangle \mid G \text{ is RG}, w \text{ is string}; w \in L(G)\}$

RG $\rightarrow$ NFA $\rightarrow$ DFA

Run DFA on w.

This always halts

$\therefore$ L is decidable lang.

- Emptiness Problem: "Given a lang. L, is L empty i.e $L = \phi$"

Ex. Which of the following is true?

(A) If for a DFA, no final states then empty lang.

(B) If for a DFA, if empty lang then no final states.

(C) If for mDFA, if empty lang then no final states.

(D) If in DFA, there is reachable final state the lang. is non-empty.

ACD are true.

B is false, since state might be unreachable.

C is true since mDFA has no unreachable state.

So lang. is non-empty iff some final state is reachable in DFA/NFA

We can check this by BFS/DFS with q_0 as source.

Ex. $L = \{\langle M \rangle \mid M \text{ is FA}, L(M) = \phi\}$

Decidable

Algo:
```
Checker(M){
    Reachable = DFS(M, q0);
    return (Reachable ∩ M.F == ϕ)
}
```

Ex. [Obviously No]

$L = \{\langle M \rangle \mid M \text{ is FA}, L(M) = \{a^n b^n \mid n \geq 0\}\}$

Decidable.

$\{a^n b^n \mid n \geq 0\}$ is not RL.

$\therefore$ For any FA M, return False/Rejected.

Ex. $L = \{ \langle N \rangle \mid \text{NFA } N,\ L(N) \text{ is Regular} \}$

Decidable

Algo: Always Yes

- Equivalence Problem: "Given two languages L_1, L_2; are they equal".

Ex. $L = \{ \langle D_1, D_2 \rangle \mid D_1, D_2 \text{ are FA},\ L(D_1) = L(D_2) \}$

Algo:

- Convert both to DFA.
- Create product automata for $D_1 \Delta D_2$ (symm. diff / XOR)
- Check D_{12} for emptiness.

- We can say that the language of a DFA D is finite if on any path from initial state to any of the final state, there is no loop.

(basically, there should be no cycle)

- Finiteness Problem: "Given a language, is it finite"

Ex. $L = \{ \langle D \rangle \mid D \text{ is DFA},\ L(D) \text{ is finite} \}$

Algo:

Check if there's any walk from initial state to any final state w/ cycle.

Decidable.

- **Containment Problem:** Given two languages, L_1, L_2; is L_1 subset of L_2.

Ex. $L = \{\langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are FA}, L(M_1) \subseteq L(M_2)\}$

If $L(M_1) \subseteq L(M_2)$ then

$$[L(M_1) \cap L(M_2)] \Delta L(M_1)$$

should be ϕ.

We can easily check this using Product Construction.

$\therefore$ Decidable.

Alternate: $L(M_1) \subseteq L(M_2)$ iff $L(M_1) - L(M_2) = \phi$.

- **Disjointness Problem:** Given two lang. L_1, L_2; is $L_1 \cap L_2 = \phi$.

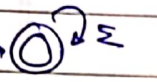
Ex. $L = \{\langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are FA}, L(M_1) \cap L(M_2) = \phi\}$

- Create Product Automata for $L(M_1) \cap L(M_2)$
- Check for ϕ emptiness.

$\therefore$ Decidable.

- Universality Problem: Given lang. L, is $L = \Sigma^*$.
(Totality)

Ex. $L = \{ \langle D \rangle \mid D \text{ is FA}, L(D) = \Sigma^* \}$

Algo 1: Minimize. Check isomorphism with: 
A2: Complement DFA. Check Emptiness.

$\therefore$ Decidable.

- Regularity Problem: Given language L, is L regular?

Ex. $L = \{ \langle D \rangle \mid D \text{ is FA}, L(D) \text{ is regular} \}$

Algo: Always Yes.

$\therefore$ Decidable.

Ex. P: Given a regular lang., is its complement regular.

Decidable.
Algo: Always Yes.

Ex. P: Given a reg. lang. is its complement non-regular.
Decidable.
Algo: Always No.

Ex. P: Given FAs, M_1 & M_2, over some Σ, is there any string $w \in \Sigma^*$ that is accepted by atleast one of M_1 & M_2.

$L = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are FA}, \exists w \in \Sigma^* (w \in L(M_1) \vee w \in (M_2)) \}$

Algo:

- Find Prod. Automata for $M_1 \cup M_2 = M$
- Check if $L(M) = \phi$
- If $L(M) = \phi$, return Reject
- Else return Accept.

$\therefore$ Decidable.

Ex. P: Given a DFA M, string $u, v \in \Sigma^*$, does $L(M)$ contain all strings that can be formed by concatenating u & v as often as desired in any order?

$L = \{ M \mid M \text{ is DFA}, u \text{ \& } v \in \Sigma^*, L((u+v)^*) \subseteq L(M) \}$

Algo:

- Convert RE $(u+v)^*$ to NFA to DFA, say M_R
- Create prod. Automata for $L(M_R) - L(M) = L(M')$
- Check $L(M') = \phi$
- If yes, return Accept
- Else reject.

$\therefore$ Decidable.

* Context-Free Lang. & Grammars :

- LMD & RMD for CFG. give the order of derivation rules used.

Ex. $S \rightarrow \overset{①}{E}E$

$E \rightarrow \underset{②}{ab} \mid \underset{③}{a}$

LMD : $S \xrightarrow{①} EE \xrightarrow{②} abE \xrightarrow{③} aba$

RMD : $S \xrightarrow{①} EE \xrightarrow{③} Ea \xrightarrow{②} aba.$

- CFLs are mostly recursively defined.

Ex. Set of palindromes over $\{0,1\}$.

Base : $\epsilon, 0, 1$ are pal.

Recursion : If w is pal.

$0w0$

$1w1$

are also pal.

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

Ex. Given $T = \{0, 1, (,), +, *, \phi, e\}$ (where e is 'ϵ' to avoid confusion). Design CFG for all Regular Expressions over $\{0,1\}$.

concat

Base Case: $0, 1, \phi, e$ are RE

Recursion : If r & s are RE

$r.s$, $r+s$, r^*, r^+, (r) are also RE

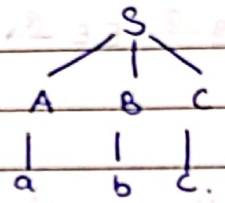
$S \rightarrow S+S \mid S^* \mid S^+ \mid (S) \mid B \mid S.S$

$B \rightarrow 0 \mid 1 \mid \phi \mid e \mid \epsilon$

- The major difference b/w Derivation & Parse Tree is that Derivations tell both the productions used and the order of their use. In PT only the productions are visible, not their order.

- For a given PT, it doesn't tell us the order of derivations so it can be converted to any derivation.

Ex. $S \rightarrow ABC$ ①
$A \rightarrow a$ ②
$B \rightarrow b$ ③
$C \rightarrow c$ ④



Derivations:

① $S \xrightarrow{①} ABC \xrightarrow{②} aBC \xrightarrow{③} abC \xrightarrow{④} abc$ (LMD)

② $S \xrightarrow{①} ABC \xrightarrow{②} aBC \xrightarrow{④} aBc \xrightarrow{③} abc$.

③ $S \xrightarrow{①} ABC \xrightarrow{③} AbC \xrightarrow{②} abC \xrightarrow{④} abc$.

④ $S \xrightarrow{①} ABC \xrightarrow{③} AbC \xrightarrow{④} Abc \xrightarrow{②} abc$.

⑤ $S \xrightarrow{①} ABC \xrightarrow{④} ABc \xrightarrow{②} aBc \xrightarrow{③} abc$.

⑥ $S \xrightarrow{①} ABC \xrightarrow{④} ABc \xrightarrow{③} Abc \xrightarrow{②} abc$. (RMD)

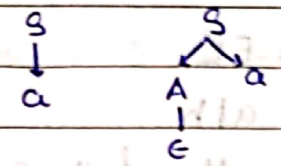
- For a PT although there may be multiple order of derivations, but there will be only one LMD & one RMD.

LMD can be found from PT by traversing in depth-first from left to right.

For RMD, right to left depth-first traversal.

Ex. $S \to \overset{①}{Aa} \mid \overset{②}{a}$

$A \to \overset{②}{aA} \mid \overset{①}{\epsilon}$



Suppose $w = a$.

$S \xrightarrow{②} a$

$S \xrightarrow{①} Aa \xrightarrow{②} a$

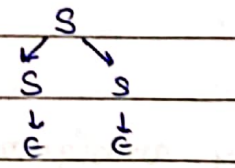
So more than one LMD for w.

$\therefore$ Grammar is ambiguous.

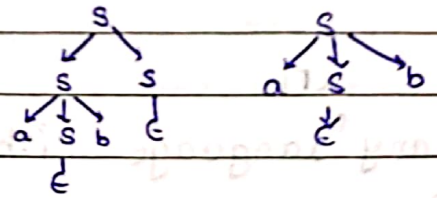
- For any string w on any grammar G, there is a bijection b/w set of all PTs for w & set of all LMDs for w, and bijection b/w set of all PTs for w & set of all RMDs for w.

Ex. $S \to SS \mid aSb \mid \epsilon$ (Balanced paranthesis)

For $w = \epsilon$



For $w = ab$



Grammar is ambiguous.

- Left Sentential Form: LSF, $S \xrightarrow{*} \alpha$ means that from S in some leftmost derivation, α would appear.
- Right Sentential Form: RSF $S \xrightarrow{*} \alpha$ means that from S, in some rightmost derivation, α would appear.

Ex. $S \to EE$

$E \to a \mid b$

which of the following is/are left sentential form:

(A) S (C) aE (E) bb (G) Eb

(B) EE (D) Ea (F) ab (H) bE

Which is/are RSF above?

A, B, D, E, F, G

Ex. Write CFG for RE $R = (0 + (10)^*)^* + 10^*$

$S \to X \mid Y$

$X \to 1 \mid X0$

$Y \to \epsilon \mid WY$

$W \to 0 \mid T$

$T \to \epsilon \mid 10T$

Ex. Is there any CFL language without any ambiguous grammar?

Yes. ϕ is the only language for which there is no ambiguous grammar.

Since for CFG to be ambiguous there has to be atleast one string with multiple PT.

Ex. A CFG G is ambiguous iff there is some string w for which there are at least two derivations.
T/F?

False.
(Either ≥ 2 PT or ≥ 2 LMD or ≥ 2 RMD)

$S \to AB$ D1: $\underline{S} \to \underline{AB} \to a\underline{B} \to ab$

$A \to a$ D2: $\underline{S} \to A\underline{B} \to \underline{A}b \to ab$

$B \to b$

Not ambiguous.

- A CFL is unambiguous iff there exists some unambiguous grammar for it.

So, a CFL is ambiguous / Inherently Ambiguous iff all the grammar that exist for it are ambiguous.

Showing that a language is Inherently Ambiguous is extremely difficult & we can't give proof for it. We can give intuition at most. (although Proof exists)

$L_1 = \{a^m b^n c^p d^q \mid m = p \text{ or } n = q\}$
This was proven by Parikh to be inherently ambiguous.

$L_2 = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$
This was proven by Hopcroft & Ullman to be inherently ambiguous.

Although more inherently ambiguous lang. exist, these two are standard & can be created questions on.

Ex. Can a regular grammar be ambiguous?

Yes. $S \to A \mid a$

$A \to a$.

Ex. Can regular language be ambiguous?

No. For every regular lang., at least one unambiguous grammar exists.

Note: Every DCFL also has at least one unambiguous grammar. So a DCFL is also not inherently ambiguous.

- Simplification of Grammar

- Unit Production: A production of the form $X \to Y$ where both X & Y are variables (or non-terminals)

Ex. $S \to A$ ← This is unit prod.

$A \to a$.

- Null Production: A production of form $X \to \epsilon$ where X is variable.

Ex. $S \to A$

$A \to \epsilon$ ← This is null prod.

- Useless Production

Ex. $S \to a \mid S$

This is useless as it never generates any string

Ex. $S \to aA \mid a$

$A \to aA$

Useless as its not generating any string.

Ex. $S \to a \mid b$

$A \to a$ } Useless as unreachable

- A useful substitution rule.

Ex. $S \to aS \mid Ab$

$A \to ab \mid Sa$.

Here we can substitute A.

$S \to aS \mid abb \mid Sab$

$A \to ab \mid Sa$.

Ex. $S \to aA \mid B$

$A \to b \mid Ba$.

$B \to aA \mid a$.

We can replace B

$S \to aA \mid a$

$A \to b \mid aAa \mid aa$.

$B \to aA \mid a$.

Ex. $S \to aB$

$A \to aaA \mid abBc$

$B \to aA \mid b$

Replacing B in $A \to abBc$

$A \to aaA \mid abaAc \mid abbc$.

Replacing B in $S \to aB$

$S \to aaA \mid ab$

Now we can delete productions of B

$S \to aaA \mid ab$

$A \to aaA \mid abaAc \mid abbc$.

- Two types of useless variables
  (1) Variables which can't produce strings
  Ex. $S \to a \mid aA$
  $A \to aA$ } Useless.

  OR

  (2) Variables which are unreachable from start state.
  Ex. $S \to a \mid b$
  $A \to aA \mid a$ } useless.

These conditions are sufficient but not necessary.

- A variable is useful iff from start variable it is reachable & it generates some string.

$$S \longrightarrow \ldots \to xAy \to \ldots \to w$$

- A production is useless iff any variable in the production $X \to \gamma$, is useless i.e. in $X \cup Y$.

- Steps to remove useless productions: (Inorder)
  ① Remove non-string-generating variables.
  ② Remove variables unreachable from start symbol.

Ex. $S \to aS \mid A \mid C$.
$C \to aCb$
$A \to a$
$B \to aa$.

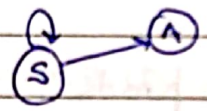
S1: Generating vars: A, B, S (since it goes to A & A is generating var).

So, Non-gen. Vars: C.

We can remove any prod which contains C either on LHS or RHS.

Removed: $S \to C$, $C \to aCb$.

S2: Reachability from S.



So, B is not reachable.

We can remove all prod $X \to \gamma$ where $B \in \{X\} \cup \gamma$

$S \to aS \mid A$

$A \to a$.

Ex. $S \to AB \mid a$

$A \to a$

S1: Gen Vars: A, S

Nongen Vars: B.

$\therefore$ $S \to a$

$A \to a$

S2: Reachability

S $\therefore$ A is Unreachable.

$\therefore S \to a$.

Eg, $S \to AA \mid CD \mid bB$

$A \to aA \mid a$

$B \to bB \mid bC$

$C \to CB$

$D \to dD \mid d$

S1: Gen. vars $\Rightarrow$ A, D, S (since S has prods AA where all vars. on RHS are generating).

So, non-gen vars: B, C.

We can remove all their prods.

$\therefore$ $S \to AA$.

$A \to aA \mid a$.

$D \to dD \mid d$.

S2: Reachability:

$S \to A$

$\therefore$ Only S, A are reachable.

$\therefore$ $S \to AA$

$A \to aA \mid a$.

- Removing of Null Productions: Given a CFG G, such that empty string doesn't belong to L(G), we want to remove all Null Productions. If $\lambda \in L(G)$ we remove all Null prods & after that we add $S \to \lambda$ where S is start symbol, but make sure S isn't on RHS of any production.

- Nullable Variable : A var. A is nullable iff $A \xrightarrow{*} \lambda$.

Ex. $S \to AB$
$A \to \lambda$
$B \to \lambda$

Nullable : A, B, S {because $S \to AB$ leads to $S \xrightarrow{*} \lambda$}

Ex. $S \to ABC$
$A \to B \mid a$.
$B \to C \mid b \mid \lambda$
$C \to AB \mid D$
$D \to Cd$.

Nullable : S1: B
S2: A {∵ $A \to B$}
S3: C {∵ $C \to AB$}
S4: S {∵ $S \to ABC$}
∴ {A, B, C, S} are nullable.

Removing $B \to \lambda$; $A \to \lambda$; $C \to \lambda$; $S \to \lambda$

$S \to ABC \mid AC \mid BC \mid C \mid AB \mid A \mid B$		$S \to ABC \mid AC \mid BC \mid C \mid AB \mid A \mid B \mid \lambda$
$A \to B \mid a$.	Since $\lambda \in L(G) \Rightarrow$	$A \to B \mid a$
$B \to C \mid b$		$B \to C \mid b$
$C \to AB \mid A \mid D \mid B$		$C \to AB \mid A \mid D \mid B$.
$D \to Cd \mid d$		$D \to Cd \mid d$.

Ex. $S \to ABaC$
$A \to BC$
$B \to b \mid \lambda$
$C \to D \mid \lambda$
$D \to d$

Nullable : S1: B, C.
S2: A {$A \to BC$}
∴ {A, B, C},

Removing : $B \to \lambda$	Removing $C \to \lambda$	Removing $A \to \lambda$
$S \to ABaC \mid AaC$	$S \to ABaC \mid AaC \mid ABa \mid Aa$	$S \to ABaC \mid AaC \mid ABa \mid Aa \mid BaC \mid aC \mid Ba \mid a$
$A \to BC \mid C$	$A \to BC \mid C \mid B \mid \lambda$	$A \to BC \mid C \mid B$
$B \to b$	$B \to b$	$B \to b$
$C \to D \mid \lambda$	$C \to D$	$C \to D$
$D \to d$	$D \to d$	$D \to d$

Ex. $S \to AB$ Nullable: A, B, S.

$A \to AaA \mid \epsilon$

$B \to BbB \mid \epsilon$

Removing $A \to \epsilon$:

$S \to AB \mid B$

$A \to AaA \mid aA \mid Aa$

$B \to BbB \mid \epsilon$

Removing $B \to \epsilon$

$S \to AB \mid B \mid A \mid \lambda$

$A \to AaA \mid aA \mid Aa$

$B \to BbB \mid Bb \mid bB$

Not removing $S \to \lambda$ since $\lambda \in L(G)$, & S not on any RHS.

Ex. $S \to SS \mid (S) \mid \epsilon$

Removing $S \to \epsilon$, but S is on RHS, so:

$S' \to S \mid \epsilon$

$S \to SS \mid (S) \mid () \mid S$

- Removal of Unit Production: If

$A \to B \mid B_1 \mid B_2$

$B \to \gamma_1 \mid \gamma_2 \mid \gamma_3$

then $A \to \gamma_1 \mid \gamma_2 \mid \gamma_3 \mid B_2 \mid B_3$

$B \to \gamma_1 \mid \gamma_2 \mid \gamma_3$

- Finally, Simplification of CFG is done in following order

1. Remove λ-productions
2. Remove unit-productions
3. Remove useless-productions.

Ex. $S \rightarrow aSSS \mid bb \mid \lambda$

S1: Removing λ-prods.

$S' \rightarrow S \mid \lambda$

$S \rightarrow aSSS \mid aSS \mid aS \mid a \mid bb$

Ex. $S \rightarrow aA \mid B \mid C$

$A \rightarrow aB \mid \lambda$

$B \rightarrow Aa$

$C \rightarrow cCD$

$D \rightarrow ddd \mid Cd.$

S1: Removing λ-prods

$S \rightarrow aA \mid B \mid C \mid a.$

$A \rightarrow aB$

$B \rightarrow Aa \mid a$

$C \rightarrow cCD$

$D \rightarrow ddd \mid Cd.$

S2: Removing unit-prods

$S \rightarrow aA \mid a \mid Aa \mid cCD$

$A \rightarrow aB$

$B \rightarrow Aa \mid a$

$C \rightarrow cCD$

$D \rightarrow ddd \mid Cd.$

S3: Removing Non-gen = {C}

$S \rightarrow aA \mid a \mid Aa.$

$A \rightarrow aB$

$B \rightarrow Aa \mid a.$

$D \rightarrow ddd.$

S4: Removing Unreachable = {D}.

$S \rightarrow aA \mid a \mid Aa$

$A \rightarrow aB$

$B \rightarrow Aa \mid a.$

Ex. $S \to XaYb$

$X \to aX \mid Y \mid b$

$Y \to Yb \mid aZ \mid X \mid \lambda$

$Z \to baa \mid SS$

S1: Removing λ-prod.

$S \to XaYb \mid aYb \mid Xab \mid ab$

$X \to aX \mid a \mid b$

$Y \to Yb \mid b \mid aZ \mid X$

$Z \to baa \mid SS$

S2: Removing U-Prod.

* Same as above *

S3: Removing Non-gen.

* Same as above *

S4: Removing Unreachable

* Same as above *

Normal Forms

Since CFGs have productions of following form:

$$V \to (V+T)^*$$

This results in both increased power (from Reg.G), but also difficulty in analysis due to unrestricted RHS.

We want a restricted form of CFG, such that we don't lose the power of CFG, but also make it easier to analyze. Such forms are called Normal forms.

Two popular Normal forms are:

(1) Chomsky Normal Form: Each production is of the form: $V \rightarrow VVIT$

(2) Greibach Normal Form: Each production is of the form:

$$V \rightarrow TV^*$$

If $\lambda \in L(G)$, then $S \rightarrow \epsilon$ is added to the grammar, provided the start symbol S doesn't appear on RHS of any prod. for the two Normal Forms.

- Chomsky Normal Form: For any CFL L, there is a grammar G, such that $L(G) = L$ & each rule in G is of the form:
  - $V \rightarrow T$: V is var, T is terminal
  - $V \rightarrow V_1 V_2$: V, V_1, V_2 are vars, neither V_1, nor V_2 are start symbol
  - If $\lambda \in L$, $S \rightarrow \epsilon$, where S is the start symbol

Furthermore, G has no useless symbols.

Ex. Convert to CNF:

$S \rightarrow ABa$

$A \rightarrow aab$

$B \rightarrow Ac$

Note: Before converting to CNF, simplify the grammar.

S1: Add following prod to grammar:

$V_a \rightarrow a$, $V_b \rightarrow b$, $V_c \rightarrow c$.

S2: Replace every terminal with non-term. above:

$S \rightarrow ABV_a$; $A \rightarrow V_a V_a V_b$; $B \rightarrow A V_c$

S3: $\{S \rightarrow AS_1 ; S_1 \rightarrow BV_a ; A \rightarrow V_a S_2 ; S_2 \rightarrow V_a V_b ; B \rightarrow AV_c ;$
$V_a \rightarrow a ; V_b \rightarrow b ; V_c \rightarrow c\}$ This is in CNF.

Note: A CNF may/maynot be ambiguous.

Ex. Convert to CNF

$S \to A \mid a$

$A \to a.$

Sol: Simplification.

① No null prod.

② $S \to A$ is unit prod. So,

$S \to a$

$A \to a.$

③ No non-gen. symbol. A is unreachable. So,

$S \to a$

$\therefore$ $G: \{S \to a\}$

This is already CNF.

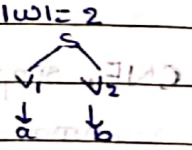
- The importance of CNF is that derivation of any string w, $|w| = n$, irrespective of order of derivation, is of length exactly $2n-1$. $(w \in L(G))$ $(w \neq \epsilon)$

Ex. Given a CFG is in CNF, what is the no. of nodes in parse tree of w, $|w| = n$? $(w \neq \epsilon;\ w \in L(G))$

let: $|w| = 1$

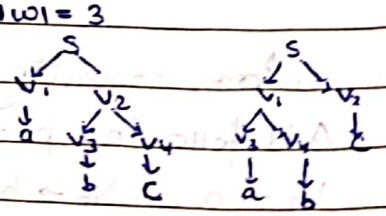
nodes = 2

$|w| = 2$



nodes = 5

$|w| = 3$



nodes = 8

$|w| = 4$

nodes = 11

So, # nodes $= 3n - 1$

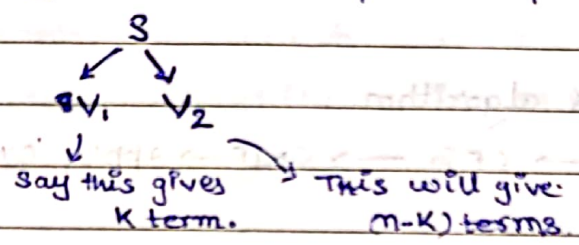
Proof: (By Induction)

Base Case: $|w| = 1$ #nodes $= 2 = (3 \times 1 - 1 = 2)$ $\}$ $|w|=2$ $\#n=5$

For $|w| = n$: #nodes $= 3n-1$

For $|w| = n+1$: #nodes $= (3n-1) + ?$

Let's say, $T(n)$ = no. of nodes for $|w| = n$, $n \geq 3$



$\therefore T(n) = T(K) + T(n-K) + \underbrace{1}_{\text{root nodes}}$

So, for $|w| = n+1$, say $K = n$.

$\cancel{T(n) =}$

$T(n+1) = T(K=n) + T(1) + 1$

$T(n+1) = (3n-1) + 2 + 1$

$= 3n+3-1$

$\Rightarrow 3(n+1) - 1$

$\therefore$ Our assumption $T(n) = 3(n) - 1$ is correct.

- In GNF, the length of derivation of any string w, $|w| = n$, irrespective of order of derivation, is $n-1$ (since in each step you get one terminal). No. of nodes in PT would be $2n$.

$$\left\{ \begin{array}{l} T(n) = 2 + T(n-1), \; n \geq 2 \\ 2, \; n = 1 \end{array} \right\}$$

- CYK Parser: Takes two input: • CFG in CNF
  • input string w, $|w| = n$

  Time Complexity: $O(n^3)$

Checks if $w \in L(G)$.

* Decision Problems of CFLs & DCFLs.

Ex. $L_{Mem} = \{\langle M, w\rangle \mid M$ is some PDA, w is some string, $w \in L(M)\}$
{Membership Problem}

Decidable.

Proof 1: CYK algorithm.

$M \longrightarrow CFG \longrightarrow CNF \rightarrow$ Apply CYK algo. $\rightarrow$ Yes / No

Proof 2: $M \rightarrow CFG \rightarrow CNF$

Now brute-force check all the derivations of len $= |w|$ & compare with w.

Ex. $L = \{\langle M, w\rangle \mid M$ is some DPDA, $w \in \Sigma^*$, $w \in L(M)\}$
{Membership problem}

Since every DPDA is also PDA, hence decidable.

- If a property P is decidable for PDA, then it is also decidable for DPDA, since every DPDA is also PDA.

Hence property P:

Decidable for PDA $\longrightarrow$ Decidable for DPDA.

& Undecidable for DPDA $\longrightarrow$ Undecidable for PDA.

Ex. $L = \{\langle M\rangle \mid M$ is PDA, $L(M) = \phi\}$ {Emptiness Problem}

Algo: Convert to CFG. Simplify the CFG. For start symbol S,

S is useless $\longleftrightarrow$ L(G) is empty

Since S is always reachable, it is useless only when it is non-generating.

$\therefore$ Decidable.

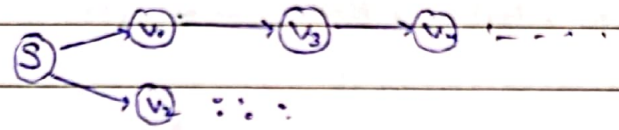
Ex. $L = \{\langle M \rangle \mid M \text{ is PDA}, L(M) \text{ is finite}\}$

{Finiteness Problem}

Decidable.

Algo:

- PDA → CFG → CNF. In CNF, there is no useless prod. or variable. So every variable is generating, and reachable.
- Make reachability graph starting from S.



- If there is loop in the graph, then language is infinite else finite.

(By the same algo, infiniteness problem is also decidable)

- Undecidable Problems for CFLs (complicated proofs)
  - Disjointness Problem
  - Equivalence Problem
  - Universality Problem
  - Containment Problem
  - Regularity Problem (Decidable for DCFG) DCFGs
  - Ambiguity Problem for CFG (Decidable for ~~DCFLs~~)
  - Inherently Ambiguous (for CFL) Problem (Decidable for DCFLs)
  - ~~Totality Problem (Dec. for DCFLs)~~

- Decidable for DCFLs: Equivalance Problem, Universality / Totality Prob, Regularity Prob, Ambiguity Problem for DCFGs, Inherent Ambiguity Problem for DCFLs.

Ex. $L = \{ \langle M \rangle \mid M \text{ is DPDA}, \overline{L(M)} \text{ is DCFL} \}$

Decidable

Algo: Always Yes. (Closure Property)

Ex. $L = \{ \langle M \rangle \mid M \text{ is PDA}, \overline{L(M)} \text{ is CFL} \}$

Undecidable.

Algo is complicated.

*

- For some TM M, if we know $w \in L(M)$ we can verify it by running M on w. M will surely halt & accept M.

- If we know that $w \notin L(M)$, we can't verify it. Running M on w may lead to M halting & rejecting or it may lead to M never halting.

- Undecidability is caused by this never halting behaviour of TM.

* Universal TM

- We know that TM can be encoded as string. Also TM takes string as input. So a TM can be given as input string to some TM.

- [Membership Problem of TM] Given a TM M, and a string w, does $w \in L(M)$?

$L_{mem} = \{ \langle M, w \rangle \mid M \text{ is TM}, w \text{ is string}, w \in L(M) \}$

This is recognizable, not decidable.

Proof:

Given a M & w:

if $w \in L(M)$, M will halt & accept. We can accept $\langle M, w \rangle$

if $w \notin L(M)$, M may halt & reject, or may never halt.

So, for member string $\langle M, w \rangle$ our checker always halts & accepts.

For non-members our checker may/maynot halt.

$\therefore$ Our checker algo is recognizable, not decidable.

- Universal TM is just a TM which can take a TM as input and simulate its behavior.

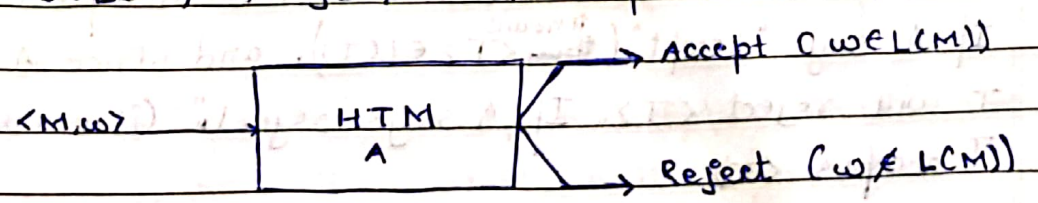
(Think of it as a C function which takes a func. pointer as input & run it on input.

```
void utm ( bool (*tm) (char* inp)) {
        bool res = tm(inp);
}
```

)

- Alternate Proof: "Membership Problem for TM is undecidable."

Proof: Assume there is a HTM, A, which accepts $\langle M, w \rangle$ if $w \in L(M)$ & rejects $\langle M, w \rangle$ if $w \notin L(M)$.



Now suppose to A, we want to give a specific TM, called T. T as input takes encoding of some TM, <P>. It then creates two copies of P, <P,P>. T treats the first P as a TM & second copy as string. Now T simulates the TM P, by giving it the input <P>.
If P accepts <P>, then TM T outputs reject. If P rejects <P>, then T outputs accept.

```
bool T ( bool (*P)(char*)) {
        char* input = stringify(P);
        bool out = A(P, input);
        return !out;
}
```

So, T on input <P> does the opposite of what P does on <P>.

~~Now suppose, to A as the (M,w) pair we give (T,<T>)~~
T runs <P> on P using the HTM A. Since A is HTM, it always halts. Basically, T when given input <P>, creates its copy, and passes one copy as M & other as w to A. A then simulates M (P in this case) on w (<P> in this case). If <P> ∈ L(P), then A halts and accepts P, else A halts & rejects. When A accepts, T rejects & vice versa.

Now suppose to T, as <P> we give <T>. T will create two copies of it & pass one as M & other as w to A. A now simulates M (T) on w(<T>). A is HTM, so it definetly halts. If A says "Accept" (it means <T> ∈ L(T)), and since A said accept, T will reject <T>. If A says "reject" (it means <T> ∉ L(T)), then T accepts <T>.

This is contradiction.

∴ Such a decider A doesn't exist.

- Halting Problem of TM: "Given a TM M, and a string w, does M halt on w?"

$L = \{ \langle M, w \rangle \mid M \text{ is TM}, w \text{ is string}, M \text{ halts on } w \}$

This is Recognizable but not decidable.

Algo: Checker (M, w) {
out = M(w);
return "Accept"
}

If M does halt on w, our checker can halt & accept. But if M doesn't halt on w, we can't say if it'll never halt or just taking too long to halt.

Proof: Assume there exists some HTM, say H, for the halting problem. It takes $\langle M, w \rangle$ input & halts & accept if M halts on w, halts & rejects when M loops on w.

Now create a TM T (just as for Membership Problem). T takes some TM P as input. It copies P. It gives one copy to H as P & and other as $\langle P \rangle$ for $\langle M, w \rangle$ pair. If $\langle P, \langle P \rangle \rangle \in L(H)$ it means P halts on $\langle P \rangle$, and if $\langle P, \langle P \rangle \rangle \notin L(H)$, it means P doesn't halt on $\langle P \rangle$.

If H halts & accepts, T intentionally loops. If H halts & rejects, T halts.

Now, give T as input to T. If H accepts $\langle T, \langle T \rangle \rangle$ (it means T halted on $\langle T \rangle$), then T wouldn't halt. If H rejects $\langle T, \langle T \rangle \rangle$ (it means T doesn't halt on $\langle T \rangle$), then T halts on $\langle T \rangle$. Hence contradiction.

∴ Such a decider (HTM) H doesn't exist.

"There exists a not RE / not Recognizable Language".

Proof:

Say we have a diagonalization language as:

$L_D = \{ \langle M \rangle \mid M \text{ is TM } \& \ \langle M \rangle \notin L(M) \}$

So, L_D is the set of all TMs which don't accept their own encoding.

Assume there is a TM P which recognizes L_D.

Now, does $\langle P \rangle$ belong to L_D.

Case 1: $\langle P \rangle \in L_D$. (i.e P is a TM which doesn't accept its own encoding)

Then P shouldn't accept $\langle P \rangle$.

But since $\langle P \rangle \in L_D$ & P recognizes L_D, then P should accept the member of L_D ($\langle P \rangle$ in this case).

$\therefore$ Contradiction.

Case 2: $\langle P \rangle \notin L_D$. (i.e. P is a TM which accepts its own encoding)

But since P accepts its encoding, & P recognizes L_D, then $\langle P \rangle \in L_D$.

Hence contradiction.

$\therefore$ No such TM P exists which can recognize L_D.

$\therefore$ L_D is a non-RE/not recognizable language.

* Rice Theorem

- For any set of values, any property over the values $P(x)$ divides the set into two ~~partitions~~ subsets. One subset is of all the elements which satisfy the property & other is of the ones which don't. So $P(x)$ is a subset of the domain.

$$P = \{x \mid x \in D \wedge P(x)\}$$

Ex. $\mathbb{N}$. $P(x) = x$ is even.

$\mathbb{N}$ → $2\mathbb{N}$, $2\mathbb{N}+1$

Ex. $\Sigma = \{0,1\}^*$. Σ^*. $P(w) = dec(w)$ is prime.

Σ^* → $\{\ldots 101, 010, 011\}$, $\{000, 001, 100, \ldots\}$

- So on ~~the~~ a set of strings (language) we can have properties like: w is palindrome, w starts with ..., w ends with ..., etc.

- For some Σ, on the set of all languages over Σ $\{P(\Sigma^*)\}$ we can have properties like:
  - Is L regular?
  - Is L infinite?
  - Is L RE?
  - Does L contain empty string?

  etc.

- Non-Trivial Property: A property $P(x)$ on base set (domain) D is non trivial iff

$$[\exists x\, P(x)] \wedge [\exists x\, \neg P(x)]$$

Means there should be some val. x which satisfies $P(x)$ & also there should be some val. x which doesn't satisfy $P(x)$.

- Note: "Rice Theorem" is applicable for "language of TMs".
(RE languages)

- i.e. "Rice Theorem" is only applicable only when decision problem is related to "language of TM". So, the decision problem must have following template:
  "Given a TM M, does L(M) satisfy some property P?"
  OR "Given a RE lang. L, does L satisfy some property P?"
- For comparision, decision problems related to TM have the following template:
  "Given a TM M, does M satisfy some property P?"
  Rice Theorem doesn't work here.

- So, the Base Set (Domain) for "Rice Theorem" is the set of all RE languages. $D = \{L \mid L \text{ is RE lang.}\}$, say RESet.
  Hence a non-trivial property on RESet would be such that: Some RE lang. satisfy the property, some RE languages don't.

- Rice Theorem: All non-trivial properties on RE languages are undecidable.
  (i.e. "All non-trivial properties P(L) on the set RESet are undecidable")

- We know that TM is analogous to Program (or Procedure). The language of TM is analogous to the functionality (or behavior) of the program. So, Rice Theorem says that every non-trivial behaviour / functionality of a program are undecidable.

- Domain: Set of all RE languages
  Monotonic Property: A property P of ~~RE language~~ RESet is monotonic iff whenever a RE lang. satisfies P, then every superset of that lang. also satisfies P.
  RE lang.

Non-Monotonic Property: A property P is non-monotonic iff there exist RE languages L_1 & L_2 ($L_1 \subset L_2$) & L_1 satisfies P, and L_2 doesn't satisfy P.

Ex. Domain: REset. Check trivial/non-trivial. Check monotonic/non-monotonic.

P1: is empty.

Non trivial. lang1 = ϕ $P_1(\phi)$ = True

lang2 = Σ^* $P_1(\Sigma^*)$ = False.

(both RE)

Non-monotonic. $L_1 = \phi$ $L_1 \subset L_2$

$L_2 = \Sigma^*$ L_1 satisfies P_1, L_2 doesn't.

(both RE)

P2: Is RE

Trivial. P_2 is True for all lang. in set REset.

Monotonic. ——— " ———

- If a property is trivial then it is monotonic.

P3: is finite

Non trivial. $L_T = \phi$ $L_F = \Sigma^*$

Non-monotonic. ——— " ———

P4: Is infinite.

~~Non~~-Monotonic. since every superset of inf. RE lang. is also inf.

Non trivial. $L_T = \Sigma^*$ $L_F = \phi$.

P_5: is ~~Empty~~ Regular.

Non-monotonic. $L_1 = \{ab\}$ $L_2 = \{a^n b^n \mid n \geq 0\}$

N Mono $\longrightarrow$ Non-trivial.

P_6: is non-regular

Non-monotonic. $L_1 = \{a^n b^n \mid n \geq 0\}$ $L_2 = \Sigma^*$

N Mono $\rightarrow$ Non Trivial.

P_7: is CFL

Non-monotonic $L_1 = \{abc\}$ $L_2 = \{a^n b^n c^n \mid n \geq 0\}$

Non-mono. $\rightarrow$ Non Trivial.

P_8: is NonCFL

Non-monotonic. $L_1 = \{a^n b^n c^n \mid n \geq 0\}$ $L_2 = \Sigma^*$

Non-mono $\rightarrow$ Non Trivial.

P_9: is Recursive.

Non mono. $L_1 = \phi$ $L_2 = \{\langle M, w\rangle \mid M \text{ is TM}, w \in \Sigma^*, w \in L(M)\}$

is Recur. Not is Rec.

Both RE.

Not mono $\rightarrow$ Not trivial.

P_{10}: contains a palindrome.

Non Trivial $L_1 = \{\}$ $L_2 = \{a\}$

Monotonic.

P_{11}: is equal to $\{ab, a\}$

Non mono. $L_1 = \{ab, a\}$ $L_2 = \{a, ab, b\}$

Non mono $\rightarrow$ Non trivial.

P_{12}: is superset of $\{ab\}$

Non tr. $L_1 = \phi$ $L_2 = \{ab, b\}$

Monotonic. $\{ab\} \subset L_1 \subset L_2$

P13: is equal to L_D (Diagonalization lang.)

Trivial. L_D is not RE, so any RE lang. can't be equal to L_D. ∴ Always No.

Trivial → Monotonic.

Rice Theorem: A property P of REset:

① If non-trivial then P is undecidable, i.e.

P is non trivial ⟷ $L = \{ l \mid l$ is RE lang. & $P(l)$ is true$\}$ is non-recursive lang.

② If non-monotonic then P is unrecognizable, i.e.

P is non-monotonic → $L = \{ l \mid l$ is RE lang. & $P(l)$ is true$\}$ is non-RE lang.

Ex. Prob: Given a TM, does it accept a?

$L = \{ \langle M \rangle \mid M$ is TM $\wedge$ $a \in L(M) \}$

{Problem/Property on lang. of TM}

Non trivial. $L_T = \{\epsilon\}$ $L_F = \phi$

Monotonic. $a \in L_1 \rightarrow a \in L_2$ if $L_1 \subset L_2$

∴ Not recursive. (Can't say RE or not using Rice Theorem)

```
Checker (M):
    out = Run(M, a);
    return out;
}
```

← if $a \in L(M)$ then it will halt & accept.
if $a \notin L(M)$, then checker may go to no halt.

∴ Not Rec. but REL.

Ex. P: Given a TM M, is L(M) infinite.

$L = \{ \langle M \rangle \mid M \text{ is TM} \wedge L(M) \text{ is } \infty \}$.

Nontrivial. $L_T = \Sigma^*$ $L_F = \phi$

Monotonic L_1 is $\infty \wedge L_1 \subset L_2 \rightarrow L_2$ is inf.

$\therefore$ Not Rec. (can't say if REL or not).

```
Checker (M) {
    n = 0;
    P = Σ* in lexicogr. order
    for P[i], i ≤ n {
        Run M on P[i] for i steps
        n++
    }
}
```

We can't verify members, since if L(M) is ∞, then the loop doesn't stop for any natural no. n.

$\therefore$ Not REL.

Ex. P: Given a TM M, M accepts at least one $w \in \Sigma^*$.

$L = \{ \langle M \rangle \mid M \text{ is TM} \wedge |L(M)| \geq 1 \}$

NonTrivial. Monotonic.

L is Not Rec.

Checker by Dovetelling can eventually get to the string accepted by M for members $\langle M \rangle$ & halt & accept. For non-members it may/may not halt.

$\therefore$ REL, but not Rec.

Ex. P: Given a TM M, L(M) is finite.

Non Trivial. Non Monotonic.

∴ Not REL.

Ex. P: Given a TM M, L(M) is regular.

Non-monotonic. $L_1 = \phi$ $L_2 = \{a^n b^n \mid n \geq 0\}$

∴ Not REL. (non-recognizable)

Ex. P: Given a TM M, $|L(M)| \leq 10$.

Non monotonic. $L_1 = \{a^n \mid n \leq 10\}$ $L_2 = \{a^n \mid n \geq 0\}$

∴ Not REL.

Ex. P: Given a REL L, L is recognized by some TM having even no. of states.

$L = \{\langle M \rangle \mid M$ is TM, $L(M)$ is recog. by some TM with even no. of states$\}$.

Trivial. (By using 0 or 1 unreachable states)

Always Yes.

∴ Decidable.

{Every automata can be made with even, odd, non-prime, prime, etc no of states by help of unreachable states.}

Ex. P: Given TM M, is L(M) non-empty?

Non triv. Monotonic.

Checker for member M's stop as soon as first string would be found, using dovetailing.

∴ RE not Rec.

Ex. P: Given TM M, $L(M)$ is empty.

Non-mono. $L_1 = \{\}$ $L_2 = \Sigma^*$

$\therefore$ Not REL.

Ex. P: Given TM M, $\overline{L(M)}$ is finite.

Non-triv. $L_1 = \{\}$ $L_2 = \Sigma^*$

Checker will find complement (let's say possible) & then will have to check finiteness. This may not stop for even members.

$\therefore$ Not REL.

(also compl. of a REL $L(M)$ may be not RE).

Note: When solving questions related to decidability & recognizability we can treat "Halt" words as "Accept".

Ex. Some properties of TM (and not of lang. of TM).

Given a TM M:

(1) Does it have start state

(2) Does it have final state

(3) Does it have less than seven states.

(4) Does it halt for all strings within 10 steps.

etc.

Note: If you can rewrite a property only w.r.t the lang. & then TM out of property, then the property is "Language Property", else "Machine Property".

Ex. Classify : Not RE (NRE), REnotRec (RENR), Recursive (R)

M is some TM

1. P: The lang. of M is unrecognizable.
2. P: M contains odd no. of states.
3. P: Every string accepted by M is of length at least 4.
4. P: The lang. of L is decidable.
5. P: No computation on M takes less than 5 steps
6. P: M has more than 3 states.
7. P: M accepts some string in ≤ 100 steps of computation.
8. P: M uses \$ as a symbol on tape.
9. P: M accepts w^R if M accepts w.

1. Lang Property.

$\therefore$ Trivial. Since M is TM. L(M) is always RE.

$\therefore$ Decidable

2. M/C Property. Decidable. Just check encoding.

3. Lang. Property. $\forall w \in L \; |w| \geq 4$.

$L_1 = \{a^4 a^*\}$ $L_2 = \Sigma^*$

$\therefore$ Non mono.

$\therefore$ NRE.

4. Lang. Property. $L_1 = \phi$ $L_2 = \{\langle M, w \rangle \mid w \in L(M)\}$

Nonmono. $\therefore$ NRE

5. M/C Property. If M takes less than 5 steps, it would have to halt within first five symbols of the string. So for all possible permutations upto length 5, Keep a counter. Run M on them & count no. of steps. If for any perm. M takes less than 5 steps & halts return "Reject", else "accept".

$\therefore$ Decidable.

6. M/C Property. Decidable. Count no. of states.

7. M/C Prop. For each perm of upto 101 len, check if M halts within 100 steps. If Yes, accept, else reject. Decidable.

8. M/C Prop. Decidable.

9. Lang. Property. P: L contains w^R if L contains w.

$L_1 = \{ab, ba\}$ $L_2 = \Sigma^*$

$\therefore$ Not mono.

$\therefore$ Not RE

10. P: L(M) is decidable.

11. P: M always halts.

10, 11. These look same but are a little diff.

In L10 we can have <M>, where M is TM but not HTM but L(M) is Recursive. In L11 we can only have <M> where M is HTM.

So, $L_{11} \subset L_{10}$.

10. Lang. Property. $L_1 = \phi$ $L_2 = \{\langle M, w\rangle \mid w \in M\}$.

$\therefore$ Non-mono. $\Rightarrow$ NRE.

11. M/C Prop. For members also we have to keep checking strings & make sure it halts for all of them.

$\therefore$ Not RE.

Alternate: Replace Halt with Accept.

P': M accepts all strings.

i.e. $L(M) = \Sigma^*$

Now this is lang. property.

Not RE.

12. P: M halts on at least one input

m/c Problem. Can be converted to lang problem.

P: M accepts at least one input

i.e. $|L(M)| > 0$. or $L(M) \neq \phi$.

RE Not R.

* Reductions

- Consider a new variant of TM: Accept/Loop TM. It either halts (which means accept) or it just doesn't halt. There is no q_f & q_r, there is just q_H (Halt) state.

- Computable Function: A func. is computable if there is an algorithm (decider/HTM) to compute it.

Ex. Consider two languages $A, B \subseteq \Sigma^*$. Assume there is a computable func. $f: \Sigma^* \to \Sigma^*$, such that

$$\forall w \in \Sigma^* \quad w \in A \longleftrightarrow f(w) \in B$$

If B is recognizable, then A recognizable?

B is recognizable, i.e. $\exists$ TM which for $\forall w \in B$, halts & accepts.
Now can we create a TM to recognize A?
Yes.

- Given a $w \in \Sigma^*$, compute $f(w)$, this will:
  - belong to B, iff $w \in A$
  - not belong to B, iff $w \notin A$.
- Run B's recognizer on $f(w)$.
- If B's recognizer halts on $f(w)$ & accepts, w must belong to A.

This is A's recognizer.

- So, for two lang. L_1 & L_2, if there is a func $f: \Sigma^* \to \Sigma^*$, such that $\forall w \in \Sigma^*$ $w \in L_1 \longleftrightarrow f(w) \in L_2$ & L_2 is decidable then L_1 is also decidable.
- If L_2 is recognizable then L_1 is also recognizable.

Ex. Given a lang L, there exists a TM M which halts & accepts all strings belonging to L. Further M may never halt on strings which don't belong to L. Then:

(A) L is RE lang.

(B) L is not Recursive lang.

A is true, since for L to be RE, there only needs to be some TM, & M is there.

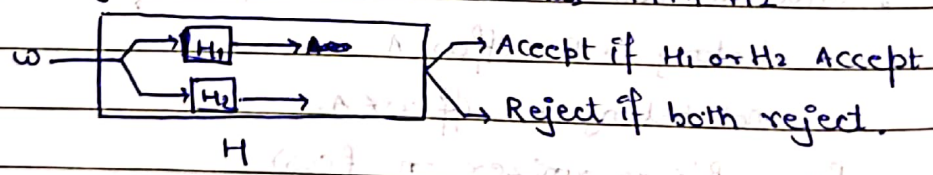
B is false, since we know M is not a HTM for L, but there might be some other HTM for L.

* Closure Properties of Recursive & RE languages.

- Union Operation

  - If L_1 & L_2 are Recursive, is $L_1 \cup L_2$ Recursive.

    Yes. Say L_1 has HTM H_1 & L_2 has HTM H_2



    ∴ H is HTM since it always halts.

  - If L_1 & L_2 are RE then $L_1 \cup L_2$ is RE.

    Yes. Similar as above, run T_1 & T_2 parallaly (or n steps alternatively). If $w \in L_1$, T_1 will def. halt, if $w \in L_2$, T_2 will definitely halt. So for $w \in L_1 \cup L_2$, our m/c halts & accepts. So T is also TM. ∴ $L_1 \cup L_2$ is RE.

- $RE \cup REC = RE$

  (may/maynot be Rec.)

  Ex. $A_{TM} \cup \Sigma^* = \Sigma^*$ is Rec.

  $A_{TM} \cup \phi = A_{TM}$ is not Rec.

  Acceptance/Membership problem for TM.

- (Rec) $\cup$ (RE but not Rec) = RE

  (may/maynot be Rec.)

- Intersection:
  - Rec. $\cap$ Rec. = Rec.
  - RE $\cap$ RE = RE

- Kleene Star
  - $(Rec)^* = Rec$
  - $(RE)^* = RE$

- Concatenation:
  - Rec. . Rec. = Rec.
  - RE . RE = RE

- Kleene Plus.
  - $(Rec)^+ = Rec.$
  - $(RE)^+ = (RE)$

- Complementation
  - $\overline{RE}$ = May/Maynot be RE.

    Ex. $L = \Sigma^*$ is RE. $\bar{L} = \phi$ is also RE

    Ex. $L = \{\langle M \rangle \mid M \text{ is TM}, \langle M \rangle \in L(M)\}$. L is RE not R.

    $\bar{L}$ is not RE. $\bar{L} = \{\langle M \rangle \mid M \text{ is TM}, \langle M \rangle \notin L(M)\}$

    Ex. $L = \{\langle M \rangle \mid |L(M)| \geq 3\}$ is RE.

    $\bar{L} = \{\langle M \rangle \mid |L(M)| \leq 2\}$ is not RE.

  - $\overline{REC}$ = Rec.

    w.k.t if L is Rec., then some HTM M exists for it.

    We know it always halts, then accepts/rejects.

    If M accepts on w, we reject & vice-versa.

    This behaves as HTM for $\bar{L}$.

Ex. If a given lang. L is RE & $\bar{L}$ is RE. Prove L is Rec.

Suppose there is a TM for L & $\bar{L}$ resp, say M & N.

Now, for a string $w \in \Sigma^*$, ~~give~~ create a TM C, which gives w to both M & N parallaly (interleaved).

If M accepts w, we can be sure N never will, & $w \in L$

If N accepts w, we can be sure M never will & $w \in \bar{L} \Rightarrow w \notin L$.

There is no condition where neither M nor N accept since $L \cup \bar{L} = \Sigma^*$. So, C always halts

$\therefore$ C behaves as HTM. $\therefore$ L is Recursive.

- Theorem: L & $\bar{L}$ both are RE iff L & $\bar{L}$ both are Recursive.

This also means, if L is RE not Rec, then $\bar{L}$ is not RE.

- if L is not RE, then $\bar{L}$ is either not RE or RE not Rec.

Ex. $L = \{\langle M \rangle \mid L(M) \text{ is regular}, M \text{ is TM}\}$ both L & $\bar{L}$ are not RE.

Ex. $L = \{\langle M \rangle \mid M \text{ is TM}, \langle M \rangle \notin L(M)\}$ L is not RE. $\bar{L}$ is RE not Rec.

Countability

- Countability answers: "Are all ∞ same?"
  (i.e $\infty = \infty$?)

- Cardinality of a set: When the set is finite we can compare the cardinality of different sets easily, but, infinity doesn't behave the same way as finite values.

- Consider a finite hotel. If the hotel is full, can you accomodate a guest? No. Answer is simple.
  Now consider a infinite hotel. 1, 2, 3, ... If the hotel is full, can you accomodate another guest? Yes. How? Shift all current occupants to the next room. Now Room 1 is empty. Put new guest there.

- Two sets S, T (finite or infinite) have same Cardinality iff there exists some bijection (one to one correspondance) b/w S & T.
- $|S| \leq |T|$ iff there exists an Injection (one to one) from S to T.
- $|S| \geq |T|$ iff there exists a Surjection (onto) from S to T.

- To prove $|S| = |T|$:
  - Either show a func. f is bijective from S to T
  - OR Show $\exists f_1 : S \to T$ is one-one & $\exists f_2 : S \to T$ is onto
  - OR Show $\exists f_1 : S \to T$ is one-one & $\exists f_2 : T \to S$ is one-one
  - OR Show $\exists f_1 : S \to T$ is onto & $\exists f_2 : T \to S$ is onto.

  (The idea is to show $(|S| = |T|)$ or $(|S| \geq |T|$ & $|S| \leq |T|)$.

Ex We say set B is proper subset of set A iff $\exists x \; x \in A \wedge x \notin B$.
For finite set A & B, $B \subset A$, w.k.t $|B| < |A|$. Is it always true for ∞ sets?
No. Consider $\mathbb{N} = \{1, 2, \ldots\}$ & $E = \{2, 4, 6, 8, \ldots\}$. $E \subset \mathbb{N}$. We can show $|E| = |\mathbb{N}|$. Consider $f : \mathbb{N} \to E$, $f(n) = 2n$. It is one-one & onto.
$\therefore$ f is bijection b/w $\mathbb{N}$ & E.
$\therefore |E| = |\mathbb{N}|$.

- For infinite sets, we use concept of Cardinality only to compare the size of two sets, not to find size of one particular set.

- For two ∞ sets A & B, we can say $|A| < |B|$ iff $\exists f: A \to B$, f is one-one but there is no surjection (onto) func. from A to B.
- $|A| > |B|$, iff there exist a onto func. but no one-one (nor a bijection) exists from A to B.

- Countable Set: Any set A is called countable iff $|A| \leq |\mathbb{N}|$.
- So, every finite set is countable, by definition.
- Alternatively, any set A is countable iff there exists a one-one func. from A to N. (OR a onto func. from N to A).
- So, the ∞ sets, such as $\mathbb{N}$, EvenN, OddN, etc are countable.

Countable Set A

A is finite — $|A| < |\mathbb{N}|$

A is infinite — $|A| = |\mathbb{N}|$

- Countable Infinite: Any set A is countable infinite iff $|A| = |\mathbb{N}|$.

- Intuition behind Countability: Given a set, can I list the elements of the set such that every element of the set appears on the list. (Such a list is called Enumerable list).

Ex. Even Natural no. set E

2	4	6	8	10	12	...
Idx: 1	2	3	4	5	6	

every element i of E appears appears at position i/2.

Ex. Set of +ve Rational numbers.

	1	2	3	4	
1	1/1	1/2	1/3	1/4	...
2	2/1	2/2	2/3	2/4	...
3	3/1	3/2	3/3	3/4	...
	⋮	⋮	⋮	⋮	

Flatten this list by stretching the thread.
(Doesn't matter that there is repetition.)

Ex. Set of Integers.

0, 1, -1, 2, -2, 3, -3, ...

Every element appears on the list.

∴ Countable Infinite set.

[Alternate Def^n] Countable Set: A set A is countable iff to every element of $\mathbb{N}$, if we can assign a finite subset of A, such that every element of A are covered, then A is countable. Coverlapping is allowed.

Ex. $A = \{a, b, c, d\}$

1: $\{a, b, c\}$ 2: $\{c, d\}$ 3: $\{d\}$ 4: $\{d\}$ 5: $\{\}$ 6: $\{\}$ ...

∴ A is countable set.

(or 1: $\{a, b, c, d\}$ 2: ϕ 3: ϕ ...)

Ex. Set of integers ($\mathbb{Z}$)

1: $\{0, -1, 1\}$ 2: $\{2, -2\}$ 3: $\{3, -3\}$ ...

Every element of $\mathbb{Z}$ is covered.

∴ $\mathbb{Z}$ is countably infinite.

Ex. Rational No. ($\mathbb{R}$)

$\forall m/n \in \mathbb{R}$ assign to $|m| + |n|$, so

1: $\{0/1, 1/1\}$ 2: $\{1/1, 0/2, -1/1, 1/-1, 0/-2\}$ ...

∴ $\mathbb{R}$ is countable.

Ex. $\Sigma = \{a, b\}$ is Σ^* countable?

- $\forall w \in \Sigma^*$ assign w to $|w|$. Assign ε to 1.

1: $\{\varepsilon, a, b\}$ 2: $\{ab, aa, ba, bb\}$ 3: ... ...

for $n \in \mathbb{N}$, the no. of elements is 2^n.

∴ Σ^* is count

- Every subset of a countable set is countable.
(We can easily show this using Alternate Def^n of Countability.)

- Uncountability: A set A is uncountable iff $|A| > |\mathbb{N}|$.
  - So every uncountable set is infinite.
  - We can also say, there doesn't exist any bijection b/w A & $\mathbb{N}$.
  - There doesn't exist an enumeration for A.

- We can show Uncountability by assuming A is countable, and then showing that there exists an element in A such that it doesn't map to any $n \in \mathbb{N}$. Hence contradiction.

Ex. S: Set of all real no. b/w 0 & 1 which only uses decimal digits 3 & 4.

$S = \{.33, .3434, .3444..., ...\}$

Let's assume S is countable, then there must exist a bijection b/w S & $\mathbb{N}$. Say $f: \mathbb{N} \to S$

So, in $\mathbb{N}$, 1, 2, 3, 4.. all map to some element in S. Say

1: .3433..
2: .4443..
3: .4334..
4: .3334..
:
...

Now, lets create a no. It differs from f(1) in 1st pos; So 1st pos = 4.
It differs from f(2) in 2nd pos. So 2nd pos = 3
& so on.
$\Rightarrow$ .4333...

This new no. maps to no $n \in \mathbb{N}$. Why? because it doesn't map to 1 (differs from f(1) in 1st pos), for $n \in \mathbb{N}$ it will not be f(n) {will differ at n^{th} pos}.

$\therefore$ Contradiction.

$\therefore$ S is uncountable.

- The previous proof is called "Cantor's Diagonalization Proof".
- Every superset of uncountable set is also uncountable.
- So set of all real no., set of all real nos. in $[0,1]$, etc are all uncountable.

- Cantor's Theorem: For any set S, $|S| < |P(S)|$, where $P(S)$ is power set of S. This means we have ∞ levels of ∞.

$\vdots$

L_3: $P(P(P(\mathbb{N})))$, ..

L_2: $P(P(\mathbb{N}))$, ..

L_1: $P(\mathbb{N})$, ...

L_0: $\mathbb{N}, \mathbb{Z}, \Sigma^*, \mathbb{Q}$, E.. [Countable level of ∞]

Proof: We have to show for any set S, we have no bijection.

- For any set S we can easily have injection (one-one).

Ex: $S = \{a, b, c, ..\}$ $P(S) = \{\phi, \{a\}, \{b\}, \{c\}, ..\}$

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

- For surjection. C1: S is finite, then $|S| = n \Rightarrow |P(S)| = 2^n$ & $\forall_{n \geq 0} n < 2^n$

Hence proved.

- For surjection. C2: S is infinite.

Assume there is a bijection $f: S \to P(S)$.

$a_1: \{a_2, a_4, ..\}$ $a_2: \{a_2, a_4, a_7, ..\}$ $a_3: \{\}$ $a_4: \{a_7, a_9, a_{11}, ..\}$

Now let's create a $x \in P(S)$.

$x = \{a_1, a_3, a_4, ..\}$. This x, $f^{-1}(x) \neq a_1$, since $a_1 \notin f(a_1)$ & $a_1 \in x$.

$f^{-1}(x) \neq a_2$, since $a_2 \in f(a_2)$ & $a_2 \notin x$. $f(a_3) \neq x$, $a_3 \notin f(a_3)$ & $a_3 \in x$,

& hence for all a_i, $x \neq f(a_i)$.

$\therefore$ x has no preimg. $\therefore$ Contradiction.

Hence S has no bijection with $P(S)$ & hence no surjection.

Hence Proved.

$\therefore$ $|S| < |P(S)|$

So, Power set of any ∞ set is uncountable.

Ex. Set of all TMs?

w.K.t set of all TMs $= \{\langle M \rangle \mid M \text{ is TM}\} \in \Sigma^*$.

$\therefore$ Countable.

Ex. Set of all RE languages?

We can show bijection f: Set of all RE lang $\rightarrow$ Set of all TM.

Every REL l has a unique TM. $\therefore$ One-One.

For every TM M, $\exists$ REL $L(M)$. $\therefore$ Onto.

$\therefore$ Bijection exist b/w RE lang & a countable set TM.

$\therefore$ |Set of all REL| = |Set of all TM| = $|\mathbb{N}|$

$\therefore$ set of all REL is countable infinite.

Ex. Set of all lang. $= P(\Sigma^*)$

Uncountable. $P(\Sigma^*)$ is power set of ∞ set.

Ex. Any lang. L over Σ.

Countable. Any lang. $L \subseteq \Sigma^*$. $\therefore$ subset of countable ∞ set.