TOC 1

DFA: $(Q, \Sigma, \delta, q_0, F)$ $\quad \delta: Q \times \Sigma \to Q$

NFA: $(Q, \Sigma, \delta, q_0, F)$ $\quad \delta: Q \times \Sigma \to 2^Q$

ϵ-NFA: $(Q, \Sigma, \delta, q_0, F)$ $\quad \delta: Q \times \{\Sigma \cup \{\epsilon\}\} \to 2^Q$

Moore: $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ $\quad \lambda: Q \to \Delta$

Mealy: $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ $\quad \lambda: Q \times \Sigma \to \Delta$

PDA: $(Q, \Sigma, \Gamma, z_0, \delta, q_0, F)$ $\quad z_0 \in \Gamma \quad F \subseteq Q$

(F can be skipped in case of acceptance by empty stack.)

DPDA: $\delta: Q \times (\Sigma \cup \epsilon) \times \Gamma \to Q \times \Gamma^*$

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

z_0 = topmost stack ele (start symbol) $\quad \Gamma$ = set of stack symbols

TM: $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Σ = i/p alphabet. $\quad \Gamma$ = Tape symbols ($\Sigma \subseteq \Gamma$)

F = Final states ($F \subseteq Q$) $\quad B$ = Blank symbol of the Tape. It is in Γ, but not in Σ.

The blank appears initially in all but finite no. of initial cells that hold i/p symbols.

$\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ DTM

NTM.

Language	Grammar	
FORMAL LANGUAGE	Regular languages (Type 3)	Regular (Type 3) grammar — Finite Automata. DFA = NFA
	Context Free Lang. (Type 2)	Context (T2) free grammar — Push down Automata. DPDA < NPDA
	Context Sensitive lang. (Type 1)	Context sensitive (T1) grammar. — Linearly Bounded Automata. DLBA ? NLBA
	Recursive Enumerable Lang. (Type 0)	Recursive Enum. (T0) gramm. — Turing Machine. DTM = NTM

→ A string $|w| = n$

substrings = $\Sigma n + 1$ # non trivial substrs = $\Sigma n - 1$

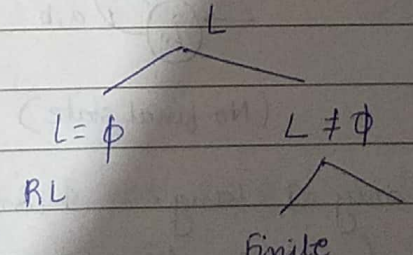
prefixes = $n + 1$ # suffixes = $n + 1$

→ Σ^* – KLEEN CLOSURE Σ^+ – POSITIVE CLOSURE

$\Sigma^+ = \Sigma^* - \epsilon$ $\Sigma^k = \{ w \mid |w| = k \}$

Σ^* = Universal language.

→



All formal languages are countable.

Closure Properties of Languages

Property	Regular	CFL	DCFL	CSL	Recursive	RE
Union	Yes	Yes	No	Yes	Yes	Yes
Intersection	Yes	No	No	Yes	Yes	Yes
Set Difference	Yes	No	No	Yes	Yes	No
Complementation	Yes	No	Yes	Yes	Yes	No
Intersection with a regular lang.	Yes	Yes	Yes	Yes	Yes	Yes
Concatenation	Yes	Yes	No	Yes	Yes	Yes
Kleen Closure	Yes	Yes	No	Yes	Yes	Yes
Kleen Plus	Yes	Yes	No	Yes	Yes	Yes
Reversal	Yes	Yes	No	Yes	Yes	Yes
Homomorphism	Yes	Yes	No	No	No	Yes
ϵ-free Homomorphism	Yes	Yes	No	Yes	Yes	Yes
Inverse Homomorphism	Yes	Yes	Yes	Yes	Yes	Yes
Substitution	Yes	Yes	No	No	No	Yes
ϵ-free Substitution	Yes	Yes	No	Yes	Yes	Yes
		3	3	2	2	2

RL ⟶ RG ⟶ FA

FA

w/o o/p : DFA, NFA, ε-NFA

w/ o/p : Moore, Mealy

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

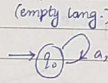
$q_0 \in Q$, $F \subseteq Q$. $\delta : Q \times \Sigma \longrightarrow Q$.

An FA is said to accept a lang. if it accepts all the strings in the lang. & rejects all other strings that don't belong to the language.

Minimal DFA

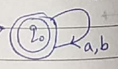
(i) Accepts ε

(ii) Accepts $L = \phi$ (empty lang.)



(No final state)

(iii) Accepts $L = \Sigma^*$ (Universal lang.)

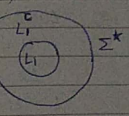


Every FA accepts only 1 lang. 1 lang. can be accepted by many FAs (some have equal states or are not minimal). 1 lang. has only 1 minimal DFA.

!!! After you draw any DFA, have a quick look to check if any minimization is possible.

COMPLEMENT OF DFA:

- Complementation exists for complete system only (DFA) & not for NFA or ε-NFA.
- Obtained after interchanging the F & NF states.



- $L(FA) \cap L(FA^c) = \phi$
- $L(FA) \cup L(FA^c) = \Sigma^*$

$M \longrightarrow (Q, \Sigma, \delta, q_0, F)$ $\quad M^c \longrightarrow (Q, \Sigma, \delta, q_0, Q - F)$

⟶ **COMPOUND AUTOMATA:**

(1) FA1 AND FA2

Q: $Q_1 \times Q_2 = \{q_0q_2, q_0q_3, q_1q_2, q_1q_3\}$

F : ∩ of final states $= \{q_1q_3\}$

δ:

$\delta(q_0q_2, a) = \delta(q_0, a) \cup \delta(q_2, a)$
$= q_1q_2$

Eg: (i) Even no. of a's & odd no. of b's.

(ii) #a's divisible by 3 & #b's divisible by 4.

(2) FA1 OR FA2

Q: $Q_1 \times Q_2 = \{q_0q_2, q_0q_3, q_1q_2, q_1q_3\}$

F: ∪ of final states $= \{q_0q_2, q_0q_3, q_1q_3\}$.

δ:

$\delta(q_0q_2, a) = \delta(q_0, a) \cup \delta(q_2, a)$
$= q_1q_2$

Eg: (i) Even no. of a's OR even no. of b's.

(ii) #a's divisible by 2 OR #b's divisible by 3.

The transition functions for both AND & OR are same. After construction we need to perform minimization, which might lead Q_{new} to be different in AND & OR. Final states in both differ.

Dead states need to be taken care of while performing $Q_1 \times Q_2$.

CROSS PRODUCT = AND.
UNION = OR.

Other operations are :

(3) CONCAT :

$L1 = aX = \{a, aa, ab, aaa, \ldots\}$. $X = (a+b)^*$

$L2 = Xb = \{b, ab, bb, aab, abb, \ldots\}$.

$L = L1 \cdot L2$ = Stat with 'a' & end with 'b'.

In above, Final states of DFA1 merged with Init states of DFA2 & transitions taken care of accordingly.

(4) COMPLEMENTATION :

$L1 = \{w \mid w$ contains a $\}$ $L1^c$ = w doesn't contain 'a'

Complementation of DFA → Interchange F & NF states.

Make finals as non-final & vice-versa.

(5) REVERSAL : Reverse all strings $\in L1$.

$L1$ = w starts with 'a' = $\{a, aa, ab, aaa, \ldots\}$.

$L1^R = \{a, aa, ba, aaa, aba, baa, bba, \ldots\}$

= end with 'a'.

DFA_reversal : (Works for both DFA & NFA) includes ε-NFA.

1. Interchange q. & final states.
2. Reverse all transitions (i.e. direction of edges).
3. Minimize if rqrd. The resultant might also turn out to be an NFA.

In resultant, If more than one initial state occurs, then convert the s/m to have only one initial state.
Join all initial states in resultant using ε-move from one initial state.

→ $NFA^c \neq$ Complement of long. accepted by NFA.

MYHIL - NERODE THEOREM :

Let L be a regular language, then it contains finite number of equivalence classes. Union of all Myhill - Nerode equivalence classes gives universal language. All the strings of universal language Partitioned into a finite number of Myhill-Nerode equivalence classes, if L is regular.

The number of Myhill - Nerode equivalence classes in a regular language ≡ Number of states in the unique minimal DFA corresponding to that language.

→ RE/FA satisfy the closure property wrt following operators:

1. Kleen closure (L1 → RE1 ⇒ (RE1)*)
2. Positive closure
3. Concatenation (RE1, RE2 ⇒ RE1 · RE2)
4. Complementation ($\bar{L} = \Sigma^* - L$, DFA^C can be constructed.)
5. Reverse (L^R, DFA^R can be constructed)
6. Prefix operator (Init) & Suffix
7. Union (finite) (R1 + R2)
8. Intersection (finite) ($\overline{\bar{L1} \cup \bar{L2}} = L1 \cap L2$)
9. Difference (L1 − L2 or L2 − L1)
   $L1 - L2 = L1 \cap \bar{L2}$
   If L1, L2 are R.L., then ⇒ L1 − L2 is also R.L.
10. Symmetric Difference ($L1 \Delta L2$)
    $L1 \Delta L2 = (L1 - L2) \cup (L2 - L1)$
    $= (L1 \cup L2) - (L1 \cap L2)$
11. Quotient ($L1 / L2$) or ($L1 \setminus L2$) [R.Q. or L.Q]
    $wx / x = w$ $\quad x \setminus xw = w$.
12. Substitution
13. $\frac{1}{2}L$, $\frac{11}{4}$, $\frac{1}{8}L$, $\frac{1}{16}L$ etc.

    $HALF(L) = \{ w \in \Sigma^* \mid wx \in L \wedge |w| = |x| \}$.
14. Min (L)
15. Max (L)
16. Cycle (L)
17. Homomorphism
18. Inverse homomorphism.
19. Isomorphism.
20. COR (Complement of OR)

$L1 \cap \bar{L2}$ $\quad L \in CFL$

$\{ \epsilon, ab, bc, abc, ab^2c^2, a^2b^2c, a^2b^4c^2, \ldots \}$

$a^n b^m c^m$ $\quad n, m \geq 0$

$a^n b^n c^m$ $\quad m > n$

$a^n b^m c^p$ $\quad m < n$

21. Finite subset of RL/NRL is always RL.

22. Finite union/intersection of RL is always RL.

Infinite union, inf. intersection, subset, superset need not be regular always (sometimes can be).

NOT CLOSED : (atleast 1 example for which not closed).

23. L/a $\qquad$ $a\backslash L$

Quotient of L & a $\qquad$ Derivative $= \frac{dL}{da}$.

$L/a = \{w \mid wa \in L\}$ $\qquad$ $a\backslash L = \{w \mid aw \in L\}$

closed. $\qquad$ closed.

24. $w = a_1 a_2 \ldots a_n$ $\qquad$ $x = b_1 b_2 \ldots b_n$ $\qquad$ $|w| = |x|$

$alt(w, x) = a_1 b_1 a_2 b_2 \ldots a_n b_n$ $\qquad$ // Shuffle

$alt(L, M) = \{ alt(w, x) \mid w \in L, x \in M \ \& \ |w| = |x| \}$

25. $even(w)$ = string consists of letters of w in even positions

$even(L) = \{ even(w) \mid w \in L \}$ $\qquad$ (L is RL)

26. $sqrt(L) = \{ x \mid \exists y \ |y| = |x|^2 \ \& \ xy \in L \}$

27. $log(L) = \{ x \mid \exists y \ |y| = 2^{|x|} \ \wedge \ xy \in L \}$

28. Subsequence (L)

$= \{w \mid w$ obtained by removing symbols from anywhere in $w \ \&$ $w \in L \}$.

DECIDABLE PROBLEMS

	FA / Regular Grammar		PDA / CFG		
Emptiness	Yes	Remove inaccessible states and check for presence of final states.	Yes	If the reduced CFG (eliminate useless productions) contains at least 1 valid production, then the CFG generates Non empty lang; otherwise Empty language. O(Size of Grammar)	
Non Emptiness	Yes		Yes		
Finiteness	Yes	1. Remove inaccessible and dead states. 2. If no cycle or infinite loops present, then finite lang, otherwise infinite lang.	Yes	1. Convert grammar to **CNF**. 2. Draw variable dependency graph. 3. If the variable dependency graph contains loops or cycles, then the grammar generates infinite language, otherwise finite. In VDG, define rank of each variable (node). If there exists a node A, such that Rank(A) = infinite, then infinite language. Rank(A) = length of longest path from A to last node.	
Infiniteness	Yes		Yes		
Equivalence (Language accepted by two machines is same.) L(M1)=L(M2)	Yes	**1.** Start construction of a new transition table with (x, y) where (x, y) are initial states of 2 machines, and the Σ. **2.** Continue construction for any new pair entry in column. Terminate if any (F, NF) or (NF, F) entry comes, and it means unequal machines. Otherwise, if all pairs are (F, F) or (NF, NF) then they are equal.	**No**		
Membership	Yes	Construct FA, and pass string through it. If it starts	Yes	CYK Algo. Grammar needs to be in	

		from q0 and reaches a final state at the end of string parsing, implies member of L.		CNF. O(n^3), n=length of w
Subset	Yes	$(L1 \subseteq L2) = (L1 \cap L2^c == \phi)$ can be checked		
Ambiguity			No	
Regularity			No	
Non Regularity			No	

*A DPDA which accepts by **empty stack** cannot accept all Regular Languages?* *TRUE*

All Regular Languages doesn't satisfy prefix property. *TRUE*

- If a language is DCFL and accepted by a DPDA with empty stack it must have the prefix property. This is because to accept a string stack must become empty. Now, when the stack becomes empty it must accept the string and due to determinism, it cannot have another move possible. So, no more characters from the input can be accepted. This is the reason why 'a' is TRUE. But the same property is not relevant for DPDA which accepts by final state.

Non-Trivial Property

A property which is satisfied by all elements of a set or NO element of a set is called a TRIVIAL property. Any other property is called non-trivial. If P is a non-trivial property of a set S, then some elements of S will satisfy the property and some will violate the property. In First Order logic we can write this as:

$$\exists x\, P(x) \wedge \exists y\, \neg P(y)$$

Now, we will see some property of RE set. Each element of this set is a recursively enumerable language. So, what can be some properties?

1. Is "0100" an element of L?
2. Are there more than 100 elements in L?
3. Is there a TM, M such that L is accepted by it?
4. Is there no TM, M such that L is not accepted by it?

Properties 3 and 4 are **trivial as 3 is true for any recursively enumerable language and 4 is FALSE for any recursively enumerable language.**

Rice's Theorem Part 1

Any non-trivial property of Recursively Enumerable set is UNDECIDABLE.

i.e., The following problems are undecidable:

1. If a given recursively enumerable language has a particular string
2. If a given recursively enumerable language is finite/regular/context-free
3. If number of strings in the given recursively enumerable language is 10 (or any other number)

→ Verify which of the following languages are regular

1. $L = \{ w \mid |w| = \text{even} \}$ — RL
2. $L = \{ w \in \Sigma^* \mid |w| \geq 10 \}$ — RL
3. $L = \{ w \in (a+b)^* \mid |w|_a = |w|_b \}$ [Same for $\leq, \geq$] — NRL
4. $L = \{ a^m b^n \mid 1 \leq m, n \leq 2 \}$ — RL
5. $L = \{ a^m b^n \mid m, n \geq 0 \}$ — RL
6. $L = \{ a^n b^n \mid 1 \leq n \leq 2^p$, p is the largest 3 digit prime $\}$ — RL
7. $L = \{ a, ab \}$ — RL
8. $L = \{ w \in (a+b)^* \mid |w| = 10 \}$ — RL
9. $L = \{ w \in (a+b)^* \mid |w|_b = r \pmod n \}$ — RL
10. $L = \{ w \in (a+b)^* \mid |w| = r \bmod n \}$ — RL
11. $L = \{ a^m b^n \mid m \geq 0 \quad n \geq 0 \}$ — RL
12. $L = \{ a^m b^n \mid m \geq 0 \quad n > 2018 \}$ — RL
13. $L = \{ a^m b^n \mid m$ — RL
14. $L = \{ a^m b^n \mid m \times n = \text{finite} \}$ — RL
15. $L = \{ a^m b^n \mid m = n \; ; \; 1 \leq m \leq 2^{2^{2010}} \}$ — RL
16. $L = \{ a^m b^n \mid m = n \; ; \; m, n \geq 10 \}$ — NRL
17. $L = \{ a^m b^n \mid m = 2n \; ; \; m, n \geq 1 \}$ — NRL
18. $L = \{ a^m b^n \mid m = n^2 \; ; \; m, n \geq 1 \}$ — NRL
19. $L = \{ a^m b^n \mid m > n \; , \; m \geq 1 \; , \; n \geq 0 \}$ — NRL
20. $L = \{ a^m b^n \mid m \neq n \; , \; m, n \geq 0 \}$ — NRL
21. $L = \{ a^m b^n \mid m$ divides $n \}$ — NRL
22. $L = \{ a^m b^n \mid m = n^p \; ; \; p \geq 1 \}$ — NRL
23. $L = \{ a^m b^n \mid GCD(m, n) = 1 \}$ — NRL
24. $L = \{ a^n b^n \mid 1 \leq n \leq 2^{37\text{th}}$ prime$) \}$ — RL
25. $L = \{ a^m b^n \mid m + n = \text{even} \}$ — RL
26. $L = \{ a^m b^n \mid m + n = \text{odd} \}$ — RL
27. $L = \{ a^m b^n c^p \mid m = n = p \}$ — NRL
28. $L = \{ a^m b^n c^p \mid m + n = p \}$ — NRL
29. $L = \{ w c w^R \mid w \in (a+b)^* \}$ — DCFL — NRL

(w: string, c: character str, w^R: string reverse)

↑ Reduced to same first & last symbol

$L = \{ wcw^R \mid w, c \in (a+b)^+ \}$ RL

$L = \{ wcw^R \mid w, c \in (a+b)^* \}$ RL

$L = \{ cwwR \mid w \in \Sigma^* \}$ NRL

$L = \{ ww^Rc \mid w \in \Sigma^* \}$ NRL

$L = \{ ww^R \mid w \in \{a,b\}^* \}$ (same for $w \in \Sigma^+$) NPDA / NRL

$L = \{ ww \mid w \in \Sigma^* \}$ CSL (also for $w \in \Sigma^+$) / NRL

$L = \{ w \in \Sigma^* \mid |w|_a = |w|_b \}$ NRL

$L = \{ wcw^R \mid w \in (a+b)^* \}$ NRL.

$L = \{ w \mid |w|_a \bmod 3 \leq |w|_b \bmod 3 \}$ $(0,1,2) \leq (0,1,2)$ RL

$L = \{ a^{n^2} \mid n \geq 1 \}$ NRL

$L = \{ a^{2^n} \mid n \geq 1 \}$ NRL

$L = \{ a^i b^{2j} \mid i,j \geq 1 \}$ [Also coz $2j$ is in AP] RL

$L = \{ a^i b^{j^2} \mid i,j \geq 1 \}$ NRL

$L = \{ a^i b^{2^n} \mid i,n \geq 1 \}$ NRL

$L = \{ wwwR \mid w \in \{a,b\}^* \}$ NRL

$L = \{ wxw^R \mid x,w \in \{0,1\}^* \ \& \ |x| = 5 \}$ NRL

$L = \{ xwwRy \mid x,y,w \in \{0,1\}^* \}$ RL

$L = \Sigma^*$ (for any string w can be taken as ϵ).

$L = \{ xwwRy \mid x,y \in (0,1)^*, \ w \in (0,1)^+ \}$ ** RL

$L = (0+1)^* 00 (0+1)^* + (0+1)^* 11 (0+1)^*$

$L = \{ cwwR \mid c,w \in (0,1)^+ \}$ ** NRL

'00100' $\in L$ but ~~RL = (0+1)*(00+11)~~ can't be represented

In (47) X & Y cover up for all except 1 char of w. However in (48) we can't say that all ww^R will have last 2 chars same (i.e 00 or 11).

$L = \{ ww^Rc \mid w,c \in (0,1)^+ \}$ NRL

Same reason as (48). Ideally 10|10|10 $\in L$, but if we try to reduce L as $00(0+1)^* + 11(0+1)^*$, then above str $\notin L$.

$L = \{ wcw \mid w \in (a,b)^* \}$ CSL [Once w is stored in a stack, it can be read back only in reverse order.]

$L = \{ wxw \mid w,x \in (a,b)^+ \}$ CSL

10|10|1010 $\in L$ & can't be covered by $1(0+1)^*1 + 0(0+1)^*0$

$L = \{ wxwy \mid w,x,y \in (a+b)^+ \}$ RL

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

RL / NRL

LIST

classmate Date Page

(53) $L = \{ xwywy \mid w, x, y \in (a+b)^+ \}$ RL

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

(54) $L = \{ wxyw \mid w, x, y \in (a+b)^+ \}$ CSL

can't be restricted to same first & last char.

(55) $L = \{ xww \mid w, x \in (a+b)^* \}$ RL

for $w = \epsilon$ $L = \Sigma^*$ means everything already contained in L.

(56) $L = \{ xww \mid w, x \in (a+b)^+ \}$ CSL

w can be read in reverse only from 1 stack.

(57) $L = \{ xww^R \mid w, x \in (a+b)^+ \}$ NPDA

CFL

REGULAR EXPRESSIONS

> The simplest way of representing a regular language is known as —.

> An expression of strings which are constructed using the operations (* , · , +) is known as

KLEEN CLOSURE ← (*) , CONCAT ← (·) , POSITIVE CLOSURE OR UNION ← (+)

Eg : $r = a^*$, $r = a^+$, $r = a \cdot b$. ($a \cdot b \neq b \cdot a$)

Types of RE :

1. RESTRICTED REG. EXPR.

   Use only (* , · , +)
   ↳ Default.

2. SEMI RESTRICTED REG. EXPR.

   Use only (* , · , + , $\cap$)

3. UNRESTRICTED REG. EXPR.

   Use ~~on~~ (* , · , + , $\cap$, ~)
   ↳ Negation.

- Algebraic expression represent the numerical value.
  Eg : $0(1+0) = 0$
- Regular expression represent the language.
  Eg : $0(1+0) \Rightarrow \{01, 00\}$

→ 'n' state DFA's that can be constructed with a designated initial state over the alphabet Σ containing 'm' symbols

$$2^n \times n^{n \times m}$$

If initial state not designated, then #DFA = $n \cdot 2^n \, n^{n \times m}$

2-state DFAs with designated initial state over $\Sigma = \{a, b\}$ that

(i) accept empty language = 16 + 4 = 20

(ii) accept universal language = 16 + 4 = 20

(iii) accept non empty language = Total − 20 = $2^4 \times 2^2 - 20 = 44$

$$n \times 2^n \times n^{n \times m}$$

- n → transitions
- selecting initial state
- 2^n subsets of n states. Subset contains Final state

δ	Σ_1	Σ_2	...	Σ_m
q_1	$*_n$			
q_2				
⋮				
q_n	[q]			

can be filled with 1 of any n states.

→ NFA : $(Q, \Sigma, \delta, q_0, F)$

$q_0 \in Q$, $F \subseteq Q$ $\quad \delta : Q \times \Sigma \longrightarrow 2^Q \quad (P(Q))$

FA has 0/1/more transitions for i/p symbol from any state — .

Accepting power (# langs. accepted) : NFA = DFA.

DFA more efficient than NFA. (In terms of response).

NFA more powerful than DFA. (In terms of construction).

NFA is an incomplete system as it responds to valid strings only.

No concept of dead state ϕ.

(ENFA is also an NFA).

Equivalence b/w NFA & DFA:

DFA m states $M = (Q, \Sigma, \delta, q_0, F)$

NFA n states $N = (Q^*, \Sigma, \delta^*, q_0^*, F^*)$

$1 \le m \le 2^n$ (DFA) States.

n-state NFA when converted to DFA can give 2^n new states at max.

- **NFA → DFA conversion :**

(i) $q_0 = q_0^*$

(ii) Construct state transition table for ease & less chances of error.

(iii) For DFA, start with q_0 in it's transition table with Σ. Find the resultant state for each i/p symbol by Union of resultant states in NFA.

$$\delta(q_1 q_2, a)_{DFA} = \bigcup_{i=1,2} \delta^*(q_i, a)$$

Repeat the above for each of new generated states in DFA. Stop when no new states in DFA appear.

(iv) F = all states in F^* that contain any of the final states of NFA (F^*).

(v) Check for minimization if reqd.

(vi) When transition is missing for an i/p symbol in NFA, make that go to a DFA's dead state (ϕ).

Eg:

δ^*	a	b
→ q_0	q_0, q_1	q_0
q_1	q_2, q_0	q_1
(q_2)	q_1, q_2	q_2

NFA

⇒

δ	a	b
→ q_0	$q_0 q_1$	q_0
$q_0 q_1$	$q_0 q_1 q_2$	q_1
($q_0 q_1 q_2$)	$q_0 q_1 q_2$	$q_0 q_1 q_2$

DFA

NOTE: Using transition table is more error free than DFA drawing. Also, T. Table is easier to minimize.

NFA Complement :

Does not exist.

$NFA^c \neq$ Complement of lang. accepted by NFA.

→ **ε-NFA** $(Q, \Sigma, \delta, q_0, F)$

$q_0 \in Q$ $F \subseteq Q$ $\delta: Q \times \{\Sigma \cup \{\epsilon\}\} \to 2^Q$.

Every NFA is an ε-NFA

Inclusion / Exclusion of ε-transition to NFA, doesn't affect lang. of NFA.

ε-NFA provides programming convenience

ε-Closure (q): Set of all states which are @ 0 distance from q. OR set of reachable state from q along ε labelled transition paths.

Also, every state is at 0 distance from itself.

ε-closure properties: (i) ε-Closure (ϕ) = ϕ

(ii) $\epsilon\text{-closure}(q_0 q_1 \cdots q_n) = \bigcup_{i \ge 0} \epsilon\text{-closure}(q_i)$

(iii) Q = total states & $q \in Q$, ε-closure (q) is an non empty finite subset of Q.

ε-closure(q) is a finite set & is hence closed.

- **ε-NFA → NFA conversion :**

states remain same. $|Q^*| = |Q|$

q_0 stays same. $q_0^* = q_0$

Final states change

$(Q, \Sigma, \delta, q_0, F)$ (ε NFA) → $(Q^*, \Sigma, \delta^*, q_0, F^*)$ (NFA)

(i) Draw the transition table (blank) for NFA containing all states of ε-NFA & Σ.

(ii) Calculate ε-closure of all states.

(iii) Fill NFA T.T using :-

$\delta^*(q, a) = \epsilon\text{-clo}(\delta(\epsilon\text{-clo}(q), a))$ ⇒ $q \to$ ε-clo → a → ε-clo → □

Transition NFA

$\delta^*(q, a)$ ⇒ q —ε-clo→ —a→ —ε-clo→ □

(iv) F^* : Every state in ε-NFA whose ε-closure contains the final state of ε-NFA is a final state in NFA.

- ε NFA ⟶ DFA conversion :

$M = (Q, \Sigma, \delta, q_0, F)$ ε-NFA $N = (Q^*, \Sigma, \delta^*, q_0^*, F^*)$ DFA

(i) Stat with an empty T.T for DFA, with q_0^* & Σ.

(ii) Initial state $q_0^* = \epsilon\text{-closure}(q_0)$

(iii) For every new state q that appears in T.T. of DFA

$\delta^*(q, x) = \epsilon\text{-closure}(\delta(q, x))$

Repeat till no new state appears.

(iv) F^* : Every state in DFA, which contains as its subset a final state of ε-NFA, belongs to F^* in DFA.

(v) ϕ can also be present as a resulting state. Minimize DFA if rqrd.

→ EQUAL / NON DISTINGUISHABLE STATES : Two states P & Q are said to be equal if both $\delta(P, x)$ & $\delta(Q, x)$ go to either F or NF for i/p string x, $\forall x \in \Sigma^*$

- Equality exists b/w pair of F or NF states.
- Two final (or) nonfinal states need not be equal.

Two states X & Y are UNEQUAL / DISTINGUISHABLE if there is atleast one string s, such that one of $\delta(X, s)$ & $\delta(Y, s)$ is accepting & the other isn't.

→ DFA MINIMIZATION

Two states X & Y in a DFA, can be combined into one state XY if they are non-distinguishable.

A DFA is minimal iff all states are distinguishable.

(I) State Equivalence Method. (State Equivalence Method / Table Filling Method)

(i) Delete all states that are unreachable from q_0.

(ii) Draw the transition table.

(iii) Find the 0-equi, 1-equi, .. sets. till we find two equivalent ~~states~~ set X_i & X_{i+1} to be exactly same.

0-equivalent set : Partition of F & NF states.

To check if 2 states X & Y in a partition belong to the same group in X_i equivalent set, $\delta(P, s)$ & $\delta(Q, s)$ should belong to the same group in X_{i-1} equivalent set.

[YDF formal lang]

For each partition in P_k, divide the states in P_k into two diff. partitions if they are k-distinguishable. Two states within this partition X & Y are k-distinguishable if there is an i/p s s.t. $\delta(X, s)$ & $\delta(Y, s)$ are (k-1) distinguishable. Repeat till $P_k \neq P_{k-1}$

(iv) (*) After minimization, check if any state unreachable from q_0 is present & eliminate.

DECIDABLE PROBLEMS : Problems for which there exists an algo to solve it.

ISOMORPHIC MACHINES: Two FSMs M1 & M2 are isomorphic to each other iff one can be obtained from the other by replacing the states. i.e. iff they :

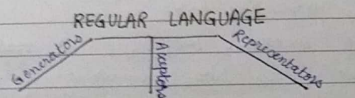
(i) accept the same lang. (ii) Contain same no. of states

(iii) have same properties (transitions).

Isomorphic machines are equal, but equal machines need not always be isomorphic.

Equal Machines:
$L(M1) = L(M2)$

Equal machines need not contain same no. of states but still accept same lang.



REGULAR GRAMMAR = FINITE AUTOMATA = REGULAR EXPRESSIONS

→ **REGULAR EXPRESSIONS**

An expression of strings with regular operators $(\cdot, *, +)$ is — Generates regular language (i.e. one R.E. generates only one Reg. L.)

$* > \cdot > +$ PRECEDENCE

Semi restricted R.E. use operators $(*, \cdot, +, \cap)$

Unrestricted RE use operators $(*, \cdot, +, \cap, \sim)$ ← Negation

R.E.	L
ϕ	$L = \{\}$ empty language
ϵ	$L = \{\epsilon\}$
$w_1 + w_2$	$L = \{w_1, w_2\}$
a	$L = \{a\}$

NOTE: 1 Reg. lang. can be generated from more than one R.E.

$r_1 = 0^*$ $r_2 = 0 + 0^*$ $r_1 \neq r_2$ but $L(r_1) = L(r_2)$.

> If r_1, r_2 are two R.E., then $r_1^*, r_1^+, r_1 \cdot r_2, r_1 + r_2$ are also RE.

$L(r_1^*) = L(r_1)^*$

$L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$

→ **Properties :**

- $\phi + R = R + \phi = R$
- $\phi \cdot R = R \cdot \phi = \phi$
- $\phi^+ = \phi$
- $\phi^* = \{\epsilon\}$.
- $\epsilon \cdot R = R \epsilon = R$
- $\epsilon^+ = \{\epsilon\}$
- $\epsilon^* = \{\epsilon\}$.
- If $r = r^+ = r^* \Rightarrow r = \epsilon$.

- $r^{***\ldots} = r^* = \epsilon + r^+$
- $r^{*+} = r^*$
- $r^* \cdot r^* = r^* = (r^*)^*$
- $r^{*+*} \cdot r^* = r^*$
- $r^{*+*} \cdot r^+ = r^+$
- $r^{+++\ldots} = r^+$
- $r^{+*} = r^*$
- $r^* \cdot r^+ = r^+ \cdot r^* = r^+$
- $r^+ \cdot r^+ = r^+ - r$

- If r_1 & r_2 are two RE, then

(i) $(r_1 + r_2)^* = (r_1^* + r_2)^*$ $(r_1^* + r_2^*)^*$ $(r_1 + r_2^*)^*$ $(r_1^* \cdot r_2^*)^*$

$r_1^* (r_2 r_1^*)^*$ $r_2^* (r_1 r_2^*)^*$ $(r_1^* r_2)^* r_1^*$

(ii) $(r_1 r_2)^* r_1 = r_1 (r_2 r_1)^*$ Shifting rule.

(iii) r^+, r^* represent infinite language except for ϕ & ϵ.

→ **Algebraic properties of RE:**

1. Closure

RE satisfy C.P wrt $(\cdot, +, *)$. If r_1, r_2 are two RE, then are all RE

① $r_1 \cdot r_2$ ② $r_1 + r_2$ ③ r_1^*, r_2^*

2. Associativity

RE satisfy associativity w.r.t $(+, \cdot)$

$r_1 + (r_2 + r_3) = (r_1 + r_2) + r_3$

$r_1 (r_2 \cdot r_3) = (r_1 r_2) r_3$

7. Annihilator $R\phi = \phi R = \phi$

8. Commutative

RE's satisfy commutativity wrt $(+)$ but generally not with $(\cdot)$

① $r_1 + r_2 = r_2 + r_1 = \{r_1, r_2\}$

② $r_1 \cdot r_2 \neq r_2 \cdot r_1$

4. Distributive

RE satisfy distr. wrt $(+, \cdot)$ [ie. concatenation then union] but not (union then concat).

① $r_1 (r_2 + r_3) = r_1 r_2 + r_1 r_3$

$(r_2 + r_3) r_1 = r_2 r_1 + r_3 r_1$

② ~~$r_1 + (r_2 + r_3) \neq r_1 r_2 + r_3$~~

$r_1 \cdot r_2 + r_3 \neq r_1 \cdot r_2 + r_1 \cdot r_3$

$r_1 + r_2 \cdot r_3 \neq r_1 \cdot r_3 + r_2 \cdot r_3$

5 Identity

$+ \longrightarrow e = \phi$ $\quad r + \phi = r = \phi + r$

$\cdot \longrightarrow e = \epsilon$ $\quad r \cdot \epsilon = \epsilon r = r$.

6 Idempotent

RE satisfy idempotent wrt union but not concat.

① $r + r = r$ $\quad$ ② $r \cdot r \neq r$.

→ Algebraic laws for languages:

For any languages L, M, N ; wrt $\cup, \cap, \cdot$

① Associativity

$L \cup (M \cup N) = (L \cup M) \cup N$

$L \cap (M \cap N) = (L \cap M) \cap N$

$L \cdot (MN) = (LM) N$

② commutative

$L \cup M = M \cup L$

$L \cap M = M \cap L$

$L \cdot M \neq M \cdot L$ (generally).

③ Distributive

$L(M \cup N) = LM \cup LN$ $\quad (M \cup N) L = ML \cup NL$

Generally, $L(M \cap N) \subseteq LM \cap LN$ & $(M \cap N) L \subseteq ML \cap NL$

④ Identity.

$\cup \longrightarrow \phi$ $\quad L \cup \phi = \phi \cup L = L$

$\cdot \longrightarrow \epsilon$ $\quad L \{\epsilon\} = \{\epsilon\} L = L$

⑤ Idempotent

① $L \cap L = L$ $\quad (L \cup L = ?,\ L \cdot L \neq)$

⑥ Annihilator

$L \phi = \phi L = \phi$

Q. RE for (i) $|w|_a \overset{?}{=} r \pmod n$ $\quad \Sigma = \{a, b\}$

$b^* (ab^*)^r [b^* (ab^*)^n]^*$

(ii) $|w| \overset{?}{=} r \pmod n$

$(a+b)^r [(a+b)^n]^*$

(iii) $a^m b^n$ / $m+n$ = even

m+n even → even + even → $(aa)^* (bb)^*$

m+n even → odd + odd → $(aa)^* a (bb)^* b$

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

(iv) $a^m b^n$ / $m+n$ odd

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

(v) start with 'a' & $|w|_a$ = even

$a (b^* a b^* a b^*)^* b^* a b^*$

(vi) start with 'a' & doesn't contain 2 consecutive 'b's

$(a + ab)(a + ab)^* = (a + ab)^+$

NOTE: $(a+ab)^+ = \{a, ab, aa, aaa, aab, aba, \ldots\}$

$a(ba+a)^* = \{a, aa, aba, aaa, \ldots\}$ ⊗

$a(ba+a)^*$ isn't the same.

a ☐☐☐☐☐

We need to generate ab, but not abb
So we can't do $a(b+ba+a)^*$ ✗.
Instead we can separate out ab in first go only.
$(a+ab)(a+ab)^* = (a+ab)^+$

NOTE: While constructing RE for similar kind, we ~~need~~ should write L once & finally check manually for each $w \in L$.

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

(vii) No 2 consecutive a's.

$(b+ba)^* + a(b+ba)^* = (\epsilon+a)(b+ba)^*$

start with b, end with b/a — start with a, end with b/a.

III^{by} $(b+ab)^* + (b+ab)^* a = (b+ab)^*(\epsilon+a)$

(viii) No 2 consecutive a's & start with a.
$a(b+ba)^*$

(ix) No 2 consecutive a's & start with b.
$(b+ba)^+$

IMP.

(vii) No 2 consecutive a's. $(b+ba)^* + a(b+ba)^* = (\epsilon+a)(b+ba)^*$
SW b, SW a/b — SW a, EW b/a

$(b+ab)^* + (b+ab)^* a = (b+ab)^*(\epsilon+a)$
SW b/a, EW b — SW b/a, EW a

(viii) No 2 consecutive a's & s.w a. $a(b+ba)^*$

(ix) No 2 cons. a's & s.w b. $(b+ba)^+$ [$\because \epsilon \notin L$, $(b+ba)^*$ ✗ & $b(b+ba)^*$ will not generate ba.]

III^{by}

(x) No 2 cons. b's. $(a+ba)^* + (a+ba)^* b = (a+ba)^*(\epsilon+b)$
$(a+ab)^* + b(a+ab)^* = (\epsilon+b)(a+ab)^*$

(xi) No 2 cons. b's & s.w. 'a'. $(a+ab)^+$

(xii) No 2 cons. b's & s.w. b. $b(a+ab)^*$

(xiii) No 2 a's & no 2 b's occur consecutively.
$L = \{\epsilon, a, b, ab, ba, aba, bab, abab, baba, \ldots\}$

L_i	S.W	E.W	RE
(a) $\{a, aba, abab a, \ldots\}$	a	a	$a(ba)^*$ OR $(ab)^* a$
(b) $\{ab, abab, ababab, \ldots\}$	a	b	$(ab)^*$ OR $a(ba)^* b$
(c) $\{ba, baba, bababa, \ldots\}$	b	a	$(ba)^*$ OR $b(ab)^* a$
(d) $\{b, bab, babab, \ldots\}$	b	b	$b(ab)^*$ OR $(ba)^* b$

'ab' term: $(\epsilon+b)(ab)^*(a+\epsilon)$
'ba' term: $(a+\epsilon)(ba)^*(\epsilon+b)$

FA → RE CONVERSION

1. ARDEN'S LEMMA.
2. STATE ELIMINATION

1. ARDEN'S LEMMA.

Works only for DFA & NFA only.

Let P & Q are R.E over Σ.

LEFT VERSION

① If P is free from ε, then

$R = Q + RP$ has UNIQUE SOL^n & sol^n is

$\boxed{R = QP^*}$

② If P contains ε, then it has INFINITELY MANY SOL^{n}S.

- Construction of eq^n $R = Q + RP$:

(i) Equation of R is constructed from incoming edges, i.e. states & i/p Σ symbol that come to R.

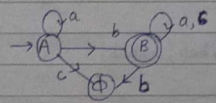
(ii) Include ε in eq^n for initial state.

(iii) RE corresponding to FA is equation of final states.

(iv) ϕ (Dead states) can be removed / ignored & equations for them need not be constructed.

Eg: $A = Aa + C$

$B = Ab + B(a+c)$



$A = \varepsilon a^*$ $\quad B = a^*b + B(a+c)$

$\Rightarrow RE = B = a^*b(a+c)^*$.

NOTE: RIGHT VERSION There is no such thing as right version.

Yet if we want, $R = Q + PR$ with $R = P^*Q$; we should first

(i) eliminate all ϕ states (i.e convert to NFA)

(ii) write eq^n in terms of outgoing edges, i.e. q$_p$ Σ symbol & next resultant state.

(iii) RE for ~~eq~~ system is eq^n of initial state.

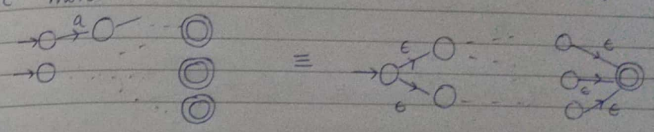
This isn't well tested & might go wrong in unknown way (The above basically tries to find all strings orginating from q_0 as they all will be accepted as it's an NFA.)

Use the left version only.

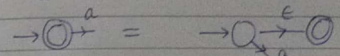
2. STATE ELIMINATION METHOD

Works not only for FA, but for TG.

① Modify to have only one initial & one final state, using ε moves.

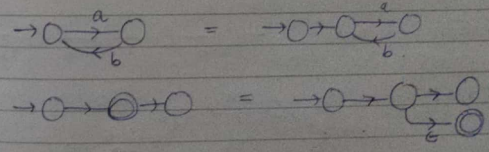


② Modify the system to have diff. initial & final states.

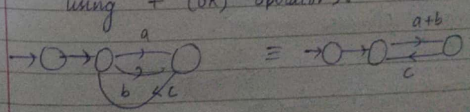


③ (i) If the initial state has an incoming edge, then add a new initial state & connect with ε-move.

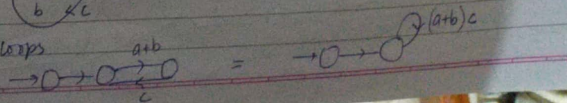
(ii) If the final state has an outgoing edge, then add a new final state (& modify current to non-final) & connect both using an ε-move.

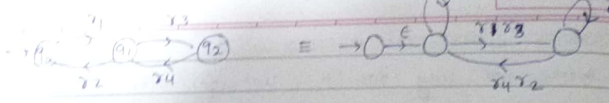


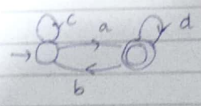
④ Try to eliminate the non final, non initial states one by one. (Also, Parallel edges can be combined to a single edge using + (OR) operator).

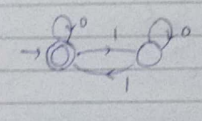


Also, loops



Eg: 


$= c^*a(d^* + bc^*a)^*$
$= (c^*ad^*b)^*c^*ad^*$

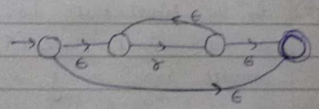

$= (0^* + 10^*1)^*$
$= (0 + 10^*1)^*$

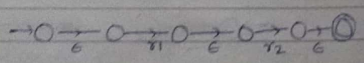
→ RE → FA CONVERSION

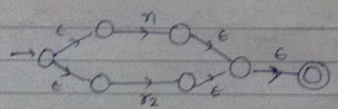
1. METHOD OF SYNTHESIS
2. METHOD OF DECOMPOSITION

1. METHOD OF SYNTHESIS.

Break the RE into pieces based on below operators & construct FA accordingly.

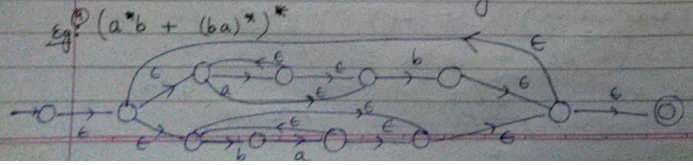
(i) Kleen closure (γ^*) 

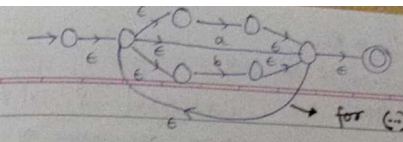
(ii) Concatenation ($\gamma_1 \gamma_2$) 

(iii) Union ($\gamma_1 + \gamma_2$) 

RE → ϵ-NFA → NFA → DFA.

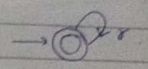
Construction with ϵ moves is easy.

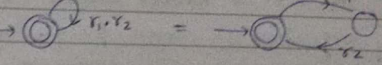
Eg: $(a^*b + (ba)^*)^*$ 

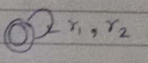
(b) $(a+b)^*$ 

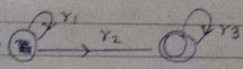
2. METHOD OF DECOMPOSITION / STATE CREATION METHOD.

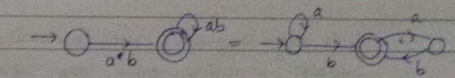
RE → TG → ϵ-NFA / NFA → DFA

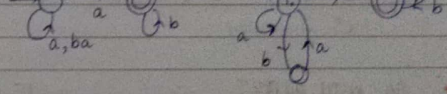
γ^* = 

$(\gamma_1 \gamma_2)^*$ = 

$(\gamma_1 + \gamma_2)^*$ = 

$\gamma_1^* \gamma_2 \gamma_3^*$ = 

Eg: (a) $a^*b(ab)^*$ 

(b) $(a+ba)^*ab^*$ 

→ PUMPING LEMMA (based on pigeon hole)

If L is an infinite regular lang. & $z \in L$ s.t. $|z| \geq n$ for some positive int n ($n \geq 1$), then it can be decomposed as $z = uvw$ s.t.

[n is dependant only on the lang.]

(i) $v \neq \epsilon$ (ii) $|uv| \leq n$

(iii) For every $i \geq 0$, $uv^iw \in L$. [$uw, uvw, uvvw, uvvvw$] $\in L$

All regular lang. satisfy P.L. property. If a lang. doesn't satisfy P.L., then it's non regular. If a lang. satisfy P.L. then it may or maynot be regular.

n = pumping length

To prove that a lang. is non regular, we use P.L. as follows:

① Assume L is a RL.
② Choose a $z \in L$, s.t. $|z| \geq n$
③ Split $z = uvw$ s.t. $|uv| \leq |z|$ & $|v| \neq 0$
④ If we are able to find even one value of i, s.t. $uv^i w \notin L$, then by contradiction, we can say that L is Non Regular.

WEAK FORM OF PUMPING LEMMA : Suppose L is an infinite lang. There are integers p & q, with $q > 0$, so that for every $n \geq 0$, L contains a string of length $p + nq$.
i.e. set of integers, length (L) = $\{|z|, z \in L\}$ contains the arithmetic procession of all integers $p + nq$, where $n \geq 0$.
[Alternative omit $|uv| \leq n$ from S.P.L.]

→ L is any language, not necessarily regular, whose alphabet contains only one symbol. Then L^* is regular.
Eg: $\Sigma = \{0\}$ $L = \{0^i \mid i = \text{prime}\}$ Non regular.
L^* is regular

→ FA WITH OUTPUT

- No need to define the F state & hence no ϕ.
  - MOORE M/c
  - MEALY M/c.

1. MOORE
   O/p is associated with state.
   For I/p of len n, o/p length is n+1. It responds for ∈ also
   $\lambda(\epsilon) = \lambda(q_0)$
   $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ Δ = O/p alphabet $\delta: Q \times \Sigma \to Q$
   # outputs = # states $\lambda: Q \to \Delta$ O/p func^n

2. MEALY
   O/p is associated with transition.
   For I/p of len n, o/p len is n. Doesn't respond to ∈.
   $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$
   Δ = o/p alphabet $\lambda: Q \times \Sigma \to \Delta$ O/p func^n
   $\delta = Q \times \Sigma \to Q$

- Moore & Mealy machines are equivalent in power.

→ Conversion:

I. MOORE → MEALY	II. MEALY → MOORE.
• O/p for a particular transition in resulting mealy m/c is the o/p associated with the state in moore m/c to which the transition goes to.	• Start with q_0 & create states for (transitions & o/p); i.e. if $q_0 \xrightarrow{a/z_1} q_2$ then create a new state as (q_2, z_1) with edge from q_0 on 'a'. Go on creating states, till for each state all transitions are present
• # states remain same.	• Mealy → Moore N states → MXN states M o/p
• Can be done using both T.D. & T.T.	• Use T.D. In case of T.T., the unique pairs of (state, o/p) in T.T. is the list of states in Moore machine. (Possibly)

→ GRAMMAR

$G = (V, T, P, S)$

(Caps) V = Variables / Non Terminals P = Production rules
(Small case) T = terminals S = Start symbol

Every grammar generates only one lang. A lang. can be generated by more than one grammar.

Process of driving a string from G using start symbol is Derivation.
Geometrical representation of derivation is called Syntax Tree / Parse tree / Derivation tree.
All intermediate prod^{n}s involved during the derivation are called Sentential form.

BNF (Backus Naur Form)
$A \to \alpha_1$, $A \to \alpha_2$ ⟹ $A \to \alpha_1 / \alpha_2$ (BNF)

- **Recursive Grammar:** G is recursive iff atleast one prodⁿ contains the same variable in both LHS & RHS. iff it generates infinite number of strings.

  Recursive prodⁿ : Eg: ① $S \rightarrow aSb/ab$.
  ② $A \rightarrow Aa/b$.

  Left Recursion : $A \rightarrow Aa$
  Right Recursion : $A \rightarrow aA$.

- **Linear Grammar:** A G is linear if it contains only one variable in LHS and atmost one variable in RHS. All regular G are linear but all linear aren't regular.

TYPE-3 OR REGULAR G	TYPE-2 OR CONTEXT FREE	TYPE-1 OR CONTEXT SENSITIVE	TYPE-0 OR RECURSIVE ENUMERABLE OR UNRESTRICTED G.
$A \rightarrow aB/a$ (RLG) OR $A \rightarrow Ba/a$ (LLG)	$A \rightarrow \alpha$	$\alpha \rightarrow \beta$	$\alpha \rightarrow \beta$
$A, B \in V$ $a \in T$ $a \in T^*$	$\alpha \in (V+T)^*$ Single var in LHS	$\lvert\alpha\rvert \leq \lvert\beta\rvert$ $\alpha, \beta \in (V+T)^*$ ϵ-free	$\alpha \in (V+T)^+$ $\beta \in (V+T)^*$
Either RLG or LLG but not both	Non preserving Long living G	Length increasing G	
Equivalence b/w LLG & RLG exists			

REGULAR GRAMMAR.

Types of regular grammar [Left linear grammar / Right linear grammar

Conversion FA → RG

I. FA → RLG

1. Initial state is the start symbol. # variables = # states.
2. If $\delta(A, \alpha) = B$ is a transition, then the corresponding production is $A \rightarrow \alpha B$
   If B is a final state, then add $A \rightarrow \alpha$ OR $B \rightarrow \epsilon$.

NOTE: In case of NFA, if a state has no outgoing edges (i.e transitions) then :

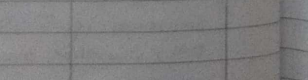
(i) if it's a final state, then add $A \rightarrow \epsilon$
(ii) if non-final, then it is a dead state & that variable will not generate any terminals, & so can be skipped from the grammar.

II. FA → LLG.

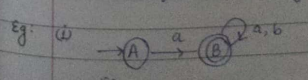
FA $\xrightarrow{\text{Reverse}}$ FA^R → RLG $\xrightarrow{\text{Reverse}}$ LLG

(L) (L^R) (L^R) $(L^R)^R$

Reversal of G: Reversal of RHS of prod^ns.
Reversal of FA: Exchange initial & final states & Reverse transitions.

Eg: ① 

FA

FA^R: 

① RLG : 
(with start symbol B) $B \rightarrow aB/bB/aA$ $A \rightarrow \epsilon$

② LLG : $B \rightarrow Ba/Bb/Aa$ $A \rightarrow \epsilon$
i.e $B \rightarrow Ba/Bb/a$ Ans

(L^R)

(ii) → (S) —a→ (A) —b→ (B)

S → aA
A → bB
B → ∈

(iii) → (S) —a→ (A) —b→ (B) ⟲ b

S → aA
A → bB
B → ∈ / bB

NOTE : FA (L) → RLG (L) —Reverse→ LLG (LR)

→ Conversion RG → FA

I. RLG → FA

1. Start symbol is the initial state.
2. For prodⁿ S → aA / bS ,
   δ(S, a) = A & δ(S, b) = S.
3. For prodⁿ S → aA / a , A → ∈ i.e. A is a final state.

II. LLG → FA

LLG (L) —Reverse→ RLG (L^R) → FA (L^R) —Reverse→ FA $(L^R)^R$

I.a. Right Grammar → ∈-NFA.

form (T^*V)

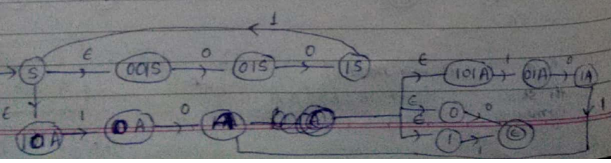
1. If A → α_1 / α_2 is a start production, then δ(A, ∈) = α_1 & δ(A, ∈) = α_2.

   (A) —∈→ (α_1), (A) —∈→ (α_2)

2. δ(aα, a) = α ; a ∈ T
3. For every terminal a ∈ T, δ(a, a) = (∈) (Final state)
4. Repeat step 2 & 3 for all states that have variables only.

eg:

(i) S → 0015 / 10A
A → 101A / 0 / 1



II.a Left Grammar → ∈-NFA

of form (VT^*)

LG (L) —Reverse→ RG (L^R) → FA (L^R) —Reverse→ FA $(L^R)^R$.

1. Reverse RHS of every prodⁿ to reverse LG to RG.
2. Obtain ∈-NFA as per method I.a.
3. Reverse FA by interchanging initial & final states & reversing transition direction.

NOTE : We will be getting NFA / ∈NFA @ many points & we will be rqrd to reverse them. So for reversal, we don't need to change it to DFA (as rqrd with implementation).

→ Conversion RG → RE

1	A → Aα / β	A → $\beta\alpha^*$
2	A → αA / β	A → $\alpha^*\beta$
3	A → α / β	A → $(\alpha+\beta)$
4	A → $\alpha_1 / \alpha_2 / \alpha_3$	A → $(\alpha_1+\alpha_2)^* / \alpha_3$
		or A → $\alpha_1 / (\alpha_2+\alpha_3)^*$

→ CONTEXT FREE GRAMMAR

Every prodⁿ is of form : A → α ; A ∈ V, α ∈ $(V+T)^*$

→ Minimization / Simplification of CFG :

Detect & eliminate :

1. ∈ - productions
2. Unit productions
3. Useless symbols (Also called Reduced CFG, but not minimal)

① ϵ - prod^{n}s.

a. Find null prod^{n}s. Eg: $A \rightarrow \epsilon$

b. Find nullable prod^{n}s Eg: $A \rightarrow (\ ) \rightarrow (\ ) \rightarrow \epsilon$

(Prod^{n}s which directly or indirectly generate ϵ).

c. Eliminate above. Write prod^n with ϵ in strings & w/o ϵ.

Eg:

$S \rightarrow AbaC$
$A \rightarrow BC$
$B \rightarrow b/\epsilon$
$C \rightarrow D/\epsilon$
$D \rightarrow d$

$\Rightarrow$

$A \rightarrow BC/C/B$
$B \rightarrow b$
$C \rightarrow D$
$D \rightarrow d$
$S \rightarrow AbaC/\ baC/\ Aba\ /\ ba$.

② Unit prod^{n}s. $(A \rightarrow B)$

$A \rightarrow B$; $A, B \in V$

Eliminate by some terminal if possible.

Eg:

$S \rightarrow aAb$
$A \rightarrow B/a$
$B \rightarrow b/C$
$C \rightarrow c$

$\Rightarrow$

$C \rightarrow c$
$B \rightarrow b/c$
$A \rightarrow b/c/a$
$S \rightarrow aAb$

③ Useless ~~prod^{n}s~~ symbols.

Variables that aren't involved in derivation of any string —

a. Remove ~~all~~ vars & their prod^{n}s, that can't be reached from the start symbol of G.

b. Remove vars & their prod^{n}s that can be reached from S but don't derive any terminals.

Eg:

$S \rightarrow ABC\ /\ BaB$
$A \rightarrow aA\ /\ BaC\ /\ aaa$
$B \rightarrow bBb\ /\ a$
$C \rightarrow CA\ /\ AC$

$\Rightarrow$

$S \rightarrow BaB$
$B \rightarrow bBb\ /\ a$

b'. Find vars. that derive atleast one terminal. Check RHS of other vars, if their RHS consists of terminals & variables found before (to derive atleast 1 terminal).

→ Popular representations of CFG

1. CNF (Chomsky Normal Form)
2. GNF (Greibach Normal Form).

CFG that is free from ϵ - prod^{n}s can be converted to CNF or GNF.

1. CNF

$A \rightarrow BC/a$; $A, B, C \in V$, $a \in T$

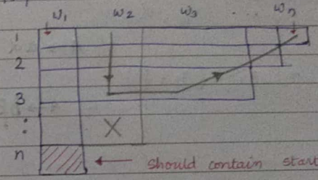
Eg:

$S \rightarrow aSb/ab \implies S \rightarrow ASB/\ AB \Rightarrow S \rightarrow XB/AB$
$A \rightarrow a$ $X \rightarrow AS$
$B \rightarrow b$ $A \rightarrow a$
$B \rightarrow b$.

- If grammar contains ϵ - prod^n, then first eliminate ϵ - prod^n before converting to CNF.
- CYK Membership Algo can be applied to CNF only.

Given $|w| = n$

$O(n^2)$ cells & $O(n)$ time to fill each cell.



Time Complexity $= O(n^3)$.

Eg: Check membership of $w = baaba$ for G. Find member substrings of w.

G: $S \rightarrow AB/BC$
$A \rightarrow BA/a$
$B \rightarrow CC/b$
$C \rightarrow AB/a$.

	b	a	a	b	a
1	B	A, C	A, C	B	A, C
2	A, (S)	B	(S), C	A, (S)	
3	ϕ	B	B		
4	ϕ	(S), C, A			
5	(S), C, A				

B A, C → BA, BC → A S

~~BA ≠ AB~~ (seq matters)

baaba ✓

Substrings: (member of G)

ba, ab, ba, aaba, baaba.

(row # 2 / # len 2) (len 4) (len 5)

	w_1	w_2	w_3	w_4	w_5
1	a	j	o	n	h
2	b	k	m	g	
3	c	l	f		
4	d	e			
5	i				

$i = ae \cup bf \cup cg \cup dh$
$e = jf \cup kg \cup lh$

$a = w_1$ $j = w_2$
$b = aj$ $k = j\theta$ $m = \sigma n$

- Parse tree or Derivation tree of CNF is always binary.

V → T
V → TV
V → TVV
:

2. GNF $A \to VT^*$

$A \to a\alpha$; $\alpha \in V^*$, $A \in V$, $a \in T$.

Useful to convert a CFG to PDA.

→ CONVERSION GNF → PDA :

1. Push the start symbol A on to the stack.
   $\delta(q_0, \epsilon, Z_0) = (q_1, AZ_0)$
2. Push RHS of A as follows :
   [α, β can be ϵ].
   (a) $\delta(q_1, a, A) = (q_1, \alpha)$
   if $A \to a\alpha$ is in G.
   (b) $\delta(q_1, b, A) = (q_1, \beta)$
   if $A \to b\beta$ is in G
3. Add final state with (After all i/ps are added)
   $\delta(q_1, \epsilon, Z_0) = (q_f, \epsilon)$

→ For a string of length 'n', # steps rqrd. to generate string :
   CNF → (2n-1)
   GNF → n

→ CSL : $\{a^n b^n c^n \mid n \geq 1\}$
G: S → A
A → aABc / abc
bB → bb
cB → Bc

classmate
Date
Page

DPDA V/s NPDA

DPDA is a PDA with the extra property that :

For each state in PDA, and for any combination of a current i/p symbol & a current stack symbol, there is at most one transition defined.

In DPDA, there is at most one legal sequence of transitions that can be followed for any input.

This doesn't preclude ϵ-transitions, as long as there is never a conflict b/w following the ϵ-transition or some other transition. However, there can be at most one ϵ-transition that could be followed at any one time.

Non Regular lang containing ϵ can be accepted by DPDA with final state & not empty stack.

$DPDA_{\text{empty stack}} \subseteq DPDA_{\text{final state}}$

$NPDA_{ES} = NPDA_{FS}$

DPDA
NPDA
are not equ

configuration

DPDA V/s NPDA.

k = Z_0, then ⇒

response. NPDA is accepted.

tack.

to make a choice
& a stack symbol.
using true i/p

e exactly those of
have the

a prefix of
guage.

PDA

$M = (Q, \Sigma, \Gamma, z_0, \delta, q_0, F)$

$\delta : Q \times (\Sigma \cup \epsilon) \times \Gamma \longrightarrow Q \times \Gamma^*$ DPDA

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

z_0 = topmost stack element ($z_0 \in \Gamma$) [start symbol]

Γ = set of stack symbols. $F \subseteq Q$

NPDA goes to more than 1 state for same configuration

$\delta(q_0, a, z_0)$.

current state, i/p alphabet, top of stack.

DPDA v/s NPDA.

PDA = FA + 1 stack.

- One convention is to say that if Top of stack = z_0, then ⇒ stack empty.
- DPDA is more efficient than NPDA in terms of response. NPDA is more powerful than DPDA in terms of # langs. accepted.

Langs (DPDA) ⊂ Langs (NPDA)

ID: $\delta(q_0, a, z_0) = (q_1, az_0)$

Acceptance by PDA: (i) Acceptance by empty stack.
(ii) Acceptance by final state.

DPDA: A PDA is deterministic if it never has to make a choice of move for a given state, i/p symbol (inc. ϵ) & a stack symbol. Also, it never has a choice b/w making a move using true i/p and a move using ϵ.

- For DPDA, langs. accepted by empty stack are exactly those of the langs. accepted by final state that have the prefix property.

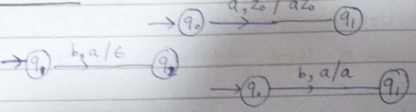
Prefix Property: No string in the lang is a proper prefix of another string in the language.

lang. accepted by empty stack ⊂ # langs. accepted by final state.

For NPDA, langs. accepted by both are same.

- PDA OPERATIONS :

1. PUSH
2. POP
3. SKIP



Whenever the PDA encounters a dead configuration i.e. which is not present, then machine will halt.

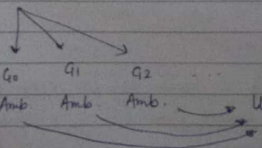
Is every regular language DCFL? (If can be unambiguous grammar. Check PK)

L(ES) = L(FS) ∧ has Prefix Property.

→ DPDA languages have unambiguous CFG. DPDA languages lie strictly between regular languages and CFL.

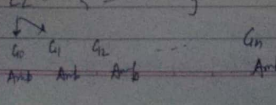
- Regular grammar can be ambiguous. Regular language can never be ambiguous.

- L1 = { . . . } Unambiguous.



- If a language is unambiguous, then there is one unambiguous G for it. All the corresponding ambiguous G can be converted to unamb G.

L2 = { . . . } Ambiguous. Inherently amb. grammar.



(All G are amb.)

Eg: L = a* G1: S → aS/ϵ Unamb. G2: S → aS/aa/ϵ Amb.

G3: S → a/aS/aa/ϵ Amb. G4: S → aS/a/ϵ Amb.

Ambiguous grammar :→ ∃a string that has more than 1 LMD or parse tree.

- Every ambiguous regular grammar can be converted to unambiguous.
- Languages above DCFL are itself ambiguous (can be both amb. & unamb.).

Inherently Ambiguous Lang : A lang that admits only ambiguous grammars.

- All the regular languages are accepted (by final state) by DPDA's and there are non regular languages accepted by DPDA's. DPDA languages lie strictly b/w regular languages & context free languages.

→ FOR DPDA: LE ⊂ LF (LE: lang. accepted by empty stack; LF: langs accepted by final state.)

FOR PDA: LE = LF

LE = LF ∧ (L has PP)

$L = \{ xy \mid |x| = |y|, x \neq y, x, y \in \{0,1\}^* \}$ CFL

(0) $L = \{ a^n b^m c^n d^m \mid n, m \geq 1 \}$ ⊗CFL.

(a) $L = \{ a^n b^k \mid n \leq k \leq 2n \}$ CFL

$S \to aSb \mid aSbb \mid \epsilon$

(b) $L = \{ a^i b^j c^k \mid (i \leq j) \text{ or } (j \leq l), j = k \}$ CFL

i, j can be anything & $j = k$

(c) $L = \{ a^i b^j c^k \mid i = j, j < k \}$ NCFL

(d) $L = \{ a^m b^n c^n d^m \mid m \neq n \}$ NCFL

(e) $L = \{ a^i b^j c^k \mid \text{if } (i = j) \text{ then } k \text{ is even} \}$ CFL

→ Unlike DFA, DPDA doesn't mean that we will have transition for each symbol defined. Whenever a PDA encounters a dead configuration i.e. which is not present, then the machine will halt.

→ PREFIX PROPERTY : The lang. L has prefix property iff it is not possible to find different string x, y in L such that one is proper prefix of other.

Eg: $L = \{ab, ba\}$ $L = \{a^n b^n \mid n \geq 0\}$ $L = \{ wxw^R \mid w \in (a+b)^+ \}$

Below don't have P.P.

$L = \{ a^m b^n \mid m > n \}$ $L = \{ w \mid w \in (a+b)^+, |w|_a = |w|_b \}$

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

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

Q. Check whether the following languages are CFL or not?

1. $a^{m+n} b^n c^m \mid n, m \geq 1$ — DCFL

$a^m a^n b^n c^m$

2. $a^m b^{m+n} c^n \mid n, m \geq 1$ — DCFL

3. $a^m b^n c^{m+n} \mid n, m \geq 1$ — DCFL.

~~$a^n b^n c^{m+n}$~~

$a^m b^n c^n c^m$

4. $a^m b^m c^n d^n \mid n, m \geq 1$ — DCFL

5. $a^m b^n c^m d^n$ — XCFL

6. $a^m b^n c^n d^m \mid n, m \geq 1$ — DCFL

7. $a^m b^i c^m d^k \mid m,$ — DCFL.

8. $a^m b^n \mid m \geq n$ — DCFL

9. $a^n b^{2n} \mid n \geq 1$ — DCFL

10. $a^n b^{n^2} \mid n \geq 1$ — XCFL.

Unbounded comparison n & n^2.

$a^n b^{2^n} \mid n \geq 1$ — XCFL.

12. $ww^R \mid w \in (a,b)^*$ — NDCFL

Non deterministic CFL.

13. $ww \mid w \in (a,b)^*$ — XCFL.

w once stored in stack can be retrieved only in reverse & hence we can't compare with the other w.

VISION
Date | Page No.

(14.) $a^n b^n c^m \mid n > m$ — X CFL

We can think of pushing 2 a's for each a in i/p. Then pop 1a for 1b in i/p. Then compare if the rest of a's in stack is more than c's. But No.
This doesn't exactly check if $a^n b^n$ is there.
Also $w = aabc$ will fail the above algo.

(15.) $a^n b^n c^n d^n \mid n \leq 10^{10}$ — RL, DCFL.

(16.) $a^n b^{2n} c^{3n} \mid n \geq 1$ — X CFL

To compare a^n & b^{2n} we exhaust everything & nothing is left to compare with c^{3n}.
Same as (14). $w = abbccc$, $abcccc$

(17.) $xcy \mid x, y \in (0,1)^*$ — RL, DCFL

(18.) $xx^R \mid x \in (a,b)^*, |x| = l$ — RL, DCFL
$|x| = l$
finite language.
(strings of len $2l$).
diff. of len l (i.e. distinct x) $= 2^l$

(19.) $ww w^R \mid w \in (a,b)^*$ — X CFL
ww can't be recognized by a PDA.
$ww w^R$,, .

(20) $a^n b^{3^n} \mid n \geq 1$ — X CFL.

$3^n \longrightarrow$ not in AP.

(21) $a^m b^n \mid m \neq n$ — DCFL

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

$(\epsilon, a/a)$ $(b, z_0/z_0)$

(22) $a^m b^{\mathbf{n}} \mid m = 2n+1$ — DCFL.

Push 1st a, then push 1 a for 2 a's in I/p

Pop 1 a for each b.

(23) $a^i b^j \mid i \neq 2j+1$ — DCFL

(22) + (a) end either b are more or a's are more.

(24) $a^{n^2} \mid n \geq 1$ — X CFL

n^2 can't be computed. Also not in AP

(25) $a^{2^n} \mid n \geq 1$ — X CFL.

(26) $a^{n!} \mid n \geq 1$ — X CFL

(27) $a^m \mid m =$ prime — X CFL

(28) $a^k \mid k =$ even — RL, DCFL.

(29) $a^i b^j c^k \mid i > j > k$ — X CFL

III^r to (16)

(30) $a^i b^j c^k d^\ell \mid i = k$ or $j = \ell$ — NDCFL.

1 PDA compares a's & c's ; other PDA compares b's & d's. Apply U.

(31) $a^i b^j c^k d^\ell \mid i = k$ and $j = \ell$ — X CFL

We can have 2 PDAs check the 2 conditions, but we won't be able to AND them. There is no framework in PDA to AND.

(32) $a^m b^\ell c^k d^n \mid m, \ell, k, n \geq 1$ — RL, DCFL

(33) $a^n b^{4m} \mid n, m \geq 1$ RL, DCFL.

(34) $\{a^3, a^8, a^{13}, \ldots\}$ RL, DCFL.

AP, $d = a^5$.

(35) $\{a^{2n+1} \mid n \geq 1\}$ RL, DCFL

AP:

(36) $a^{n^n} \mid n \geq 1$ X CFL

n^n can't be computed.

(37) $w \mid w \in (a,b)^*$ & $|w| \geq 100$ RL, DCFL

(38) $w \mid w \in \{a,b,c\}^*$ & $n_a(w) = n_b(w) = n_c(w)$

can't compare ∞ a, b X RL

PDA can compare a's & b's but nothing to count c's. X CFL.

also $a^n b^n c^n \subset$ (38) & $a^n b^n c^n$ is CSL.

(38) $w \mid w \in \{a,b\}$, $n_a(w) \geq n_b(w) + 1$

(ϵ, a) or DCFL

(a, z_0) X RL.

Add from surrounding pages.

(9) $L = \{ a^m b^n \mid m > n ;\ m \geq 1, n \geq 0 \}$

DCFL

(10) $L = \{ a^m b^n \mid m < n ,\ m \geq 0,\ n \geq 1 \}$ DCFL

) $L = \{ a^m b^n \mid m = 2n ,\ m, n \geq 0 \}$ DCFL

OR $n = 2m$

) $L = \{ a^m b^n \mid m = 2n+1 ,\ m, n \geq 1 \}$ DCFL

) $L = \{ a^m b^n c^p \mid m = n \text{ OR } n = p ;\ m, n, p \geq 1 \}$

(NPDA) CFN

) $w x w^R \mid w \in (a+b)^*$ DCFL

) $x w w^R \mid w \in (a+b)^*$ NPDA CFL

) $w w^R x \mid w \in (a+b)^*$ NPDA

) $w w^R \mid w \in (a+b)^*$ NPDA

) $L = \{ a^n b^n \mid n \geq 1 \} \cup \{ a^n b^{2n} \mid n \geq 1 \}$ NPDA

DCFL ∪ DCFL

$\{ a^n b^{2n} c^{3n} \mid n \geq 1 \}$ Non CFL

) $\{ a^i b^j c^k d^l \mid i = k \text{ OR } j = l \}$ NPDA

PUMPING LEMMA FOR CFL

VISION
Date | Page No.

Let L be a context free language, & $w \in L$ s.t. $|w| \geq n$ for some positive integer n. Then w can be decomposed as $w = abcde$ such that

(i) $bd \neq \epsilon$

(ii) $|bcd| \leq n$

(iii) for all $i \geq 0$, the string $ab^icd^ie \in L$.

NOTE: (i) Every CFL satisfies PL property. If a lang. L doesn't satisfy PL property then L is not CFL.

If a lang. L satisfies the ~~pr~~ P.L. property then L need not be CFL necessarily.

TURING MACHINE

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Q = set of all states $\quad q_0 \in Q \quad F \subseteq Q$

Σ = finite set of i/p symbols

Γ = complete set of tape symbols. $(\Sigma \subseteq \Gamma)$

B : Blank symbol. B is in Γ but not in Σ.

The blank appears initially in all but the finite number of initial cells that hold i/p symbols.

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

$\delta(q, X) = (p, Y, D)$

- q : current state
- X : tape symbol
- p : next state
- $Y \in \Gamma$: Y is written in the cell being scanned, replacing whatever symbol was there.
- D : direction (Left / Right) — tells the direction in which the head moves.

- The tape is unbounded, so any no. of left & right moves possible.

Types of TM
1. Language Recognizers.
2. O/p Generators
3. Language Generators.

HALT : The state where transition isn't defined is called — It can be final or non final.

Acceptance of TM — After taking i/p, 3 possibilities:
(i) TM may go to final halt.
(ii) TM may go to non final halt.
(iii) TM may go in an infinite loop.

After reading complete i/p string, if the TM goes to :

(i) FH → i/p string is accepted by the TM.

(ii) NFH ⇒ " rejected by the TM.

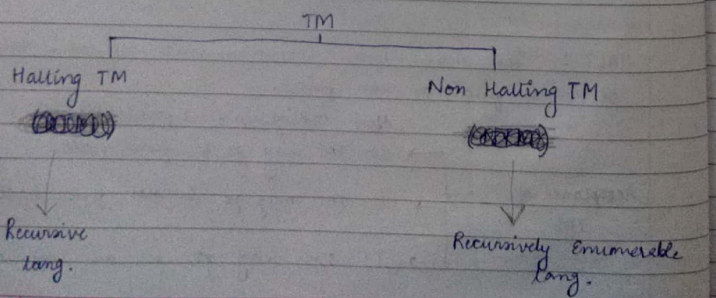
(iii) Infinite loop ⇒ i/p string w is neither accepted nor rejected

The set of languages accepted by a TM is called the RECURSIVELY ENUMERABLE languages or RE langs..

Acceptance
- After reading i/p, TM eventually enters an accepting (final) state.
- Acceptance by halting.
  A TM halts, if it enters a state q, scanning a tape symbol X, and there is no move in this situation, i.e $\delta(q, X)$ is undefined.

- We assume that a TM always halts when it is in an accepting state.
  However, it is not always possible to require that a TM halts even if it doesn't accept.

RE language also called TURING RECOGNIZABLE languages.



(TURING DECIDABLE langs.)

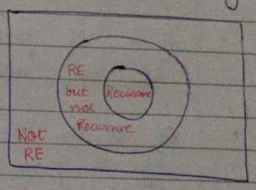
Recursive set and Recursively Enumerable Set :

L for some TM M

If we think of a language L as a 'problem' (as will be the case frequently), ;

L is called decidable if it is a recursive language.
undecidable if it is not a recursive lang.

L_d - Diagonalization lang.
↳ Not RE.
No TM to accept it.



Recursive : TM recognizes the language.
Also tells us when it has decided the i/p string is not in the lang.
TM always halts, regardless of whether it reaches an accepting state.

RE but not Recursive : No guarantee of halting.
Lang. Acceptance in an inconvenient way :- if the i/p is in lang., we will eventually know that, but if the i/p is not in the lang., then the Turing machine may run forever & we shall never be sure the i/p won't be accepted eventually.

Complementation :

- If L is recursive, L^c ($\Sigma^* - L$) is also recursive.

If both L & L^c are RE, then L is recursive & hence L^c is also recursive.

- It is not possible that L and L^c both are RE but not recursive.

- Of all the 9 possible ways to place a lang. L & L^c in fig 1, only following 4 are possible:

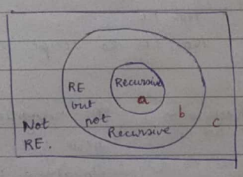


fig 1.

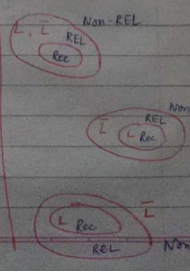
(i) Both L & L^c are recursive.
(ii) Neither L nor L^c is RE. (region c)
(iii) L is RE but not recursive, L^c is not RE (b & c)
(iv) L^c is RE but not recursive, L is not RE (a & b)

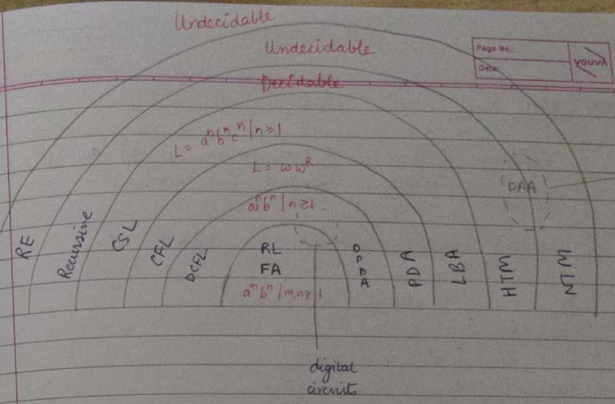
Suppose L_x is not-RE. Then $\overline{L_x}$ can either be non-RE or RE but not recursive.

→ Standard examples :

(i) $L = \{ a^{n!} \; ; \; n \geq 0 \}$
(ii) $L = \{ a^n b^n c^n \mid n \geq 1 \}$.
(iii) $L = \{ a^n \mid n \text{ is prime} \}$
(iv) $L = \{ a^n \mid n \text{ is not prime} \}$
(v) $L = \{ a^n \mid n = m^2, \; m \geq 1 \}$
(vi) $L = \{ ww \mid w \in (a+b)^+ \}$
(vii) $L = \{ w^n \mid w \in (a+b)^+, \; n \geq 1 \}$
(viii) $L = \{ ww^R \mid w \in (a,b)^* \}$

Impossible cases of Comp





- No membership algo for REL.

→ lang. derived by following unrestricted grammar :

(i) $S \to S_1 B$
$S_1 \to a S_1 b$
$bB \to bbbB$
$aS_1 b \to aa$
$B \to \lambda$

$L = \{ a^{n+1} b^{n+k} \mid n > 1 \}$, $k = 1, 1, 3$ }

(ii) $S \to abc \mid aAbc$
$Ab \to bA$
$Ac \to Bbcc$
$bB \to Bb$
$aB \to aa \mid aaA$

$L = a^n b^n c^n \mid n \geq 1$

→ LBA : (CSL)

Input tape is finite. 2 end markers ($, ¢) as left & right end markers. Read & write header.

Restricted non deterministic TM = LBA

Eg: (i) $L = \{ a^n b^n c^n \mid n \geq 1 \}$
(ii) $L = \{ a^{n^2} \mid n \geq 1 \}$
(iii) $L = \{ a^p \mid p \text{ is prime} \}$
(iv) $L = \{ ww \mid w \in (a,b)^+ \}$.
(v) $L = \{ a^n b^n c^{2n} \mid n \geq 1 \}$.

→ Σ^* is countable. 2^{Σ^*} is uncountable

→ $CFL^c = CSL^c = CSL$
Every CSL is Recursive.
Complement of every CFL is Recursive.
However
(i) $L_1 = \{ wcw \mid w \in \{0,1\}^* \}$ CFL
$L_1^c = \{ xy \mid |x| = |y|, x \neq y, x \in \{0,1\}^* \}$ CFL
(ii) $L_2 = \{ a^n b^n c^n \mid n \geq \}$ CSL
L_2^c = CFL (and hence CSL).

→ Let G be an ambiguous grammar and D be its disambiguated version. Let the language recognized by two grammars be L(G) & L(D).

$L(D) = L(G)$

→ Every non-deterministic TM can be converted to an equivalent deterministic TM.

DTM — Deterministic TM
NTM — Non Deterministic TM

Powers of DTM & NTM are same.

If L is a recursive lang., there is atleast 1 TM for L that is halting.

- L_1 is a REL. Then there is an infinite list of TMs $M_0, M_1, M_2, \ldots$ such that $L_1 = \mathcal{L}(M_i)$
- L_1 is a recursive lang. & $M_0, M_1, M_2, \ldots$ a list of TM s.t. $L_1 = \mathcal{L}(M_i)$.
for any $i \geq 0$, does M_i always halt? Maybe, maybe not.
but at least one M_i is halting.
- L_1 = REL, $M_0, M_1, \ldots$ = TMs s.t. $L_1 = \mathcal{L}(M_i)$
for any $i \geq 0$, does M_i halt always?
Maybe, maybe not. If L_1 is recursive (& hence REL), at least one M_i will always halt.
- L_1 = REL but not recursive. $M_0, M_1, \ldots$ = TMs s.t. $L_1 = \mathcal{L}(M_i)$
None of the M_i will be halting TMs.

→ NTM may have more than one choice of next move (but finite) i.e. (state, new symbol, head move) for each state & symbol scanned.

Every NTM can be converted to an equivalent DTM.

Basically NTM ~~is~~ is a TM in which there may be multiple transitions defined for a particular state/i/p combination.

→ A language is called RECOGNIZABLE or REL if it is the language of some TM.

DECIDER — TM that always halts (never goes into an infinite loop).

A language L_1 is called DECIDABLE iff there is a decider M s.t. $L_1 = L(M)$.
These langs are also called RECURSIVE.

→ $L = \{ a^l b^m c^n \mid l \neq m \text{ or } m \neq n \}$
CFL but not DCFL

$L_1 = \{ 0^{n+m} 1^n 0^m \mid n, m \geq 0 \}$ CFL
$L_2 = \{ 0^{n+m} 1^{n+m} 0^m \mid n, m \geq 0 \}$ X CFL
$L_3 = \{ 0^{n+m} 1^{n+m} 0^{n+m} \mid n, m \geq 0 \}$ X CFL

↳ $\subseteq \{ 0^x 1^x 0^x \mid x \geq 0 \}$

$L_4 = \{ ww \mid w \in (0,1)^* \}$ CSL

→ There are some CFLs which are inherently ambiguous. That is all the CFGs generating them are ambiguous.

→ $S \to aSb \mid SS \mid \epsilon$ or $S \to (S) \mid SS \mid \epsilon$

$L(G)$: Balanced parenthesis.
(each left paren has a matching right paren & are well nested).
$L(G)$ = DCFL but not regular

→ Main diff. b/w Recursive & REL is membership property

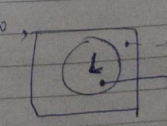
	Membership
Recursive	✓
REL	X (No algo)

THEOREM : If HALTING PROBLEM were decidable, then every REL would be recursive. Hence consequently the halting problem is undecidable.

(However it is recognizable).

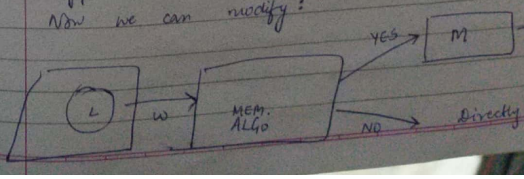
Suppose L = REL and a TM M s.t. $L(M) = L$.

Now,



so M may or may not halt for $w \notin L$.

Suppose we have an algo for membership.
Now we can modify:



This means that if a mem. algo is there, all REL are Recursive, which have already been proved false. So no membership algo is there.
~~no halting~~

⇒ HALT is undecidable.

→ How does a TM halt?
When it reaches a state with dead configuration. Even final states have dead config (it is not difined where to go next).

→ Every TM can be rep. as a string of 0s & 1's.
(Universal TM & encoding of TM).
We can encode all states, δ etc as a string of 1s & using 0 as a seperator.

Not every string of 0's & 1's ~~be~~ might be a TM.

→ Σ = alphabet Σ* = set of all strings
2^{Σ^*} = set of all languages

Σ^* = Countable 2^{Σ^*} = Uncountable

↓
Superset of all TMs

⇒ #TMs < #languages

Any language L_i is a subset of Σ^*
i.e $L_i \subseteq \Sigma^*$

⇒ every lang is countable

→ If S is countably infinite, then 2^S is uncountable.

→ Diagonalisation method to prove that set of all langs are uncountable. MITTO == BLOWN

$\Sigma = \{0,1\}$ Σ^* = countable 2^{Σ^*} = Uncountable

Σ^*	ε	a	b	aa	ab	ba	bb	aaa	...
1)	0	1	0	1	0	0	0	0	
2)	1	0	0	0	0	0	1	0	
3)	1	1	0	0	1	0	0	0	
4)	0	0	0	1	0	0	0	0	
5)	1	0	0	0	0	0	0	0	
6)									

$\overline{00010}$ = 11101
↳ will not be present

Q: P1 = decidable P2 = undecidable.

(a) P1 ~~red~~ P3 P3 D if P1 $\xrightarrow{red}_{D}$ P3 F.

(b) P3 UD if P3 $\xrightarrow{red}_{UD}$ P2 F
P3 can be reduced to something else also which might be simpler than P2 & be D.

Eg: a^* can be accepted by TM
" by FA.

(c) P3 UD if P2 $\xrightarrow{red}_{UD}$ P3

10001...
01110

Reduction (std. many-to-one).

$A \xrightarrow{red} B$

$B \in X$

A can't be harder than X.

Upper bound of $A = X$

$A \leq X$

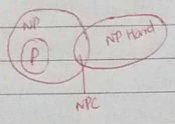
$A \xrightarrow{red} B$

$A \in X$

B can't be easier than X

Lower bound of $B = X$

$B \geq X$



→ $P \subseteq NP \subseteq$ Recursive

→ L = CFL Is $\bar{L}$ also context free? Undecidable.

→ EXPRESSIVE POWER

Single Tape TM ≡ Multi Tape TM.

DTM = NTM.

→ L1 = Recursive L2, L3 = REL but not Recursive.

$L2 - L1 = L2 \cap L1^c$

$= REL \cap Recursive = REL \cap REL$

(but not Rec)

$= REL$

$L1 - L3 = L1 \cap L3^c = Rec \cap (Rel\ but\ not\ rec)^c$

$= Rec \cap Non\ REL$

$= Non\ REL$

→ A lang is recursive iff it can be effectively enumerated in lexicographic order.

→ The state entry problem is given a TM, a state $q \in Q$ and $w \in \Sigma^+$, decide whether or not the state 'q' is ever entered when M is applied to 'w'. This is undecidable.

Given a TM M, whether or not M halts if it started with a blank tape. This is undecidable.

Almost any problem related to RE language is undecidable.

L is Recursively enumerable:

TM exist: M_0, M_1, ...

They accept string in L, and do not accept any string outside L

L is Recursive:

at least one TM halts on L and on $\sum$-L, others may or may not*

L is Recursively enumerable but not Recursive:

TM exist: M_0, M_1, ...

but none halts on all x in $\sum$-L*

M_0 goes on infinite loop on a string p in $\sum$-L, while M_1 on q in $\sum$*-L*

However, each correct TM accepts each string in L, and none in $\sum$-L*

L is not R.E.:

no TM exists

- **Let *M* be a TM that halts on all inputs:**
  - Question: Is L(M) recursively enumerable?
  - Answer: Yes! By definition it is!
  - Question: Is L(M) recursive?
  - Answer: Yes! By definition it is!
  - Question: Is L(M) in r.e – r?
  - Answer: No! It can't be. Since *M* always halts, L(M) is recursive.

- **Let *M* be a TM.**
  - Question: Is L(M) r.e.?
  - Answer: Yes! By definition it is!
  - Question: Is L(M) recursive?
  - Answer: Don't know, we don't have enough information.
  - Question: Is L(M) in r.e – r?
  - Answer: Don't know, we don't have enough information.

- **Let *M* be a TM, and suppose that *M* loops forever on some string *x*.**

  - Question: Is L(M) recursively enumerable?
  - Answer: Yes! By definition it is. But, obviously *x* is not in L(M).

  - Question: Is L(M) recursive?
  - Answer: Don't know. Although *M* doesn't always halt, some other TM *M'* may exist such that L(M') = L(M) and *M'* always halts.

  - Question: Is L(M) in r.e. – r?
  - Answer: Don't know.

  May be another *M'* will halt on *x*, and on all strings! May be no TM for this L(M) does halt on all strings! We just do not know!

- ***Let M be a TM.***

  - As noted previously, L(M) is recursively enumerable, but may or may not be recursive.

  - Question: Suppose, we know L(M) is recursive. Does that mean *M* always halts?
  - Answer: Not necessarily. However, some TM *M'* must exist such that L(M') = L(M) and *M'* always halts.

  - Question: Suppose that L(M) is in r.e. – r. Does *M* always halt?
  - Answer: No! If it did then L(M) would be recursive and therefore not in r.e. – r.