

GATE CSE NOTES

by

UseMyNotes

FA, RL

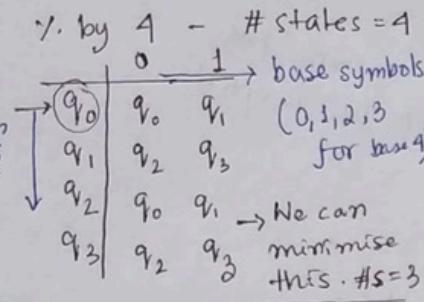
TOC

* DFA = $(\emptyset, \Sigma, S, q_0, F)$ $S: \emptyset \times \Sigma \rightarrow Q$

* # states ($|w|=m : n+2$) ($|w| \leq n : n+2$) ($|w| \geq n : m+1$) $\Sigma = \{a, b\}$
 $(|w| \leq n : n) \quad (n_a = m : n+2) \quad (n_a \geq n : n+1)$ $\xrightarrow{\text{mth symbol from RHS}}$
 $(n_a \leq n : n) \quad (n^{\text{th symbol from LHS is 'a'}} : n+2) \quad \text{is } a \Rightarrow \text{need}$

* Cross product method $\frac{\{A, B\} \times \{C, D\}}{m \quad m} = \frac{\{AC, AD, BC, BD\}}{mm}$ 2^n states for min DFA even when symbols changed in Σ .

* Short-cut ($\Sigma = \{0, 1\}$, DFA for binary number when interpreted as decimal divisible by n)



If ternary number:

| | | |
|-------|-------|-------|
| 0 | 1 | 2 |
| q_0 | q_1 | q_2 |
| q_1 | q_3 | q_0 |
| q_2 | q_2 | q_3 |
| q_3 | q_1 | q_2 |

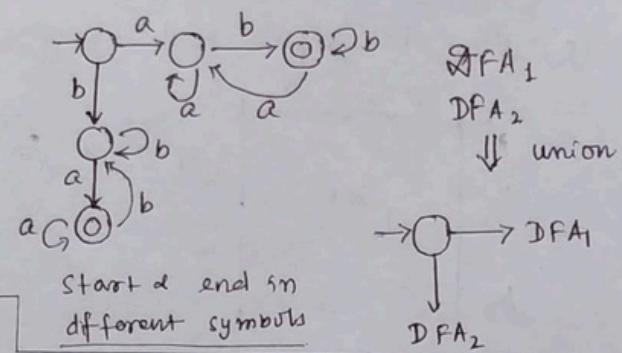
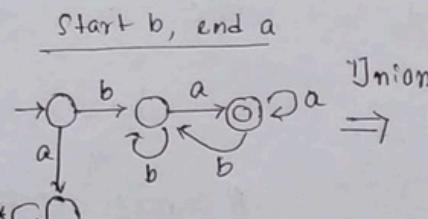
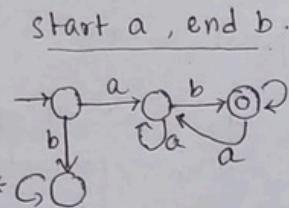
* Complement of DFA!
eg: (Start with a, not start with a)
(even length string, odd length string)
F becomes $Q - F$

* Operations on DFA.

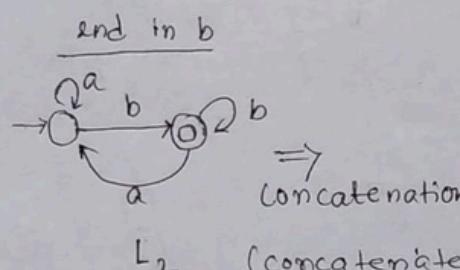
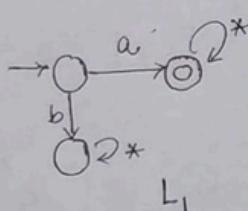
a - b or b - a

- 1) Union (Starts & ends with different symbols)
- 2) Concatenation (Starts with 'a', ends with 'b')
- 3) Cross product (Even # a's, even # b's)
- 4) Complement (not containing a)
- 5) Reversal ($awb \rightarrow bwa$)

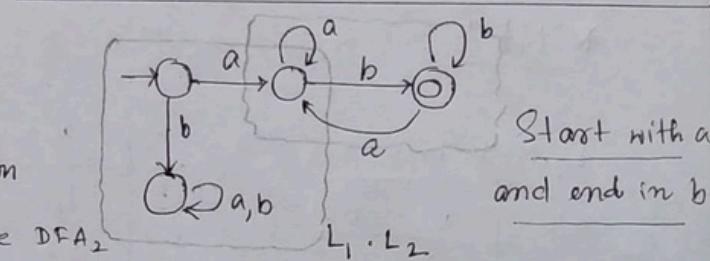
eg ①. $\Sigma = \{a, b\}$



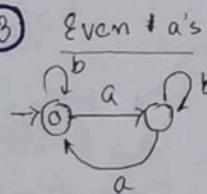
②. Start with a



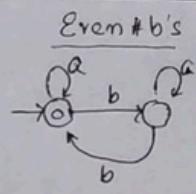
(concatenate DFA₂ at the final state of DFA₁) (Final, & initial merged)



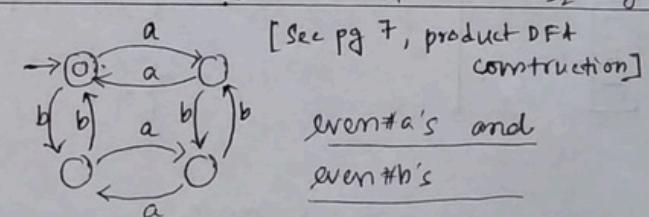
③ Even # a's



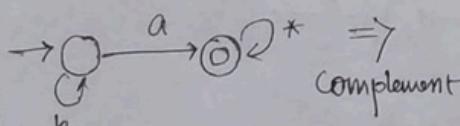
Even # b's



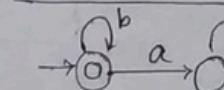
Cross product (of states)



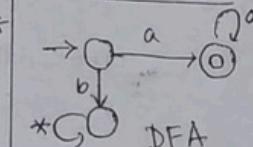
④ Contains a



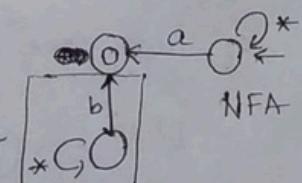
not containing a



Start with a



reverse



* (Q) # 2-state dfa's with designated initial state that accept

| | | | | |
|----------------------------------|-------------------|---|---|-------------------|
| <u>empty language</u> | over $\{a, b\}$. | $\begin{array}{c cc} & a & b \\ \rightarrow x & x/y & x/y \\ y & x/y & x/y \end{array}$ | $\begin{array}{c cc} & a & b \\ \rightarrow x & x & x \\ y & x/y & x/y \end{array}$ | $\Rightarrow 20.$ |
| $\{\} \Leftrightarrow \emptyset$ | | $2^4 = 16$ | $2^2 = 4$ | |

(Q) # 2-state dfa's with designated initial state over $\{a, b\}$.

| | | | | |
|---|---|---|---|-------------------|
| $\begin{array}{c cc} & a & b \\ \rightarrow x & 2^4 & 2^4 \\ y & 2^4 & 2^4 \end{array}$ | $\begin{array}{c cc} & a & b \\ \rightarrow x & 2^4 & 2^4 \\ y & 2^4 & 2^4 \end{array}$ | $\begin{array}{c cc} & a & b \\ \rightarrow x & 2^4 & 2^4 \\ y & 2^4 & 2^4 \end{array}$ | $\begin{array}{c cc} & a & b \\ \rightarrow x & 2^4 & 2^4 \\ y & 2^4 & 2^4 \end{array}$ | $\Rightarrow 64.$ |
|---|---|---|---|-------------------|

(Q) # 2-state dfa's with de. init. state that accept universal language. (Σ^*)

| | | | | |
|---|---|-------------------------------------|---|---|
| <p>init. state has to be a final state to accept the null string.</p> | $\begin{array}{c cc} & a & b \\ \rightarrow x & 2^4 & 2^4 \\ y & 2^4 & 2^4 \end{array}$ | <p>Everything gets accepted</p> | $\begin{array}{c cc} & a & b \\ \rightarrow x & x & x \\ y & x/y & x/y \end{array}$ | <p>everything has to be accepted in x</p> |
| | | | | $\Rightarrow 20.$ |

(Q) # 3-state dfa's with desig. init. state over $\{a, b\}$

| | |
|--|--|
| $\begin{array}{c cc} & a & b \\ \rightarrow x & 3^6 & 3^6 \\ y & 3^6 & 3^6 \\ z & 3^6 & 3^6 \end{array}$ | $3^6 \times (\text{Choosing final state})$ |
| | $= 3^6 \times \left(\sum_{r=0}^3 3C_r \right) = 3^6 \times 2^3 = 5832.$ |

(Q) # m-state dfa's with designated init. state over an alphabet having m symbols

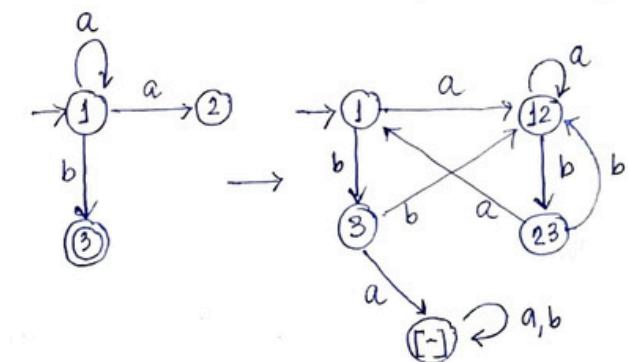
| | | | |
|-------------------|---|---------------------------|---|
| $\rightarrow A_1$ | $\begin{array}{c cc} a_1 & a_2 & \dots & a_m \\ \hline A_1 & & & \\ A_2 & & & \\ \vdots & & & \\ A_n & & & \end{array}$ | $(m^{mn} \times 2^n) A_m$ | Choosing final states \Downarrow |
| | | | # subsets possible for a set of m elements. |

$[U\{\epsilon\}]$ for ϵ -NFA

* NFA = $(\Theta, \Sigma, \delta, q_0, F)$ $\delta: \Theta \times \Sigma \rightarrow$ Power set of Θ . $\Theta = \{A, B\}$

$$\text{or } 2^\Theta = \{\emptyset, \{A, B\}, A, B\}$$

| <u>NFA to DFA</u> | | <u>a</u> | <u>b</u> |
|-------------------|---|---|---------------|
| $\rightarrow 1$ | $\begin{array}{c cc} a & b \\ \hline 1,2 & 3 \end{array}$ | $\xrightarrow{1} \begin{array}{c cc} & a & b \\ \hline 1 & 1,2 & 3 \\ 1,2 & 1,2 & 2,3 \\ [-] & [-] & 1,2 \end{array}$ | \Rightarrow |
| 2 | $\begin{array}{c cc} a & b \\ \hline 1 & 2 \end{array}$ | $\xrightarrow{2} \begin{array}{c cc} & a & b \\ \hline 1 & 1 & 1,2 \\ 1,2 & - & [-] \end{array}$ | |
| 3 | $\begin{array}{c cc} a & b \\ \hline - & 1,2 \end{array}$ | $\xrightarrow{3} \begin{array}{c cc} & a & b \\ \hline - & [-] & [-] \end{array}$ | |

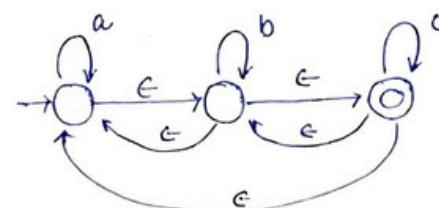


* NFA with ϵ move.

e.g. any combⁿ of a, b, c

DFA: $\rightarrow \bigcup a, b, c$

NFA with:
 ϵ -move





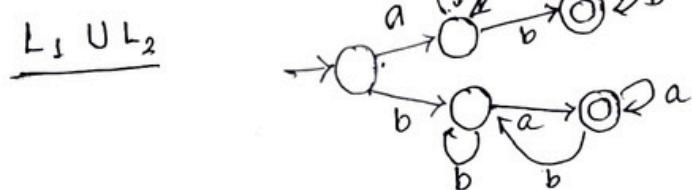
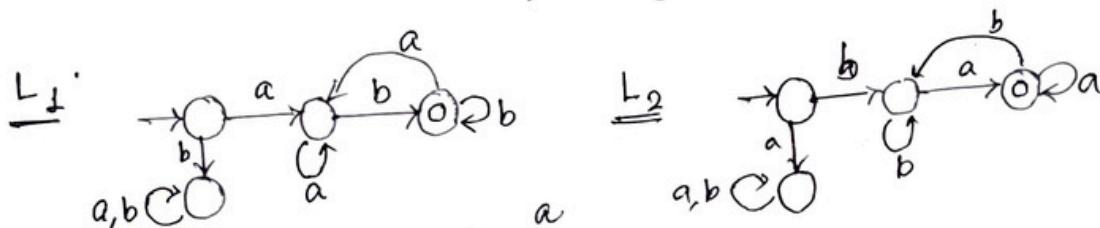
Report on

* Processes on DFA.

> Union eg. Set of strings s.t. string of the language start & end with different symbols.

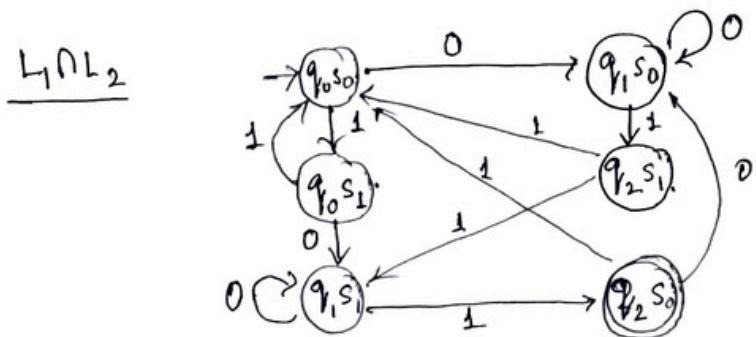
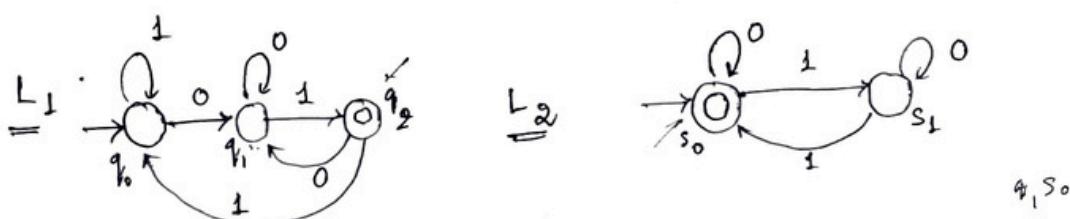


$L_1 \cup L_2$ where $L_1 = \{ \text{starts with } a \text{ and ends with } b \}$
 $L_2 = \{ \text{starts with } b \text{ & ends with } a \}$



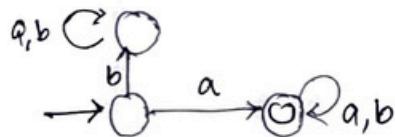
> Intersection eg. Set of strings over $\{0, 1\}$ s.t. it ends with 01 & has even # 1's.

$L = L_1 \cap L_2$ where $L_1 = \{ \text{ends with } 01 \}$
 $L_2 = \{ \text{Even } \# 1's \}$

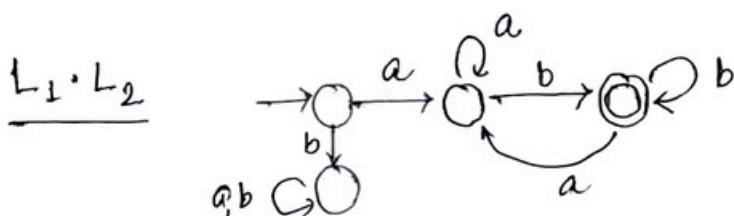
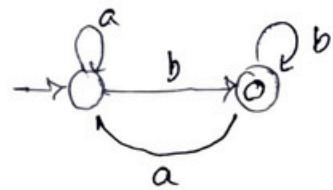


> Concatenation. Start with a and end with b, $L_1 \cdot L_2$

L_1 : Start with a



L_2 : Ends with b



> Complement final \leftrightarrow Non-final states

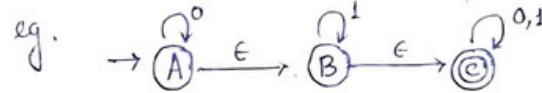
(Don't change transition direction)

> Reverse

- 1) States remain same
- 2) Initial \leftrightarrow Final
- 3) Reverse transition direction
- 4) Loops remain same
- 5) Remove inappropriate states.

- Not each reversal of DFA leads to DFA.

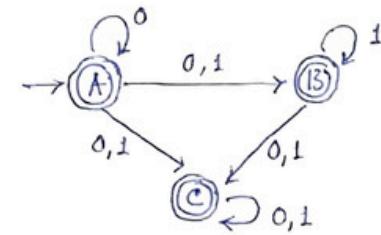
* ϵ -NFA \rightarrow equivalent NFA w/o ϵ -move (by finding ϵ -closure) ϵ^*



final state: any state that can reach final state only by seeing ϵ

| | ϵ^* | IP0 | ϵ^* | G^* | IP1 | ϵ^* |
|---|--------------|-----|--------------|-------|-----|--------------|
| A | ABC | AC | <u>ABC</u> | ABC | BC | <u>BC</u> |
| B | BC | c | <u>c</u> | BC | BC | <u>BC</u> |
| C | c | c | <u>c</u> | c | c | <u>c</u> |

| | 0 | 1 |
|-----------------|-----|----|
| $\rightarrow A$ | ABC | BC |
| $\rightarrow B$ | c | BC |
| $\rightarrow C$ | c | c |



[# states remain same from ϵ -NFA to NFA]

* 2nd symbol from rhs is 'a'

Build NFA $\rightarrow \textcircled{1} \xrightarrow{a} \textcircled{0} \xrightarrow{a,b} \textcircled{0}$ \rightarrow Convert to DFA.

(Same way for - 3rd symbol from rhs is 'a')

* If NFA contains n states, DFA can have at most 2^n states.

* # states in DFA \geq # states in corresponding NFA.

* When we complement DFA, the language gets complemented.

When we complement NFA, the n may/may not get complemented.

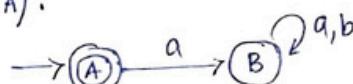
* Complement of language accepted by NFA \neq Language accepted by complement of NFA

eg. NFA $\rightarrow \textcircled{A} \xrightarrow{a} \textcircled{B} \xrightarrow{a,b} \textcircled{0}$

Language = $\{a, aa, ab, aab, abb, \dots\}$

$$\text{Comp}(L) = \{\epsilon, b, bb, ba, \dots\}$$

Comp(NFA):



$$\text{Lang} = \{\epsilon\}$$

not equal.

* Minimisation of DFA (Partitioning method)

| | 0 | 1 |
|-------------------|---|---|
| $\rightarrow A$ | B | C |
| B | B | D |
| C | B | C |
| D | B | E |
| \textcircled{E} | B | C |

0 equivalence: $\{A, B, C, D\} \{E\}$

1 equivalence: $\{A, B, C\} \{D\} \{E\}$

2 equivalence: $\{A, C\} \{B\} \{D\} \{E\}$

3 equivalence: $\{A, C\} \{B\} \{D\} \{E\}$ — same, so stop

* For some language, DFA is not unique.

* Dead state, Inaccessible state.

* k -equivalent: q_i, q_j if both $\delta(q_i, x)$ & $\delta(q_j, x)$ produce final states or both produce nonfinal states for all $x \in \Sigma^*$ of length k or less ($|x| \leq k$).

* Equivalent or indistinguishable states:

$$\delta(q_i, x) \rightarrow F \text{ and } \delta(q_j, x) \rightarrow F$$

or

$$\delta(q_i, x) \not\rightarrow F \text{ and } \delta(q_j, x) \not\rightarrow F$$

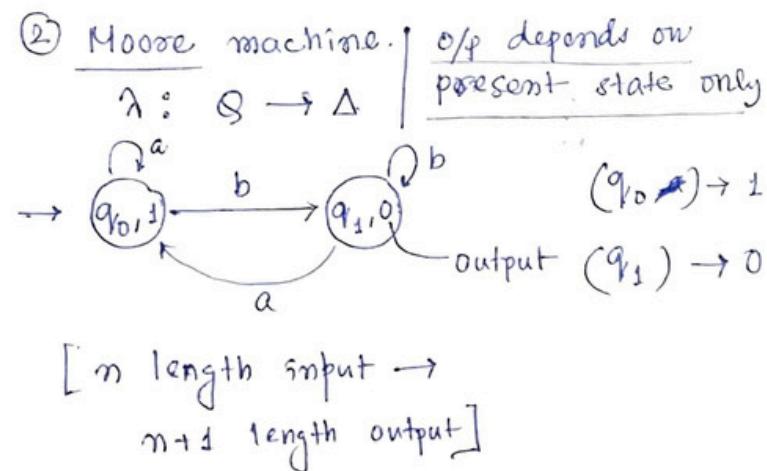
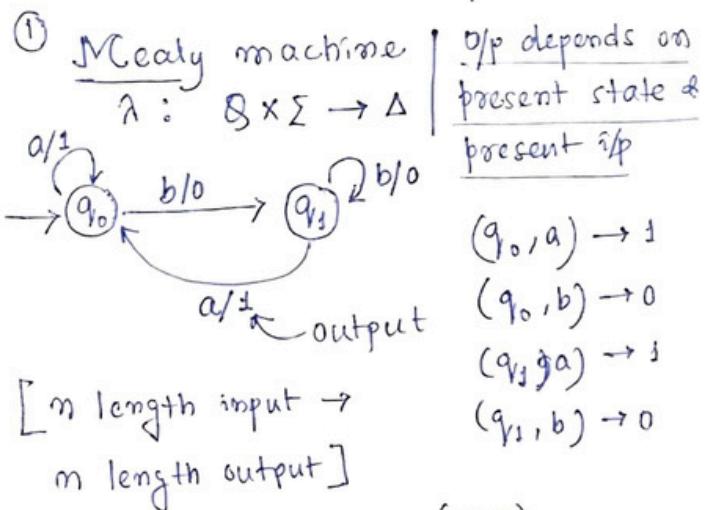
| | 0 | 1 |
|-------------------|---|----|
| $\rightarrow AC$ | B | Ac |
| B | B | D |
| D | B | E |
| \textcircled{E} | B | Ac |

* Distinguishable

$$\delta^*(q_i, w) \in F \text{ and }$$

$$\delta^*(q_j, w) \notin F \text{ for } w \in \Sigma^*$$

* FA w/o/p = $(Q, \Sigma, S, q_0, \Delta, f)$ $\delta : Q \times \Sigma \rightarrow S$
 (FST) o/p alphabet o/p function

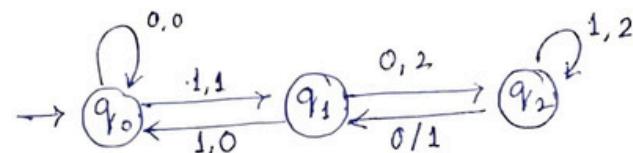


- * Finite state transducer produces a string from an output alphabet Δ in response to a given read-once-only input string from input alphabet Σ . * FST is deterministic.
- * There are no final states for FST. * Classes of Mealy, Moore are equivalent.
- * Mealy : $\delta(q_i, a) = q_j$ next state
 $\lambda(q_i, a) = b$ o/p | Moore : $\delta(q_i, a) = q_j$
 $\lambda(q_i) = b$

* Construction of FST : TOC1 + RBR Note

e.g. Mealy that o/p's the residue mod-3 of binary numbers. (considering its decimal

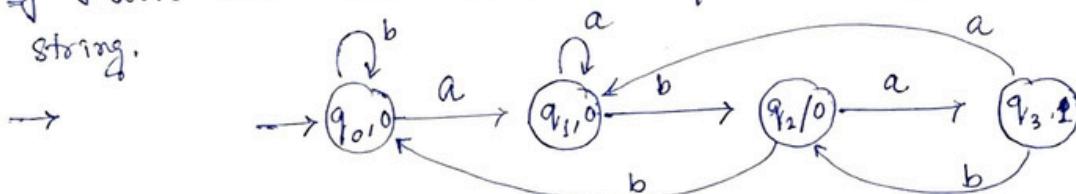
| | 0 | 1 | 0/P |
|---|-----|---|-----|
| 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 2 | 10 | 2 | 2 |
| 3 | 11 | 0 | 0 |
| 4 | 100 | 1 | 1 |
| 5 | 101 | 2 | 2 |
| 6 | 110 | 0 | 0 |
| 7 | 111 | 1 | 1 |



- for each o/p different state.

- transition to the state that outputs same residue for given number.

e.g. Moore that counts occurrence of substring 'aba' in a given input string.

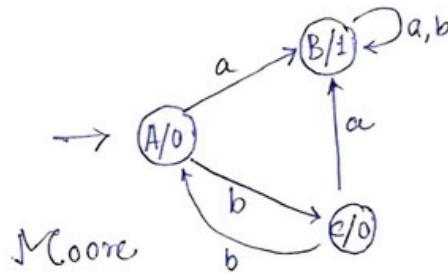


* Conversion Mealy \rightleftarrows Moore.

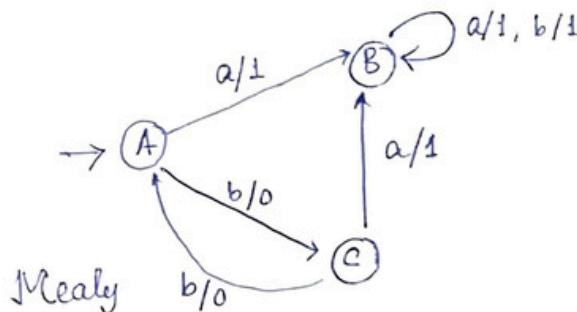
1) Moore \rightarrow Mealy : Copy corresponding next states as it is for different input symbols. Put the o/p of q_i from the Moore machine to the places of Mealy where next state is q_i .

eg. (Use tabular method)

otherwise



Moore



Mealy

2) Mealy \rightarrow Moore : eg.

(# states increases)

i) break q_i from Mealy
to multiple states if for same
 q_i state we have different ops
for different ips. [break q_1 to q_{10} & q_{11} ; q_2 to q_{20} & q_{21}]

| Mealy. PS | 0 | | 1 | |
|-----------|----------|----|----------|----|
| | NS | OP | NS | OP |
| q_0 | q_0 | 1 | q_{10} | 0 |
| q_1 | q_3 | 1 | q_3 | 1 |
| q_2 | q_{11} | 1 | q_{21} | 1 |
| q_3 | q_{11} | 0 | q_0 | 1 |

| PS | NS | 0 | OP | NS | 1 | OP |
|-------------------|----------|---|----|----------|---|----|
| $\rightarrow q_0$ | q_0 | | 1 | q_{10} | 0 | |
| q_{10} | q_3 | | 1 | q_3 | 1 | |
| q_{11} | q_3 | | 1 | q_3 | 1 | |
| q_{20} | q_{11} | | 1 | q_{21} | 1 | |
| q_{21} | q_{11} | | 1 | q_{21} | 1 | |
| q_3 | q_{20} | 0 | | q_0 | 1 | |

| PS | NS | | O/P |
|----------|----------|----------|------------------------------|
| | IP 0 | IP 1 | |
| q_0 | q_0 | q_{10} | 1 (as for all q_0 op is 1) |
| q_{10} | q_3 | q_3 | 0 |
| q_{11} | q_3 | q_3 | 1 |
| q_{20} | q_{11} | q_{21} | 0 |
| q_{21} | q_{11} | q_{21} | 1 |
| q_3 | q_{20} | q_0 | 1. |

* FA can only count a finite number of input scenarios.

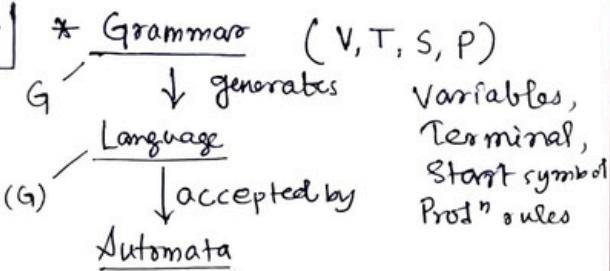
* Shortcut : Moore machine base-n number, residue modulo-k as o/p.

eg. base-4 number, residue mod-5 as o/p

| | NS 0 | 1 | 2 | 3 | O/P |
|-------|-------|-------|-------|-------|-----|
| q_0 | q_0 | q_1 | q_2 | q_3 | 0 |
| q_1 | q_1 | q_0 | q_1 | q_2 | 1 |
| q_2 | q_3 | q_4 | q_0 | q_1 | 2 |
| q_3 | q_2 | q_3 | q_4 | q_0 | 3 |
| q_4 | q_1 | q_2 | q_3 | q_1 | 4 |

* $L \subseteq \Sigma^*$, $CL \subseteq 2^{\Sigma^*}$ [L : Formal language, Σ : Alphabet (vocab. of symbols), Σ^* : Set of all strings made using symbols from Σ , CL : Class of languages, 2^{Σ^*} : Power set of Σ^* or set of all subsets of Σ^* . or set of languages over Σ]

* Chomsky Hierarchy of classes of languages.



$$L(G) = \{ w \in T^* : s \xrightarrow{*} w \}$$

* Language \leftrightarrow Grammar [Derivation tree for gram \rightarrow Lang]

$$\text{eg. } L = \{ a^n b^m \mid n \geq m \}$$

$$\begin{cases} S \rightarrow aSb / aAb \\ A \rightarrow \epsilon / aA \end{cases}$$

$$\text{eg. } L = \{ a^n b^n c^n, n \geq 1 \} \text{ eq } L = \{ a^n b^{n+1}, n \geq 0 \}$$

$$\begin{cases} S \rightarrow aSAc / abc \\ bA \rightarrow bb \\ cA \rightarrow Ac. \end{cases}$$

$$\begin{cases} S \rightarrow aSb / aAb \\ A \rightarrow aAb / b \end{cases}$$

* Regular Expressions

eg. Every 'a' followed by 'b'. $(b + ab)^*$

eg even length

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

eg odd length

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

• ϵ null string, ϕ empty language set • Closure applied on RE, Kleene's closure applied on Alphabet. eg. $r_1 = 01$, $r_1^* = \epsilon, 01, 0101, \dots$; $\Sigma = \{0, 1\}$, $\Sigma^* = \{\epsilon, 0, 1, 01, \dots\}$

• Precedence: $*$ > \cdot > $+$

• Properties: $a+b = b+a$, $a+(b+c) = (a+b)+c$, $a \cdot b \neq b \cdot a$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$, $a \cdot (b+c) = a \cdot b + a \cdot c$, $(a+b) \cdot c = ac + bc$, $a+(b \cdot c) \neq (a+b) \cdot (a+c)$,

~~$(a+b)+c \neq (a+c) \cdot (b+c)$~~ , $a+a = a$, $a \cdot a \neq a$, $r + \phi = \phi + r = r$,

$$\epsilon \cdot r = r \cdot \epsilon = r, \phi r = r \phi = \phi, r + \Sigma^* = \Sigma^*$$

• Equivalences: $L(r_1 + r_2) = L(r_1) \cup L(r_2)$, $L(r_1 r_2) = L(r_1) \cdot L(r_2)$,

$$L(r^*) = (L(r))^*, r_1 \cdot r_2 \neq r_2 \cdot r_1, \phi r = r \phi = r^* \phi = \phi \cdot \phi = \phi^+ = \phi, \epsilon^* = \epsilon = \epsilon^*$$

$$\checkmark \phi^* = \epsilon, r^* \cdot r^+ = r^+, r^* \cdot r^* = r^*, (r^+)^* = (r^*)^+ = (r^+)^* = r^*, \checkmark \epsilon + rr^* = r^*$$

$$\checkmark \epsilon + r^* r = \epsilon + r^+ = r^*, \checkmark p(qp)^* = (pq)^* p, \checkmark (p+q)^* = (p^*q^*)^* = (p^*+q^*)^*$$

$$\checkmark (p+q)^* p^* q^* = (p+q)^* = \phi^* (qp^*)^* = (p^*q^*)^* p^* = (p+q^*)^* = (p^*+q^*)^* =$$

$$p^* (p+q)^* = (p+q)p^* , rr^* = r^*r = r^+$$

\checkmark at most 2 b's

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

\checkmark no 2 a's come together. $\{\epsilon, ab, aba, bab, babb, bba, \dots\}$

$$\text{or } (b+ab)^* + (b+ab)^*a = (b+ab)^*(\epsilon+a)$$

$$\text{or } (b+ba)^* + a(b+ba)^* = (\epsilon+a)(b+ba)^*$$

* Regex examples.

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

1) Strings start, end same symbol

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

2) Start a, not having 2 consecutive b's

$$(a + ab)^+$$

3) Len of string = 2

$$(a+b)(a+b)$$

$$\text{len} \geq 2$$

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

$$\text{len} \leq 2$$

$$(a+b+\epsilon) (a+b+\epsilon)$$

4) Len of string even

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

5) Len of string div. by 3

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

6) Len of string odd

$$\underbrace{((a+b)(a+b)(a+b))^*}_{\text{+1 even}}$$

7) Exactly one b

$$a^* b a^*$$

$$2 \text{ b's}$$

$$a^* b a^* b a^*$$

✓ 8) At most 2 b's

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

9) abb as substring

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

10) At least 2 b's

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

✓ 11) At least 1 a and at least 1 b.

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

✓ 12) Not containing 01 as substring.

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

$$\{ \epsilon, 0, 1, 11, 111, \dots, 10, 100, 100, \dots \}$$

$$\rightarrow 1^* 0^*$$

✓ 13) Every 0 followed by 11. $(011+1)^*$

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

✓ 14) At least 2 occurrences of b b/w any two occurrences of a. $b^* + b^* (abb^+)^* ab^*$

3a

* Primitive regex

- i) $a \in \Sigma$ (a a symbol)
 - ii) ϵ
 - iii) ϕ .
- If $|\Sigma| = n$ there are $n+2$ primitive regular expressions.

✓ 15) Length of string $3n$ or $3n+1$.

$$(a+b)(a+b)(a+b)(\epsilon + a+b)$$

✓ 16) At least one occurrence of each symbol in $\Sigma = \{a, b, c\}$
a, b, c can be arranged in $3!$ ways

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

$$\begin{matrix} m & a & n & c & n & b & n & + \\ m & b & n & a & n & c & n & + \end{matrix}$$

$$\begin{matrix} n & b & n & c & n & a & n & + \\ n & c & n & a & n & b & n & + \end{matrix}$$

$$\begin{matrix} n & c & n & b & n & a & n & + \\ n & c & n & b & n & a & n & \end{matrix}$$

17) All runs of a's have lengths multiple of - 3.

$$(aaa + b+c)^*$$

✓ 18) Every 0 is followed (immediately) by k 1's and preceded by at least k 1's.

$$1^* 1^k (01^k)^* + 1^*$$

✓ 19) $L = \{a^n b^m \mid m+n \text{ is even}\}$

$$\# a \text{ even} \# b \text{ even } (aa)^* (bb)^* \Rightarrow a(aa)^* b(bb)^* + (aa)^* (bb)^*$$

$$\# a \text{ odd} \# b \text{ odd } a(aa)^* b(bb)^*$$

20) Not ending with 11 $(0+1)^* (00+01+10)$

21) # a's is divisible by 3. $(b^* a b^* a b^* a b^*)^*$

22) Containing no double symbols {00 or 11}

$$(01+10)^* + \epsilon + 0 + 1$$

no 2 a's, no 2 b's come together.

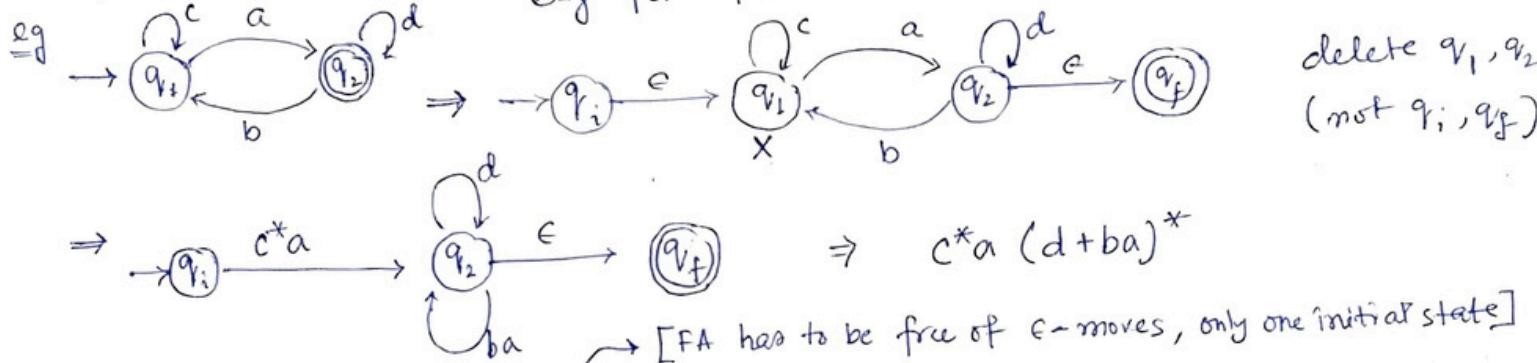
4 classes Start in \bullet End in (no 2 a's, no 2 b's together)

| | | | | | |
|------|---|---|-------------------------------|---|-----------------------------------|
| i) | a | a | $\{a, aba, ababa, \dots\}$ | - | $(ab)^*a \Leftrightarrow a(ab)^*$ |
| ii) | a | b | $\{ab, abab, ababab, \dots\}$ | - | $(ab)^*$ or $a(ab)^*b$ |
| iii) | b | a | $\{ba, baba, bababa, \dots\}$ | - | $(ba)^*$ or $b(ab)^*a$ |
| iv) | b | b | $\{b, bab, babab, \dots\}$ | - | $(ba)^*b \Leftrightarrow b(ab)^*$ |

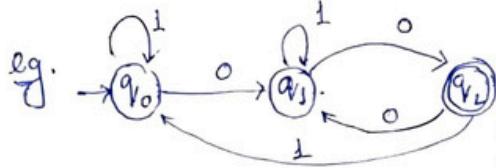
$$\begin{aligned} A_{fa} &= (ab)^*a + (ab)^* + b(ab)^*a + b(ab)^* \\ &= (ab)^*(a+\epsilon) + b(ab)^*(a+\epsilon) \\ &= (\epsilon+b)(ab)^*(a+\epsilon) \end{aligned}$$

* FA \rightarrow RE. [More examples at ME-TH]

1. By state elimination [no incoming edge for init. state, no outgoing edge for final state, delete states one by one]



2. Using Arden's theorem [P & Q be 2 re's. If P does not contain ϵ , the $R = Q + RP$ has a unique sol'n $R = QP^*$]



Solve $\left\{ \begin{array}{l} q_0 = q_0 \cdot 1 + q_1 \cdot 1 + \epsilon \\ q_1 = q_0 \cdot 0 + q_1 \cdot 1 + q_2 \cdot 0 \\ q_2 = q_1 \cdot 0 \end{array} \right.$

$q_2 = \checkmark$

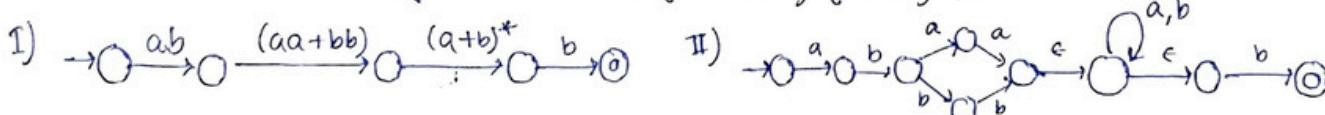
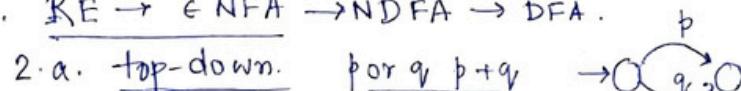
* RE \rightarrow FA

1. Direct conversion RE \rightarrow DFA.

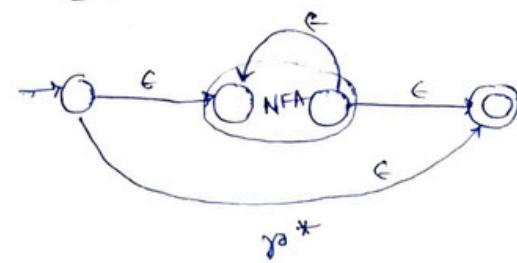
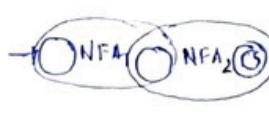
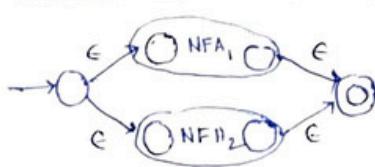
Using firstpos, followpos

Refer SK book

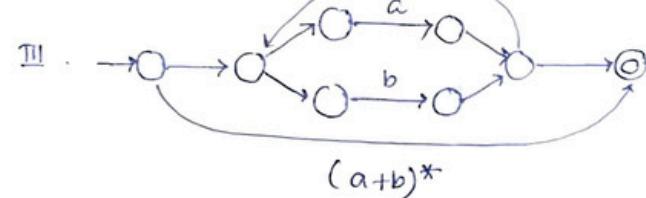
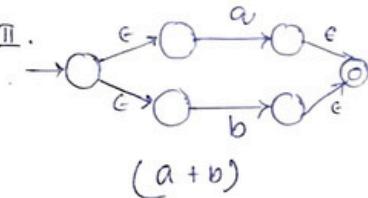
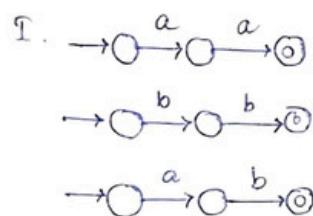
2. RE $\rightarrow \epsilon$ NFA $\xrightarrow{\epsilon\text{-closure}}$ NDFA \rightarrow DFA.



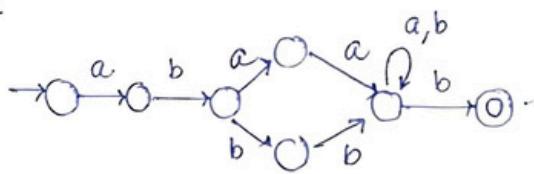
2.b. Bottom up $\rightarrow \bullet \xrightarrow{\epsilon} \bullet \xrightarrow{a} \bullet$



$$\frac{a}{(ab)(aa+bb)} (a+b)^* b.$$



IV



* Equivalence of 2 FAs

Produce same re.

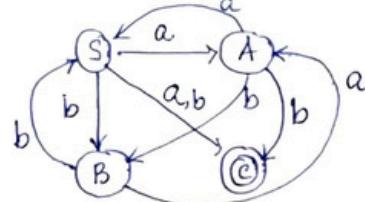
* Equivalence of 2 re's

Have same FA.

* Regular grammar \rightarrow FA. [• for grammar producing ^{not} null string, # states = # non-terminals + 1, each non-terminal corresponds to a state, extra state - final state. • for grammar ~~not~~ producing null, # states = # non-terminals] [$A \rightarrow aB : S(A,a) \rightarrow B ; A \rightarrow a : S(A,a) \rightarrow \text{final state} ; A \rightarrow \epsilon : S(A,\epsilon) \rightarrow A$ & A is final state]

e.g. $S \rightarrow aA/bB/a/b$
 $A \rightarrow aS/bB/b$
 $B \rightarrow aA/bS$

| 3 NTs S,A,B ; final state C



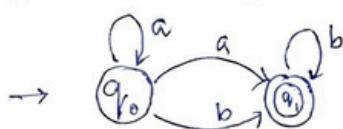
* RE \rightarrow Regular grammar.

i) RE \rightarrow FA (eliminate ϵ moves)

ii) # nonterminals in the grammar = # states of FA

iii) $S(q_1, a) \rightarrow q_2$: $A_1 \rightarrow aA_2$ or $q_1 \rightarrow aq_2$, } if q_2 final, both
 $S(q_1, a) \rightarrow q_2$: $q_1 \rightarrow a$ [when q_2 is final state]
 $S(q_1, \epsilon) \rightarrow q_1$: $q_1 \rightarrow \epsilon$ [when q_1 is final state].

e.g. $a^*(a+b)b^*$ Prodⁿ rules



$S(q_0, a) \rightarrow q_0, q_1$: $q_0 \rightarrow aq_0, q_0 \rightarrow aq_1, q_0 \rightarrow a$
 $S(q_0, b) \rightarrow q_1$: $q_0 \rightarrow bq_1, q_0 \rightarrow b$
 $S(q_1, b) \rightarrow q_1$: $q_1 \rightarrow b, q_1 \rightarrow bq_1$

so, $\begin{cases} q_0 \rightarrow a/b/aq_0/aq_1/bq_1 \\ q_1 \rightarrow b/bq_1 \end{cases}$

[FA \leftrightarrow RE \leftrightarrow Re.grammar
 \leftrightarrow RL

All interchangeable,
convertible to each
other]

* Pumping lemma for regular expression [Used to prove that a language is not regular, proof by contradiction technique using Pigeonhole principle].

> for any regular language L , there exists an integer n (pumping length), such that for all $x \in L$ with $|x| \geq n$, there exists $u, v, w \in \Sigma^*$, s.t. $x = uwv$ and 1) $|uv| \leq n$ 2) $|v| \geq 1$ 3) for all $i \geq 0$, $uv^i w \in L$. [If v is pumped any no. of times, resulting string remains in L .]

> If at least one string made from pumping is found to be not in L , then we say L isn't regular.

> Steps: i) Assume L is regular. n be the #states of FA accepting L
ii) Choose string w ($w \in L$) s.t. $|w| \geq n$. Then $w = xyz$ with $|xy| \leq n$ and $|y| > 0$ iii) Find suitable integer i s.t. $xy^i z \notin L$.

Eg. $L = \{a^p : p \text{ is prime}\}$ is not regular.

$w = a^p$ | $w \in L$, $p \geq n$ where n is #states, assume L is regular

$w = xyz$ with $|xy| \leq n$ & $|y| > 0$

y is a string of a 's. Assume, $y = a^m$, $1 \leq m \leq n$

Let's take $i = p+1$.

$$\begin{aligned} |xy^i z| &= |xyz| + |y^{i-1}| \\ &= p + (i-1)m \\ &= p + pm \\ &= p(1+m) \text{ not a prime.} \end{aligned}$$

So, $xy^i z \notin L$. Hence, not regular.

If L is regular, then
 $\exists n \geq 1$, such that

$\forall w \in L$, where $|w| \geq n$

$\exists xyz$ such that $w = xyz$

and $|xy| \leq n$

and $|y| \geq 1$

and $\forall i \geq 0$, $xy^i z \in L$.

try to disprove to prove irregularity.

* Testing regularity: If L is finite, then it's regular. If infinite we disprove by Pumping lemma for RL. [Set of a regular language is countable.] FA has finite memory, so it can't count infinite occurrence of some string. Eg. While $a^n b^n$; $n \leq 10^{10}$ is regular, $a^n b^n$, $n \geq 1$ is not. ($\underline{\text{is}} \rightarrow \text{next pg}$)

Eg. ww^R | $w \in \{a, b\}^*$: Irregular, w can be of any size and as we have to check it to produce w^R , FA can't do it. (Comparison)

Eg. $a^n b^m c^k$ | $n, m, k \geq 1$: No comparison \Rightarrow No need of memory. So, regular.

eg. a^p ; p prime.
 no repeating pattern
 ↓
 irregular

eg. $a^i b^{j^2}$: $i, j \geq 1$
 even after a, b generated independently, no pattern of b^{j^2} is found
 \Rightarrow irregular

eg. $w | n_a = n_b$
 irregular as needs counting infinitely

eg. $www^R, a^n b^n c^n, a^n b^{n+m} c^m$ not regular

eg. $\{1^p \mid p \text{ prime}\}, \{0^i 1^j \mid i > j\}$ non-regular.

eg. $L = \{0^n 1^{2n} \mid n \geq 1\}$ use Pumping lemma.

states = n

$$w = 0^n 1^{2n} \quad | \quad 3n > n$$

$$= xyz \quad |xy| \leq n \quad |y| > 0$$

i) $\lambda = 0$ $w = 0^{n-k} 0^k 1^{2n}$

$$xy^0z = xy^0z = 0^{n-k} * 1^{2n} \quad 2(n-k) \neq 2n$$

ii) $i = 0$ $w = 0^n 1^k 1^{2n-k}$

$$xy^0z = 0^n 1^{2n-k} \quad 2n \neq 2n-k$$

$$w = 0^{n-k} 0^k 1^k 1^{2n-k}$$

iii) $i = 2$

$$w = \frac{0^{n-k}}{x} \frac{0^k 1^k}{y} \frac{1^{2n-k}}{z}$$

$$\begin{aligned} xy^2z &= 0^{n-k} (0^k 1^k) (0^k 1^k) 1^{2n-k} \\ &= 0^n 1^k 0^k 1^{2n}. \text{ not in the form } 0^n 1^{2n}. \end{aligned}$$

So, L is not regular.

Only way to generate/accept an infinite language with an FA is to use Kleene star (in regex) or cycles/loops (in FA). This forces some sort of repetitive cycle within the strings of language. If we are unable to find one, it must be irregular.

* Myciull-Nerode Theorem [Used to test regularity, also to minimise DFA]

Given a language L , 2 strings x & y are said to be in same class if, for all possible strings $w \in \Sigma^*$, xw & yw are in L or

Right-invariant property:
 2 strings x, y from language \rightarrow
 any $w \in \Sigma^*$ appended to right of x, y
 still they are equivalent (in the language or not)

both are not in L.

(i) L divides the set of all possible strings into mutually exclusive equivalence classes.

(ii) A language L is regular if & only if the index number (number of equivalence classes created by L) is finite.

e.g. $L = \{ \text{string ending in '0'} \}$ $\Sigma = \{0, 1\}$

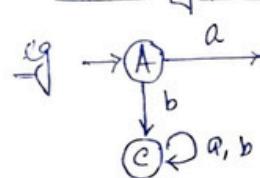
$$L = \{0, 00, 10, 000, 100, 110, \dots\}$$

The equivalence classes are

- i) String ending in 0
 - ii) $m \neq n \neq l$
- Finite index
↓
Regular language

* # Myhill-Nerode equivalence classes for a regular language \equiv Number of states in the minimal DFA recognising the language.

* Finding equivalence classes of FA. [More at ME-TH 75]

eg.  - DFA minimised ✓
- #states = 3 = #eqv. classes
- Partitions or eqv. classes:

$$\left. \begin{aligned} [A] &= \{\epsilon\} \\ [B] &= a(a+b)^* \\ [C] &= b(a+b)^* \\ \therefore [A] \cup [B] \cup [C] &= \\ &= \epsilon + a(a+b)^* \not\approx 0 + b(a+b)^* \\ &= (a+b)^* \end{aligned} \right\}$$

* Summary - Testing regularity

1. Finite set is regular.
2. Finite comparison (like $n \leq 10^{10}$) \Rightarrow regular
3. Pattern of strings forms AP \Rightarrow regular
4. pattern with GP not regular. (rather CSL)
5. No pattern, that can be repeated to generate language wholly \Rightarrow not regular
6. Concatenation of regular, non-regular \Rightarrow non-regular $(a^n b^n, n \geq 1) \cdot (a^n, n \geq 1)$
7. Unbounded storage needed \Rightarrow non-regular
e.g. $\{w \mid n_a / 3 > n_b / 3\}$ bounded memory, finite #states \Rightarrow regular

8. $\{wxw^r \mid |x|=5\}$ not regular $\downarrow w, x \in (a+b)^*$

$\{wxw^r\}$ regular

$\{w x w^r\} \quad w, x \in (a+b)^+, \quad x \in (a+b)^{+ \text{ regular}}$

$$w, x \in (a+b)^*$$

~~$\{ww^ry \mid w,y \in \{0,1\}^+\}$~~ not regular

$\{ \text{...} \}$ | $x, y, w \in \{0,1\}^*$ regulare

$\checkmark \{w x w \mid w, x \in \{0,1\}^*\}$
- regular

$$(x = (a+b)^*, \omega = \epsilon)$$

$$(x = (a+b)^*, w = c)$$

$\{wxw \mid w \in \{0,1\}^*\}$
- not regular

- not regular

CCSI

$\checkmark \{ w x w \mid w, x \in \{0,1\}^+ \}$ - not regular
 $\{ w \mid w \text{ is multiple of } 7 \}$ regular

Will you be my res.

multiple of π , regular

$\checkmark \{ w : w \text{ is unary notation for natural number multiple of } 7 \}$ reg

~~w~~, w: w is decimal n

+ } regular

{ w : w is base b number string divisible by n } regular
(Check TOC folder for further understanding)
+ first pg ① for FA shortcut

$\{ w : w \text{ is unary notation for } 10^n \text{ for some } n \geq 1 \}$

$\{ w : w \text{ is decimal notation for } 10^n \text{ for some } n \geq 1 \}$

$\{ w : w \text{ is of the form } x\#y \text{ where } x, y \in \{1\}^+ \text{ & } y = x+1$
 when x & y are interpreted as unary numbers }

$\{a^n b^j : |n-j| = \text{any} - \text{not regular}\}$

$\{ w \in \{a,b\}^*: \text{for each prefix } x \text{ of } w, \#a(x) \geq \#b(x) \}$
 - not regular

Every subset of a regular language isn't regular.

e.g. $L = a^*$. $L_i = a^p : p$ is prime

$$L_1 \subset L_4$$

✓ 10. If L is regular, then so is $L' = \{xy \mid x \in L \text{ & } y \notin L\}$

Proof $y \notin L \Rightarrow y \in \bar{L}$ \bar{L} is regular (as RL closed under complement). So, \bar{L} is concat. of 2 regular languages $\Rightarrow L'$ is regular.

11. $\{w \mid w = w^R\}$ not regular

$L = \{ayx^R \mid a, y \in \Sigma^*\}$ regular

$\{a^n b^m \mid n \neq m\}$ not regular

12. If we can build NFA for an infinite language then regular.

* 13. $L_1 = \{a^p \mid p \text{ is prime}\}$, $L_2 = \{a^{2^n} \mid n > 0\}$

$L_3 = \{a^n b^n \mid n > 0\}$

L_1^* , L_2^* are regular. L_3^* is not regular.

[If L is non-regular, L^* may / may not be regular.]

[Class of non-regular languages not closed under Kleene star.]

Closure properties of RL.

(1) Closed under cycle opⁿ
 $\text{cycle}(L) = \{xy \mid x, y \in \Sigma^* \wedge yx \in L\}$

RL is closed under 1. Kleene closure (*) 2. Positive closure (+)

3. Complement, 4. Reverse operator, 5. ^{Finite} Union (LUM regular when L, M regular), 6. ^{Finite} Intersection, 7. ^{Finite} Difference, 8. Homomorphism

[gives a string for each symbol of alphabet. e.g. $\Sigma = \{0, 1\}$
homomorphism f^n $h(0) = ab$, $h(1) = \epsilon$, extend to other strings
 $\in \{0, 1\}^*$ such as $h(01010) = ababab$] 9. Inverse

homomorphism [$h^{-1}(L) = \{w \mid h(w) \text{ is in } L\}$] 10. ^{Finite} Concatenation

11. Intersection with regular language.

Closed under intersection

$L \cap M$, final states of product consist of final states of both automata for L, M. $F_N = F \times F'$.

Closed under union

$F_N = (F \times Q') \cup (Q \times F')$
[any state containing final state of M or N]

Closed under difference $L - M$

Final states where L has final but M has not.

Product DFA construction

(e.g. #a even, #b even).

$M = (\emptyset, \Sigma, \delta, q_0, F)$ $M' = (\emptyset', \Sigma', \delta', q_0', F')$

Product automaton $N = (\emptyset_N, \Sigma_N, \delta_N, (q_0, q_0'), F_N)$

$\emptyset_N = \emptyset \times \emptyset'$, $\delta_N: Q_N \times \Sigma \rightarrow Q_N$

For any $q \in \emptyset$, $q' \in \emptyset'$ & $c \in \Sigma$,

$\delta_N(\underbrace{(q, q')}, c) = (\delta(q, c), \delta'(q', c))$
state of N

> Infinite union, inf. intersection, inf. difference
inf. concatenation of regular languages need not be regular.

12. Closed under posfix
{e, apb, abc, abcd}
Prefix Prefix(L) = {abc, abcd} also regular

13. Closed under posfix
{abc} of L = {abc, abcd}

H.

* If we know some specific seed language to be not regular, we can use closure properties (homomorphism, intersection, union,..) to show that lots of other languages are not regular.

→ Say, we want to find out about L' .

Suppose, L' is regular. Use closure properties on L' to prove L (already known to be irregular) is regular. So, L' is irregular by contradiction.

↳ Using closure properties

e.g. $\Sigma = \{a, b\}$ $L = \{w \mid w \text{ has } n_a = n_b\}$

Now, a^*b^* is regular (known). Hence, if we assume L is regular, $L \cap a^*b^*$ is also regular. But $L \cap a^*b^*$ is precisely $\{a^n b^n, n \geq 0\}$, which is irregular! Contradiction.

So, L is irregular.

[Don't use closure property backwards! e.g. If L_1, L_2 are regular then also $L_3 = L_1 \cap L_2$. But, if L_1 and L_2 are regular, we can't say anything about L_3 . $ab = ab \cap a^n b^n$]

1. Use $\emptyset, \Sigma^*, a^n b^n$, complement of $a^n b^n$ to prove using closure properties. [$\emptyset \rightarrow$ intersection, $\emptyset \cap$ any lang = \emptyset ; $\Sigma^* \rightarrow$ union]

✓ 2. If L is a regular language, so is $L' = \{w \mid w \in L \text{ and } w^R \in L\}$
→ $w^R \in L$ means $w \in L^R$. So, $w \in L$ and $w \in L^R \Rightarrow w \in L \cap L^R$. ∴ $L' = L \cap L^R$. L is regular. L^R also regular (closed under reverse). So, L' too is regular.

✓ 3. If C is any set of regular languages, $U \subseteq$ (union of all elements of C) is also a regular language, only when C is finite set.

✓ 4. Every regular language except \emptyset has a regular proper subset.

✓ 5. If L_1, L_2 are non-regular, we can't say anything about $L_1 \cup L_2$ being regular or not. 8

$$[\{a^n b^m \mid n \geq m\} \cup \{a^n b^m \mid n \leq m\} = a^* b^*]$$

* Decision problems on Regular Languages. [More on SK book]

All decidable :

[If any decision (yes/no) problem has a correct algorithm that runs finite amt of time]

1. $w \in L(G)$? (Membership)

2. $L(G) = \emptyset$? (Emptiness problem)

3. $L(G) = \Sigma^*$? ↗

4. $L(G_1) \subseteq L(G_2)$. (Containment)

5. $L(G_1) = L(G_2)$

6. $L(G_1) \cap L(G_2) = \emptyset$

7. $L(G)$ is regular?

8. $L(G)$ is finite? (finiteness)

* Regular Grammar. [Type-3] Generates RL. [Grammar $G = (V, T, S, P)$]

A grammar is regular iff it's left \Leftrightarrow right linear.

- Class of linear grammar is a subset of CFG & superset of RLG (Regular lang. gram.). $V \rightarrow T^* V T^* / T^*$

- Left linear grammar: $V \rightarrow VT^* / T^*$ LHS of prod^n is a non-terminal, RHS contains either a non-t followed by a terminal sequence or terminal sequence. e.g. $S \rightarrow Sa / Sb / a / b$.

- Right linear grammar: $V \rightarrow T^* V / T^*$ e.g. $S \rightarrow aS / bS / a / b$

$\begin{cases} S \rightarrow aX \\ X \rightarrow Sb \end{cases}$ } not left linear } We can't mix two,
not right linear } So, it's not regular grammar.

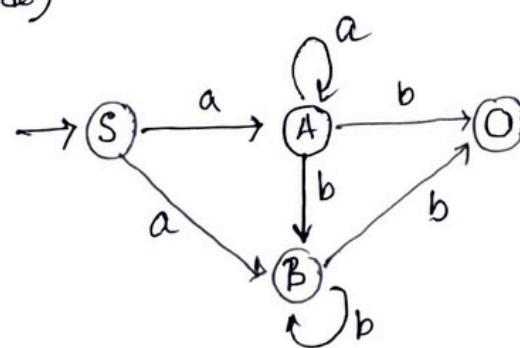
- Left lin, right lin. grammar & regular grammar are equivalent i.e. they generate same class of languages - regular lang.

- Regular grammar (RLG) $\xrightarrow{\text{to}}$ FA.

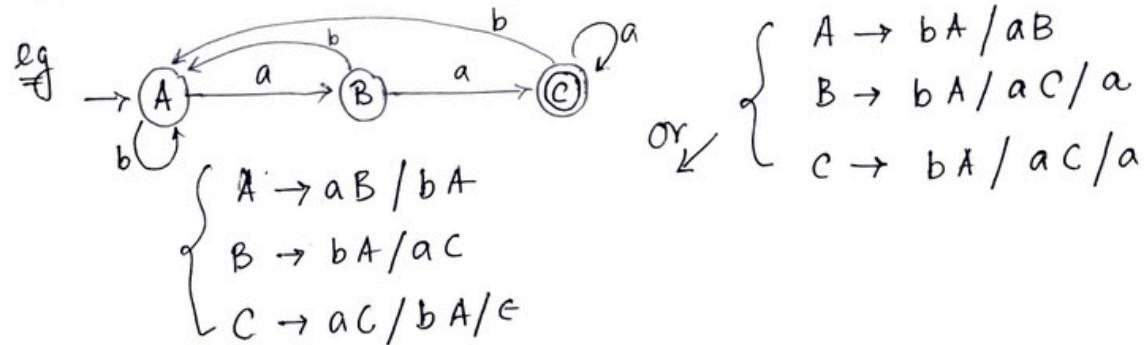
> See pg 4 (reverse side)

e.g. $S \rightarrow aA / aB$
 $A \rightarrow aA / bB / b$
 $B \rightarrow bB / b$

states = 4



- FA \rightarrow RLG \Rightarrow See pg 4 (reverse side).



- For any given language, we may have > 1 grammars generating the language.

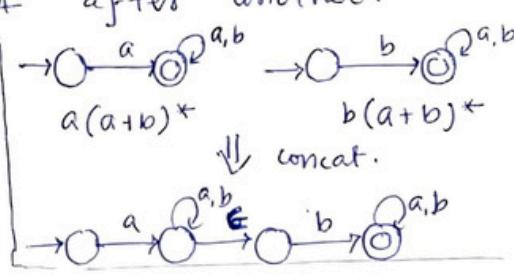
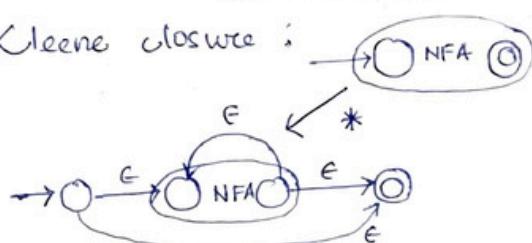
* Closure operations on FA.

1. Union, Intersection, Difference : We use product DFA of constituent FAs. [see pg 7]

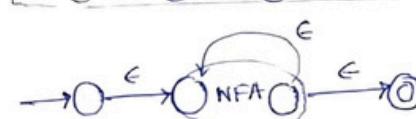
2. Complement : Final interchanged with non final states.

3. Concatenation : Append one FA after another.

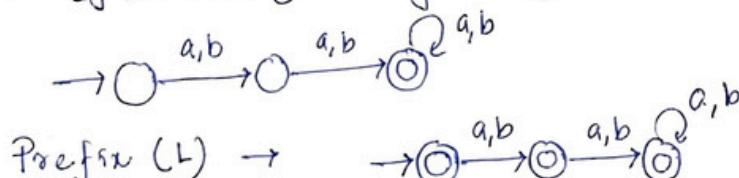
4. Kleene closure :



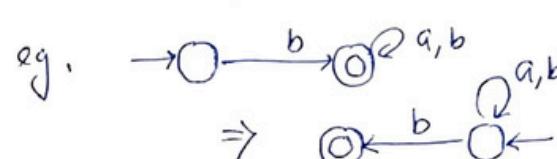
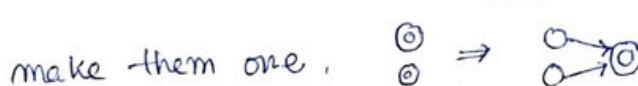
5. Positive closure :



6. Prefix : eg L = Binary strings length at least 2



7. Reverse operation : (Interchange final & initial states) & (reversing transition directions) If > 1 final states, make them one.



* Grammatical examples

1. $a^n b^m \mid n+m \text{ is even}$

regex $(aa)^* (bb)^* + a(aa)^* b(bb)^*$

$$\left\{ \begin{array}{l} S \rightarrow \cancel{AB} \ AB / aAbB \\ A \rightarrow aaA / \epsilon \qquad \qquad \qquad (aa)^* \\ B \rightarrow bbB / \epsilon \qquad \qquad \qquad (bb)^* \end{array} \right.$$

2. $a^n b^m \mid n \geq 3, m \geq 2$

$$\left\{ \begin{array}{l} S \rightarrow aaaA \ bb \ B \\ A \rightarrow aA / \epsilon \\ B \rightarrow bB / \epsilon \end{array} \right.$$

3. $\underline{a^n b^n c^m} \mid n \geq 1, m \geq 0$

$$S \rightarrow AB$$

$$A \rightarrow aAb / ab$$

$$B \rightarrow cB / \epsilon$$

4. $\underline{a^n b^n c^m} \underline{b^m} \mid n, m \geq 0$

$$S \rightarrow AB$$

$$A \rightarrow aAb / \epsilon$$

$$B \rightarrow cBb / \epsilon$$

$$5. \quad \{a^n b^m c^m d^n \mid m, n \geq 1\}$$

$$\begin{cases} S \rightarrow aAd / asd \\ A \rightarrow bAc / bc \end{cases}$$

$$6. \quad \{a^{m+n} b^n c^m \mid m, n \geq 1\}$$

$$\begin{cases} S \rightarrow aAc / asc \\ A \rightarrow aAb / ab \end{cases}$$

~~✓~~ 7. $\{a^n b^m \mid n \neq m ; m, n \geq 1\}$

$$\begin{cases} S \rightarrow asb / aAb / aBb \\ A \rightarrow aA / a \text{ — more a's} \\ B \rightarrow bB / b \text{ — more b's} \end{cases}$$

say $n \neq m$ & $n > m$

1. use $S \rightarrow asb$ m times to make $a^m b^m$
2. then use $S \rightarrow aAb$, $A \rightarrow$ to make $a^n b^m$.

$$8. \quad \{a^n b^m \mid n \geq m\}$$

$$S \rightarrow asb / A$$

$$A \rightarrow aA / \epsilon$$

~~✓~~ 9. Start, end same symbol.

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

$$S \rightarrow aAa / bAb / a / b$$

$$A \rightarrow aA / bA / \epsilon$$

~~✓~~ 10. Length of string odd.

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

or

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

$$(a+b) \rightarrow A \rightarrow a/b$$

$$(a+b)(a+b) \rightarrow B \rightarrow AA$$

$$((a+b)(a+b))^* \rightarrow C \rightarrow BC/\epsilon$$

(C)

$$\left\{ \begin{array}{l} S \rightarrow A \cdot C \\ C \rightarrow BC/\epsilon \\ B \rightarrow AA \\ A \rightarrow a/b \end{array} \right.$$

$\underbrace{a + \text{odd}}_{\text{even}}$ $\underbrace{b + \text{even}}_{\text{odd}}$

↑ Hint

11. { start with a \rightarrow total len even

{ start with b $\overset{\text{or}}{\rightarrow}$ total len odd

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

$(a+b)$ $A \rightarrow a/b$ $(a+b)(a+b)$ $B \rightarrow AA$ $((a+b)(a+b))^* \quad C \rightarrow BC/\epsilon$

$$\left\{ \begin{array}{l} S \rightarrow aAC / bC \\ C \rightarrow BC / \epsilon \\ B \rightarrow AA \\ A \rightarrow a/b \end{array} \right.$$

[More on SK book]

12. $\{a^i b^j \mid i, j \geq 0, i \neq 2j\} = L_1 \cup L_2$

 $L_1 = \{a^i b^j \mid i, j \geq 0, i > 2j\}$ $L_2 = \{a^i b^j \mid i, j \geq 0, i < 2j\}$

L_1 $S_1 \Rightarrow aA$ for each addition of b
 $A \rightarrow aaAb / aA / \epsilon$ add 2 a 's at least

L_2 $S_2 \rightarrow Bb / aBb$ there are non-0 b 's
 $B \rightarrow Bb / aBb / aaBb / \epsilon$ base case

1b: b or ab

for each addition of b add at most 2 a 's ($0, 1, 2$)

 $S \rightarrow S_1 / S_2$

* Other closure properties of RL.

i) Family of regular languages is closed under COR operation.

$$\text{COR}(L_1, L_2) = \{ w : w \in \bar{L}_1 \text{ or } w \in \bar{L}_2 \} \quad (\bar{L}_1 \cup \bar{L}_2)$$

ii) Under XOR operation. iii) Under NOR operation

iv) Symmetric difference. v) Square root $\sqrt{L} = \{ w \mid ww \in L \}$
 (while square of L is not regular
(need not be) L is regular)

vi) Min $\text{min}(L) = \{ w \mid w \in L, \text{ no prefix of } w \in L \}$

vii) Max $\text{max}(L) = \{ w \mid w \in L, \text{ but for no } x \text{ other than } \epsilon \quad w x \in L \}$

[eg. $L = \{a^n \mid n \geq 2\}$ $\text{min}(L) = \{aa\}$ $\text{max}(L) = \emptyset$]

$\left\{ \begin{array}{l} \text{Min}(L) = \{ w \mid w \in L, \text{ if } (ay = w, x \in L) \text{ then } y = \epsilon \} \\ \text{Max}(L) = \{ w \mid w \in L, \text{ if } wy \in L \text{ then } y = \epsilon \} \end{array} \right.$

[eg. $L = \{a^n b^m c^t \mid t = n \text{ or } t = m\}$

$\text{Max}(L) = \{a^n b^m c^t \mid t = \max\{m, n\}\}$

$\text{Min}(L) = \{a^n b^m c^t \mid t = \min\{m, n\}\} \quad \text{if } m, n > 0$
 $= \{\epsilon\} \quad \text{if } m, n \geq 0$

viii) Shuffle $(L_1, L_2) = \{x_1 y_1 x_2 y_2 \dots x_k y_k \mid x_1 \dots x_k \text{ is in } L_1, y_1 \dots y_k \text{ is in } L_2\}$

ix) Substitution f (it is a mapping of alphabet Σ onto subsets of Δ^* for some alphabet Δ .

A map $f : \Sigma^* \rightarrow 2^{\Delta^*}$ s.t.

2^{Δ^*} - power set of Δ^*

i) $\forall u, v \in \Sigma^* ; f(uv) = f(u)f(v)$

ii) $f(\epsilon) = \{\epsilon\}$

x) Suffix(L) = $\{v \mid \text{for some } u, uv \text{ is in } L\}$

xi) Half of a regular language $\text{Half}(L) = \{u \mid uv \in L \text{ & } |u| = |v|\}$

xii) One-third (L) = $\{u \mid uvw \in L, |u| = |v| = |w|\}$

xiii) Quotient (L_1/L_2) = $\{x \mid xy \in L_1 \text{ for some } y \in L_2\}$

xiv) Subsequence $\{w \mid w \text{ obtained by removing symbols from anywhere in strings from } L\}$

xv) Sub-word (L) = $\{v \mid \text{for some } u, w, uwv \text{ is in } L\}$

* M_{min} number of states in DFA [Also on pg 1]

a) ~~Binary strings whose decimal equivalent is~~

~~divisible by n.~~

> If n is odd, # states = n

[eg. divisible by 7]

> If $n = 2^k$, # states = $k+1$

> If $n = 2^k \times m$, where m is odd, then #states = $k+m$

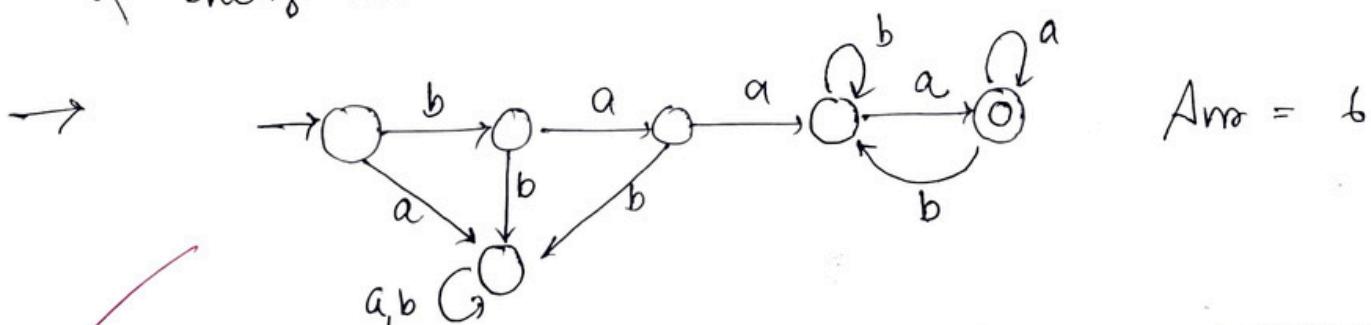
> Binary strings divisible by m or n

↳ Check for divisibility of $\text{lcm}(m, n)$

> Binary strings divisible by m or n #states = lcm(m, n)

↳ Check for $\text{lcm}(m, n)$ divisibility.

b) Min #states in DFA for string starting with baa
of ending with a. $\Sigma = \{a, b\}$

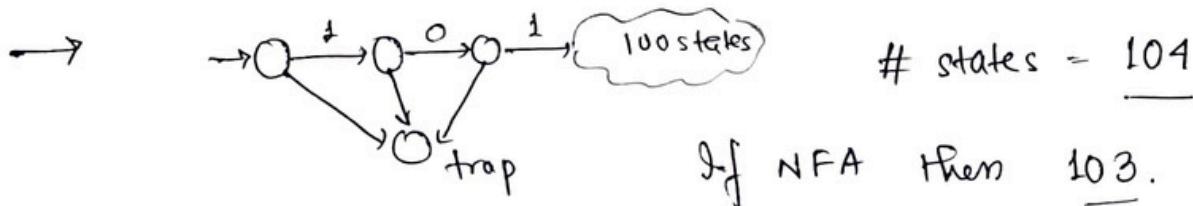


c) Min #states in DFA accepting strings containing
0's divisible by p & # 1's divisible by q = pq



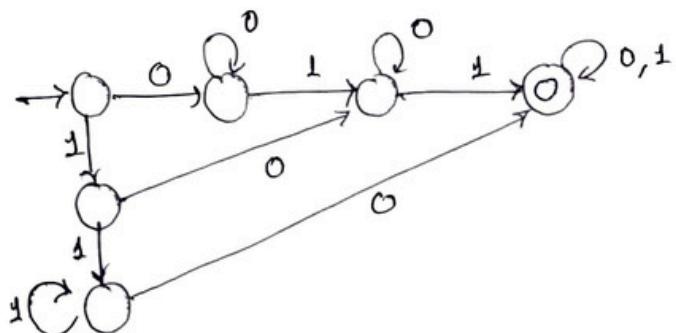
Report on

- Q # states in min DFA that accepts all binary strings that starts with 101 & divisible by 100.



- Q # states in min DFA in

$L = \{ w \mid w \in \{0,1\}^* \text{ & contains at least one } 0 \text{ & two } 1's \}$



CFG, CFL, PDA

* Context free grammar: $\alpha \rightarrow \beta$, $\alpha \in V$, $\beta \in (V \cup T)^*$, $|\alpha| = 1$
 (only one non-terminal at the LHS).

[If any symbol is present with the producing non-terminal at the LHS, then that symbol is called context. In CFG, no context is added, only one non-t. So, it's called CFG.]



Ex CFG examples.

1. Balanced parentheses. 2. $\{ w c w^r \mid w \in \{a,b\}^* \}$
 $S \rightarrow SS / (S) / \epsilon$ $S \rightarrow asa / bsb / c$

3. $(0+1)^* 01^*$ ✓ 4. Equal no. of 0's & 1's
 $S \rightarrow AOB$ $S \rightarrow SS / 0S1 / 1S0 / \epsilon$
 $A \rightarrow 0A / 1A / \epsilon$ 5. $\{ abba(baa)^n aab(aaba)^n \mid n \geq 0 \}$
 $B \rightarrow 1B / \epsilon$ $S \rightarrow abba A \quad A \rightarrow (baa) A \quad (aab) A \rightarrow aab$

6. $\{ a^n b^m c^m d^n \mid m, n \geq 1 \}$ 7. $\{ a^n b^n c^m d^m \mid m, n \geq 1 \}$
 $S \rightarrow aSd / aAd$ $S \rightarrow AB$
 $A \rightarrow bAc / bc$ $A \rightarrow aAb / ab \quad B \rightarrow cBd / cd.$

✓ 8. Consecutive 0 can occur, but consecutive 1's cannot.

$$\begin{aligned} S &\rightarrow 0S / 0 / 1 / 1A \\ A &\rightarrow 0 / 0S \end{aligned}$$

✓ 9. Set of all odd integers. | 10. Even integers.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow +/- \\ B &\rightarrow CD \\ D &\rightarrow 1/3/5/7/9 \\ C &\rightarrow CE / \epsilon \\ E &\rightarrow 0/1/2/3/4/5/6/7/8/9 \end{aligned}$$

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow +- \\ B &\rightarrow CD \\ C &\rightarrow CE / \epsilon \\ D &\rightarrow 0/2/4/6/8 \\ E &\rightarrow 0/.../9 \end{aligned}$$

- > DCFL has a one-one correspondence with LR(1) grammar & since LR(k) grammar is unambiguous, no DCFL is inherently ambiguous.
- > A DCFL can have an ambi. grammar
- > If lang. of the grammar is non-deterministic, the grammar can be ambig. or not-ambi.

Q. Grammar that generates inherently ambiguous language.

~~language~~

✓ 1. $S \rightarrow AB/CD, A \rightarrow aAb/\epsilon, B \rightarrow dB/\epsilon, C \rightarrow aC/\epsilon$
 $D \rightarrow bDd/\epsilon$

2. $S \rightarrow AB/CD, A \rightarrow aAb/\epsilon, B \rightarrow bB/\epsilon, C \rightarrow aC/bC/\epsilon$

3. Both 4. Neither $D \rightarrow bB/\epsilon$

→ 1. $\text{Lang} = (a^n b^n d^m \mid m, n \geq 0) \cup$

$(a^k b^l d^k \mid \cancel{m, n}, l, k \geq 0)$

Inherently ambiguous.

2. $\text{Lang} = (a+b)^* = (a^n b^{m+n} \mid m, n \geq 0) \cup$

$(w b^k \mid w \in (a,b)^*, k \geq 0)$

= DCFL \cup RL

= DCPL

(never ~~ambiguous~~)

* Backus Naur form (BNF) $\stackrel{\text{is defined as}}{:=}$ | or | < > category names.

e.g. Identifier in a programming language.

`<identifier>` := letter (letter|digit)*

$$\langle \text{letter} \rangle ::= a | \dots | z | A | \dots | Z$$

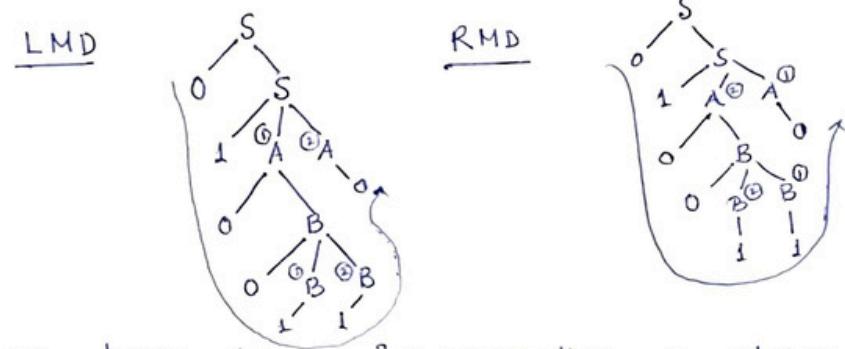
$\langle \text{digit} \rangle \quad ; = \quad 0 | \dots | 9$

* Derivation & Parse tree. Left & right most derivation

(Parse tree can be generated only for RLG, CFG, Not for CSG, as CSG contains prod^n like $aA \rightarrow bb$ or $AA \rightarrow B$.)

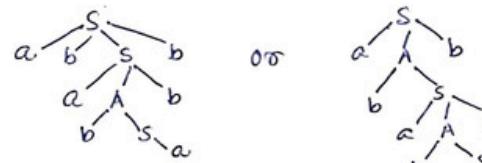
CFG: $S \rightarrow OS/1AA$
 $A \rightarrow 0/1A/0B$
 $B \not\Rightarrow 1/0BB$

String : 0100110



Ambiguity : More than one parse trees for generating a string.

e.g. $S \rightarrow a/abSb/aAb$ for string abababb,
 $A \rightarrow bS/aAAb$.



> Ambiguous CFL : w G L(G) that has at least 2 parse trees.

Inherently ambiguous CFL If all grammars for CFL L are ambiguous.

> Unambiguous CFL If \exists an unambiguous grammar for CFL (even if one is unambiguous).

(no particular rule to remove ambiguity). Some CFLs have only ambiguous grammars.)

- > Checking ambiguity has no algorithm. Undecidable. (Equivalent to post correspondence problem).

> Not always possible to make a grammar unambiguous.

> Try to remove ambiguity: i) Adding precedence rules

> Try to remove ambiguity: i) Adding precedence rules
 ii) Using semantics & choosing one that makes more sense . iii) Adding grouping rules iv) Fixing grammar (removing unit, null prodⁿs, useless prodⁿs).

> Removing ambiguity by precedence & associativity rules.

- Precedence constraint: Higher the level of prodⁿ, lower the priority of the operator.

- Associativity constraint: Operator left associative \rightarrow induce left recursion in its prodⁿ. If right associative \rightarrow right recursion.

eg $R \rightarrow R + R \mid R \cdot R \mid R^* \mid a \mid b$ (ambiguous)

Priority * > . > + and + is left, . is left associative

Unambiguous grammar $E \rightarrow E + T \mid T \quad | \quad + \text{ lowest priority}$
 $T \rightarrow T \cdot F \mid F \quad | \quad \downarrow \text{highest level of prod}^n$
 $F \rightarrow F^* \mid G$
 $G \rightarrow a \mid b$

eg $\text{exp} \rightarrow \text{exp or exp } \mid \text{exp and exp } \mid \text{not exp } \mid T \mid F$

Priority not > and > or | and, or. left associative

$\text{exp} \rightarrow \text{exp or exp}^1 / \text{exp}^2$

$\text{exp}^1 \rightarrow \text{exp}^1 \text{ and exp}^2 / \text{exp}^2$

$\text{exp}^2 \rightarrow \text{not exp}^2 / \text{exp}^3$

$\text{exp}^3 \rightarrow T / F$.

> If n parse trees are there for a string w, there are n corresponding lmds & n rmds.

> For ambiguous grammar, lmd & rmd represent different parse trees.

> For unambiguous grammar, lmd, rmd represent the same parse tree.

> There may exist derivations that are neither lm nor rm.

eg. $S \rightarrow ABC, A \rightarrow a, B \rightarrow b, C \rightarrow \epsilon$

$S \rightarrow ABC \rightarrow aBC \rightarrow aB \rightarrow abc \Rightarrow \text{neither lm nor rm.}$

> There may exist a grammar for which lmd & rmd for all strings are same.

- > For given parse tree, we may have its LRD, RLD same.
- > If for all strings, LRD is same as RLD, then that grammar may be ambiguous or unambiguous.

* Evaluating expression based on grammar.

a) By drawing parse tree (time taking)

b) By designing rules for operators (Precedence, associativity)
[↑ level of prod^n → ↓ priority
or priority

[↑ level of prod^n → ↓ priority

left recursion → left associative
 right n → right n
~~both~~ neither]

e.g. $E \rightarrow E \uparrow T / T \quad T \rightarrow T + F / F \quad F \rightarrow G - F \mid G \quad G \rightarrow id$

Precedence id > - > + > ↑

Associativity ↑, + left assoc. - right assoc.

Evaluate. $2 \uparrow 1 \uparrow 3 + 5 - 6 - 8 - 5 + 10 + 11 \uparrow 2$

✓

$$\begin{aligned}
 &= ((2 \uparrow 1) \uparrow 3 (((3 + (5 - (6 - (8 - 5))))) + 10) + 11) \uparrow 2 \\
 &\quad \uparrow \\
 &\quad \text{start from here while parenthesizing} \\
 &= (2^{2^6})^2
 \end{aligned}$$

Q Consider id > (x, ÷) > (+, -)

Given all are left associative construct grammar.

$$\begin{array}{l}
 \rightarrow \quad \left\{ \begin{array}{l} E \rightarrow E + T \mid E - T \mid T \\ T \rightarrow T \times F \mid T \div F \mid F \\ F \rightarrow id \end{array} \right.
 \end{array}$$

* Left recursion and Left factoring

> Left recursion $A \stackrel{+}{\Rightarrow} A\alpha \quad A \in V, \alpha \text{ any string}$

- Direct left recursion $A \rightarrow A\alpha$

- Indirect left recursion $A \rightarrow B\alpha$

$B \rightarrow A\beta$

$A_0 \rightarrow A_1\alpha_1$
 $A_1 \rightarrow A_2\alpha_2$
 \vdots
 $A_n \rightarrow A_0\alpha_{(n+1)}$

> Removing left recursion (as top down parser can't handle it)

a) Immediate/Direct (by Moore's proposal)

$A \rightarrow A\alpha / \beta \Rightarrow \boxed{A \rightarrow \beta A'}$ β not starting with A
 (starts in β , then any number of α)
 $A' \rightarrow \alpha A' / \epsilon$

In general, $A \rightarrow A\alpha_1 | \dots | A\alpha_m | \beta_1 | \dots | \beta_n$ and β_i not starting with A

$$\begin{aligned} &\Downarrow \\ A &\rightarrow \beta_1 A' | \dots | \beta_n A' \\ A' &\rightarrow \alpha_1 A' | \dots | \alpha_m A' | \epsilon. \end{aligned}$$

b) Indirect; Arrange non-terminals in some order A_1, \dots, A_n

Algo for i from 1 to n

for j from 1 to $i-1$

replace each prodⁿ $A_i \rightarrow A_j \gamma$ by

$A_i \rightarrow \alpha_1 \gamma | \dots | \alpha_k \gamma$ where $A_j \rightarrow \alpha_1 | \dots | \alpha_k$

eliminate left recursions among A_i prodⁿs.

eg $E \rightarrow E + T | T$ | 2 immediate LR's
 $T \rightarrow T * F | F$
 $F \rightarrow a | (E)$

removing LRs \downarrow

$$\begin{aligned} T &\rightarrow T * F \\ &\Downarrow \\ T &\rightarrow F T' \\ T' &\rightarrow * F T' / \epsilon \end{aligned}$$

$$\left(\begin{array}{lll} E \rightarrow T E' & T \rightarrow F T' & F \rightarrow a | (E) \\ E' \rightarrow + T E' / \epsilon & T' \rightarrow * F T' / \epsilon & \end{array} \right)$$

eg $S \rightarrow Aa / b$ | Has indirect LR. $S \rightarrow Aa$, $A \rightarrow Sd$
 $A \rightarrow Ac / Sd / f$ | Rename $A_1 \rightarrow A_2 a / b$ $A_2 \rightarrow A_2 c / A_1 d / f$

$i=1 j=1$ no prodⁿ as $A_1 \rightarrow A_1 \alpha$

$i=2 j=1 A_2 \rightarrow A_1 d \Rightarrow A_2 \rightarrow A_2 ad / bd$.

Prodⁿ for A_2 becomes $A_2 \rightarrow A_2 c / A_2 ad / bd / f$

Removing direct LR of A_2 prodⁿs $\rightarrow \begin{cases} A_2 \rightarrow bd A_2' / f A_2' \\ A_2' \rightarrow c A_2' / ad A_2' \end{cases}$

Now, change names to original.

> Left factoring [We remove, ~~as it requires~~ backtracking]

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n$$

\Downarrow

$$A \rightarrow \alpha A_1 \quad A_1 \rightarrow \beta_1 \mid \dots \mid \beta_n.$$

Non-redundant grammar:
Without any useless symbols (Reduced grammar)

* Simplification of CFG. (Removal of useless, null, unit prod's)

non-generating non-reachable

> Useless symbols.

eg. non-generating. [terminal]

$$\begin{cases} S \rightarrow AC \mid BA \\ C \rightarrow \cancel{B} \mid CB \mid AC \\ A \rightarrow a \\ B \rightarrow aC \mid b \end{cases}$$

\Downarrow remove prod's having C

$$\begin{cases} S \rightarrow BA \\ A \rightarrow a \\ B \rightarrow b \end{cases}$$

eg. non-reachable

$$\begin{cases} S \rightarrow aC \mid SB \\ A \rightarrow bSCa \mid ad \\ B \rightarrow aSB \mid bBC \\ C \rightarrow aBC \mid ad \end{cases}$$

\Downarrow remove non-generating (B)

$$\begin{array}{l} S \rightarrow aC \\ A \rightarrow bSCa \mid ad \\ C \rightarrow ad \Rightarrow A \text{ non-reachable} \\ S \rightarrow aC \\ C \rightarrow ad \end{array}$$

> Unit prod's ($A \rightarrow B$) ($NT \rightarrow NT$)

eg. $S \rightarrow AB, A \rightarrow E, B \rightarrow C, C \rightarrow D, D \rightarrow \underline{B}, E \rightarrow a$
 \Downarrow removing

$$S \rightarrow AB, A \rightarrow a, B \rightarrow b.$$

eg. $S \rightarrow \alpha X / \gamma b / \gamma, X \rightarrow S, Y \rightarrow Yb/b$

\Downarrow

$$S \rightarrow \alpha X / \gamma b / \gamma b/b, X \rightarrow \alpha X / \gamma b / \gamma, Y \rightarrow Yb/b.$$

> Null prod's ($NT \rightarrow \epsilon$) ✓ [If we get $S \xrightarrow{*} \epsilon$ then that null prod can't be removed]

(remove one by one) (find on the RHS where the NT exists)

$$eg. S \rightarrow aA \quad A \rightarrow b/\epsilon$$

\Downarrow

$$S \rightarrow aA/a$$

$$A \rightarrow b$$

$$eg. S \rightarrow ABaC \quad A \rightarrow BC \quad B \rightarrow b/\epsilon \quad C \rightarrow D/\epsilon \quad D \rightarrow d$$

$\Downarrow B \rightarrow \epsilon, C \rightarrow \epsilon$

$$S \rightarrow ABaC / Aac / ABA / Aa$$

$$A \rightarrow BC / B / C / \epsilon$$

$$B \rightarrow b \quad C \rightarrow D \quad D \rightarrow d$$

$A \rightarrow \epsilon$

$$\Rightarrow S \rightarrow ABac / AaC / ABa / Aa / Bac / ac / Ba / a$$

$$A \rightarrow B / C / BC \quad B \rightarrow b \quad C \rightarrow D \quad D \rightarrow d.$$

13

eg $S \rightarrow aS / A \quad \} \quad S \rightarrow A \rightarrow \epsilon \quad \text{so } A \rightarrow \epsilon \text{ can't be removed.}$
 $A \rightarrow aA / \epsilon \quad \}$

* Linear grammar: Grammar not having context (only NT in LHS) and at most 1 NT on the RHS. $S \rightarrow abSc$

> Left linear: $A \rightarrow Ba$. NT on the left end

> Right linear: $A \rightarrow aB$. NT on the right end

} Regular grammar

> Converting linear grammar into regular grammar

(not possible if not left or right linear)

right linear $\left\{ \begin{array}{l} S \rightarrow bas / aA \\ A \rightarrow bbA / bb \end{array} \right. \Rightarrow$

$$\left\{ \begin{array}{l} S \rightarrow bB / aA \\ B \rightarrow as \\ A \rightarrow bb / bc \\ C \rightarrow bA \end{array} \right.$$

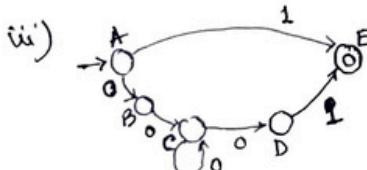
Left linear ↗
 ↖ Right linear

- i) regex from given grammar
- ii) reverse regex
- iii) build FA
- iv) generate regular right linear grammar from FA
- v) reverse RHS of each prodⁿ

eg $S \rightarrow 10A / 1 \quad A \rightarrow 0A / 00$

i) $(100^*00 + 1)$

ii) $(1 + 000^*01)$



iv) $A \rightarrow 0B / 1E / 1$ v) $A \rightarrow B0 / 1$
 $B \rightarrow 0C$ $B \rightarrow CO$
 $C \rightarrow 0C / 0D$ $C \rightarrow CO / DO$
 $D \rightarrow 1E / 1$ $D \rightarrow 1$

E useless

Right linear ↗
 ↖ Left-linear

i) reverse right LHS of each prodⁿ

ii) regex

iii) reverse regex

iv) FA

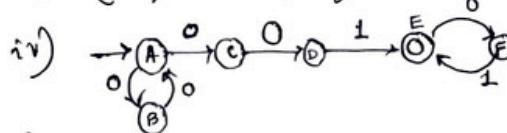
v) generate right linear grammar from FA.

eg $S \rightarrow S10 / A1 \quad A \rightarrow A00 / 00$

i) $S \rightarrow 01S / 1A \quad A \rightarrow 00A / 00$

ii) $(01)^* 1 00 (00)^*$

iii) $(00)^* 00 1 (01)^*$



v) $A \rightarrow 0B \quad B \rightarrow 0A$
 $A \rightarrow 0C \quad C \rightarrow 0D$
 $D \rightarrow 1E / 1 \quad E \rightarrow 0F$
 $F \rightarrow 1E / 1$

* Normal forms.

1. Chomsky Normal Form. If all prodⁿs of CFG are in the form:

$$\begin{array}{ll} (\text{CNF}) & \text{NT} \rightarrow \text{string of exactly 2 NTs} \\ & \text{NT} \rightarrow \text{Single terminal.} \end{array} \quad \begin{array}{l} A \rightarrow BC \\ A \rightarrow a \end{array}$$

> Converting to CNF

1. Eliminate ~~wild~~, unit, null prodⁿs.

*check if
in CNF
already*

2. Eliminate terminals on the RHS of length 2 or more.

3. Restrict # NTs on RHS to two.

4. Convert string containing NTs & Ts to string of only NTs on the RHS.

eg $S \rightarrow bA / aB \quad A \rightarrow bAA / as / a$
 $B \rightarrow aBB / bs / b$
 ↓ (already simplified)

$$\left\{ \begin{array}{l} N_a \rightarrow a \quad N_b \rightarrow b \\ S \rightarrow N_a B | N_b A \\ A \rightarrow N_b AA | N_a S | a \\ B \rightarrow N_b BB | N_b S | b \end{array} \right. \rightarrow$$

$$\left. \begin{array}{l} S \rightarrow N_a B | N_b A \\ A \rightarrow N_b S | N_a S | a \\ B \rightarrow N_a T | N_b S | b \\ N_a \rightarrow a \quad N_b \rightarrow b \\ S \rightarrow AA \quad T \rightarrow BB \end{array} \right\} \text{in CNF}$$

eg $\left\{ \begin{array}{l} S \rightarrow abAB \quad A \rightarrow \epsilon \\ A \rightarrow bAB / \epsilon \quad \rightarrow \\ B \rightarrow BAa / \epsilon. \end{array} \right.$

$$\begin{array}{l} S \rightarrow abAB / abB \quad B \rightarrow \epsilon \\ A \rightarrow bAB / bB \quad \rightarrow \\ B \rightarrow BAa / \epsilon / Ba \end{array}$$

$$\begin{array}{l} S \rightarrow abAB / abA / abB / ab \\ A \rightarrow bAB / bA / bB / b \\ B \rightarrow BAa / Aa / Ba / a \end{array}$$

$$\left\{ \begin{array}{l} S \rightarrow CD | CA | CB | N_a N_b \\ A \rightarrow N_b D | N_b A | N_b B | b \\ B \rightarrow EN_a | AN_a | BN_a | a. \\ C \rightarrow N_a N_b \\ D \rightarrow AB, \quad E \rightarrow BA \end{array} \right.$$

$$\left. \begin{array}{l} N_a \rightarrow a \quad N_b \rightarrow b \\ S \rightarrow N_a N_b AB / N_a N_b A / \\ N_a N_b B / N_a N_b \\ A \rightarrow N_b AB / N_b A / N_b B / b \\ B \rightarrow BAN_a / AN_a / BN_a / a \end{array} \right. \leftarrow$$

2. Greibach Normal form.

$$\left\{ \begin{array}{l} \text{NT} \rightarrow (\text{single T}) (\text{string of NTs}) \\ \text{NT} \rightarrow (\text{single T}) \end{array} \right.$$

$$\text{NT} \rightarrow (\text{single T}) (\text{NT})^*$$

⇒ Converting to GNF.

- Eliminate null, unit prod's.
 - Rename NTs of V as A_1, A_2, \dots, A_n . Start symbol A_1
 - Make all prod's of the form $A_i \rightarrow A_j \cup \dots \cup A_m$
- \vdots
- $$A \rightarrow A\alpha / \beta_1 / \beta_2 / \dots / \beta_m$$
- $$\downarrow$$
- $$A \rightarrow \beta_1\alpha / \beta_2\alpha / \dots / \beta_m\alpha$$

iv) Use this -

$$A \rightarrow A\alpha_j / \beta_i \quad | \quad 1 \leq j \leq m \\ | \quad 1 \leq i \leq n$$

↓ Introduce a NT X

$$A \rightarrow \beta_i / \beta_i X$$

$$X \rightarrow \alpha_j / \alpha_j X$$

(X , prod's not in GNF) →

$$X_1 \rightarrow aA_1A_2 / aA_1X_1A_2 / bA_1 / \\ bX_1A_2 / aA_1A_1X_1 / aA_1X_1A_1X_1 / \\ bA_1X_1 / bX_1A_1X_1$$

So, the GNF →

$$A_1 \rightarrow aA_1A_2 / bA_2 / bX_1A_1 / aA_1X_1A_2 / a$$

$$A_2 \rightarrow aA_1 / aA_1X_1 / b / bX_1$$

$$X_1 \rightarrow \dots$$

> Advantages of GNF

- # steps reqd to generate string w is $|w|$.

- Useful to convert CFG to PDA.

↳ Converting to GNF.

$$S \rightarrow AA / a \quad A \rightarrow SS / b.$$

(Already in CNF)

Rename $S \rightarrow A_1$, $A \rightarrow A_2$

$$A_1 \rightarrow A_2A_2 / a \quad \underbrace{A_2 \rightarrow A_1A_3 / b}_{\text{use (iv)}}$$

$$\left\{ \begin{array}{l} A_2 \rightarrow aA_1 / aA_1X_1 \\ X_1 \rightarrow A_2A_1 / A_2A_1X_1 \\ A_2 \rightarrow b / bX_1 \end{array} \right. \quad \begin{array}{l} \xleftarrow{\text{use (iv)}} \\ A_1 \\ \xrightarrow{\text{not GNF.}} \end{array}$$

$$A_1 \rightarrow aA_1A_2 / bA_2 / bX_1A_2 / \\ / aA_1X_1A_2 / a$$

• Important points on Normal forms

1. CNF ≈ Binary standard form

2. CNF # prod's reqd = $2^{|w|} - 1$

3. G be CNF grammar & T be derivation tree for

string 'x' in G. If length of longest path is $= k$,

then yield length $\leq 2^{k-1}$

4. For generation of l length yield, min height of derivation tree for given CNF CFG is $\lceil \lg_2 l \rceil + 1$ + max

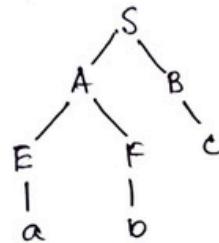
height is h . $h = 2^{h-1} \Rightarrow h = \log_2 h + 1$

✓ 5. CFG G w/o null prodⁿs & unit prodⁿs of K be the max # symbols on the RHS of any prodⁿ.

Then the eqn. CNF contain a max of $(K-1) |P| + |T|$ prodⁿs. [$|P| \rightarrow \# \text{prod}^n \text{s of } G, |T| \rightarrow \# \text{terminals}$.]

✓ eg $\begin{array}{l} S \rightarrow AB \\ A \rightarrow EF \\ C \rightarrow a \\ B \rightarrow C \\ F \rightarrow b \end{array}$ find max. yield length.

String abc



max path length = 3

max yield length = $2^{3-1} = 4$.

]

6. No CNF or GNF CFGs generate the null string.

7. No of prodⁿs needed in GNF for string $w = |w|$

* CFG examples.

1. $\{a^m b^n \mid m \neq n\}$ $S \rightarrow A \mid B$ $A \rightarrow aAb \mid aA/a$
 $B \rightarrow aBb \mid Bb \mid b$.

2. Odd len. palindromes.

$$S \rightarrow aSa \mid bSb \mid a \mid b$$

3. $n_a = n_b$.

$$S \rightarrow aSbS \mid bSaS \mid \epsilon$$

or
$$S \rightarrow aB \mid bA \mid \epsilon$$
$$A \rightarrow aS \mid bAA \mid a$$
$$B \rightarrow bS \mid aBB \mid b$$

4. $\{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$
 $S \rightarrow A \mid B$
 $A \rightarrow aAb \mid ab$
 $B \rightarrow aBbb \mid abb$

* Closure Properties of CFG: Closed under —

- a) Union, b) ~~Intersection~~ Reversal, c) concatenation, d) Star Closure,
- e) Homomorphism, f) Inverse homomorphism g) Substitution
- h) Prefix i) Right quotient with RL. j) Cycle $\xleftarrow{\text{op}}$
- k) Union with regular language l) Intersection with RL

m) Right difference with RL.

→ CFLs are not closed under →

a) Intersection. b) Complement c) Difference (Set)

d) XOR, NAND, NOR, e) Subset/Superset f) Left difference
with RL (RL - CFL)

Examples. a) Union. $L_1 = \{a^n b^n \mid n \geq 1\}$ $A \rightarrow aAb / ab$
 $L_2 = \{a^n b^{2n} \mid n \geq 1\}$ $B \rightarrow aBbb / abb$

$L_1 \cup L_2 : S \rightarrow A1B \quad A \rightarrow \dots \quad B \rightarrow \dots \quad L_1 \cup L_2$ is CFL

✓ b) Intersection $L_1 = \{a^n \mid n \geq 0\}$ $L_2 = \{a^n \mid n \text{ is even}\}$
 $L_1 \cap L_2$ is CFL as $L_1 \cap L_2 = \{a^n \mid n \text{ is even}\}$

[But, $L_3 = \{a^n b^n c^m \mid n, m \geq 1\}$ $L_4 = \{a^n b^m c^n \mid n, m \geq 1\}$
 $L_3 \cap L_4 = \{a^n b^n c^n \mid n \geq 1\}$ is not CFL.]

✓ c) Complement. $L = \{\overline{ww} \mid w \in (a+b)^*\}$ is CFL.
 $\bar{L} = \{ww \mid w \in (a+b)^*\}$ is not CFL.

$L_1 = \{ww^R \mid w \in (a+b)^*\}$ is CFL.
 $\bar{L}_1 = \{\overline{ww^R} \mid w \in (a+b)^*\}$ is also CFL.

✓ d) Concatenation $L_1 = \{a^n b^n \mid n \geq 1\}$ $L_2 = \{c^m d^m \mid m \geq 1\}$
 $S \rightarrow AB \quad A \rightarrow \dots \quad B \rightarrow \dots \quad \underbrace{L_1 \cdot L_2 \text{ is CFL.}}_{\downarrow} \quad A \rightarrow aAb / ab \quad B \rightarrow cBd / cd$

e) Kleene closure $L = \{a^n b^n \mid n \geq 1\}$ $A \rightarrow aAb / ab$
 $L^* \text{ is CFL. } L^* \rightarrow S \rightarrow \epsilon / AS \quad A \rightarrow aAb / ab$

f) Reversal $L = \{a^n b^n \mid n \geq 1\}$ $L^R = \{b^n a^n \mid n \geq 1\}$ also CFL.

g) Substitution $L = \{a^n b^n \mid n \geq 1\}$ $f(a) = 0^k 1^k \quad f(b) = 1^m 0^m$
 $CFG(L) : A \rightarrow aAb / ab$
 $CFG(f(L)) : A \rightarrow XAY / XY \quad \left. \begin{array}{l} X \rightarrow 0^k 1^k / \epsilon \\ Y \rightarrow 1^m 0^m / \epsilon \end{array} \right\} CFL. \checkmark$

b) Homomorphism. $L = \{a^n b^n \mid n \geq 1\}$ $h(a) = 01$ $h(b) = 111$

$CFG(L) : A \rightarrow aAb/ab$

$CFG(h(L)) : A \rightarrow 01A111 / 01111. \rightarrow CFL \checkmark$

* Closure properties of DCFL.

Closed under - i) Complement, ii) Inverse homomorphism.

iii) Union with RL iv) Intersection with RL v) Right difference with RL (DCFL - RL)

vi) Right quotient with RL (DCFL / RL)

vii) Left diff. with RL (RL - DCFL) viii) Prefix, ix) Min x) Max.

Not closed under - i) Union ii) Intersection iii) Difference

iv) Concat v) Kleene closure vi) Positive closure vii) Reversal

viii) Homomorphism ix) Substitution.

* Testing CFL or not.

> Pumping Lemma for CFL : For any CFL L , it's possible to find 2

substrings that can be pumped any number of times & still be in the same language.

- Let L be a CFL. Then we can find a natural number n such that

1. Every $z \in L$ with $|z| \geq n$ can be written as

$\boxed{z} = uvwxy$ for some strings u, v, w, x, y .

2. $|vwx| \leq n$

3. $|zx| \geq 1$

4. $uv^iwx^i y \in L$ for all $i \geq 0$.

> Proving a L is not CFL

I. Assume L is CFL. Find a natural number n , $|z| \geq n$.

II. $z = uvwxy$

III. Find k s.t. $uv^kwx^ky \notin L$. Contradiction!

eg. $\{a^n b^m c^n \mid n \geq 1\}$

$$z = a^n b^n c^n \quad |z| = 3n > n$$

$$z = uvwxy = a^n b^n c^n. \quad \left| \begin{array}{l} 1 \leq |vwx| \leq n \quad \text{as } |vwz| \leq n \\ \Rightarrow v \text{ or } x \text{ can't contain all the 3 symbols.} \end{array} \right.$$

v or x will be in any of

these forms:

1. Contain only a & b . $a^i b^j$

2. Contain n b & c $b^i c^j$

3. Contain n repetition of any of the ~~the~~ symbols.

take value of $k=2$.

(1,2) v^2 or x^2 in the form $a^i b^j a^i b^j$ or $b^i c^j b^i c^j$.

pump up

$\therefore uv^2 w x^2 y$ not in the form $a^n b^n c^n$. So, $uv^2 w x^2 y \notin L$.

(3) v or x of the form a^i , b^i or c^i . Take k as 0.

$uv^0 w x^0 y = uw$ not in the form as number of occurrences of one of the other 2 symbols in uw is less than n . So, $uv^0 w x^0 y \notin L$.

\therefore it's not CFL.

Observations

1. Every RL is CFL. eg. $\{a^m b^p c^k d^n \mid m, p, k, n \geq 1\}$
RL, CFL

2. (CFL when) given an expⁿ s.t. it's possible to obtain a center or mid point in the string^s, so we can carry out comparison of left & right sub-parts using stack.

eg. $a^n b^n \mid n \geq 1$
CFL

$a^m b^n c^{m+n}$.
CFL

$a^n b^{2n}$
CFL ✓ $\begin{pmatrix} \text{push 2 a's for +a} \\ \text{pop 1 a when } b \end{pmatrix}$

$a^m b^n c^n$

X CFL

✓ Given \exp^n is a combination of multiple \exp^n 's with mid-points in them, such that each sub-expression is independent of other sub-expressions, then it is context-free.

e.g. $\{a^m b^m c^n d^n\}$ | $\{a^m b^n c^m d^n\}$ is not CFL
CFL

✓ Given \exp^n consists of an opⁿ where mid point could be found along with some independent regular expressions in between, results into CFL.

e.g. $a^m b^i c^m d^n$ | b^i, c^m are regular in between
CFL. that does not affect stack

✓ An \exp^n that doesn't form a pattern on which linear comparison could be carried out using stack is not CFL.

e.g. $a^m b^{n^2}$ not CFL | $\{a^m \mid m \text{ prime}\}$ not CFL
 a^{n^2} not CFL

6. An \exp^n that involves counting & comparison of 3 or more variables independently is not CFL as stack allows comparison of 2 variables at a time.

e.g. $a^n b^n c^n$ not CFL | $\{a^i b^j c^k \mid i > j > k\}$
 $\{w \mid n_a = n_b = n_c\}$ not CFL not CFL

7. Counting & comparison only with top of stack.
Language exhibiting characteristics that involves comparison with bottom of stack is not CFL.

e.g. $\{a^m b^n c^m d^n\}$ not CFL | $\{ww \mid w \in \{a,b\}^*\}$ not CFL

8. If we can find mid-pt in the \exp^n even in a non deterministic way, then it's CFL.

e.g. $ww^R \mid w \in (a+b)^*$ | $a^i b^j c^k d^l \mid i=k, j=l$
CFL (not DCFL) CFL (not DCFL)

g. DCFL or CFL? For a language to be DCFL, it should be clear when to push

or pop. $\{a^n b^n\}$ is DCFL (as well as CFL, DCFL ⊂ CFL) as we know to push a's, pop b's always. But, ww^r is CFL or it is recognised by NPDA as we don't exact point where w starts.

eg. CFL but not DCFL - $\{ww^r\}$, - $\{a^i b^j c^k \mid i=j \text{ or } k=i\}$,
 $\{-\{a^n b^n c^m \cup a^n b^m c^n\}\}$, ✓ $\{a^n b^m c^x d^y\}$

✓ $\{a^n b^x c^m d^y \mid n=m \text{ or } x=y\}$

↳ 2 copies of PDA : (push a's, ignore b)
 (pop c, ignore d)

(ignore a, push b, ignore c, pop d)

$n=m$ or
 $x=y$

↓ 2 copies
 of PDA

{ push a; pop b; ignore c;
 or
 ignore a,b; push c, pop d }

- $\{wxw^r \mid w, x \in \{0,1\}^*\}$.

• ✓ DCFL ∪ RL = DCFL
 DCFL ∩ RL = DCFL

* Examples. (CFL/DCFL/RL)

1) $\{a^n b^n c^m \mid n > m\}$ not CFL

2) $\{zcy \mid z,y \in \{0,1\}^*\}$ RL, DCFL, CFL

3) $\{a^i b^j c^k \mid j = i+k\}$ DCFL, CFL [$k = j-i$]

4) $\{w \mid w \in \{a,b\}, m_a = m_b + 1\}$ DCFL, CFL

Hierarchy

5) $\{a^n b^n c^n \mid n \geq 0\}$ not CFL

{ If only DCFL written, it's not RL, and it's CFL.

6) $\{ww \mid w \in \{0,1\}^*\}$ not CFL

{ If only RL → RL, DCFL, CFL all

7) $\{a^i b^j c^k \mid 0 \leq i \leq j \leq k \leq n\}$ not CFL

{ If CFL only → not RL or not DCFL

8) $\{a^n \mid n \geq 0\}$ not CFL

✓ 9) $\{a^n b^j \mid n = j^2\}$ not CFL

10) $\{a^{rs} \mid r,s \text{ prime}\}$ not CFL.

11) $a^{m+n} b^n c^m \mid n, m \geq 1$ XRL, DCFL, CFL

12) $a^m b^{m+n} c^n \mid m, n \geq 1$ XRL, DCFL, CFL

13) $a^m b^n c^{m+n}$ DCFL

- DCFL
- 14) $a^m b^m c^n d^n$ DCFL
 16) $a^m b^m c^n d^m$ DCFL
 18) $a^m b^m \mid m > n$ DCFL
 20) $a^n b^{n^2}$ X CFL
 22) $ww \mid w \in (a+b)^*$ X CFL
 24) $a^n b^n c^n d^n \mid n \leq 10^{10}$
 RL, DCFL, CFL
 26) $wwr \mid |w| = \ell$
 ℓ known RL, DCFL, CFL
 28) $a^m b^{3^n}$ X CFL
 ✓30) $a^m b^n \mid m = 2n+1$ DCFL
 32) $a^k \mid k \text{ even}$ DCFL
 1st a push, 2nd a pop,
 3rd a n , 4th a pop.
 ✓33) $a^i b^j c^k \mid i > j > k$ X CFL
 ✓36) $a^i b^j c^k d^l \mid i=k \text{ and } j=l$ X CFL
 38) $a^n b^{4m} \mid n, m \geq 1$ RL
 40) a^{n^n} X CFL
 42) $w \mid n_a = n_b \neq n_c$ X CFL
 44) $wwr wr$ X CFL
 46) $0^i 1^j \mid j = i^2$ X CFL
 48) $a^n b^{n+m} c^p$ DCFL
 ✓50) $a^m b^n c^p \mid m = n \text{ or } n = p$
 ↑ CFL
 52) $wwr w$ X CFL
 54) $wwr w^r$ X CFL
 ✓56) $\{canb^n \mid n \geq 0\} \cup \{danb^{2n} \mid n \geq 0\}$ DCFL

- 15) $a^m b^n c^m d^n$ X CFL
 ✓17) $a^m b^i c^m d^k$ DCFL
 19) $a^n b^{2n} \mid n \geq 1$ DCFL
 21) $a^n b^{2^n}$ X CFL
 23) $a^n b^n c^m \mid m > n$ X CFL
 ✓25) $a^n b^{2n} c^{3n} \mid n \geq 1$ X CFL
 27) www^r X CFL
 ✓29) $a^m b^n \mid m \neq n$ DCFL
 ✓31) $a^i b^j \mid i \neq 2j+1$ DCFL
 (whether by the time a's are over,
 b's remaining (or) by the time
 b's over, a's remaining)
 34) $a^i b^j c^k \mid j = i+k$ DCFL
 ✓35) $a^i b^j c^k d^l \mid i=k \text{ or } j=l$
 CFL (not DCFL)
 37) $a^m b^l c^k d^n \mid m, l, k, n \geq 1$
 RL
 39) $\{a^3, a^8, a^{13}, \dots\}$ RL
 41) $w \mid |w| \geq 100$ RL
 43) $w \mid n_a \geq n_b + 1$ DCFL
 45) $a^p \mid p \text{ is composite number}$
 X CFL
 ✓47) $0^i 1^j 0^k \mid j = \max(i, k)$ X CFL
 ✓49) $w \mid n_a = n_b + n_c$ DCFL
 ✓51) $aww \mid a, w \in \{0,1\}^*$ RL
 ✓53) $a^n b^n c^i \mid i \neq n$ X CFL
 55) $a^m b^{3n}$ DCFL
 56) 8tring of balanced
 parentheses DCFL
 ✓58) $\{an^cb^n\} \cup \{an^db^{2n}\}$ DCFL

> Some points regarding identification of language.

- [- $CFL \cap RL = CFL$ (need not be RL) - $RL - CFL =$ need not be CFL]
- [- $CFL - RL = CFL$, $CFL \cup RL = CFL$, $CFL / RL = CFL$]
- [- $DCFL - RL = DCFL$, $DCFL \cap RL = DCFL$, $DCFL \cup RL = DCFL$]
- [- $DCFL / RL = DCFL$, $RL - DCFL = DCFL$]
- CFL not closed under complement, difference, intersection.
- $DCFL$ n n n union, intersection, difference, concat, *, +, reversal, homomorphism, substitution.

* Decision problems in CFG, CFL

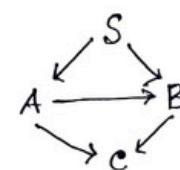
1. Emptiness : If we can't derive any string of terminals from given grammar, then it's empty language.

- Algo
- > Eliminate useless symbols (unreachable, nongenerating)
 - > If the start symbol itself is useless, then empty.
Otherwise, non-empty.

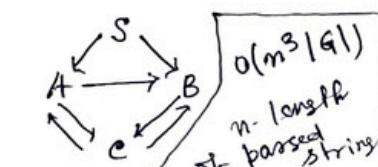
2. Finiteness : Finite if there are no cycles in the directed graph generated from prodⁿ rules of the grammar.

- Algo
- > Eliminate ϵ , unit, useless prodⁿs. (Convert to CNF)
 - > Draw directed graph from the grammar prodⁿ rules.
 - > If cycle present then infinite. Otherwise not.

eg $S \rightarrow AB$ $A \rightarrow BC/a$ $B \rightarrow cc/b$ $C \rightarrow a$
No cycle, so finite language.



eg $S \rightarrow AB$ $A \rightarrow BC/a$ $B \rightarrow cc/b$ $C \rightarrow a/AB$
Has cycle, infinite language.



3. Membership : CYK (Cocke-Younger-Kasami) Algorithm. $O(n^3 |G|)$
(DP algo, bottom-up)
Build a triangular table where each row
of the table corresponds to one particular
length of sub string. [Bottom most - of length n]
(Grammar has to be in CNF.) m - length of string

$|G|$ - # rules in G.

Eg $S \rightarrow AB/BC$

$A \rightarrow BA/a$

$B \rightarrow CC/b$

$C \rightarrow AB/a$

Check if $baaba \in L(CFG)$?

| | | 5 ^a | 4 ^b | 3 ^a | 2 ^a | 1 ^b |
|----------------|---------------------|--------------------|--------------------|------------------|----------------|----------------|
| 1 ^b | x_{15} A, S, C | x_{14} ϕ | x_{13} ϕ | x_{12} A, S | x_{11} B | |
| 2 ^a | x_{25} S, A, C | x_{24} B | x_{23} B | x_{22} A, C | | |
| 3 ^a | x_{35} B | x_{34} S, C | x_{33} A, C | | | |
| 4 ^b | x_{45} A, S | x_{44} B | | | | |
| 5 ^a | x_{55} A, C | | | | | |

If x_{1n} can
be generated,
then done.

First fill
 $x_{11}, x_{12}, x_{33}, x_{44},$
 x_{55}
 then $x_{12}, x_{23},$
 x_{34}, x_{45}
 ...

x_{ij} in the table

represents - x (string)
starting from location
i and ~~ending in locⁿ j~~

~~ending in locⁿ j.~~

Replace x_{ij} with v_{ij} . (as $x_{11} \rightarrow B$)

(v_{ij} - set of variables in the grammar that can derive
the substring x_{ij} . If the set consists of start symbol,
then it is clear that substring x_{ij} can be derived.)

Explanation: ① $x_{12} \rightarrow \underline{ba, a ba}$ to generate ba \rightarrow

$$(12) = (11)(22) = \{B\} \{A, C\} - \{BA, BC\}$$

Check if any prod^n's rhs contains BA or BC.

$BA \rightarrow$ from A, $BC \rightarrow$ from S. Put A, S.

$$\textcircled{2} \quad x_{23} \rightarrow aa \quad (23) = (22)(33) = \{A, C\} \{A, C\} \\ = \{AA, AC, CA, CC\}$$

$$\textcircled{3} \quad x_{13} \quad (13) = (11)(23) = \{B\} \{B\} = \{BB\} \\ = (12)(33) = \{A, S\} \{A, C\} = \{AA, AC, SA, SC\}$$

$$\textcircled{4} \quad x_{14} \quad (14) = (11)(24) \\ = (12)(34) \\ = (13)(44) \quad \textcircled{5} \quad (15) = (11)(25) \\ = (12)(35) \\ = (13)(45) \\ = (14)(55)$$

Go back through the process to get the derivation
tree for baaba.

Another question that can be asked - How many substrings of baaba can be generated from the grammar?

→ Checks on which cells there is S ($\alpha_{12}, \alpha_{34}, \alpha_{45}, \alpha_{25}$). So, the substrings are: ba, ab, ab, aaba
total 3 substrings.

| | | $L(G)$ regular | | $L(G)$ finite | |
|------|--------------|--------------------|-------------------|---------------------------|-------------------|
| | | DCFL | | D | |
| | | CFL | | . | |
| DCFL | $w \in L(G)$ | $L(G) = \emptyset$ | $L(G) = \Sigma^*$ | $L(G_1) \subseteq L(G_2)$ | $L(G_1) = L(G_2)$ |
| | D | D | D | UD | D |
| CFL | D | D | UD | UD | UD |

Application of directed transition graph of CFG

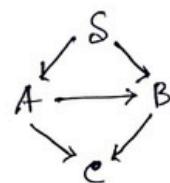
Rank of a variable can be computed if the graph has no cycle [for finite language rank is computed.]

Rank of a var = length of longest path from that var.

If a var. has rank r then any string generated from the var. is of length $\leq 2^r$.

If $A \rightarrow BC$ (non-redundant CNF) then $r(A) > r(B), r(C)$.

eg
 $S \rightarrow AB$
 $A \rightarrow BC / a$
 $B \rightarrow CC / b$
 $c \rightarrow a$

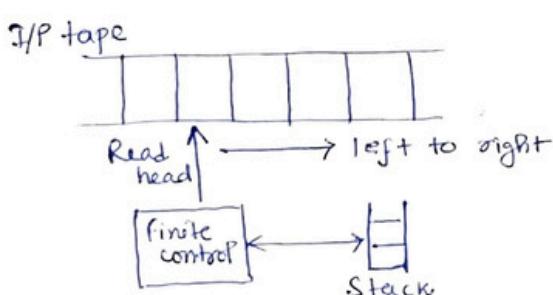


$$\begin{aligned}
 r(S) &= 3 & S \rightarrow A \rightarrow B \rightarrow C \\
 r(A) &= 2 \\
 r(B) &= 1 & r(c) = 0
 \end{aligned}$$

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

Γ - stack symbols (alphabet)
 z_0 - stack bottom or initial top

$$\begin{aligned}
 DPDA \rightarrow \delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma &\rightarrow Q \times \Gamma^* && \text{initial top} \\
 NPDA \rightarrow \delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma &\rightarrow 2^{Q \times \Gamma^*} && \text{(only finite subsets)}
 \end{aligned}$$

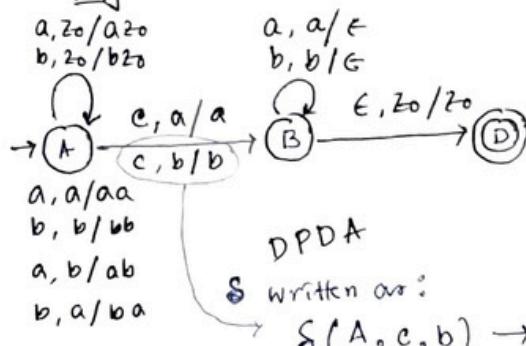
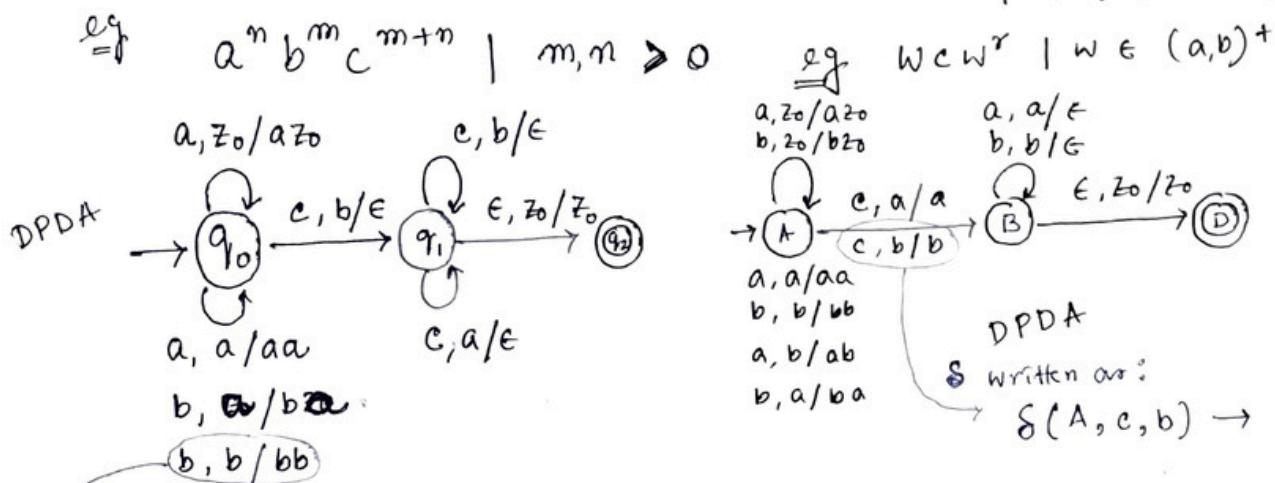


Acceptance by -

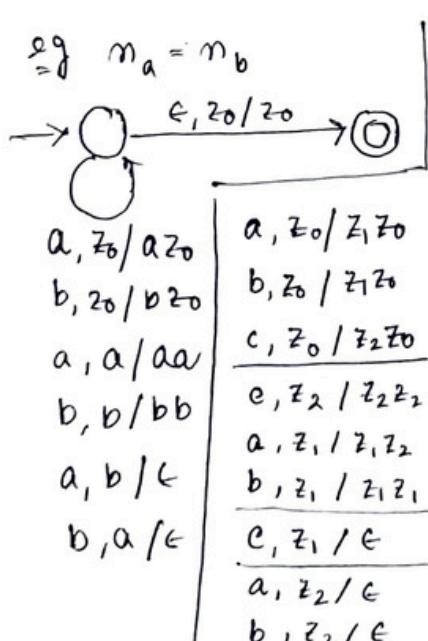
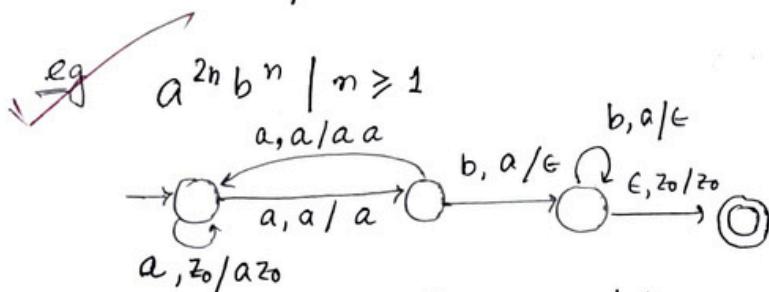
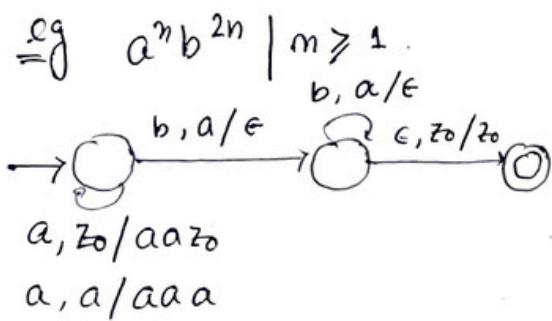
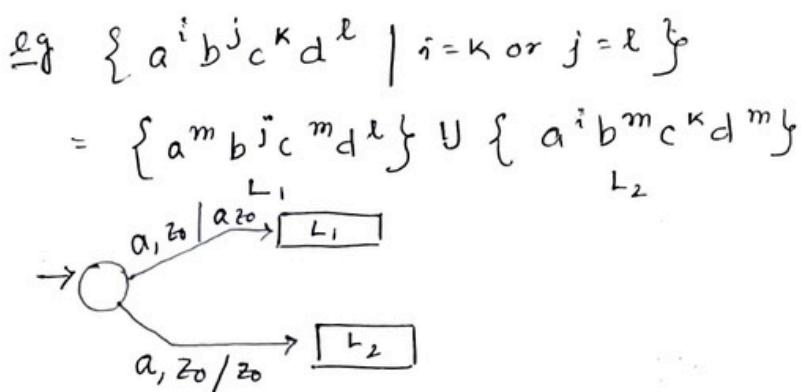
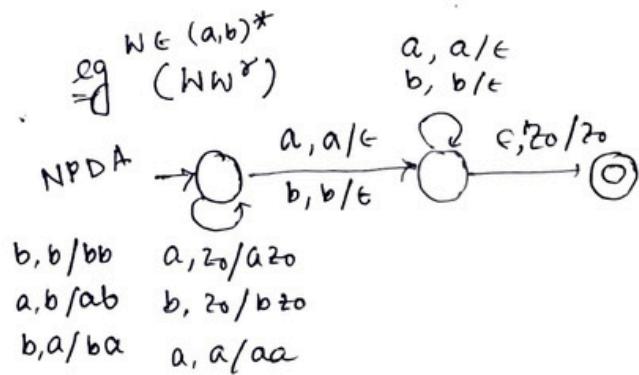
- Empty stack
- final state

> Constructing PDA

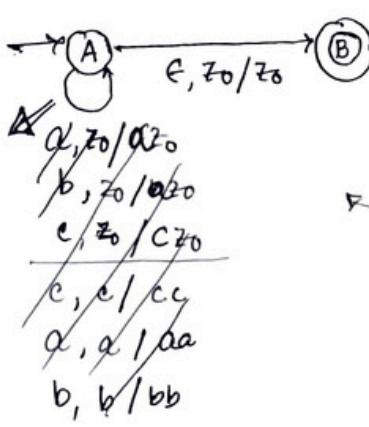
δ in the form $A, B \rightarrow C$ or $A, B/C$:
reads A as input, pops B, pushes C.



read as: if the head reads 'b' as input, and finite control checks the top of the stack to be 'b' then pop 'b' and push 'bb' in the stack.



eg $n_a + n_b = n_c$



* ID (Instantaneous descⁿ)
(state, input, stack)

current remaining contents

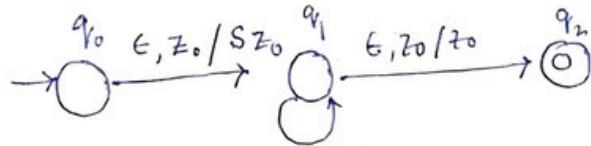
eg. $c a b c$
 $A, cabc, z_0$
 $A, abc, z_2 z_0$
 A, bc, z_0
 $A, c, z_1 z_0$
 A, ϵ, z_0

✓ CFG \rightarrow PDA. a) CFG \rightarrow GNF CFG, b) S(start) put to the stack.

$S(q_0, \epsilon, z_0) \rightarrow (q_1, Sz_0)$ c) for prodⁿ NT $\rightarrow T \cdot NT^*$

single terminal, followed by string of non-terminal), $S(q_1, T, NT_1) \rightarrow (q_1, \text{string of NT})$, d) for prodⁿ like $NT_1 \rightarrow \{\text{single terminal}\}$, $S(q_1, T, NT_1) \rightarrow (q_1, \epsilon)$ e) for accepting a string, 2 transitional functions are added. ① $S(q_1, \epsilon, z_0) \rightarrow (q_1, \epsilon)$ by empty stack
② $S(q_1, \epsilon, z_0) \rightarrow (q_{\text{final}}, z_0)$ by final state.

eg $S \rightarrow aA$, $A \rightarrow aABC/bB/a$, $C \rightarrow c$, $B \rightarrow b$



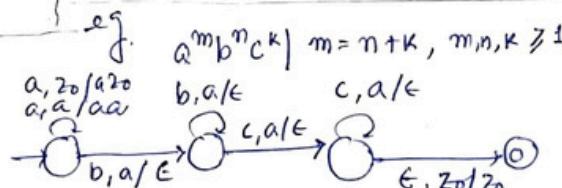
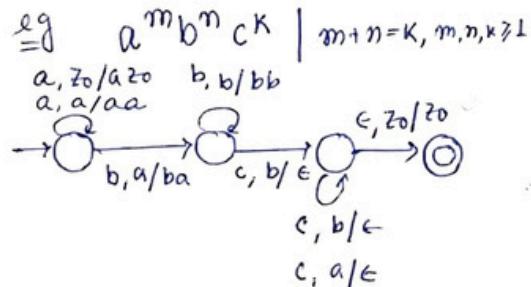
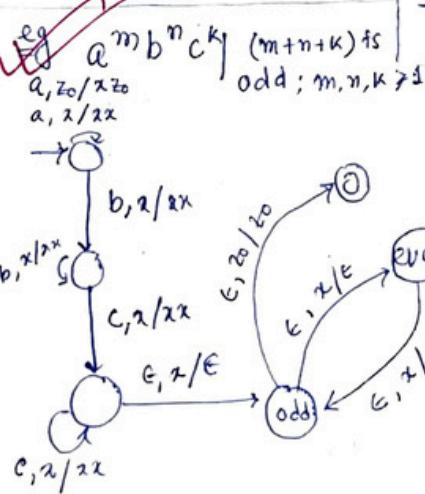
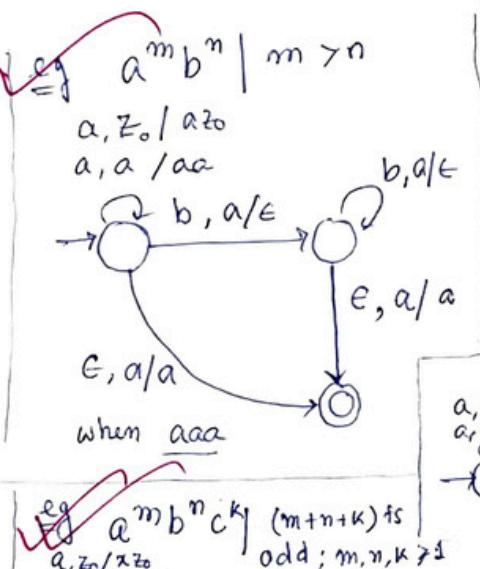
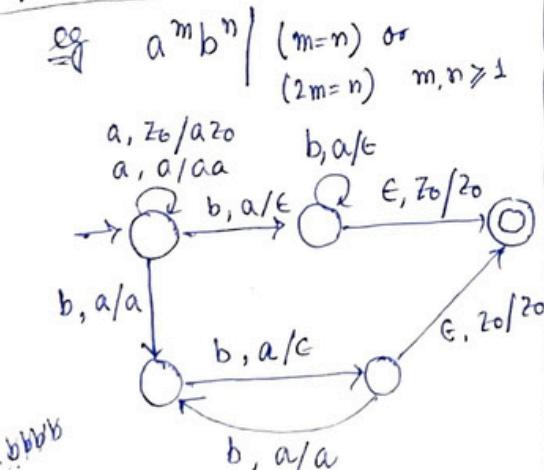
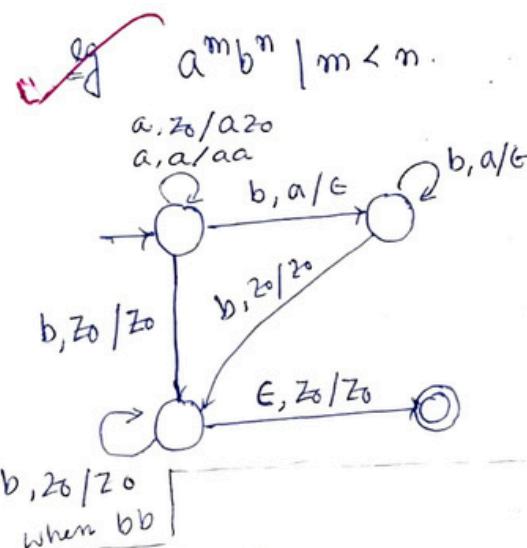
(a, S/A), (a, A/ABC), (b, A/B)
(b, B/ε), (a, A/ε); (c, C/ε), (ε, Z0/ε)

■ PDA \rightarrow CFG SK books (103)

ID for aabbcc

$(q_0, aabbcc, z_0) \rightarrow (q_1, aabbcc, Sz_0)$
 $\rightarrow (q_1, abbbc, Az_0) \rightarrow (q_1, bbbcc, ABCz_0) \rightarrow (q_1, bbcc, BBCz_0) \rightarrow (q_1, bc, BCz_0) \rightarrow (q_1, c, CZ_0) \rightarrow (q_1, \epsilon, Z_0) \rightarrow (q_2, Z_0)^*$

* CFL \leftrightarrow PDA



After pushing all a, b, c, check how many x's are there - # x's even or odd.

* More PDA convⁿ on ME-TH (108), SK book Chapter end (PDA)

TM and Undecidability

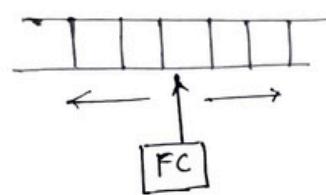
* $TM = (\Sigma, \Gamma, \delta, q_0, B, F)$. [Γ : Allowable tape symbol, B : Blank]

for DTM, $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, H\}$ Left, Right, Halts

for NTM, $\delta: Q \times \Gamma \rightarrow Q^{\Gamma \times \Gamma \times \{L, R, H\}}$

* $(A, \alpha, \beta, \delta) \rightarrow (B, \beta, \delta)$
 $\delta(A, \alpha) = (B, \beta, \delta)$
 $\alpha \in \Sigma^*, \beta \in \Gamma^*$, accepted by Halting TM = Recursive lang.

$\delta \in \{L, R\}^*$ * Lang. accepted by TM



* TM as acceptor, transducer

* TM can't accept ϵ .

* There are TMs that never halts.

* Universal TM:
 (Encoding TM as string in $(0,1)^*$)

(Unrestricted grammar generates REL)
 $(VUT)^+ \rightarrow (VUT)^*$



* Theorem If L & \bar{L} are both rec. enumerable, then both are recursive. If L is recursive, then \bar{L} also recursive & consequently both are RE.

$$(L, \bar{L} \text{ RE}) \rightarrow (L, \bar{L} \text{ REC})$$

$$(L \text{ REC}) \rightarrow (\bar{L} \text{ REC}; (L, \bar{L}) \text{ RE})$$

* Countable A set is countable if \exists an enumeration method using which all the elements of the set can be generated & for any particular elem, it takes only finite # steps to generate it. This # steps can be used as its index & hence a mapping into natural numbers.

e.g. $\frac{p}{q}, p, q \in \mathbb{Z}^+$

$\Sigma, \Sigma^*, 2^{\mathbb{Z}}$ countable
 2^{Σ^*} not countable

... So, uncountable

Similar way, \mathbb{Z}^* countable.

* every subset of countable set is finite or countable.

* Every lang. L is countable as $L \subseteq \Sigma^*$.

* Set of all TMs, $S \subseteq \Sigma^*$. So, TM set is countable. \Rightarrow REL, REC L, CSL, CFL, RL are countable. \Rightarrow PDAs, LBAs, FAs are countable.

$$L \in RL, CFL, CSL, REL, REC$$

> Diagonalisation method (to prove 2^{Σ^*} is uncountable). Power set of any countably infinite set is uncountable.

> There are some langs that are not RE (not accepted by any TM)

> If S_1, S_2 are countable, $S_1 \cup S_2, S_1 \times S_2$ are countable.

> Cartesian prod of finite no. of countable sets is countable.

> Set of all lang. that are not RE is uncountable.

* Undecidability

Algo \rightarrow HTM | TM - countable \therefore Some problems that have no algo

Comp^y: If there's algo for a $\neg P^n$

Dec^y: For a prob, yes/no answer by TM

> Halting prob. is undecidable.

- State entry prob. is undecidable.

- Halting after starting w/ blank tape is undecidable.

- Almost any problem rel^d to RE langs. is undecidable.

- PCP is undecidable.

• Lne is REL, not REC.

Lc is non-REL., undecidable.

Reduction $P_1 \rightarrow P_2$ if exists,

i) If P_1 is undec., P_2 too undec.

ii) If P_1 is non-REL, P_2 is also non-REL

iii) If P_2 dec, P_1 too dec.

• Property of a REL: Set of REL that satisfy a property P.

e.g. P_{CFL} : Set of REL that are CFL.

Trivial property $P = \emptyset$, or $P = \text{Set of all RELs}$.

• Rice theorem Every non-trivial prop of REL is undecidable.

• If a lang. is RE, but not REC, then its complement is non-RE. If $L \& \bar{L}$ are RE, then both are RFC.

• Any particular instance of a prob is always decidable.

• For REC^y lang, TM may halt or not halt, but for each REC lang (TM acceptable)

• For REC^x (rec. ~~lang~~), TM must halt. (TM decidable)

> no membership algo for RE.

No have mem algo for REC.

REC — HTM

RE — TM

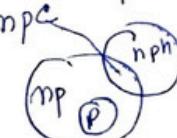
• P accepted by DTM in polynomial time.

NP acc. by NTM in polynomial time.

- 3SAT is np-complete

- Any np-easy is decidable prob.

Any np-comp prob is n.



- Some np-hard are decidable some are not

L_d (Diagonalisation lang) - non REL

L_u (universal lang) - REL
↑ undecidable

- P: \exists DTM or HTM, for input of size n it halts in (polynomial of n) time.
- NP: \exists NTM that always halt.

NPH: Each problem in NP can get reduced to it in poly time.

NPC: NP and NPH.

- UTM: $M \# x$

- 1) M accepts x : UTM accepts $M \# x$
 - 2) M rejects x : UTM rejects $M \# x$
 - 3) M loops on x : UTM loops on $M \# x$
(SemiDec, undec.)
- $\triangleright \{M \# x \mid M \text{ accepts } x\} \rightarrow \text{REL}$ as we can design TM for this
- $\triangleright \{M \# x \mid M \text{ halts on } x\} \rightarrow \text{SemiDecidable, Undecidable.}$
- \hookrightarrow Halting problem: RE but not REC.

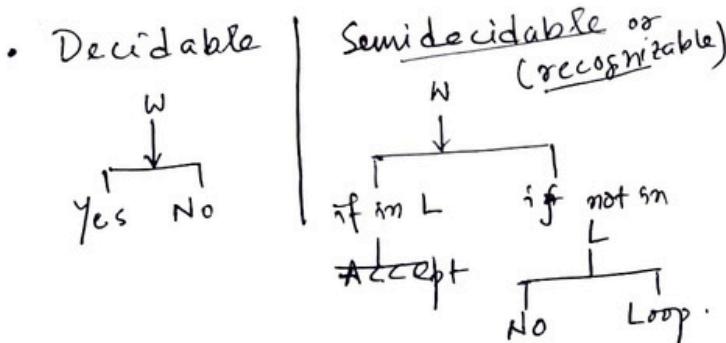
- Propositional logic is decidable. First order logic is undecidable.

- Decidable \rightarrow If recursive, otherwise undecidable.

Semi-decidable \rightarrow If RE. [eg. Halting problem, Blank tape halting prob, State entry prob, PCP, MPCP]
(also undecidable)



- Decidable lang. closed under union, intersection, complement.



- To show a lang. L is not in REC, it suffices to show that some L' is not in REC and $L' \rightarrow L$ (L' reduces to L).

$$\text{Showing } L' \rightarrow L = \overline{L'} \rightarrow \overline{L}$$

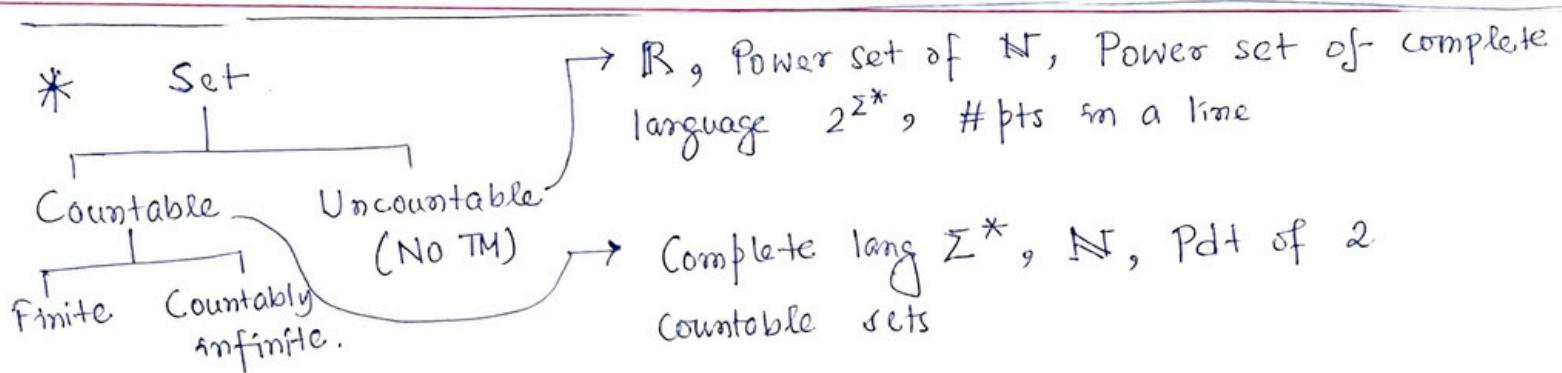
(Problem \approx language)

- Recognizable lang. closed under union, intersectn.
 - A lang. L is decidable iff L, \overline{L} both are recognizable.
 - A lang. L is recognizable iff it is enumerable.
- semi-decidable \leftrightarrow recognizable
 \leftrightarrow enumerable.
- There exists an unrecognizable lang.
 - There's L that's semiDec. but not dec.
 - If L is undecidable, it can be REL or non-REL (can't be REC).

\Rightarrow Rice's Th. Let C be a proper, non-empty subset of REL . Then the language $L_C = \{\langle M \rangle \mid L(M) \in C\}$ is (not recursive) (or undecidable).

To show L is not recursive: pick appropriate C (i.e. a C s.t. $L_C = L$), then prove 1) $C \subseteq \text{REL}$, 2) $C \neq \text{REL}$ & 3) $C \neq \emptyset$. So, L is not recursive hence, undecidable.

\triangleright If L is RE, then whether \bar{L} will be RE is undecidable. (i.e. RL/CFL/CSL/RECL/RE)



- \triangleright All uncountable sets are non REL (as no TM exists)
- \triangleright Uncountable sets are undecidable.
- \triangleright # uncountable sets are uncountable.
- \triangleright # langs for which no TM exists - uncountable.
- \triangleright # langs which are undecidable - uncountable.
- \triangleright # langs for which we can construct FA or PDA or TM is countable.
- \triangleright # decidable problems ~~is~~ is countable.

- * • RECL (Decidable)
 - a) Checking whether TM enters into state Q or not within 100 steps, b) checking TM having 100 steps or not, c) checking whether string is member of TM or not within 2020 steps.

- \triangleright Difference of REC & RE - RECL has membership algo, REL does not. for RECL membership is decidable, for REL not decidable (undecidable).

REL but not RECL (undecidable).

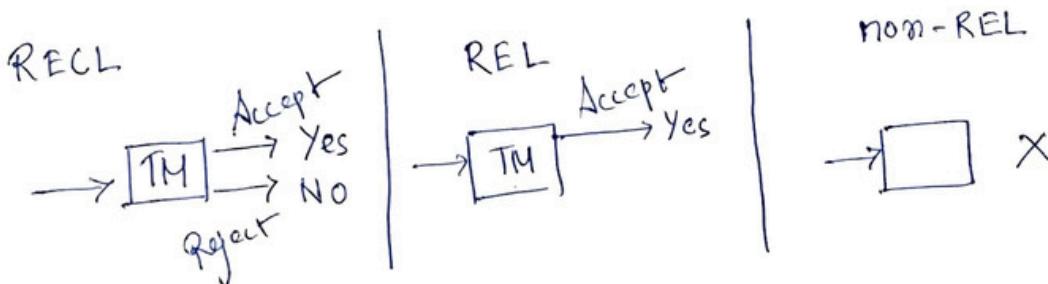
- a) Ambiguity of a language b) Membership of REL (TM) :
Checking whether a string is accepted by a TM or not.
c) Checking whether TM enters into state q or not.
d) Checking whether L generated by a TM is regular or
not (regularity), e) Checking TM accepts ϵ or not [although
TM accepts ϵ , but $L = \{\langle M \rangle \mid \epsilon \in L(M)\}$ is undecidable]

Non - REL. (Undecidable)

- a) Uncountable sets b) set of all unambiguous grammars,
c) Complement of (REL but not RECL) d) Checking
whether 2 CFGs generate same lang. or not (Eqv. prob.),
e) Checking whether given CFG generates complete
lang. or not f) Eqv. prob. of TM g) Checking
whether TM accepts Σ^* (compl. language) or not,
h) Emptiness prob. of TM.

* Decidable problem: \exists a HTM or algorithm for
that problem. All RECL.

* Undecidable problem $\exists!$ a HTM or algo for the prob.
All non-recursive lang. (REL or
non-REL)



* Reduction

$A \rightarrow B$ (A is reducible to B)

> B is harder than A .

> If B decidable, A decidable.

> If A undec, B undec.

> If B is RECL, A is RECL too.

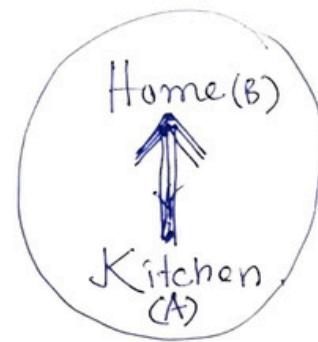
> If B is REL, A is REL too.

> If A is not RECL, B is not RECL.

> If A is not REL, B is not REL.

> If B is Yes, A is Yes.

If A is No, B is No.



* Decidable/Undecidable Table (@ E1)

* Undecidable probs abt TM.

i) HP (~~undec.~~ for REL, decid. for RECL),

ii) State entry prob (reading string w , whether TM enters a particular state Q or not. Undec. for REL, dec. for RECL), iii) Checking whether TM accepts L or not (by Rice's), iv) Checking TM accepts RL or not, v) TM accs CFLs or not., vi) TM accepts RECL or not.

* Decidable probs abt TM

i) Checking whether TM having 2021 states or not, ii) Checking whether TM takes 100 steps

- or not on some i/p., iii) Checking whether TM accepts string x or not within 2018 steps,
 iv) Checking TM takes at least 100 steps or not at some i/p, v) TM takes at least 100 steps or not on all i/p's.

* Undecidable probs. abt programming langs.

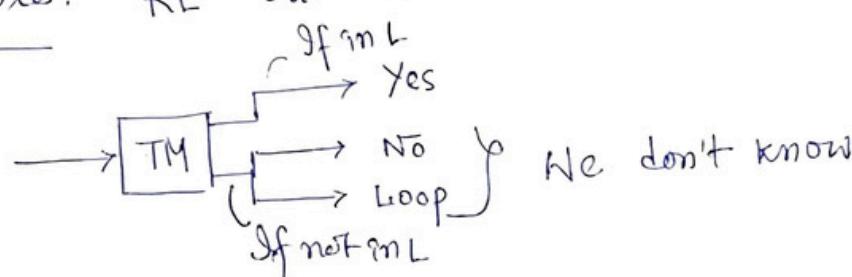
1. Checking w. given 2 programs produce same o/p or not for same i/p, 2. Checking w. given prog. ever produces an o/p, 3. C.w. given prog. can loop forever or not on some i/p,

* Post Correspondence Problem. (PCP) $\text{MPCP} \rightarrow \text{PCP}$.

It is RE but not REC. It is undecidable.

Modified PCP is reducible to PCP. MPCP is undecidable prob. [Both PCP & MPCP are decidable if the list contains one elem only.]

* Semi-Decidable. RE but not REC.



- Points
- * A lang is decidable (recursive) iff both of it & its complement are recognisable (REL but not RECL).
 - * Enumerable \Leftrightarrow Recognisable \Leftrightarrow (RECL but not REC)

* Halting set = $\{(M, x) \mid \text{TM } M \text{ halts on input } x\}$

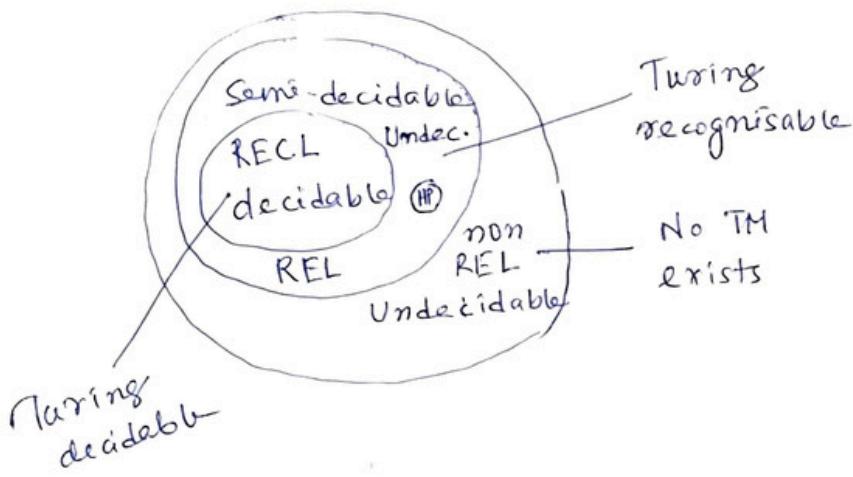
— this set is REL. But, complement of it is not RE.
↳ but not RECL (i.e. set is semidecidable).

* Complement of decidable lang is decidable (If L is recursive, \bar{L} too).

* All REL are not undecidable (easy fact $RL \subset DCFL \subset CFL$

$\subset CSL \subset RECL \subset REL$

* All REL are semi-decidable.



* HP - Semidecidable, undecidable.

* Computer - LBA (finite memory)

* $L = \{\langle M, w \rangle \mid M \text{ enters into state } Q \text{ on input } w\}$

* Dovetailing

(A, B, C, D, E)

| | | | | |
|-----|----|---|---|--|
| A | | | | |
| | I | | | |
| A | B | | | |
| II | I | | | |
| A | B | C | | |
| III | II | I | | |
| A | B | C | D | |
| III | II | I | | |
| : | | | | |

problems regarding property of TM

a) TM having finite ~~set~~ states? [trivial]
→ Recursively, Decidable

b) Will a TM ever make a left move on an input —
decidable, recursive

c) Will TM ever reach a state Q on an input w. — undecidable,

semidecidable

* Rice's • Any non-trivial 2-way property of REL is undecidable. (Trivial - property that's always true). — No trivial property is undecidable e.g. of trivial prop': $L = \{\langle M \rangle \mid L(M) \text{ is REL}\}$ as for all REL we have TM M.

- A non-trivial prop^y is one in which we have at least one elem not satisfying the property i.e. for some M , $L(M)$ must not have the given prop^y.
- e.g. of non-trivial prop^y:
- so has M that doesn't satisfy property.*
- i) $L = \{ \langle M \rangle \mid L(M) \text{ is regular} \}$ — $L(T_{yes}) = a^n$
ii) $L = \{ \langle M \rangle \mid L(M) \text{ is recursive} \}$ — $L(T_{no}) = a^n b^n$
- iii) $L = \{ \langle M \rangle \mid L(M) = \emptyset = \{\} \}$ — there are M , for which $L(M) \neq \emptyset$
- iv) $L = \{ \langle M \rangle \mid L(M) \text{ contains } 0 \}$ — there are M , where $L(M) \neq 0$
- v) $L = \{ \langle M \rangle \mid M \text{ accepts at least 10 strings} \}$
→ corresponds to lang. accepted by the machine M .

- Any non-monotonic property of REL (lang of TM) (1-way) is (not & even semi-decidable). (or unrecognizable)

Non-monotonic — stricter than non-trivial.

↳ prop^y must be non-trivial & $L(T_{yes})$ must be a proper subset of $L(T_{no})$.

e.g. non-monotonic prop^y

1. $L = \{ \langle M \rangle \mid L(M) = \{\} \}$ Non-RE, Undec.

$$L(T_{yes}) = \{\perp\} \quad L(T_{no}) = \Sigma^*$$

2. $\{ \langle M \rangle \mid L(M) = \{\} \}$ Non-RE, Undec.

$$L(T_{yes}) = \emptyset = \{\} \quad L(T_{no}) = \Sigma^*$$

3. $\{ \langle M \rangle \mid L(M) \text{ has exactly 100 strings} \}$ Non-RE, Undec.

$$L(T_{yes}) = \{1, 11, 111, \dots, 100 \text{ 1's}\} \quad L(T_{no}) = \Sigma^*$$

4. $L = \{ \langle M \rangle \mid L(M) \text{ is finite} \}$ non RE, undec.

$L(T_{yes}) = \text{any finite set}$ $L(T_{no}) = \Sigma^*$

✓ 5. $L = \{ \langle M \rangle \mid L(M) \text{ is infinite} \}$ [Go starred]

can't apply Rice's as can't find $L(T_{yes})$, $L(T_{no})$

[$L(T_{yes})$ itself is infinite, if we take $L(T_{no})$ as Σ^*
it's also infinite], and it's not a non-trivial
prop'. But it is REL. It is non-REL. (Comp.
of this ↑ (4) is too R non-REL).

✓ 6. $L = \{ \langle M \rangle \mid L(M) \text{ is } \Sigma^* \}$ [Go starred]

L and \bar{L} both non-RE.

* $L = \{ \langle M \rangle \mid L(M) \text{ is recognisable by a TM having even \# states} \}$ [or odd # states].

→ REL [as it's trivial] — We can just add new states to the desc' of TM.

Applying Rice's

1. Find $L(T_{yes})$, $L(T_{no})$. If both exist i.e. we have such lang. not ~~satisfying~~ satisfying the prop' & another lang. satisfying the prop', then, undecidable (^{non-}~~recursive~~).

2. If $L(T_{yes}) \subset \bar{L}(T_{no})$ then, not even RE (or semi-decidable). It's non-RE.

Examples. (Applying Rice's)

In form $L = \{ \langle M \rangle \mid L(M) \text{ is } \dots \}$

1. $L(M)$ has at least 10 strings. $L(T_{yes}) = \Sigma^*$, $L(T_{no}) = \emptyset$
So, undecidable. [In fact, RE]
Turing recognizable.

$L(T_y) \not\subseteq L(T_{no})$ non-monotonic
not applicable

2. $L(M)$ has at most 10 strings. $L(T_{yes}) = \emptyset$, $L(T_{no}) = \Sigma^*$
So, undecidable i.e. RE or non-RE.

$\emptyset \subset \Sigma^*$
 $L(T_y) \not\subseteq L(T_{no})$ for all cases

3. $L(M)$ is recognised by a TM having even # states (=REL)
Any REL can be recognised by a TM having even # of states. So, it's a trivial prop^y on REL. \Rightarrow Decidable.

Even if for a REL we have a TM having odd # of states, we can make an eqv. TM having even # states by adding one extra state. $s(q, \epsilon) \rightarrow (q', \epsilon, \text{stay})$

4. $L(M)$ is a subset of Σ^* . (= REL)

Trivial prop^y. All REL are subset of Σ^* . So, decidable.

5. $L(M)$ is finite. $L(T_y) = \emptyset$ $L(T_{no}) = \Sigma^*$
non-turing recognizable or
 $L(T_y) \subset L(T_{no})$ So, non-RE & undecidable.

6. $L(M)$ is $\{0\}$ $L(T_y) = \{0\}$ $L(T_n) = \Sigma^*$
 $L(T_y) \subset L(T_{no})$ non-RE, undecidable.

7. $L(M)$ is regular. ($L(T_y) = \emptyset$) \subset (Any $\xrightarrow{L(T_{no})}$ non RL)
non-RE, undec.

8. $L(M)$ is not regular [$L(T_y) = \{a^n b^n \mid n > 0\} \subset L(T_n) = \Sigma^*$]
non-RE, undec.

9. $L(M)$ is infinite. $L(T_y) = \Sigma^*$, $L(T_n) = \{\epsilon\}$. Can't find $L(T_y) \subset L(T_n)$. So, only comment is undecidable. [But, it is non-RE, although we can't apply Rice's 2nd part here, as ^{not a} non-monotonic prop^y.]

- (26)
- > $L_1 = \{ \langle M \rangle \mid M \text{ visits state } Q \text{ on some ip within 10 steps}\}$
Semi-Decidable, RE
 - > $L_2 = \{ \langle M \rangle \mid |M| < 100, \text{ where } |M| \text{ is \# states in } M\}$ Decidable
 - > $L = \{ \langle M \rangle \mid M \text{ is a TM and } \exists \text{ an input on which } M \text{ halts in less than } |M| \text{ steps}\}$
Decidable, RECL.
 - > $\{ \langle M \rangle \mid M \text{ is a TM \& } |L(M)| \leq 3\}$.
 $L(T_{yes}) = \{0, 1\} \quad L(T_{no}) = \Sigma^* \quad | \quad L(T_{yes}) \subset L(T_{no})$
 So, non-RE, undecidable.
 - > $\{ \langle M \rangle \mid |L(M)| = 2\}$
 $L(T_y) = \{0, 1\} \quad L(T_n) = \Sigma^* \quad | \quad L(T_y) \subset L(T_n)$
 Non-RE.
 - > $\{ \langle M \rangle \mid |L(M)| \geq 3\}$
 $L(T_y) = \{0, 1, 2\} \quad L(T_n) = \emptyset$
 Undecidable
 - > $\{ \langle M \rangle \mid M \text{ accepts all even numbers}\}$
 $L(T_y) = \{\text{set of all even numbers}\} \quad | \quad \text{non-RE. or}$
 $L(T_n) = \Sigma^* \quad L(T_y) \subset L(T_n) \quad | \quad \text{not even semi-decidable}$
 - > $\{ \langle M \rangle \mid M \text{ does not accept all even numbers}\}$
 - > $\{ \langle M \rangle \mid M \text{ rejects all even numbers}\}$
 - > $\{ \langle M \rangle \mid L(M) \text{ is countable}\}$ Trivial property ($= \text{REL}$)
— Decidable / Recursive
 - > $\{ \langle M \rangle \mid L(M) \text{ is uncountable}\}$ Trivial prop'g ($= \emptyset$)
 (over finite alphabet & finite length strings)
— Decidable / Recursive.

> $\{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are 2 TMs and } \in \in L(M_1) \cup L(M_2) \}$

$$L(T_{\text{rec}}) = L(M_1) \cup L(M_2) = \Sigma^* \quad \begin{cases} \text{RE, undecidable} \\ (\text{not Rec.}) \end{cases}$$

$$L(T_{\text{no}}) = \{0, 1\}$$

> $\{ \langle M_1, M_2 \rangle \mid \in \in L(M_1) \cap L(M_2) \}$

$$L(T_{\text{yes}}) = \Sigma^* \quad L(T_{\text{no}}) = \emptyset. \quad \begin{cases} \text{RE, undecidable,} \\ (\text{not Recursive}) \end{cases}$$

> $\{ \langle M_1, M_2 \rangle \mid \in \in L(M_1) \setminus L(M_2) \}$
non RE.

> $\{ \langle M \rangle \mid M_0 \text{ is a TM that halts on all inputs}$
and $M_0 \in L(M) \}$
non REC, RE.

> $\{ \langle M \rangle \mid M_0 \text{ is a TM that halts on all inputs}$
 $\& M \in L(M_0) \}$ REC.

> $L = \{ \langle M, x \rangle \mid x \text{ is a string } \& \exists \text{ a TM, } M' \text{ such that}$
 $x \notin L(M) \cap L(M') \}$ REC.

> $\{ \langle M \rangle \mid M \text{ is a TM } \& \exists \text{ an input on which } M \text{ halts}$
within 1000 steps } REC.

> $\{ \langle M \rangle \mid \exists \text{ an input whose length is less than 100,}$
on which M halts } RE not REC

> $\{ \langle M \rangle \mid M \text{ is the only TM that accepts } L(M) \}$
REC. (Every lang. has an
infinite # TMs that accept it)

> $\{ \langle n, x, M_1, M_2, \dots, M_k \rangle \mid n \text{ is a natural no., } x \text{ string, } M_i \text{ is}$
a TM for all $1 \leq i \leq k$, & at least $\frac{k}{2}$ TM's of M_1, \dots, M_k
halt on $x \}$ RE not Recursive.

- > $\{\langle M \rangle \mid |M| < 1000\}$ Recursive
- > $\{\langle M \rangle \mid \exists x, |x| \equiv_5 1 \text{ & } x \in L(M)\}$ RE not REC.
- > $\{\langle M \rangle \mid M \text{ halts on all palindromes}\}$ Non RE.
- > $\{\langle M \rangle \mid L(M) \cap \{a^{2^n} \mid n \geq 0\} \text{ is empty}\}$ Non RE
- > $\{\langle M, k \rangle \mid |\{w \in L(M) : w \in a^*b^*\}| \geq k\}$ RE, not REC
- > $\{\langle M \rangle \mid L(M) = L' \text{ for some undecidable } L' \}$ Rec
and M halts on all inputs $= \emptyset$
- > $\{\langle M \rangle \mid M \text{ accepts at least 2 strings of different lengths}\}$ RE, not Rec.
- > $\{\langle M \rangle \mid L(M) \text{ & } \overline{L(M)} \text{ both infinite}\}$ non RE.
- > $\{\langle M \rangle \mid |L(M)| \text{ is prime}\}$ Non RE.
- > $\{\langle M, a, k \rangle \mid M \text{ does not halt on } a \text{ within } k \text{ steps}\}$ Rec.
- > $\{\langle M \rangle \mid \exists x \in \Sigma^* \text{ s.t. for every } y \in L(M), xy \notin L(M)\}$
- non RE.
- > $\{\langle M \rangle \mid \exists x, y \in \Sigma^* \text{ s.t. either } x \in L(M) \text{ or } y \notin L(M)\}$
Rec.
- > $\{\langle M \rangle \mid \exists \text{ a TM } M' \text{ s.t. } \langle M \rangle \neq \langle M' \rangle \text{ and } L(M) = L(M')\}$
(larg. of all TMs)
- Rec.
- > $\{\langle M_1, M_2 \rangle \mid L(M_1) \leq_m L(M_2)\}$. non-RE.
↑ mapping reducibility.
- > $\{\langle M \rangle \mid M \text{ doesn't accept any string } w \text{ s.t. } 001 \text{ is a prefix of } w\}$. non RE.

- > $\{\langle M, \alpha \rangle \mid M \text{ does not accept any string } w \text{ s.t. } \alpha \text{ is a prefix of } w\}$ non-RE
- > $\{\langle M, \alpha \rangle \mid \alpha \text{ is prefix of } \langle M \rangle\}$ Recursive
- > $\{\langle M_1, M_2, M_3 \rangle \mid L(M_1) = L(M_2) \cup L(M_3)\}$ non-RE.
- > $\{\langle M_1, M_2, M_3 \rangle \mid L(M_1) \subseteq L(M_2) \cup L(M_3)\}$ non-RE
- > $\{\langle M_1 \rangle \mid \exists \text{ 2 TMs } M_2, M_3 \text{ s.t. } L(M_1) \subseteq L(M_2) \cup L(M_3)\}$
 - Rec.
- > $\{\langle M, w \rangle \mid M \text{ is a TM that accepts } w \text{ using at most } 2^{|w|} \text{ squares of its tape}\}$
 - Rec.
- > L_D (Diagonalization lang.) = $\{\langle M \rangle \mid M \text{ is a TM \& }$
 - [Set of descriptions of TMs that don't accept themselves]
 - $\langle M \rangle \notin L(M)$
- > $A_{TM} = L(U_{TM}) = \{\langle M, w \rangle \mid M \text{ is a TM \& } M \text{ accepts } w\}$ - RE
- > $\overline{A_{TM}} = \overline{L(U_{TM})} = \{\langle M, w \rangle \mid M \text{ does not accept } w\}$ non-RE.

Note : If both L & L' are RE, then L will be REC.

> Complement of a language that is RE but not REC, will be a non RE language.

- A TM is said to halt whenever it reaches a config' for which δ isn't defined. [for final state no δ defined, so TM halts @ final].
 - A TM for $\{\epsilon\}$ -
 - if tape[1] is blank then accept
 - else reject
 - A TM for $\phi = \{\}$ -

| | | | |
|--------|--|----|---------------------------------|
| reject | | or | if tape[1] is blank then reject |
| | | | else reject |
 - TM accepts null string, ϵ .
 - A TM must have at least 2 states, (accept & reject)
 - (?) as the TM can't simultaneously accept and reject.
(may have only one based on rep'.) :-
 - Complement of mon-regular is always non-regular.
 - DCFL \cup CFL = CFL
-
-

1. $\overline{\text{RECL}} \rightarrow \text{RECL}$, $\overline{\text{REL}} \rightarrow$ either $\overline{\text{RECL}}$ or
 $\overline{\text{non-REL}}$,

REL but not $\text{RECL} \rightarrow \text{non-REL}$.

2. Decidable : Recursive, Turing decidable,
Valid, invalid strings Effectively enumerable, Acceptable by HTM
both have logic

Semidecidable : RE, Turing recognizable, Turing
Only valid strings have acceptable, Turing computable, Enu-
logic. merable, Partially decidable.



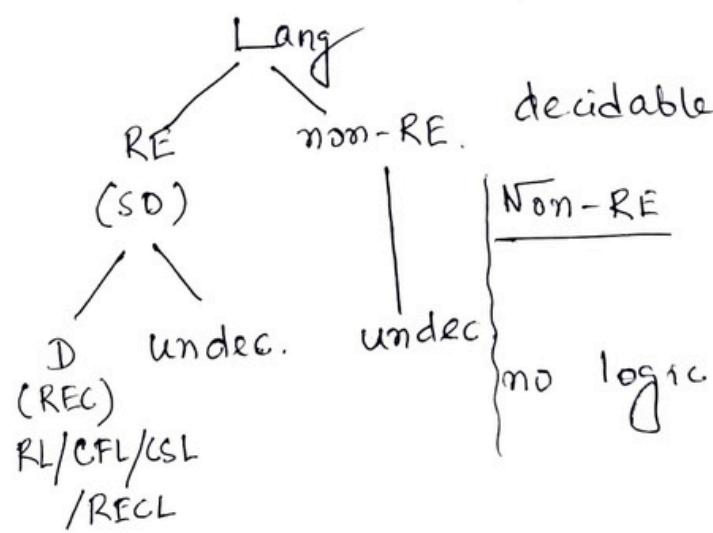
Undecidable

: Non-rec., semidecidable but not decidable, no HTM, no algo.

RE but not REC

Semidecidable but not recursive, SD but not

decidable



Undec., Not semidecidable, has no TM, valid strings have

$$\rightarrow \text{CFG} \quad L = \{ \text{CFG} \mid \text{CFG is ambiguous} \}$$

has logic for yes, no logic for no.

— Semidecidable; RE, but not REC.

$$\rightarrow L = \{ \text{CFG} \mid \text{CFG is unambiguous} \}$$

no logic for yes (CFG is unambi)

→ non-RE., UD, not Semidec.

→ if ambiguous, we can find a string that has ≥ 2 parse trees in finite time.

if a CFG is unambiguous, we can never find a string for which we will have ≥ 2 parse trees. We can keep on searching infinitely. Hence, no logic for yes case even.

$$\rightarrow L = \{ \text{CFG} \mid \text{CFG is ambiguous} \}$$

$$\bar{L} = (\Sigma^* - L)$$

here all are unambiguous → non RE.

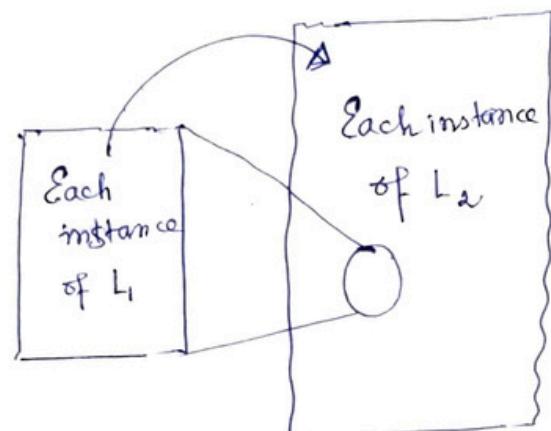
Reducibility

| | | |
|--|---|---|
| P_1 reduces to P_2 | $P_1 \leq_m P_2$ many to one reducibility | HP for FA, PDA, DPDA, HTM is decid. |
| $P_1 \leq_p P_2$ or $P_1 \alpha P_2$ | $P_1 \leq_p P_2$ polynomially reducible | |

P_2 is at least as hard as P_1 .

$P_1 \leq P_2$ { Each instant of P_1 will be mapped to some instance of P_2 .

- $L_1 \leq L_2$
 - L_2 decidable $\Rightarrow L_1$ decidable.
 - L_1 undecidable $\Rightarrow L_2$ undec.
 - L_2 finite lang $\Rightarrow L_1$ finite lang
 - L_1 infinite $\Rightarrow L_2$ infinite
 - L_2 is RE $\Rightarrow L_1$ is RE
 - L_1 is not RE $\Rightarrow L_2$ is not RE.
 - L_2 is REC $\Rightarrow L_1$ is REC
 - L_1 not REC $\Rightarrow L_2$ not REC.
 - L_2 is Yes. $\Rightarrow P_1$ is Yes.
(P_2 is no $\Rightarrow \times$)
 - P_1 is no $\Rightarrow P_2$ is no.



Project (L_2)
↑
module (L_1)

$$\bullet L_1 \leq L_2, L_2 \leq L_3$$

> If L_3 is decidable, L_2 & L_1 decidable.

> If L_3 is undec., L_2, L_1 undec.

> If L_2 is decidable, L_1 decidable (don't know of L_3).

> If L_2 is und., L_3 undec. (don't know of L_1).

- $P_1 \leq P_2 \wedge P_2 \leq P_3$, P_1 decidable, P_3 undec.
 $\Rightarrow P_2$ (can't say anything)

> Countability

If X & Y are countable,

- $X \cup Y$, $X \cap Y$, subset of X or Y , $X \times Y$ are countable.

- $P(X)$ powerset is countable if X is finite.

is uncountable if X is infinite.

Lang. over Σ

Finite Lang.

Regular Lang.

Decidable

Infinite Lang.

Decidable

RE, not REC
(SD, but UD)

Not RE
(Not SD)

Uncountable

Countable

All REL are countable.

Among non-REL some are countable, some are not.

e.g. $L = \{ \langle M \rangle \mid M \text{ is a TM and } M \text{ does not accept a given string } w \}$

→ non-RE and countable

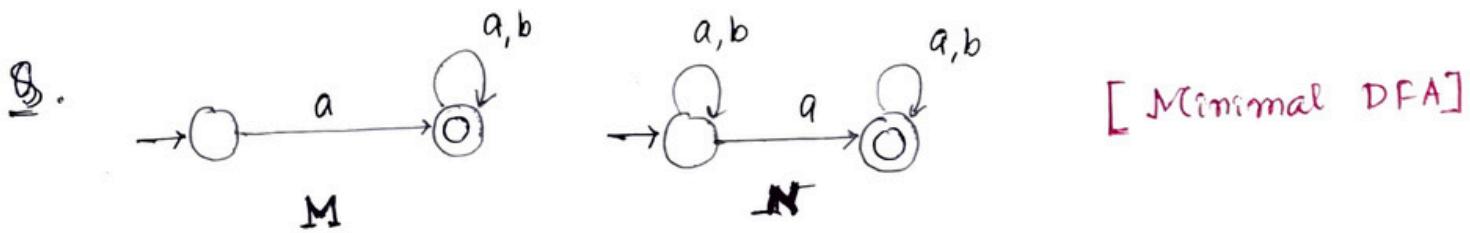
e.g. $L = \{ \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2), M_1, M_2 \text{ are TMs} \}$

→ non-RE & uncountable.

• Σ be a finite alphabet, then

> Σ , Σ^* , 2^Σ are countable.

> 2^{Σ^*} is uncountable.

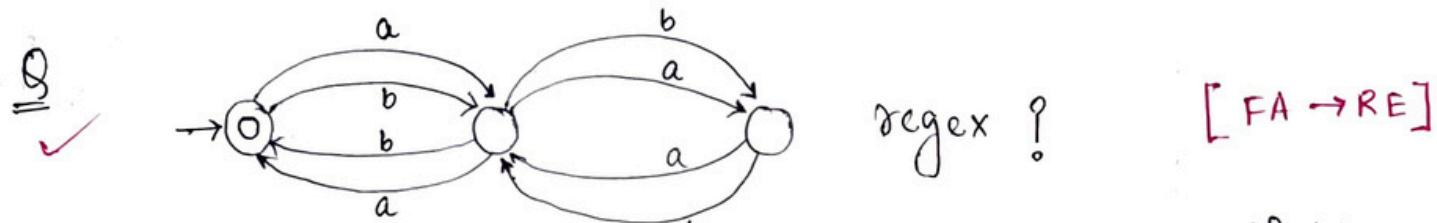
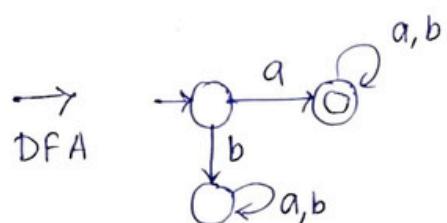


Minimally # states in a DFA or # states in a minimal DFA that accepts $L(M) \cap L(N)$.

$$\Rightarrow L(M) = a(a+b)^* \quad L(N) = (a+b)^*a(a+b)^*$$

$$L(M) \cap L(N) = L(M) = a(a+b)^*$$

Ans = 3.



Eliminating options by taking some string that's accepted by the FA.

$$\Rightarrow (aa + bb + ab + ba)^*$$

| | | |
|----------|---------------------------------------|--|
| <u>Q</u> | $L_1 = \{a^n b^n c^m \mid n, m > 0\}$ | $L_1 \cap L_2 = a^n b^n c^n \mid n > 0$ CSL, \times CFL |
| | $L_2 = \{a^m b^n c^n \mid m, n > 0\}$ | $L_1 \cup L_2 =$ CFL, \times DCFL $L_1 \cdot L_2 =$ CFL |

[Closure properties of CFL]

Q ✓ $L_1 = a^{3n} \# a^{2n} \# a^n \mid n > 0$

$$L_2 = a^n b^n c^m \mid \underline{n \neq m} \wedge n, m > 0 \quad [\text{Identification of Langs}]$$

→ More than 1 comparisons for both.

\times CFL, both are CSL.

Q Decidable?

[Decidability] [Closure prop^y]

- a) Whether inverseⁿ of a CFL & a finite regular lang is infinite. ✓
- b) Whether CFL is regular. X
- c) Whether 2 DPDA's accept same lang. ✓
- d) Whether given grammar is Context free. ✓

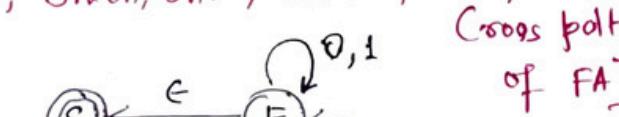
Q. $(a+b)^* b (a+b)^* b (a+b)^*$ [Regex descⁿ]

→ Containing at least 2 b's.



- Reversal of the FA :

When F is only final



both accepts same lang.

$(0+1)^*$

- If F is the only final state then the

corresponding minimal dfa eqv. to given NFA
will have exactly 1 state.

If S is the only final state corr. minimal
dfa has exactly 2 states.

Q Decidable?

- a) $\{\langle M \rangle \mid M \text{ is a TM} \& M \text{ has } 21 \text{ states}\}$ ✓
- b) $\{\langle M \rangle \mid M \text{ does not have a halt state (final)}\}$ ✓
- c) $\{\langle M, w \rangle \mid M \text{ accepts } w \text{ string}\}$ X
- d) $\{\langle M \rangle \mid L(M) \text{ is finite}\}$ X

Q L is CFL, not DCFL. Conclusion on \bar{L} -

> \bar{L} is not closed on CFL, i.e., \bar{L} may / may not be CFL.

X \bar{L} must be CSL.

X \bar{L} cannot be DCFL.
 \hookrightarrow If DCFL, then

$L = \bar{L}$ is also DCFL as closed under complement for DCFL.

$\Rightarrow \bar{L}$ is not DCFL.

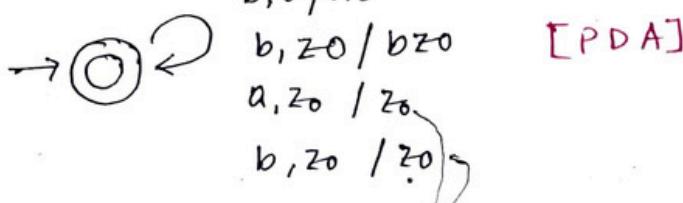
Every CFL \rightarrow CSL \Rightarrow Complement closed for CSL
 So, \bar{L} must be CSL.

$WW^R \rightarrow$ all non-complement \downarrow palindromes

We can make NPDAs, so, CFL.

Q

PDA.

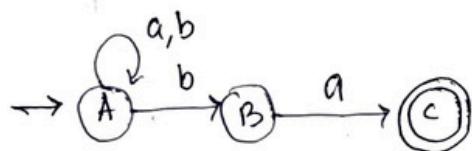


- It accepts $(a+b)^*$

- It's non-deterministic (for b, as have 3 transitions or actions)

ME Q [Regex] ends with a, but not ends with aa.

\rightarrow Check with answers.



$(a+b)^*ba$

$(a+b)^*ba + a \cancel{aa}$

$$\left. \begin{array}{l} L_1 = a^n b^n \mid n \geq 0 \\ L_2 = a^n b^{2n} \mid n \geq 0 \end{array} \right\} L_1 \cdot L_2^R = a^n b^n \cdot b^m a^m \leftarrow \text{DCFL} \quad [\text{Closure prop}^b]$$

$\bar{L}_2 \leftarrow \text{DCFL}$ [closure]

$L_1^3 = a^n b^n a^m b^m a^p b^p \leftarrow \text{DCFL}$

L^R reverse of L.

$L_1 \cup L_2 \leftarrow \text{CFL, not DCFL}$

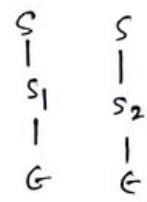
~~Q~~ Ambiguous but has an equiv. unamb. grammar:

G_1 :

$$S \rightarrow S_1 / S_2$$

$$S_1 \rightarrow aS_1b / \epsilon$$

$$S_2 \rightarrow aS_2bb / \epsilon$$



grammars
ambiguous

$$L(G_1) = \frac{a^n b^n}{\text{can't collapse}} \cup \frac{a^n b^{2n}}{\text{can't collapse}}$$

$$a^n b^n \cap a^n b^{2n} = \emptyset \neq \infty$$

So, we can have
an equiv. unamb. grammar.



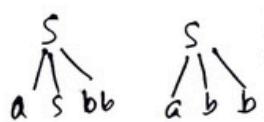
equiv
unamb.
grammars

$$\left\{ \begin{array}{l} S \rightarrow S_1 / S_2 / \epsilon \\ S_1 \rightarrow aS_1b / ab \\ S_2 \rightarrow aS_2bb / abb. \end{array} \right.$$

G_2 :

$$S \rightarrow aSbb / abb / G$$

(Ambiguous grammar)



$$L(G_2) = a^n b^{2n} \leftarrow \text{DCFL}$$

[can never have lang ambiguity]

Equiv. unamb. grammar:

G_3 :

$$S \rightarrow (S_1 C) / (AS_2)$$

$a^n b^n c^m$

$$S_1 \rightarrow aS_1b / \epsilon \quad - a^n b^n$$

$$C \rightarrow cC / \epsilon \quad - c^m$$

$$S_2 \rightarrow bS_2c / \epsilon \quad - b^m c^m$$

$$A \rightarrow aA / \epsilon \quad - a^k$$

= Can't have equiv. amb.
grammar.

~~✓~~ Checking ambiguity

Amb. if —

1. — U — ≥ 2 parts

2. parts don't collapse

3. — ∩ — $\neq \emptyset$ may be ∞

$$\text{eg } L = \left(\frac{a^n b^n c^m}{①} \cup \frac{a^n b^m c^m}{②} \right)$$

2 parts, can't
collapse as ① has
 $a = b$ & ② has
 $b = c$. \cap of them
as $a^n b^n c^m \Rightarrow \infty$
strings in the lang.

So, L is ambiguous.

CFL may or
may not have
lang. ambiguity.

$$L(G_3) = \frac{a^n b^n c^m}{\rightarrow} \cup \frac{a^k b^m c^m}{\rightarrow}$$

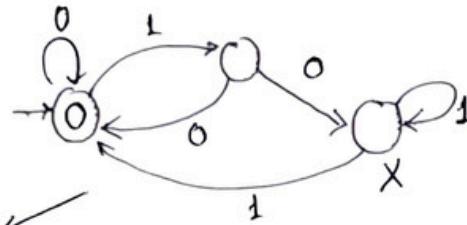
2 parts, can't collapse,
 \cap is $a^n b^n c^n \rightarrow \infty$ lang
so, lang is ambiguous

↓
all grammar for $L(G_3)$
is ambiguous.

Q [Regex] $\Sigma = \{0, 1\}$ Every 0 is followed by exactly 1 or 2 one's.

- a) $(01 + 011)^*$ It can't be generated X
- b) $(01 + 011)^* 1^*$ 0111 gets generated X
- c) $(01 + 011)^* + 1^*$ 101 doesn't get accepted X
- ✓ d) $1^* (01 + 011)^*$

Q [Regex] from DFA.



$$\begin{aligned}
 & \left((x^*y^*)^* \right)^* \\
 & \left((x+y) \right)^* \\
 & \rightarrow \text{DFA} \quad \xrightarrow{\quad} \quad \xrightarrow{\quad} \\
 & (0 + 01^*1) \\
 & = 0(\epsilon + 1^*1) \\
 & = 01^*
 \end{aligned}$$

also can be written as -

~~$0 + 1(0 + 01^*1)^*$~~

~~$(0 + 101^*)^*$~~

↑ Ans.

~~$(0^*(101^*)^*)^*$~~ *

Q CFL?

a) Complement of $\{x \in \{a,b\}^* \mid x \text{ is not WW for } w \in \{a,b\}^*\}$

X CFL \hookrightarrow becomes WW form when complemented
CSL ✓

b) $\{a^i b^{i+k} a^k \mid k \neq i\}$ Not CFL 2 condⁿ checking.

c) Compl. of $\{a^n b^n c^m \mid m = n\}$
 \hookrightarrow becomes $\{a^m b^n c^p \mid m \neq n \text{ or } n \neq p\}$

d) $\{a^n b^{2n} c^{3n} \mid n \geq 1\} - \{a, b, c\}^*$ → becomes \emptyset
RL, so CFL ✓

~~RE~~ but not REC

After deciding, undecidable

Every superset of

$L(T_{yes})$ if satisfies
the property then

it is monotonic
property \Rightarrow the lang is RE, not
mon-RE

If superset of $L(T_y)$

not satisfies the
property then \Rightarrow
non-monotonic.

So, non-RE, not RE.

~~Q T/F~~

a) If $L \subseteq \{0\}^*$ & L is a CFL, then
it is regular. (T) [as unary Σ]

b) $L \subseteq \{0,1\}^*$ & L is CFL, then
L may be regular. (T) $[0^*, 01^*]$

c) $L \subseteq \{0\}^*$ & it's a CSL, L is not regular. (F)

* ~~L $\subseteq \{0,1\}^*$ & L is the lang. of all palindromes,
then every infinite subset of L is non-regular. (F)~~

$\hookrightarrow 0^*, 1^*$ also in L. They are regular.

~~Q~~

CFL?

~~a) $\{a^m b^n \mid m = km, k=0,1,2,\dots\} = \{a^m b^m v a^{m^2} b^{m^2} v \dots\}$~~
~~= infinite union \times CFL~~

~~b) $\{a^m b^n \mid n = 3m+2 \text{ if } m \text{ odd}$
 $\quad \& n = 4m+5 \text{ if } m \text{ even}\}$ \leftarrow CFL ✓~~

~~c) $\{a^m b^n c^p \mid m < \min(m,p)\}$ to find min \leftarrow comparison +
another \times CFL~~

~~d) $\{a^m b^n c^p \mid \gcd(m,n) = p\}$. \times CFL.~~

↳ (CSL)