GATE CSE NOTES

by

Joyoshish Saha

TOC

FA, RL

* DFA = $(Q, \Sigma, \delta, q_0, F)$ $\delta : Q \times \Sigma \to Q$
* # states ($|w| = n : n+2$) ($|w| \le n : n+2$) ($|w| \ge n : n+1$) ($|w| \% n = 0 : n$) ($n_a = n : n+2$) ($n_a \ge n : n+1$) ($n_a \% n = 0 : n$) (n^{th} symbol from LHS is 'a' : $n+2$)
* Cross product method $\{A,B\} \times \{C,D\} = \{AC, AD, BC, BD\}$ ($n \times m$, nm)

$\Sigma = \{a, b\}$: n^{th} symbol from RHS is a => need 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)

% by 4 - # states = 4

	0	1
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3

(0, 1, 2, 3 for base 4) — base symbols; remainders

→ We can minimise this. #s = 3

If ternary number:

	0	1	2
→ q_0	q_0	q_1	q_2
q_1	q_3	q_0	q_1
q_2	q_2	q_3	q_0
q_3	q_1	q_2	q_3

* Complement of DFA:

eg: (Start with a, not start with a) (even length string, odd length string)

F becomes Q − F

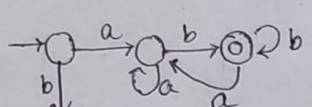
* Operations on DFA.

a_b or b_a

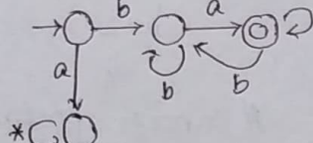
1) Union (Starts & ends with different symbols) 2) Concatenation (Starts with 'a', ends with 'b') 3) Cross product (Even # a's, even # b's) 4) Complement (not containing a) 5) Reversal (awb → bwa)

eg (1). $\Sigma = \{a, b\}$

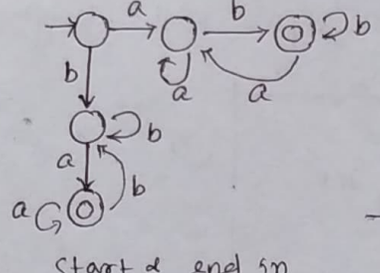
Start a, end b.



Start b, end a

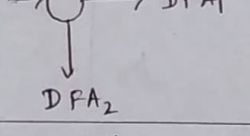


Union ⇒

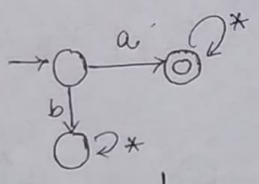


Start & end in different symbols

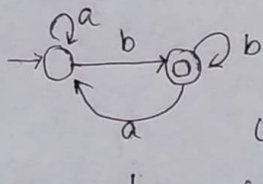
DFA1, DFA2 ⇓ union



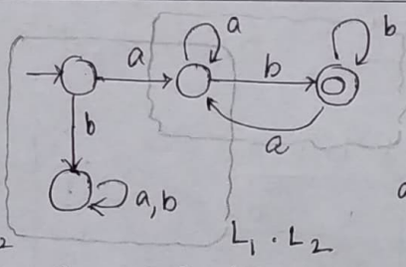
(2). Start with a



end in b



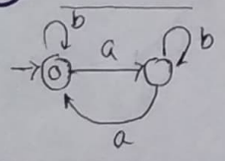
⇒ Concatenation (concatenate DFA_2 at the final state of DFA_1) (Final$_1$ & initial$_2$ merged)



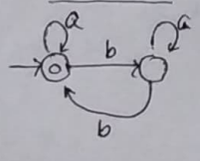
Start with a and end in b

L_1, L_2, $L_1 \cdot L_2$

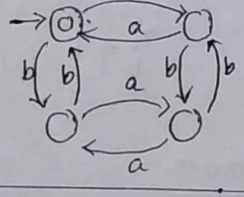
(3) Even # a's



Even # b's



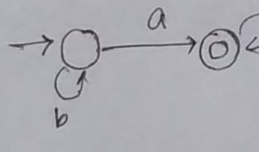
⇒ Cross product (of states)



[See pg 7, product DFA construction]

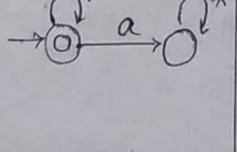
even # a's and even # b's

(4) Contains a

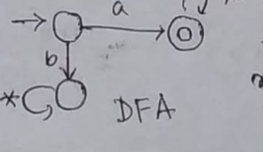


⇒ Complement

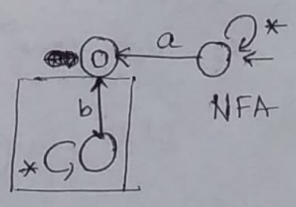
not containing a



(5) Start with a



DFA ⇒ reverse



NFA

(start, δ: Q×Σ → Q, final) → # Possibles DFAs with n states, m symbols in Q, = $2^n \times n^{mn} \times m$ (if we choose start also) = $2^n n^{mn}$ (start state can be any - normal case)

* (Q) # 2-state dfa's with designated initial state that accept empty language over {a,b}.

{} or ϕ

	a	b
→X	X/Y	X/Y
Y	X/Y	X/Y

$2^4 = 16$

	a	b
→X	X	X
(Y)	X/Y	X/Y

$2^2 = 4$

$\Rightarrow 20$.

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

	a	b
→X	2^4	
Y		

	a	b
→X	2^4	
(Y)		

	a	b
→(X)	2^4	
Y		

	a	b
→(X)	2^4	
(Y)		

$\Rightarrow 64$.

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

init. state has to be a final state to accept the null string.

	a	b
→(X)	2^4	
(Y)		

Everything gets accepted

	a	b
→(X)	X	X
Y	X/Y	X/Y

2^2

Everything has to be accepted in X

$\Rightarrow 20$.

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

	a	b
→X	3^6	
Y		
Z		

$3^6 \times$ (Choosing final state)

$= 3^6 \times \left(\sum_{r=0}^{3} {}^3C_r\right) = 3^6 \times 2^3 = 5832$.

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

	a_1	a_2	...	a_m
→A_1	n^{mn}			
A_2				
⋮				
A_n				

$(n^{mn} \times 2^n)$ Ans

Choosing final states

⇓

subsets possible for a set of n elements.

* NFA = $(Q, \Sigma, \delta, q_0, F)$ $\delta : Q \times \Sigma \to$ Powerset of Q. or 2^Q

$\Sigma \cup \{\epsilon\}$ for ϵ-NFA

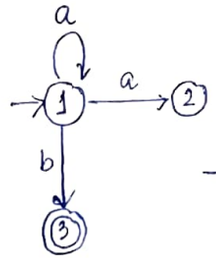
$Q = \{A, B\}$

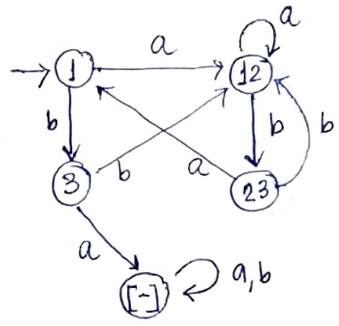
$2^Q = \{\phi, \{A\}, \{B\}, \{A,B\}\}$

↳ NFA to DFA

	a	b
→1	1,2	3
2	1	2
(3)	-	1,2

	a	b
→1	1,2	3
1,2	1,2	2,3
(3)	[-]	1,2
(2,3)	1	1,2
[-]	[-]	[-]





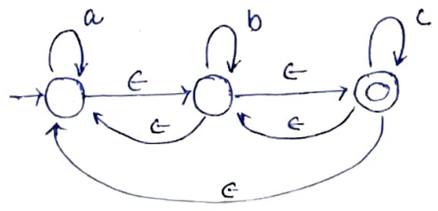
TOC 1 - copy

* NFA with ϵ move.

eg any comb^n of a,b,c

DFA:

NFA with ϵ-move :



Indian Institute of Engineering Science and Technology, Shibpur
Department of Information Technology

Report on. .....

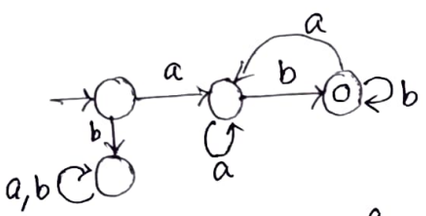
* Processes on DFA.

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

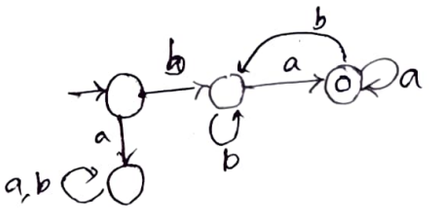
⇓

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

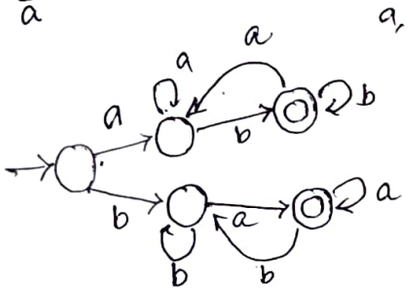
L_1:



L_2



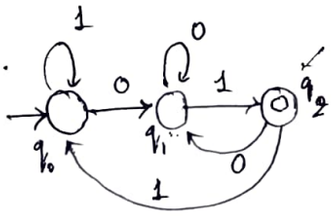
$L_1 \cup L_2$



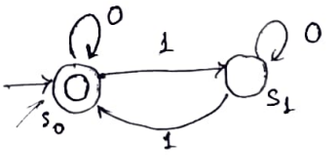
➢ **Intersection** eg. Set of strings over {0,1} s.t. it ends with 01 & has even # 1's.

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

L_1:

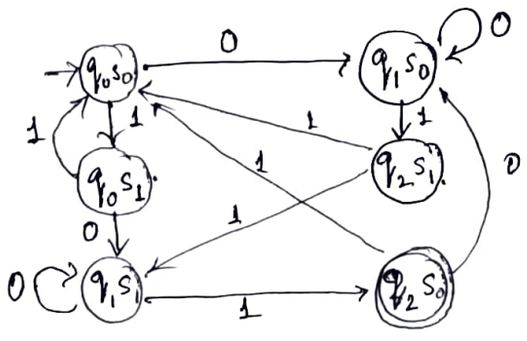


L_2



$q_1 s_0$

$L_1 \cap L_2$

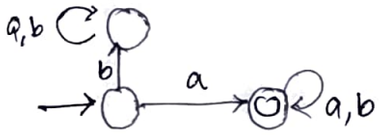


Date .....

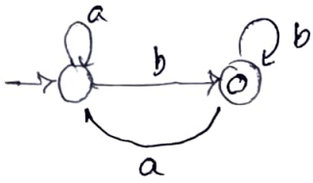
Signature .....

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

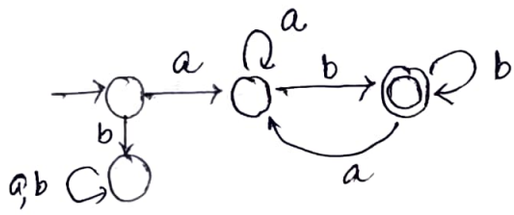
L_1: Start with a



L_2: Ends with b



$L_1 \cdot L_2$



> **Complement** Final ↔ Non-final states

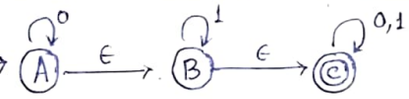
(Don't change transition direction)

> **Reverse**

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

— Not each reversal of DFA leads to DFA.

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

eg. 

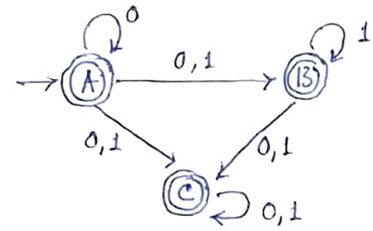
Final state: any state that can reach final state only by seeing ϵ

TOC 1

	ϵ^*	IP 0	ϵ^*	ϵ^*	IP 1	ϵ^*
A	ABC	AC	ABC	ABC	BC	BC
B	BC	C	C	BC	BC	BC
C	C	C	C	C	C	C

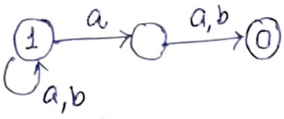
NFA transition table

	0	1
→ A	ABC	BC
B	C	BC
C	C	C



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

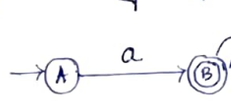
* 2nd symbol from rhs is 'a'

Build NFA → 

→ Convert to DFA.

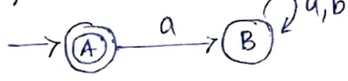
(Same way for - 3rd symbol from rhs is 'a')

* If NFA contains n states, DFA can have at most 2^n states.
* # states in DFA ≥ # 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 ≠ Language accepted by complement of NFA

↳ eg. NFA 

↳ Language = {a, aa, ab, aab, abb, ...}

Comp(L) = {ε, b, bb, ba, ...} — not equal.

Comp(NFA): 

Lang = {ε}

* Minimisation of DFA (Partitioning method)

eg

	0	1
→ A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C

0 equivalence : {A, B, C, D} {E}
1 equivalence : {A, B, C} {D} {E}
2 " : {A, C} {B} {D} {E}
3 " : {A, C} {B} {D} {E} — same, so stop

⇓

	0	1
→ AC	B	AC
B	B	D
D	B	E
(E)	B	AC

* For some language, DFA is not unique.
* Dead state, Inaccessible state.

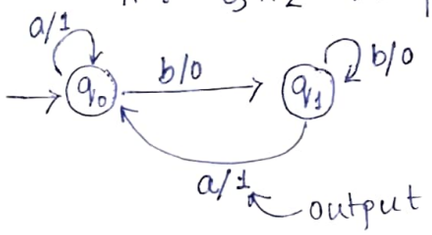
* κ-equivalent: q_i, q_j if both $\delta(q_i, x)$ & $\delta(q_j, x)$ produce final states or both produce nonfinal states for all $x \in \Sigma^*$ of length κ or less ($|x| \leq \kappa$).
* Equivalent or indistinguishable states:
  $\delta(q_i, x) \to F$ and $\delta(q_j, x) \to F$
  or
  $\delta(q_i, x) \not\to F$ and $\delta(q_j, x) \not\to F$

* Distinguishable
  $\delta^*(q_i, w) \in F$ and $\delta^*(q_j, w) \notin F$ for $w \in \Sigma^*$

* FA w/ o/p (FST) = $(Q, \Sigma, \delta, q_0, \Delta, \lambda)$ $\quad \delta : Q \times \Sigma \to Q$

Δ: o/p alphabet; λ: o/p function

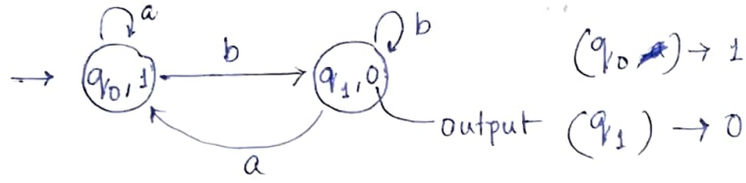
① Mealy machine $\quad \lambda : Q \times \Sigma \to \Delta$ | o/p depends on present state & present i/p



$(q_0, a) \to 1$
$(q_0, b) \to 0$
$(q_1, a) \to 1$
$(q_1, b) \to 0$

[n length input → n length output]

② Moore machine. $\quad \lambda : Q \to \Delta$ | o/p depends on present state only



[n length input → n+1 length output]

* Finite state transducer (FST) 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.

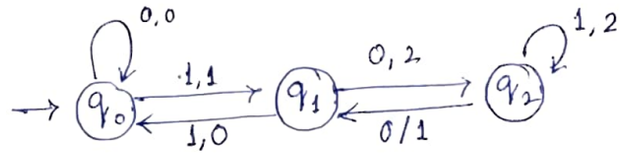
* Mealy : $\delta(q_i, a) = q_j$ next state; $\lambda(q_i, a) = b$ o/p | Moore : $\delta(q_i, a) = q_j$; $\lambda(q_i) = b$

* Construction of FST : TOC1 + RBR Note

eg. Mealy that o/ps the residue mod-3 of binary number. (considering its decima

→ Observe

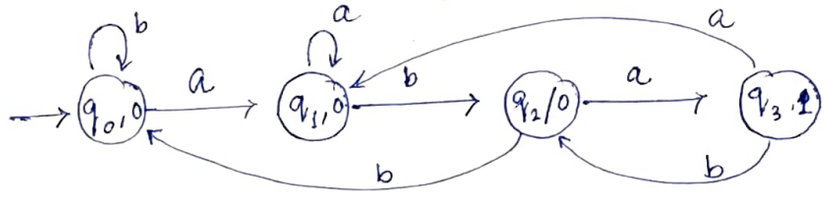
		o/p
0	0	0
1	1	1
2	10	2
3	11	0
4	100	1
5	101	2
6	110	0
7	111	1



- for each o/p different state.
- transition to the state that outputs same residue for given number.

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

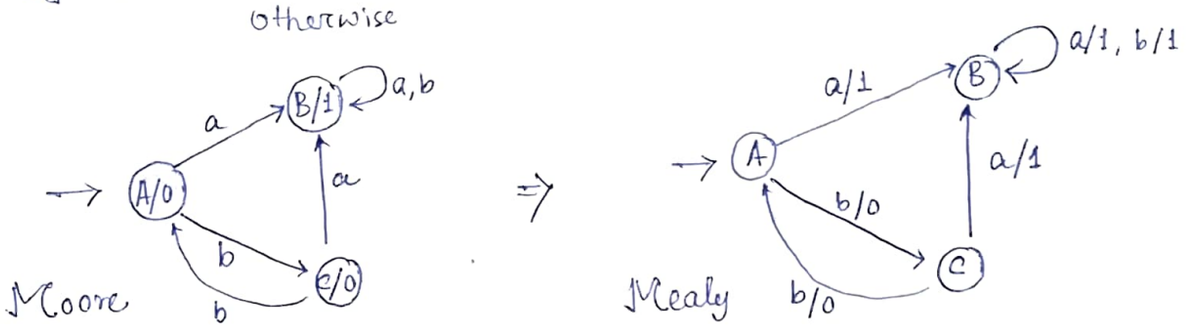
→



* Conversion Mealy ⇌ Moore.

1) Moore → 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. (# states remain same)

eg. (Use tabular method) otherwise



2) Mealy → Moore : (# states increases)

eg. Mealy.

PS	NS (0)	OP (0)	NS (1)	OP (1)
→ q_0	q_0	1	q_1	0
q_1	q_3	1	q_3	1
q_2	q_1	1	q_2	1
q_3	q_2	0	q_0	1

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}]

PS	NS (0)	OP (0)	NS (1)	OP (1)
→ 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: IP 0	NS: IP 1	O/P
→ 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.

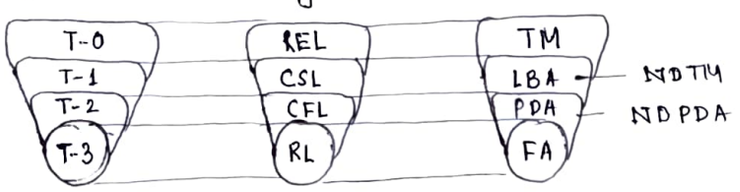
* Shortcut : Moore machine base-n number, residue modulo-k as o/p.

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_4	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_4	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.



* Grammar (V, T, S, P)
  - G → generates → Language L(G)
  - Language → accepted by → Automata
  - Variables, Terminal, Start symbol, Prod^n rules

$L(G) = \{w \in T^* : S \overset{*}{\Rightarrow} w\}$

* Language ↔ Grammar [Derivation tree for gram → Lang]

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

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

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

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

eg $L = \{a^n b^{n+1}, n > 0\}$

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

* Regular Expressions

eg. Every 'a' followed by a '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. $r_1 = 01$, $r_1^* = \epsilon, 01, 0101, \ldots$; $\Sigma = \{0,1\}$, $\Sigma^* = \{\epsilon, 0, 1, 01, \}$
- 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: $L(r_1 + r_2) = L(r_1) \cup L(r_2)$, $L(r_1 r_2) = L(r_1) \cdot L(r_2)$, $L(r^*) = (L(r))^*$, $r_1 \cdot r_2 \neq r_2 r_1$, $\phi r = r\phi = r^*\phi = \phi \cdot \phi = \phi^+ = \phi$, $\epsilon^* = \epsilon = \epsilon^*$, $\phi^* = \epsilon$, $r^* r^+ = r^+$, $r^* r^* = r^*$, $(r^*)^* = (r^*)^+ = (r^+)^* = r^*$, $\epsilon + rr^* = r^*$, $\epsilon + r^* r = \epsilon + r^+ = r^*$, $\phi(qp)^* = (pq)^* p$, $(p+q)^* = (p^* q^*)^* = (p^* + q^*)^*$, $(p+q)^* p^* q^* = (p+q)^* = \phi^* (qp^*)^* = (p^* q)^* p^* = (p+q^*)^* = (p^* + q)^* = p^* (p+q)^* = (p + qp^*)^*$, $rr^* = r^* r = r^+$

eg at most 2 b's
$a^* (b+\epsilon) a^* (b+\epsilon) a^*$

eg no 2 a's come together. {ε, ab, aba, bab, babb, bba, ..}
$(b+ab)^* + (b+ab)^* a = (b+ab)^* (\epsilon + a)$
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 | len ≥ 2 | len ≤ 2

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

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

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

(a+b) → odd; ((a+b)(a+b)) → even

7) Exactly one b | 2 b's

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

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. → $1^* 0^*$

$\Sigma = \{0, 1\}$ $\{\epsilon, 0, 1, 11, 111, \ldots, 10, 100, 100, \ldots\}$

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.

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

- 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

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

$$\text{"}\ a\ \text{"}\ c\ \text{"}\ b\ \text{"}\ +$$

$$\text{"}\ b\ \text{"}\ a\ \text{"}\ c\ \text{"}\ +$$

$$\text{"}\ b\ \text{"}\ c\ \text{"}\ a\ \text{"}\ +$$

$$\text{"}\ c\ \text{"}\ a\ \text{"}\ b\ \text{"}\ +$$

$$\text{"}\ c\ \text{"}\ b\ \text{"}\ a\ \text{"}$$

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

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

18) Every 0 is followed (immediately) by k 1's & 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 even #b even $(aa)^*(bb)^*$

#a odd #b odd $a(aa)^* b(bb)^*$

$\Rightarrow\ a(aa)^* b(bb)^* + (aa)^*(bb)^*$

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

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

22) Containing no double symbols (00 or 11)

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

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

4 Classes

	Start in	End in	(no 2 a's, no 2 b's together)		
i)	a	a	{a, aba, ababa, ...}	—	$(ab)^*a$ or $a(ba)^*$
ii)	a	b	{ab, abab, ababab, ...}	—	$(ab)^*$ or $a(ba)^*b$
iii)	b	a	{ba, baba, bababa, ...}	—	$(ba)^*$ or $b(ab)^*a$
iv)	b	b	{b, bab, babab, ...}	—	$(ba)^*b$ or $b(ab)^*$

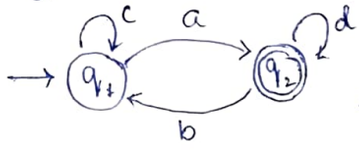
$$\text{Ans} = (ab)^*a + (ab)^* + b(ab)^*a + b(ab)^*$$

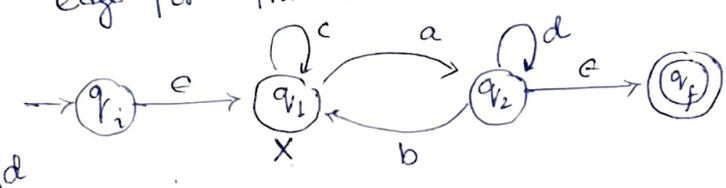
$$= (ab)^*(a+\epsilon) + b(ab)^*(a+\epsilon)$$

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

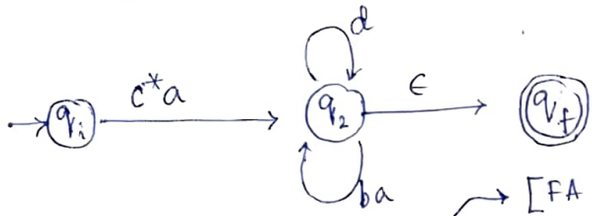
* FA → RE. [More examples at ~~IMP~~ IME-TH]

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

eg 

⇒ 

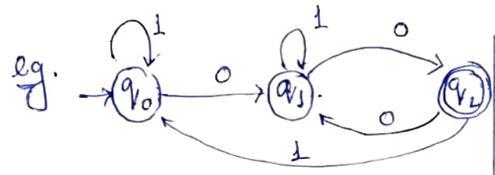
delete q_1, q_2 (not q_i, q_f)

⇒ 

⇒ $c^*a(d+ba)^*$

2. Using Arden's theorem [FA has to be free of ϵ-moves, only one initial state]

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

eg. 

Solve
$$q_0 = q_0 \cdot 1 + q_2 \cdot 1 + \epsilon$$
$$q_1 = q_0 \cdot 0 + q_1 \cdot 1 + q_2 \cdot 0$$
$$q_2 = q_1 \cdot 0$$

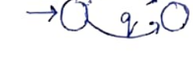
$q_2 = $ ✓

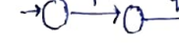
---

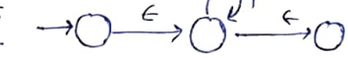
* RE → FA

1. Direct conversion RE → DFA. Using firstpos, followpos } Refer SK book

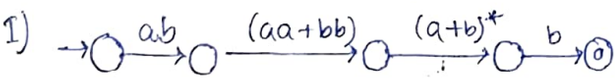
2. RE → ϵ NFA (ϵ-closure) → NDFA → DFA.

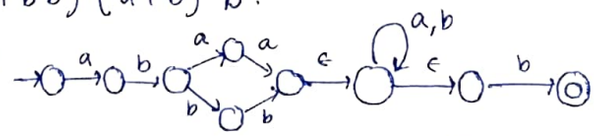
2.a. top-down. p or q p+q 

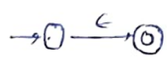
pq 

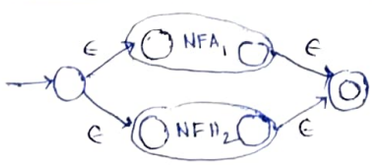
p^* 

eg re = $ab(aa+bb)(a+b)^*b$.

I) 

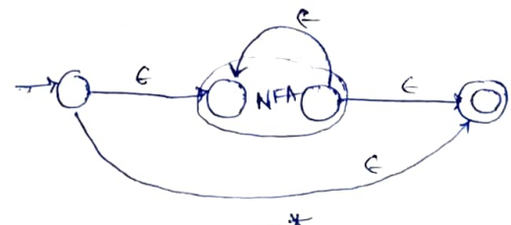
II) 

2.b. Bottom up 



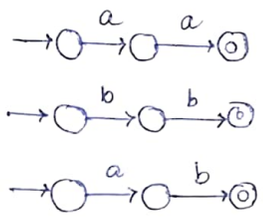
$r_1 + r_2$

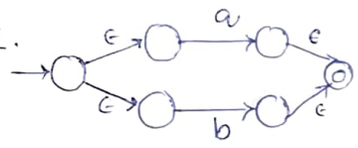
$r_1 \cdot r_2$



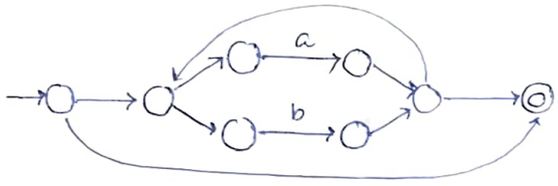
r_0^*

eg. $(ab)(aa+bb)(a+b)^*b$.

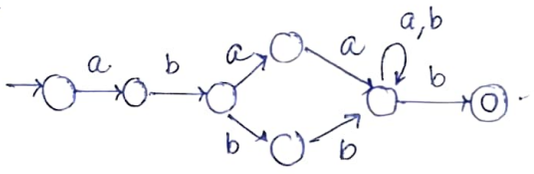
I. 

II. 

$(a+b)$

III. 

$(a+b)^*$

IV 

* Equivalence of 2 FAs

Produce same re.

* Equivalence of 2 re's

Have same FA.

* Regular grammar → FA. [for grammar not producing null string, # states = # non-terminals + 1, each non terminal corresponds to a state, extra state - final state. • for grammar producing null, # states = # non-terminals]

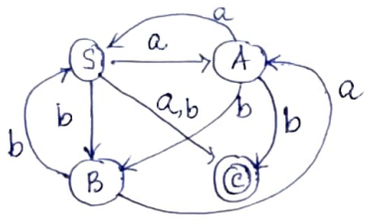
[$A \to aB$: $\delta(A,a) \to B$; $A \to a$: $\delta(A,a) \to$ final state ; $A \to \epsilon$: $\delta(A,\epsilon) \to A$ & A is final state]

eg. $S \to aA/bB/a/b$

$A \to aS/bB/b$

$B \to aA/bS$

3 NTs S, A, B ; final state C



* RE → FA → Regular grammar.

i) RE → FA (eliminate ϵ moves)

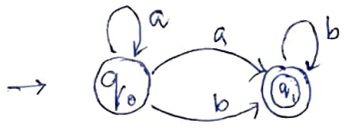
ii) # nonterminals in the grammar = # states of FA

iii) $\delta(q_1,a) \to q_2$: $A_1 \to aA_2$ or $q_1 \to aq_2$ } If q_2 final, both

$\delta(q_1,a) \to q_2$: $q_1 \to a$ [when q_2 is final state]

$\delta(q_1,\epsilon) \to q_1$: $q_1 \to \epsilon$ [when q_1 is final state].

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



Prodⁿ rules

$\delta(q_0,a) \to q_0, q_1$: $q_0 \to aq_0$, $q_0 \to aq_1$, $q_0 \to a$

$\delta(q_0,b) \to q_1$: $q_0 \to bq_1$, $q_0 \to b$

$\delta(q_1,b) \to q_1$: $q_1 \to b$, $q_1 \to bq_1$

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

[FA ↔ RE ↔ Re. grammar ↔ RL

All interchangable, 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. $x = uvw$ and 1) $|uv| \leq n$ 2) $|v| \geq 1$ 3) for all $i \geq 0$, $uv^iw \in L$

[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. $xy^iz \notin L$.

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

$w = a^p \mid w \in L, p \geq n$ 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 $i = p + 1$.

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

So, $xy^iz \notin L$. 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 $\forall i \geq 0$, $xy^iz \in L$.
↑ 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 $a^nb^n; n \leq 10^{10}$ is regular, $a^nb^n, n \geq 1$ is not. (* → next pg)

Eg. $ww^R \mid w \in \{a,b\}^*$: 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 $a^nb^mc^k \mid n,m,k \geq 1$: No comparison ⇒ 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

⇒ irregular

eg $w \mid n_a = n_b$

irregular as needs counting infinitely

eg ww^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.

$\{0^m 1^m \mid m \neq n\}$ not regular

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

states = n

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

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

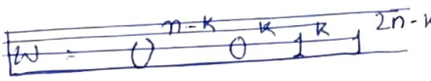
cases i) y only of 0's, $y = 0^k$

ii) y " " 1's, $y = 1^k$

iii) y of 0's, 1's, $y = 0^k 1^k$

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

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



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

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

iii) $i = 2$

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

$xy^2 z = 0^{n-k} (0^k 1^k)(0^k 1^k) 1^{2n-k}$

$= 0^n 1^k 0^k 1^{2n}$ not in the form $0^n 1^{2n}$.

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.

* Myhill-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 $w \in \Sigma^*$, both xw & yw are in L or

Right-invariant property:

2 strings x, y equivalent from language → any $w \in \Sigma^*$ appended to right of x, y still they are equivalent (both 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.

eg. $L = \{$ string ending in '0' $\}$ $\Sigma = \{0, 1\}$

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

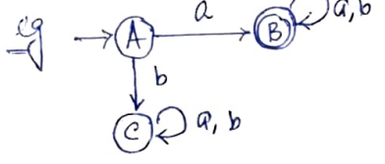
The equivalence classes are

i) String ending in 0
ii) " " 1

} Finite index ⇓ Regular language

* # Myhill-Nerode equivalence classes for a regular language ≡ 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:

$[A] = \{\epsilon\}$
$[B] = a(a+b)^*$
$[C] = b(a+b)^*$
$[A] \cup [B] \cup [C] =$
$\epsilon + a(a+b)^* + b(a+b)^*$
$= \epsilon + (a+b)(a+b)^*$
$= (a+b)^*$

* Summary – Testing regularity

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

2(a) If L is infinite over 2 or >2 symbols & that don't contain any dependency b/w symbols then L is regular.

8. $\{wxw^r \mid |x|=5\}$ not regular

$\{wxw^r\}$ regular

$\{wxw^r\}$ $w, x \in (a+b)^*$

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

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

$\{$ [redacted] $\mid x, y, w \in \{0,1\}^+\}$ regular

$xww^r y$

$\{wxw \mid w, x \in \{0,1\}^*\}$ - regular ($x = (a+b)^*$, $w = \epsilon$)

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

$\{wxw \mid w, x \in \{0,1\}^+\}$ - not regular (CSL)

$\{xww \mid w, x \in (a+b)^*\}$ regular

$\{xww \mid w, x \in (a+b)^+\}$ not regular

$\{xwwR \mid w, x \in (a+b)^+\}$ not regular

$\{xwy \mid w, x, y \in (a+b)^+\}$ regular

$\{xwyw \mid w, x, y \in (a+b)^+\}$ regular

$\{xwy^3 \mid w, x, y \in (a+b)^+\}$ not regular

$\{w$: w is unary notation for natural number multiple of $n\}$ regular

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

$L = \{1^{10}, 1^{100}, 1^{1000}, \ldots\}$ - not regular

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

$L = 100^*$ - regular

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

eg. $11\#111$ ✓

$11\#11$ ✗

- not regular

$\{a^n b^j : |n-j| = 2\}$ - not regular

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

9. Every subset of a regular language isn't regular.

eg. $L = a^*$ $L_1 = a^p : p$ is prime

$L_1 \subset L$

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, L is concat. of 2 regular languages $\Rightarrow L'$ is regular.

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

$= L = \{ xyx^R : x, y \in \Sigma^* \}$ regular

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

12. $L = \{ w : n_a = n_b \}$

L is not regular.

[* is 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.

(13) Closed under cycle of p^n

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

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

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

[gives a string for each symbol of alphabet eg. $\Sigma = \{0, 1\}$

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.

12. Closed under prefix
$L = \{abcd\}$ Prefix$(L) = \{\epsilon, ab, abc, abcd\}$ also regular

We use product DFA of L & M.

↳ Closed under intersection

$L \cap M$, final states of product consist of final states of both automatons 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 M']

↳ Closed under difference L − M

Final states where L has final but M has not.

Product DFA construction

(eg. #a even, #b even).

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

Product automaton $N = (Q_N, \Sigma, \delta_N, (q_0, q_0'), F_N)$

$Q_N = Q \times Q'$, $\delta_N : Q_N \times \Sigma \to Q_N$

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

$\delta_N((q, q'), c) = (\delta(q, c), \delta'(q', c))$

(q, q') : state of N

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

* 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.

L> Using closure properties

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

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! eg If L_1, L_2 are regular then also $L_3 = L_1 \cap L_2$. But, if L_1 and L_3 are regular, we can't say anything about L_2. $ab = ab \cap a^n b^*$

1. Use ϕ, Σ^*, $a^n b^n$, complement of $a^n b^n$ to prove using closure properties. [ϕ for intersection, $\phi \cap$ any lang $= \phi$; Σ^* for 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 $w \in L^R \Rightarrow w \in L \cap L^R$. $\therefore L' = L \cap L^R$. L is regular. L^R also regular (closed under reverse). So, L' too is regular.

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

4. Every regular language except ϕ has a regular proper subset.

• Subset of a RL need not be a RL. $a^n b^n \subseteq \Sigma^*$.

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

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

* 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. $L(G) = \phi$? (Emptiness problem)
3. $L(G) = \Sigma^*$? ✗
4. $L(G_1) \subseteq L(G_2)$. (containment)
5. $L(G_1) = L(G_2)$
6. $L(G_1) \cap L(G_2) = \phi$
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 or right linear.

- Class of linear grammar is a subset of CFG & superset of RLG (Regular lang. gram.). $V \to T^* V T^* / T^*$
- Left linear grammar : $V \to VT^* / T^*$ LHS of prod^n is a non-terminal, RHS contains either a non-t followed by a terminal sequence or terminal sequence. eg. $S \to Sa / Sb / a / b$.
- Right linear grammar : $V \to T^* V / T^*$ eg. $S \to aS / bS / a / b$
  - $S \to aX$, $X \to Sb$ } not left linear, not right linear > We can't mix two, 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) to FA.

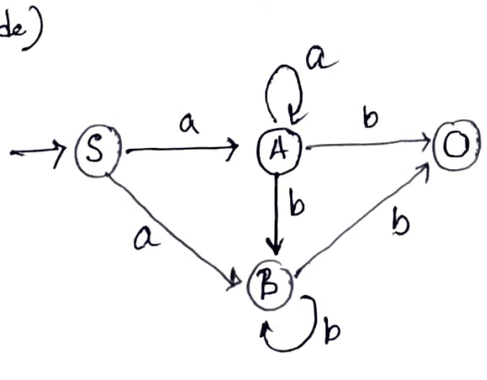
> See pg 4 (reverse side)

eg. $S \to aA / aB$

$A \to aA / bB / b$

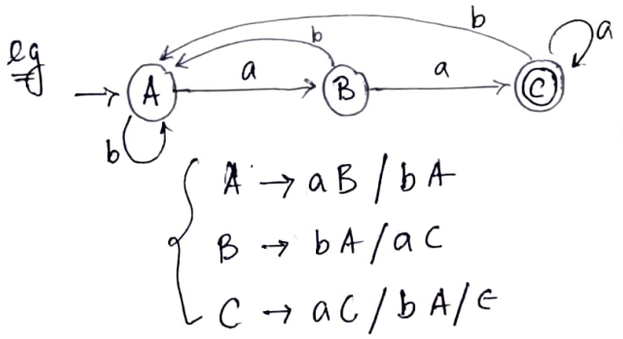
$B \to bB / b$

states = 4



- FA → RLG > See pg 4 (reverse side).

eg



$$\begin{cases} A \to aB \mid bA \\ B \to bA \mid aC \\ C \to aC \mid bA \mid \epsilon \end{cases}$$

or

$$\begin{cases} A \to bA \mid aB \\ B \to bA \mid aC \mid a \\ C \to bA \mid aC \mid a \end{cases}$$

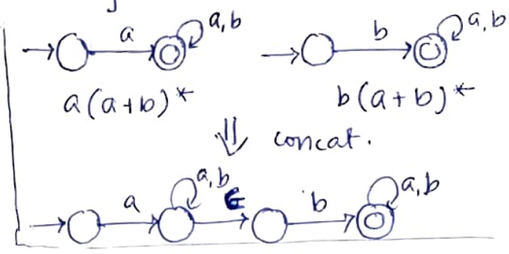
- For any given language, we may have >1 grammars generating the language.

* Closure operations on FA.

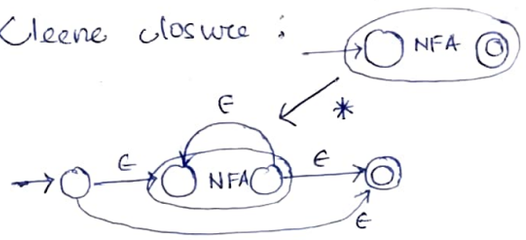
1. Union, Intersection, Difference : We use product DFA of constituent FAs. [see pg 7]

2. Complement : Final interchanged with non final states.

3. Concatenation : Append one FA after another.

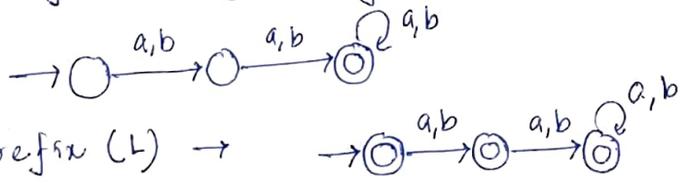


4. Kleene closure :

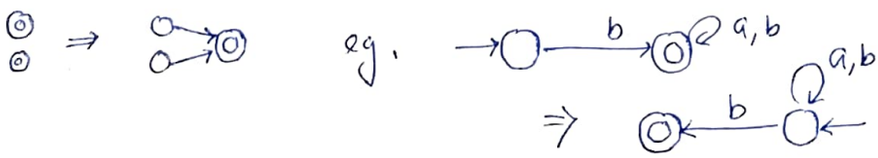


5. Positive closure :

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.



Grammar examples

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

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

$$\begin{cases} S \to AB \mid aAbB \\ A \to aaA \mid \epsilon & \text{---} (aa)^* \\ B \to bbB \mid \epsilon & \text{---} (bb)^* \end{cases}$$

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

$$\begin{cases} S \to aaaA\ bb\ B \\ A \to aA \mid \epsilon \\ B \to bB \mid \epsilon \end{cases}$$

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

$S \to AB$

$A \to aAb \mid ab$

$B \to cB \mid \epsilon$

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

$S \to AB$

$A \to aAb \mid \epsilon$

$B \to cBb \mid \epsilon$

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

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

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

$$\begin{cases} S \rightarrow aAc \mid aSc \\ A \rightarrow aAb \mid ab \end{cases}$$

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

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

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

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

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

$S \rightarrow aSb \mid A$

$A \rightarrow aA \mid \epsilon$.

9. Start, end same symbol.

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

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

$A \rightarrow aA \mid bA \mid \epsilon$

10. Length of string odd.

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

or

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

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

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

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

((

$S \rightarrow A.C$

$C \rightarrow BC/\epsilon$

$B \rightarrow AA$

$A \rightarrow a/b$

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

hint

11. start with a → total len even

or

start with b → total len odd

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

$(a+b)$ $\quad A \to a/b$

$(a+b)(a+b)$ $\quad B \to AA$

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

$$\begin{cases} S \to aAC / bC \\ C \to BC / \epsilon \\ B \to AA \\ A \to a/b \end{cases}$$

[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

$S_1 \to aA$

$A \to aaAb / aA / \epsilon$ } for each addition of b add 2 a's at least

L_2

$S_2 \to Bb / aBb$

$B \to Bb / aBb / aaBb / \epsilon$ } there are non-0 b's

base case

1b : b or ab

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

$S \to S_1 / S_2$

* Other closure properties of RL.

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

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

ii) Under XOR operation. iii) Under NOR operation

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

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

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

$Min(L) = \{ w \mid w \in L, \text{ if } (ay = w, x \in L) \text{ then } y = \epsilon \}$

$Max(L) = \{ w \mid w \in L, \text{ if } wy \in L \text{ then } y = \epsilon \}$

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

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

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

$= \{\epsilon\}$ if $m, n \geq 0$

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

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

A map $f : \Sigma^* \longrightarrow 2^{\Delta^*}$ s.t. 2^{Δ^*} - power set of Δ^*

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

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

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

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

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

$L = a^*$ $Half(L) = a^*$ | $L = a(aa)^*$ $Half(L) = a^*$ | $L = \phi$ $Half(L) = a^*b^*$ $L = a^*b^*$ $Half(L) = a^*b^*$

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

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

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

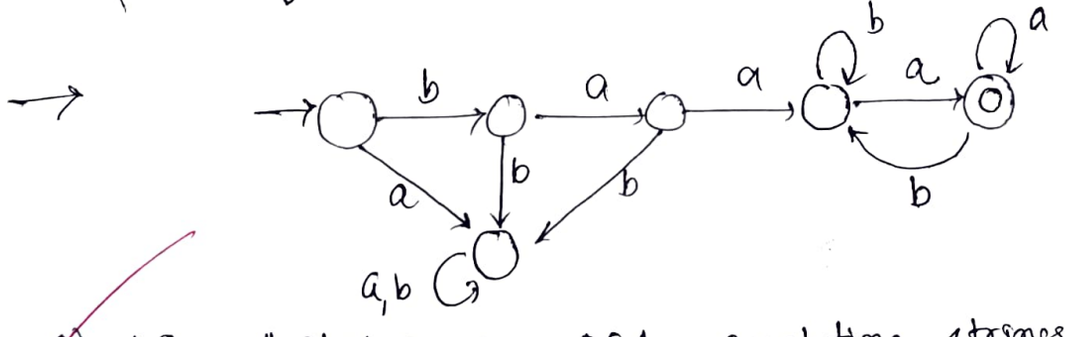
* Minimum number of states in DFA [Also on pg 1]

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 & n
  ↳ Check for divisibility of $lcm(m,n)$
- Binary strings divisible by m or n #states $= lcm(m,n)$
  ↳ check for $lcm(m,n)$ divisibility.

d) Set of strings having exactly m a's & n b's. Min. DFA has $(m+1)(n+1)+1$ states.

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



Ans = 6

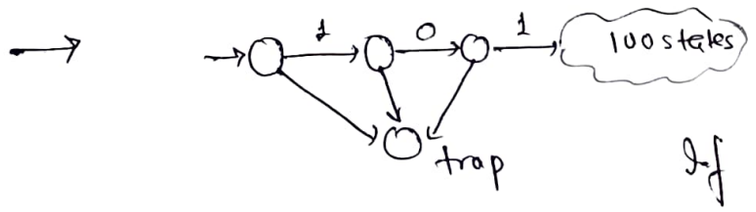
c) Min #states in DFA accepting strings containing #0's divisible by p & #1's divisible by q = pq

Indian Institute of Engineering Science and Technology, Shibpur

Department of Information Technology

Report on. .....

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

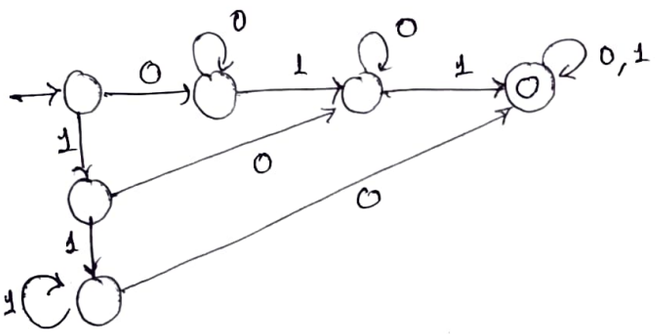


states = 104

If NFA then 103.

Q # states in min DFA in

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



CFG, CFL, PDA

* Context free grammar: $\alpha \to \beta$, $\alpha \in V$, $\beta \in (V \cup T)^*$, $|\alpha| = 1$
(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.]

$aA \to bB$ ← Context

$A \to aB$ ← no context

Eg CFG examples.

1. Balanced parantheses.
$S \to SS / (S) / \epsilon$

2. $\{ w c w^r \mid w \in \{a,b\}^* \}$
$S \to aSa / bSb / c$

3. $(0+1)^* 01^*$
$S \to AOB$
$A \to 0A / 1A / \epsilon$
$B \to 1B / \epsilon$

4. Equal no. of 0's & 1's
$S \to SS / 0S1 / 1S0 / \epsilon$

5. $\{ abba (baa)^n aab (aaba)^n \mid n \geq 0 \}$
$S \to abbaA$ $A \to (baa) A (aaba)$ $A \to aab$

6. $\{ a^n b^m c^m d^n \mid m, n \geq 1 \}$
$S \to aSd / aAd$
$A \to bAc / bc$

7. $\{ a^n b^n c^m d^m \mid m, n \geq 1 \}$
$S \to AB$
$A \to aAb / ab$ $B \to cBd / cd$.

8. Consecutive 0 can occur, but consecutive 1's cannot.
$S \to 0S / 0 / 1 / 1A$
$A \to 0 / 0S$

9. Set of all odd integers.
$S \to AB$
$A \to +/-$
$B \to CD$
$D \to 1/3/5/7/9$.
$C \to CE / \epsilon$
$E \to 0/1/2/3/4/5/6/7/8/9$

10. Even integers.
$S \to AB$
$A \to +-$
$B \to CD$
$C \to CE / \epsilon$
$D \to 0/2/4/6/8$
$E \to 0/..../9$

> 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.

1. $S \to AB/CD$, $A \to aAb/\epsilon$, $B \to dB/\epsilon$, $C \to aC/\epsilon$
   $D \to bDd/\epsilon$
2. $S \to AB/CD$, $A \to aAb/G$, $B \to bB/\epsilon$, $C \to aC/bC/\epsilon$
3. Both 4. Neither $D \to bB/\epsilon$

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

Inherently ambiguous.

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

Unambiguous.

$= DCFL \cup RL$

$= DCFL$

(never ambiguous)

* Backus Naur form (BNF)

::= is defined as | | or | < > category names.

eg. identifier in a programming language.

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

<letter> ::= a | .. | z | A | .. | Z

<digit> ::= 0 | .. | 9

* Derivation & Parse tree.

Left & right most derivation

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

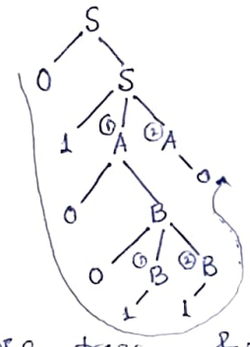
eg. CFG: $S \to 0S/1AA$

$A \to 0/1A/0B$

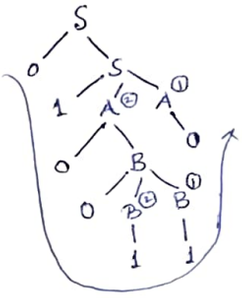
$B \to 1/0BB$

String: 0100110

LMD



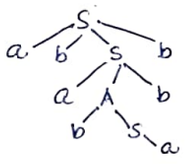
RMD



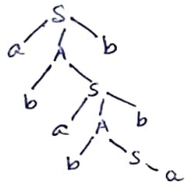
* Ambiguity : More than one parse trees for generating a string.

eg $S \to a/abSb/aAb$ for string ababbb,

$A \to bS/aAAb$.



or



> Ambiguous CFL : $\exists w \in L(G)$ 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 ii) Using semantics & choosing parse tree that makes more sense. iii) Adding grouping rules iv) Fixing the grammar (removing unit, null prodⁿs, useless prodⁿs).

> All DCFLs are unambiguous (although they may have ambiguous grammar; but some (≥ 1) unambiguous grammar must exist. > NCFL may be ambi. or unambi. > Non-deterministic CFL does not imply inherent ambiguity, but inherent ambiguous language is always non-deterministic.

> Removing ambiguity by precedence & associativity rules.

- Precedence constraint: Higher the level of prod^n, lower the priority of the operator.
- Associativity constraint: Operator left associative → induce left recursion in its prod^n. If right associative → right recursion.

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

Priority $* > \cdot > +$ and $+$ is left, $\cdot$ is left associative

Unambiguous grammar

$E \to E + T \mid T$
$T \to T \cdot F \mid F$
$F \to F^* \mid G$
$G \to a \mid b$

| + lowest priority ⇓ highest level of prod^n

eg exp → exp or exp | exp and exp | not exp | T | F

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

exp → exp or exp1 / exp1
exp1 → exp1 and exp2 / exp2
exp2 → not exp2 / exp3
exp3 → 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 → ABC, A → a, B → b, C → c

S → ABC → aBC → aBc → abc ⇒ neither lm nor rm.

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

> Unambiguous grammar does not always generate DCFL. eg. Given length palindrome on {0,1} $S \to 0S0 / 1S1 / \epsilon$ (this is unambiguous), we have only NPDA for it. > Example of inherently ambi. language: 1. $\{a^n b^m c^d \mid m,n > 0\} \cap \{a^n b^n c^m d^m \mid m,n > 0\}$ [$a^n b^m c^d$ common subset is not context-free] , 2. $\{a^n b^n c^m\} \cup \{a^n b^m c^m\}$, 3. $a^i b^j c^k \mid i=j \text{ or } j=k$ and for w = abcd, we can choose to go to first or 2nd language set for prod^n of abcd.

> For given parse tree, we may have its lmd, rmd same.

> If for all strings, lmd is same as rmd, 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) or priority

[↑ level of prod^n → ↓ priority

left recursion → left associative

right " → right "

both → neither]

eg E → E ↑ T / T T → T + F / F F → G - F | G G → id

Precedence id > - > + > ↑

Associativity ↑, + left assoc. - right assoc.

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

$= ((2 \uparrow 1) \uparrow (((3 + (5 - (6 - (8 - 5)))) + 10) + 11)) \uparrow 2$

↑ Start from here while parenthesizing

$= (2^{26})^2$

Q Consider id > (×, ÷) > (+, -)

Given all are left associative construct grammar.

→

$$\begin{cases} E \to E + T \mid E - T \mid T \\ T \to T \times F \mid T \div F \mid F \\ F \to id \end{cases}$$

* Left recursion and Left factoring

> Left recursion $A \overset{+}{\Rightarrow} A\alpha$ $A \in V$, α any string

- Direct left recursion $A \to A\alpha$
- Indirect left recursion $A \to B\alpha$, $B \to A\beta$

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

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

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

$A \to A\alpha / \beta \Rightarrow$ $\boxed{\begin{array}{l} A \to \beta A' \\ A' \to \alpha A' / \epsilon \end{array}}$ β not starting with A

(starts in β, then any number of α)

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

$$\Downarrow$$

$$A \to \beta_1 A' | \cdots | \beta_n A'$$

$$A' \to \alpha_1 A' | \cdots | \alpha_m A' | \epsilon .$$

b) Indirect : Arrange non-terminals in some order $A_1, \ldots, A_n$

Algo
```
for i from 1 to n
    for j from 1 to i-1
        replace each prodn Ai → Aj γ by
            Ai → α1 γ | ... | αk γ where Aj → α1 | ... | αk
    eliminate left recursions among Ai prodns.
```

eg
$E \to E+T | T$
$T \to T*F | F$
$F \to a | (E)$

removing LRs ↓

2 immediate LRs

$E \to E+T$ ⇓ $E \to TE'$, $E' \to +TE' / \epsilon$

$T \to T*F$ ⇓ $T \to FT'$, $T' \to *FT' / \epsilon$

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

eg
$S \to Aa | b$
$A \to Ac | Sd | f$

Har indirect LR. $S \to Aa$, $A \to Sd$

Rename $A_1 \to A_2 a | b$ $A_2 \to A_2 c | A_1 d | f$

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

$i=2$ $j=1$ $A_2 \to A_1 d \Rightarrow A_2 \to A_2 a d / bd$.

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

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

Now, change names to original.

$A \to \alpha\beta_1 / \alpha\beta_2$

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

Grammar

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

$\Downarrow$

$A \to \alpha A_1 \quad A_1 \to \beta_1 \mid \dots \mid \beta_n$

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

* Simplification of CFG. (Removal of useless, null, unit prod^{n}s)

useless: non-generating, non-reachable

> Useless symbols.

eg non-generating. [terminal]

$S \to AC \mid BA$
$C \to CB \mid AC$
$A \to a$
$B \to aC \mid b$

$\Downarrow$ remove prod^{n}s having C

$S \to BA$
$A \to a$
$B \to b$

eg non-reachable

$S \to aC \mid SB$
$A \to bSCa \mid ad$
$B \to aSB \mid bBC$
$C \to aBC \mid ad$

First remove non-g^{ns} then non-reachable

$\Downarrow$ remove non-generating (B)

$S \to aC$
$A \to bSCa \mid ad$
$C \to ad$

$\Rightarrow$ A non-reachable

$S \to aC$
$C \to ad$

> Unit prod^{n}s (A → B) (NT → NT)

eg. $S \to AB,\ A \to E,\ B \to C,\ C \to D,\ D \to b,\ E \to a$

$\Downarrow$ removing

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

eg. $S \to aX / Yb / Y \quad , X \to S, \quad Y \to Yb / b$

$\Downarrow$

$S \to aX / Yb / Yb / b,\ X \to aX / Yb / Y,\ Y \to Yb / b.$

> Null prod^{n}s (NT → ϵ) [If we get $S \overset{*}{\Rightarrow} \epsilon$ then that null prod^n can't be removed]

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

eg $S \to aA \quad A \to b / \epsilon$

$\Downarrow$

$S \to aA / a$

$A \to b$

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

$\Downarrow B \to \epsilon,\ C \to \epsilon$

$S \to ABaC / AaC / ABa / Aa$

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

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

$A \to \epsilon$

$\Rightarrow$ $S \to ABaC / AaC / ABa / Aa / BaC / aC / Ba / a$

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

eg $\left.\begin{array}{l} S \to aS / A \\ A \to aA / \epsilon \end{array}\right\}$ $S \to A \to \epsilon$ so $A \to \epsilon$ can't be removed.

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

> Left linear : $A \to Ba$ NT on the left end
> Right linear : $A \to aB$. NT on the right end
} Regular grammar

> Converting linear grammar into regular grammar (not possible if not left or right linear)

right linear $\left\{\begin{array}{l} S \to baS / aA \\ A \to bbA / bb \end{array}\right. \Rightarrow \left\{\begin{array}{l} S \to bB / aA \\ B \to aS \\ A \to bb / bC \\ C \to bA \end{array}\right.$

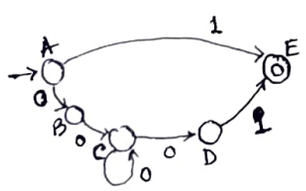
> Left linear
← Right linear

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

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

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

ii) $(1 + 0 \bullet 00^* 0 1)$

iii) 

iv) $A \to 0B / 1E / 1$
$B \to 0C$
$C \to 0C / 0D$
$D \to 1E / 1$
E useless

v) $A \to B0/1$
$B \to C0$
$C \to C0 / D0$
$D \to 1$

> Right linear
← Left-linear

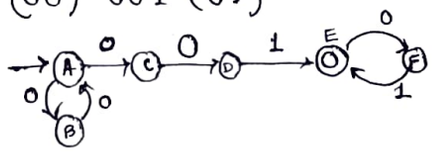
i) reverse right HS of each prod^n
ii) regex
iii) reverse regex
iv) FA
v) generate right linear grammar from FA.

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

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

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

iii) $(00)^*001\,(01)^*$

iv) 

v) $A \to 0B \mid A \to 0C \mid D \to 1E/1$
$B \to 0A$
$C \to 0D$
$E \to 0F$
$F \to 1E/1$

* Normal Forms.

1. Chomsky Normal Form. (CNF)

If all prod^{n}s of CFG are in the form:

NT → string of exactly 2 NTs $A \to BC$

NT → Single terminal. $A \to a$

> Converting to CNF

1. Eliminate useless, unit, null prod^{n}s.
2. Eliminate terminals on the RHS of length 2 or more. (Check if in CNF already)
3. Restrict # NTs on RHS to two.
4. Convert string containing NTs & Ts to string of only NTs on the RHS.

eg $S \to bA / aB \quad A \to bAA / aS / a$
$B \to aBB / bS / b$

(already simplified)

$$\begin{cases} N_a \to a \quad N_b \to b \\ S \to N_aB \mid N_bA \\ A \to N_bAA \mid N_aS \mid a \\ B \to N_bBB \mid N_bS \mid b \end{cases} \longrightarrow \begin{cases} S \to N_aB \mid N_bA \\ A \to N_bS \mid N_aS \mid a \\ B \to N_bT \mid N_bS \mid b \\ N_a \to a \quad N_b \to b \\ S \to AA \quad T \to BB \end{cases}$$ in CNF

eg $\begin{cases} S \to abAB \\ A \to bAB / \epsilon \\ B \to BAa / \epsilon \end{cases} \xrightarrow{A \to \epsilon} \begin{cases} S \to abAB / abB \\ A \to bAB / bB \\ B \to BAa / \epsilon / Ba \end{cases} \xrightarrow{B \to \epsilon} \begin{cases} S \to abAB / abA / abB / ab \\ A \to bAB / bA / bB / b \\ B \to BAa / Aa / Ba / a \end{cases}$

$$\downarrow$$

$$\begin{cases} N_a \to a \quad N_b \to b \\ S \to N_aN_bAB / N_aN_bA / N_aN_bB / N_aN_b \\ A \to N_bAB / N_bA / N_bB / b \\ B \to BAN_a / AN_a / BN_a / a \end{cases} \leftarrow \begin{cases} S \to CD \mid CA \mid CB \mid N_aN_b \\ A \to N_bD \mid N_bA \mid N_bB \mid b \\ B \to EN_a \mid AN_a \mid BN_a \mid a \\ C \to N_aN_b \\ D \to AB, \quad E \to BA \end{cases}$$

Advantages of CNF i) Len of prod^n restricted. ii) Parse tree obtained from CNF is always binary tree. iii) # states to derive a string of length $|w|$ is $2|w|-1$. iv) It's easy to apply CYK algo on CNF. → (Why? - refer CNF Intro)

$|w|-1$ steps to make a string of len $|w|$ containing only NTs. Then $|w|$ steps to change each non-ter. to a termt.

$S \to AB \quad A \to a \quad B \to b$

$S \to AB \to aB \to ab$. (3 steps)
(1) (2)

---

2. Greibach Normal form.

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

$$\begin{cases} NT \to (\text{single T})(\text{string of NTs}) \\ NT \to (\text{single T}) \end{cases}$$

> Converting to GNF.

i) Eliminate null, unit prod^{n}s.

ii) Rename NTs of V as $A_1, A_2, \ldots, A_n$. Start symbol A_1

iii) Make all prod^{n}s of the form $A_i \to A_j V$ where $i \le j$.

[$A \to A\alpha / \beta_1 / \beta_2 \cdots / \beta_n$

$\downarrow$

$A \to \beta_1\alpha / \beta_2\alpha / \cdots / \beta_n\alpha$

iv) Use this –

$A \to A\alpha_j / \beta_i \;\Big|\; 1 \le j \le m$, $1 \le i \le n$

$\downarrow$ Introduce a NT X

$A \to \beta_i / \beta_i X$

$X \to \alpha_j / \alpha_j X$

> Advantages of GNF

i) # steps reqd to generate string w is $|w|$.

ii) Useful to convert CFG to PDA.

---

eg Converting to GNF.

$S \to AA / a \qquad A \to SS / b.$

(Already in CNF)

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

$A_1 \to A_2 A_2 / a \qquad A_2 \to A_1 A_1 / b$

$A_2 \to A_2 A_2 A_1 / a A_1 / b$ ($A\alpha_j$ / β_i)

use (iv)

$A_2 \to a A_1 / a A_1 X_1$

$X_1 \to A_2 A_1 / A_2 A_1 X_1$

$A_2 \to b / b X_1$

↳ in GNF.

(but $A_1 \to A_2 A_2$ is not GNF.)

$\downarrow$

$A_1 \to a A_1 A_2 / b A_2 / b X_1 A_2 / a A_1 X_1 A_2 / a$

(X_1 prod^{n}s not in GNF) $\downarrow$

$X_1 \to a A_1 A_1 / a A_1 X_1 A_1 / b A_1 / b X_1 A_1 / a A_1 A_1 X / a A_1 X_1 A_1 X / b A_1 X_1 / b X_1 A_1 X_1$

So, the GNF $\Rightarrow$

$A_1 \to a A_1 A_2 / b A_2 / b X_1 A_1 / a A_1 X_1 A_2 / a$

$A_2 \to a A_1 / a A_1 X_1 / b / b X_1$

$X_1 \to$ ...

- **Important points on Normal forms**

1. CNF ≈ Binary standard form
2. CNF # prod^{n}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 $\le 2^{k-1}$
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. $\quad l = 2^{h-1} \Rightarrow h = \log_2 l + 1$

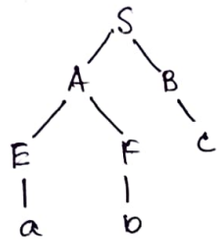
5. CFG G w/o null prod^{n}s & unit prod^{n}s of k be the max # symbols on the RHS of any prod^n.

Then the eqv. CNF contain a max of $\boxed{(k-1)\,|P| + |T|}$ prod^{n}s. [$|P| \to$ # prod^{n}s of G, $|T| \to$ # terminals].

eg CNF CFG: $S \to AB \quad B \to C \quad F \to b \quad A \to EF \quad E \to a$

String abc

Find max. yield length.



max path length = 3

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

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

7. No of prod^{n}s needed in GNF for string $w = |w|$

* CFG examples.

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

2. Odd len. palindromes.

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

3. $n_a = n_b$.

$S \to aSbS \mid bSaS \mid \epsilon$

or

$S \to aB \mid bA \mid \epsilon$

$A \to aS \mid bAA \mid a$

$B \to bS \mid aBB \mid b.$

4. $\{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$

$S \to A \mid B$

$A \to aAb \mid ab$

$B \to aBbb \mid abb$

* Closure Properties of CFG (or CFL class): Closed under —

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

• Cycle of Σ: cycle(L) = $\{xy \mid x, y \in \Sigma^* \text{ and } yx \in L\}$

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. $L_1 = \{a^n b^n \mid n \geq 1\}$ $A \to aAb \mid ab$

$L_2 = \{a^n b^{2n} \mid n \geq 1\}$ $B \to aBbb \mid abb$

$L_1 \cup L_2$: $S \to A \mid B$ $A \to \cdots$ $B \to \cdots$ $L_1 \cup L_2$ 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 $L_1 = \{a^n b^n \mid n \geq 1\}$ $L_2 = \{c^m d^m \mid m \geq 1\}$

$S \to AB$ $A \to \cdots$ $B \to \cdots$ $L_1 \cdot L_2$ is CFL. $A \to aAb/ab$ $B \to cBd/cd$

e) Kleene closure $L = \{a^n b^n \mid n \geq 1\}$ $A \to aAb/ab$

L^* is CFL. $L^* \to S \to \epsilon / AS$ $A \to aAb/ab$

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 $L = \{a^n b^n \mid n \geq 1\}$ $f(a) = 0^k 1^k$ $f(b) = 1^m 0^m$

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

CFG (f(L)) : $A \to XAY \mid XY$, $X \to 0X1 \mid \epsilon$, $Y \to 1Y0 \mid \epsilon$ } CFL. ✓

h) 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)) : $A \rightarrow 01A111 / 01111$. → CFL ✓

* 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 $z = uvwxy$ for some strings u, v, w, x, y.
2. $|vwx| \leq n$
3. $|vx| \geq 1$
4. $uv^i wx^i y \in L$ 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. $uv^k wx^k y \notin L$. Contradiction!

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

$Z = a^n b^n c^n \qquad |Z| = 3n > n$

$Z = uvwxy = a^n b^n c^n.$ | $1 \leq |vx| \leq n$ as $|vwx| \leq n$

$\Rightarrow$ v or x can't contain all the 3 symbols.

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 ~~three~~ symbols.

take value of $k = 2$.

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

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

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

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

So, it's not CFL.

Observations

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

2. (CFL when) given an exp^n s.t. it's possible to obtain a center or mid point in the string, 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 ✓ (push 2 a's for 1 a, pop 1 a when b)

$a^n b^n c^n$

X CFL

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

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

4. Given exp^n consists of an op^n where mid point could be found along with some independent regulare expressions in between, results into CFL.

eg. $a^m b^i c^m d^k$ CFL. b^i, c^k are regular in between that does not affect stack

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

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

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

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

7. Counting & comparison only with top of stack.

Language exhibiting characteristics that involves comparison with bottom of stack is not CFL.

eg. $\{a^m b^n c^m d^n\}$ 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.

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

9. 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 $\subset$ 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^r starts.

eg. CFL but not DCFL - $\{ww^r\}$ (palindrome), - $\{a^i b^j c^k \mid i=j \text{ or } k=i\}$,

- $\{a^n b^m c^m \cup a^n b^m c^m\}$, - $\{a^n b^m c^x d^y \mid n=m \text{ (or) } x=y\}$ ↓ 2 copies of PDA

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

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

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

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

- DCFL $\cup$ RL = DCFL
  DCFL $\cap$ RL = DCFL

* Examples. (CFL/DCFL/RL)

1) $\{a^n b^n c^m \mid n > m\}$ not CFL
2) $\{xcy \mid x,y \in (0,1)^*\}$ RL, DCFL, CFL
3) $\{a^i b^j c^k \mid j = i+k\}$ DCFL, CFL [$k = j-i$]
4) $\{w \mid w \in (a,b), n_a = n_b + 1\}$ DCFL, CFL
5) $\{a^m b^n c^n \mid n \geq 0\}$ not CFL
6) $\{ww \mid w \in \{0,1\}^*\}$ not CFL
7) $\{a^i b^j c^k \mid 0 \leq i \leq j \leq k \leq n\}$ not CFL
8) $\{a^{n!} \mid n \geq 0\}$ not CFL
9) $\{a^n b^j \mid n = j^2\}$ not CFL
10) $\{a^{rts} \mid r \text{ is prime}\}$ 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

Hierarchy

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

{ If only RL → RL, DCFL, CFL all

{ If CFL only → not RL or not DCFL

14) $a^m b^m c^n d^n$ DCFL

15) $a^m b^n c^m d^n$ X CFL

16) $a^m b^n c^n d^m$ DCFL

17) $a^m b^i c^m d^k$ DCFL

18) $a^m b^n \mid m > n$ DCFL

19) $a^n b^{2n} \mid n \geq 1$ DCFL

20) $a^n b^{n^2}$ X CFL

21) $a^n b^{2^n}$ X CFL

22) $ww \mid w \in (a+b)^*$ X CFL

23) $a^n b^n c^m \mid n > m$ X CFL

24) $a^n b^n c^n d^n \mid n \leq 10^{10}$
RL, DCFL, CFL

25) $a^n b^{2n} c^{3n} \mid n \geq 1$ X CFL

26) $ww^r \mid |w| = l$
l known RL, DCFL, CFL

27) WWW^r X CFL

28) $a^n b^{3^n}$ X CFL

29) $a^m b^n \mid m \neq n$ DCFL

30) $a^m b^n \mid m = 2n+1$
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?)

32) $a^k \mid k$ even DCFL
1st a push, 2nd a pop,
3rd a push, 4th a pop.

34) $a^i b^j c^k \mid j = i+k$ DCFL

33) $a^i b^j c^k \mid i > j > k$
X CFL

35) $a^i b^j c^k d^l \mid i = k$ or $j = l$
CFL (not DCFL)

36) $a^i b^j c^k d^l \mid i = k$ and $j = l$
X CFL

37) $a^m b^l c^k d^n \mid m, l, k, n \geq 1$
RL

38) $a^n b^{4m} \mid n, m \geq 1$ RL

39) $\{a^3, a^8, a^{13}, \ldots\}$ RL

40) a^{n^n} X CFL

41) $w \mid |w| \geq 100$ RL

42) $w \mid n_a = n_b = n_c$ X CFL

43) $w \mid n_a \geq n_b + 1$ DCFL

44) $ww^r w^r$ X CFL

45) $a^p \mid p$ is composite number
X CFL

46) $0^i 1^j \mid j = i^2$ X CFL

47) $0^i 1^j 0^k \mid j = \max(i, k)$ X CFL

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

49) $w \mid n_a = n_b + n_c$ DCFL

50) $a^m b^n c^p \mid m = n$ or $n = p$
↳ CFL

51) $xww \mid x, w \in \{0,1\}^*$ RL

52) $ww^r w$ X CFL

53) $a^n b^n c^i \mid i \neq n$ X CFL

54) $ww^r w^r$ X CFL

55) $a^m b^{3m}$ DCFL

56) Strings of balanced parantheses DCFL

57) $\{ca^n b^n \mid n \geq 0\} \cup \{da^m b^{2m} \mid n \geq 0\}$ DCFL

58) $\{a^n c b^n \} \cup \{a^n d b^{2n}\}$ DCFL

59) $a^n b^m \cup a^n b^{2n}$ CFL (not DCFL)

> Some points regarding identification of language.

- CFL ∩ RL = CFL (need not be RL) - RL - CFL = need not be CFL
- CFL − RL = CFL, CFL ∪ RL = CFL, CFL / RL = CFL
- DCFL − RL = DCFL, DCFL ∩ RL = DCFL, DCFL ∪ RL = DCFL
- DCFL / RL = DCFL, RL − DCFL = DCFL
- CFL not closed under complement, difference, intersection.
- DCFL ∩ ∪ − union, intersection, difference, concat, *, +, reversal, homomorphism, substitution.

* Decision problems on CFG, CFL

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

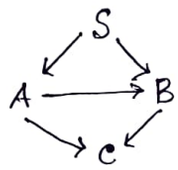
Algo > Eliminate useless symbols (unreachable, nongenerating)
> If the start symbol itself is useless, then empty. Otherwise, non-empty.

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. (nodes = nonterms)
> If cycle present then infinite. Otherwise not.

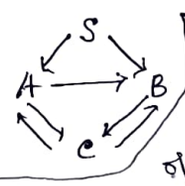
eg $S \to AB \quad A \to BC/a \quad B \to cc/b \quad C \to a$

No cycle, so finite language.



eg $S \to AB \quad A \to BC/a \quad B \to CC/b \quad C \to a/AB$

Has cycle, infinite language.



3. **Membership** : CYK (Cocke-Younger-Kasami) Algorithm.

Build a triangular table where each row of the table corresponds to one particulare length of sub strings. [Bottom most - of length n]
n - length of string

(Grammar has to be in CNF.)

$O(n^3|G|)$
n - length of passed string
$O(n^3 \times |G|)$
(DP algo, bottom-up)
|G| - # rules in G.

eg $S \to AB/BC$

$A \to BA/a$

$B \to CC/b$

$C \to AB/a$

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				

x_{ij} in the table represents - x (string) starting from location i and ending in loc^n j.

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

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

First fill $x_{11}, x_{22}, x_{33}, x_{44}, x_{55}$ then $x_{12}, x_{23}, x_{34}, x_{45}$ ....

(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: ⊙ $x_{12} \to$ b a a b a to generate ba $\to$

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

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

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

⊙ $x_{23} \to$ aa $\quad (23) = (22)(33) = \{A,C\}\{A,C\}$

$= \{AA, AC, CA, CC\}$ ✓

⊙ x_{13} $\quad (13) = (11)(23) = \{B\}\{B\} = \{BB\}$

$= (12)(33) = \{A,S\}\{A,C\} = \{AA, AC, SA, SC\}$ ×

⊙ x_{14} $\quad (14) = (11)(24)$

$= (12)(34)$

$= (13)(44)$

⊙ $(15) = (11)(25)$

$= (12)(35)$

$= (13)(45)$

$= (14)(55)$

↩ 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?

→ Check on which cells there is S (x_{12}, x_{34}, x_{45}, x_{25}). So, the substrings are: ba, ab, ab, aaba

total 3 substrings.

> Decision problems on DCFL, CFL

	L(G) regular	L(G) finite
DCFL	D	D
CFL	UD	

	$w \in L(G)$	$L(G) = \phi$	$L(G) = \Sigma^*$	$L(G_1) \subseteq L(G_2)$	$L(G_1) = L(G_2)$	$L(G_1) \cap L(G_2) = \phi$
DCFL	D	D	D	UD	D	UD
CFL	D	D	UD	UD	UD	UD

> Application of directed transition graph of CFG

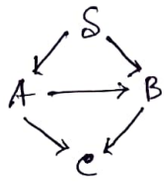
Rank of a variable can be computed if the graph has no cycle [for finite language rank is computed.]

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

eg $S \to AB$, $A \to BC \mid a$, $B \to CC \mid b$, $C \to a$



$r(S) = 3$ $S \to A \to B \to C$

$r(A) = 2$

$r(B) = 1$ $r(C) = 0$

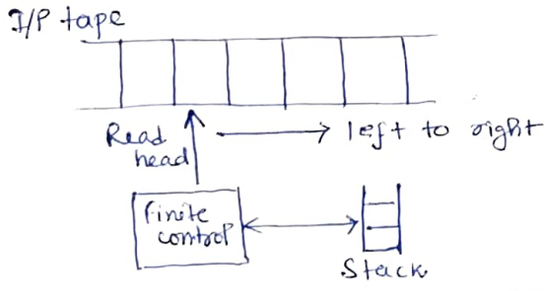
* PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.

Γ – stack symbols (alphabet)

Z_0 – stack bottom or initial top

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

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



> Acceptance by –

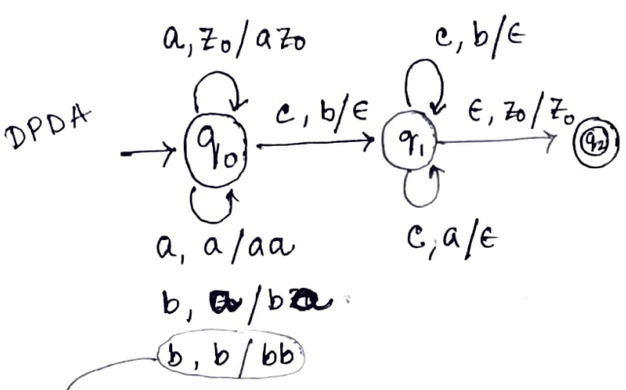
i) Empty stack

ii) final state

> Constructing PDA

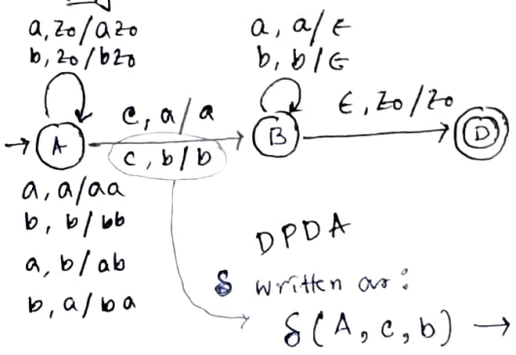
δ in the form A, B → c or A, B/c :
reads A as input, pops B, pushes c.

eg $a^n b^m c^{m+n} \mid m, n \geq 0$



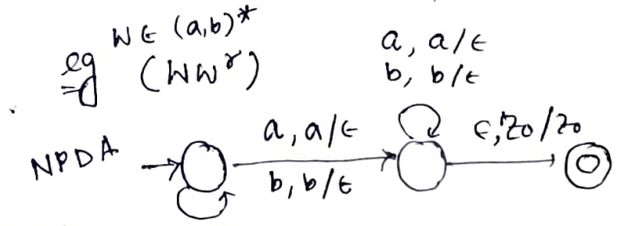
read as: if the head reads 'b' as input, and finite control checks the top of the stack to be 'b' then pop 'b' and push 'bb' in the stack.

eg $wcw^r \mid w \in (a,b)^+$



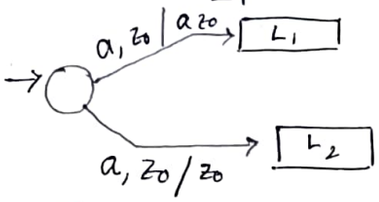
DPDA δ written as: $\delta(A, c, b) \rightarrow (B, b)$

eg (ww^r) $w \in (a,b)^*$

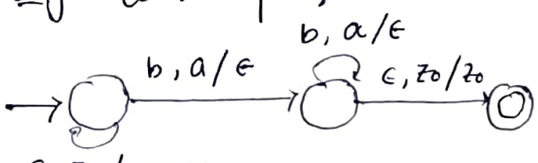


eg $\{ a^i b^j c^k d^l \mid i = k \text{ or } j = l \}$

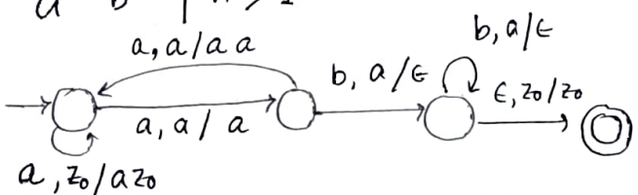
$= \{ a^m b^j c^m d^l \} \cup \{ a^i b^m c^k d^m \}$
L_1 L_2



eg $a^n b^{2n} \mid n \geq 1$

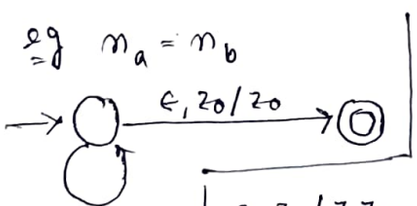


eg $a^{2n} b^n \mid n \geq 1$



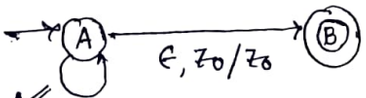
a a a a b b
push push pop
ignore ignore

eg $n_a = n_b$



	a, Z0/Z1Z0
	b, Z0/Z1Z0
	c, Z0/Z2Z0
	e, Z2/Z2Z2
	a, Z1/Z1Z2
	b, Z1/Z1Z1
	c, Z1/ϵ
	a, Z2/ϵ
	b, Z2/ϵ
	ϵ, Z0/ϵ (by empty stack)

eg $n_a + n_b = n_c$



a, Z0/aZ0
b, Z0/bZ0
c, Z0/cZ0
c, c/cc
a, a/aa
b, b/bb

* ID (Instantaneous desc^n)
(state, input, stack)
current remaining contents

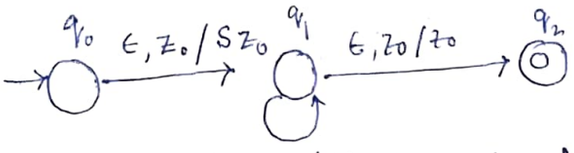
eg. c a b c

A, cabc, Z0
A, abc, Z2Z0
A, bc, Z0
A, c, Z1Z0
A, ϵ, Z0 ✓

CFG → PDA a) CFG → GNF CFG, b) S (start) put to the stack $\delta(q_0, \epsilon, Z_0) \to (q_1, SZ_0)$ c) for prodⁿ $NT_1 \to T \cdot NT$ [single terminal, followed by string of non-terminal), $\delta(q_1, T, NT_1) \to (q_1$, string of NT), d) for prodⁿ like $NT_1 \to$ {single terminal}, $\delta(q_1, T, NT_1) \to (q_1, \epsilon)$ e) for accepting a string, 2 transitional functions are added. (i) $\delta(q_1, \epsilon, Z_0) \to (q_1, \epsilon)$ by empty stack (ii) $\delta(q_1, \epsilon, Z_0) \to (q_{final}, Z_0)$ by final state.

(More on SK book)

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



$(a, S/A), (a, A/ABC), (b, A/B)$

$(b, B/\epsilon)$, $(a, A/\epsilon)$; $(\epsilon, C/\epsilon)$, $(\epsilon, Z_0/\epsilon)$

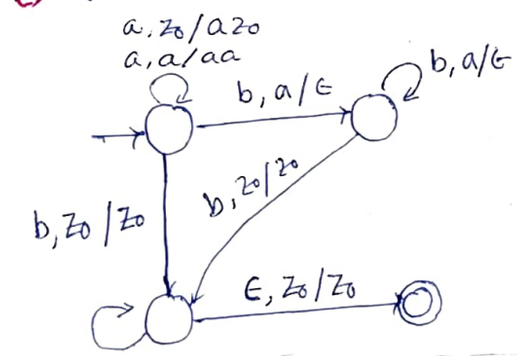
ID for aabbbc

$(q_0, aabbbc, Z_0) \to (q_1, aabbbc, SZ_0)$
$\to (q_1, abbbc, AZ_0) \to$
$(q_1, bbbc, ABCZ_0) \to$
$(q_1, bbc, BBCZ_0) \to$
$(q_1, bc, BCZ_0) \to$
$(q_1, c, CZ_0) \to$
$(q_1, \epsilon, Z_0) \to (q_2, Z_0)$ ✓

PDA → CFG SK book (403)

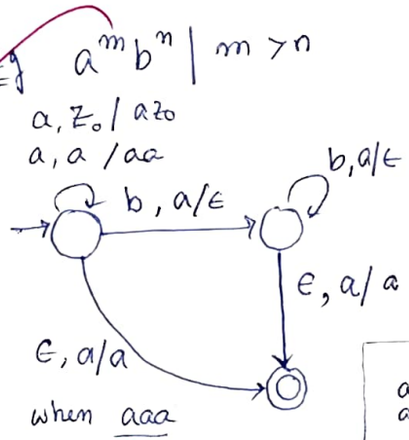
* CFL ↔ PDA

eg $a^m b^n \mid m < n$.



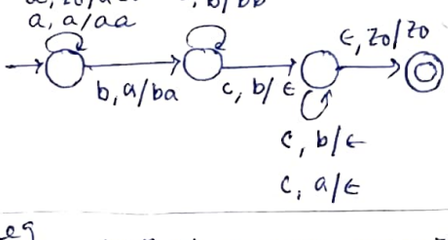
b, Z₀/Z₀ when bb

eg $a^m b^n \mid m > n$

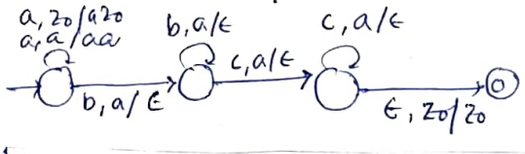


when aaa

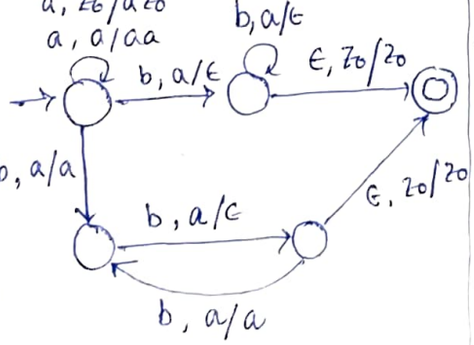
eg $a^m b^n c^k \mid m+n=k, m,n,k \geq 1$



eg $a^m b^n c^k \mid m = n+k, m,n,k \geq 1$

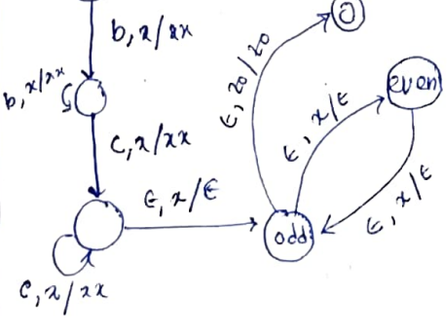


eg $a^m b^n \mid (m=n)$ or $(2m=n)$ $m,n \geq 1$



aabbbb

eg $a^m b^n c^k \mid (m+n+k)$ is odd; $m,n,k \geq 1$



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

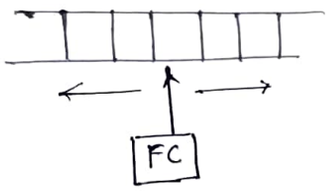
* More PDA conv^n on ME_TH (105), SK book Chapter end (PDA)

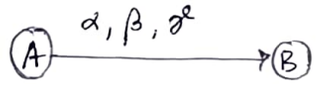
TM and Undecidability

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

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

for NTM, δ : $Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R,H\}}$



* 

input tape i/p α
replace with β
go to direction ℓ = L or R

$\delta(A, \alpha) = (B, \beta, \ell)$

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

$\ell \in \{L, R\}$

* Lang. accepted by Halting TM = Recursive lang.

* Lang. accepted by TM = REnumerable Lang.

(Unrestricted grammar generates REL)

$(VUT)^+ \rightarrow (VUT)^*$

* TM as acceptor, transducer

* TM can't accept ϵ.

* There are TMs that never halts.

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

* 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.

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

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

* CSG

$(VUT)^+ \rightarrow (VUT)^*$

$x \rightarrow y$

$|x| \leq |y|$

* For RE lang. TM may or may not halt.

* No membership algo for RE lang. For REC Lang. exists.

* Countable A set is countable if $\exists$ an enumeration method using which all the elem 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 onto natural numbers.

$\Sigma, \Sigma^*, 2^\Sigma$ countable

2^{Σ^*} not countable

eg. p/q $p, q \in Z^+$

1	2	3	4
1/1			
	1/2	2/1	
		1/3	2/2	3/1
			1/4	...

So, countable

Similar way, Σ^* countable.

> every subset of countable set is finite or countable.

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

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

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

> 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)

• A TM must contain ≥ 2 states.

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

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

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

* Undecidability

Algo → HTM | TM - countable ∴ Some problems that have no algo

Comp^y : If there's algo for a f^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 lang. is undecidable.
- PCP is undecidable.

- L_{ne} is REL, not REcL.
  L_e 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.
  eg. P_{CFL} : Set of REL that are CFL.

Trivial property $P = \phi$, or P = Set of all RELs.

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

- If a lang. is RE, but not REC, then it's complement is non-RE. If L & $\bar{L}$ are RE, then both are REC.

- Any particular instance of a prob is always decidable.

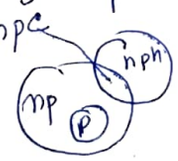
- For RE (rec. enu) lang, TM may halt or not halt, but for each REC TM exists. Lang (TM acceptable)
- For REC (rec. ~~enum~~), TM must halt. (TM decidable)

> no membership algo for RE.
We 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 ~~comp~~ prob. is decidable
  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

Undecidable ⇔ not recursive
⇔ REL or non REL

- **P** : ∃ DTM or HTM , for input of size n it halts in (polynomial of n) time.

NP : ∃ 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.)
  - > {M#x | M accepts x} — REL as we can design TM for this
  - > {M#x | M halts on x} — Semidecidable, Undecidable.
    - Halting problem : RE but not REC.

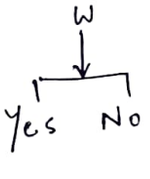
- Propositional logic is decidable. First order logic is undecidable.

- Decidable → If recursive , otherwise undecidable

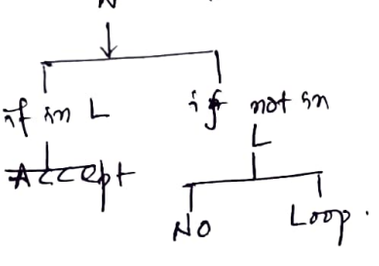
  Semidecidable → If RE. [eg. Halting problem, Blank tape halting prob, State entry prob, PCP, MPCP]
  (also undecidable)

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

- Decidable



Semidecidable or (recognizable)



- To show a L is not in RECL, it suffices to show that some L' is not in REC L and L' → L (L' reduces to L).
- Showing L' → L ≡ $\overline{L'}$ → $\overline{L}$

- Recognizable langs. closed under union, intersec".
- A lang. L is decidable iff L, $\overline{L}$ both are recognizable.
- A lang L is recognizable iff it is enumerable.
- semi-decidable ⇔ recognizable ⇔ 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 RECL).

(Problem ≈ language)

⇒ 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) $C \subseteq REL$, 2) $C \neq REL$ & 3) $C \neq \phi$. So, L is not recursive hence, undecidable.

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

---

* Set

- Countable
  - Finite
  - Countably infinite. → Complete lang Σ^*, $\mathbb{N}$, Pdt of 2 countable sets
- Uncountable (No TM) → $\mathbb{R}$, Power set of $\mathbb{N}$, Power set of complete language 2^{Σ^*}, # pts in a line

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

> 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.

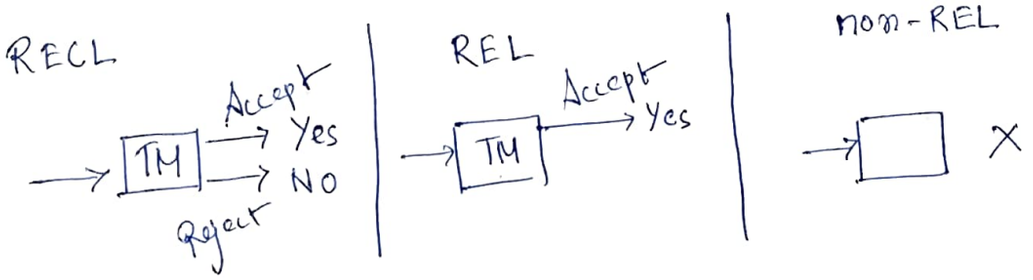
d) Checking whether L generated by a TM is regulare or not (regularity), e) Checking TM accepts ϵ or not [although TM accepts ϵ, but $L = \{\langle M \rangle \mid \epsilon \in L(M)\}$ is undecidable

Non - REL. (Undecidable)

a) Uncountable sets b) set of all unambiguous grammare, e) 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. ∃ a HTM or algorithm for that problem. All RECL.

* Undecidable problem ∃! 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 $\overline{No}$.

* 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 e 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.

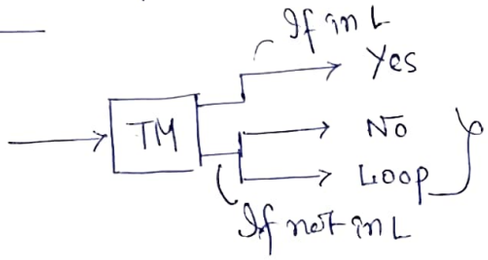
* Undecidable probs. abt programming langs.

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,

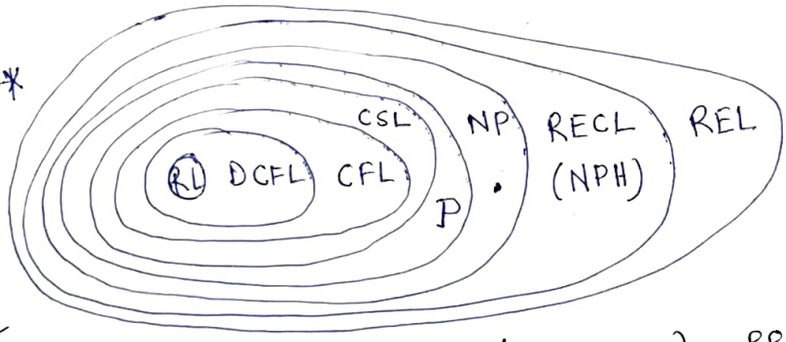
* Post Correspondence Problem. (PCP) MPCP → PCP.

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.



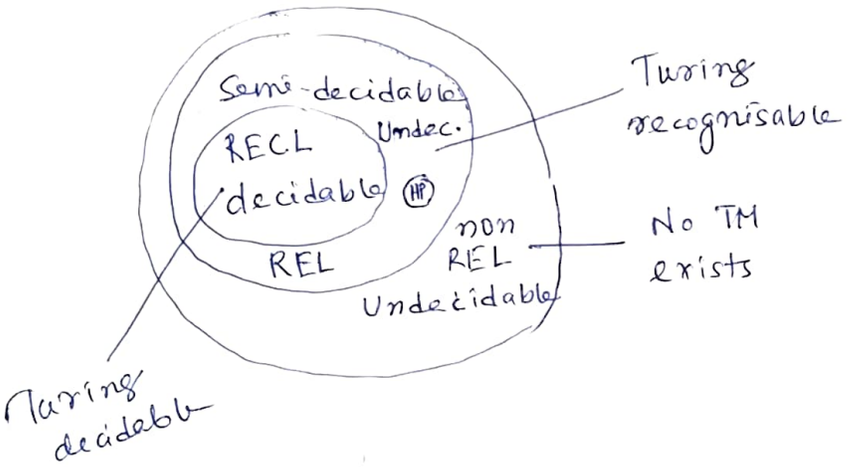
*



Points

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

* Halting set = $\{(M, x) \mid$ TM M 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 ⊂ DCFL ⊂ CFL ⊂ CSL ⊂ RECL ⊂ REL)
* All REL are semi-decidable.



* HP – Semidecidable, undecidable.
* Computer – LBA (finite memory)

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

* Dovetailing (A, B, C, D, E)

A
A B
A B C
A B C D
⋮

* Problems regarding property of TM
  a) TM having finite ~~set~~ states ? [trivial] – prop of TM
     – Recursive, 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 (2-way) • Any non-trivial property of REL is undecidable. (Trivial – property that's always true). – No trivial property is undecidable
  eg. of trivial prop^y : $L = \{\langle M\rangle \mid L(M)$ 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.

eg. of non-trivial prop^y:

non-trivial so, undecidable

i) $L = \{ \langle M \rangle \mid L(M)$ is regular $\}$ — $L(T_{yes}) = a^n$, $L(T_{no}) = a^n b^n$ (so has M that doesn't satisfy property.)

ii) $L = \{ \langle M \rangle \mid L(M)$ is recursive $\}$

iii) $L = \{ \langle M \rangle \mid L(M) = \phi = \{\} \}$ — there are M, for which $L(M) \neq \phi$

iv) $L = \{ \langle M \rangle \mid L(M)$ contains 0 $\}$ - there are M, where $L(M) \not\ni 0$

v) $L = \{ \langle M \rangle \mid$ M accepts at least 10 strings $\}$

↳ corresponds to lang. accepted by the machine M.

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

Non-monotonic - stricter than non-trivial.

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

eg non-monotonic prop^y

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

$L(T_{yes}) = \{1\}$ $L(T_{no}) = \Sigma^*$

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

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

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

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

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

$L(T_{yes})$ = 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^y. But, this is REL. It is non-REL. (Comp. of this (4) is too ✗ non-REL).

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

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

---

* $L = \{ \langle M \rangle \mid L(M)$ 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^n of TM.

* Applying Rice's

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

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

$L = \{\langle M \rangle \mid$ ∃ an ip of length < 100 on which M halts$\}$ REL but not RECL. [finite # ips whose len <100. Run the TM M in interleaved mode (Dovetailing), if ∃ an ip among them for which M halts and that's in the Language. But, in worst case M can loop forever if for none of these ips M halts. So, (RE) not (REC).]

Examples. (Applying Rice's) In form $L = \{\langle M \rangle \mid L(M) \text{ is } \dots\}$

1. $L(M)$ has at least 10 strings. $L(T_{yes}) = \Sigma^*$, $L(T_{no}) = \phi$

So, undecidable. [In fact, RE] Turing recognizable.

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

2. $L(M)$ has at most 10 strings. $L(T_{yes}) = \phi$, $L(T_{no}) = \Sigma^*$

So, undecidable i.e. RE or non-RE.

non-RE $\phi \subset \Sigma^*$ $L(T_y) \not\subseteq L(T_{no})$ 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. $\delta(q, \epsilon) \rightarrow (q', \epsilon, \text{stay})$

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. $L(T_y) = \phi$ $L(T_{no}) = \Sigma^*$

non-turing recognizable or non-RE & undecidable.

$L(T_y) \subset L(T_{no})$ So,

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 $(L(T_y) = \phi) \subset (\text{Any } {}^{L(T_{no})}_{\downarrow} \text{ non RL})$

non-RE, undec.

8. $L(M)$ is not regular $[L(T_y) = \{a^n b^n \mid n > 0\}] \subseteq (L(T_n) = \Sigma^*)$

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.]

10. Checking equivalence of 2 REL is non-REL.

> $L_1 = \{ \langle M \rangle \mid M$ visits state Q on some ip within 10 steps$\}$
Semi-Decidable, RE

> $L_2 = \{ \langle M \rangle \mid |M| < 100$, where $|M|$ is # states in M $\}$ Decidable

> $L = \{ \langle M \rangle \mid M$ is a TM and $\exists$ an input on which M halts in less than $|M|$ steps $\}$
— Decidable, RECL.

> $\{ \langle M \rangle \mid M$ is a TM & $|L(M)| \leq 3 \}$.
$L(T_{yes}) = \{0,1\}$ $L(T_{no}) = \Sigma^* \mid L(T_{yes}) \subset L(T_{no})$
So, non-RE, undecidable.

> $\{ \langle M \rangle \mid |L(M)| = 2 \}$
$L(T_y) = \{0,1\}$ $L(T_n) = \Sigma^* \mid L(T_y) \subset L(T_n)$
non-RE.

> $\{ \langle M \rangle \mid |L(M)| \geq 3 \}$
$L(T_y) = \{0,1,2\}$ $L(T_n) = \phi$
Undecidable

> $\{ \langle M \rangle \mid$ M accepts all even numbers $\}$
$L(T_y) = \{$set of all even numbers$\}$
$L(T_n) = \Sigma^*$ $L(T_y) \subset L(T_n)$
] non-RE. or not even semidecidable

> $\{ \langle M \rangle \mid$ M does not accept all even numbers $\}$

> $\{ \langle M \rangle \mid$ M rejects all even numbers $\}$

> $\{ \langle M \rangle \mid L(M)$ is countable $\}$ Trivial property (= REL)
— Decidable / Recursive

> $\{ \langle M \rangle \mid L(M)$ is uncountable $\}$ Trivial prop^y (= ϕ)
— Decidable / Recursive.

(over finite alphabet & finite length strings)

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

$L(T_{yes}) = L(M_1) \cup L(M_2) = \Sigma^*$
$L(T_{no}) = \{0,1\}$
} RE, undecidable (not Rec.)

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

$L(T_{yes}) = \Sigma^* \quad L(T_{no}) = \phi$. RE, undecidable, (not Recursive)

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

non RE.

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

non REC, RE.

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

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

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

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

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

> $\{\langle k, x, M_1, M_2, \ldots, M_k\rangle \mid k$ is a natural no., x strix, M_i is a TM for all $1 \le i \le k$, & at least $k/2$ TM's of $M_1, \ldots, M_k$ 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

> $\{\langle M\rangle \mid M$ halts on all palindromes$\}$ Non RE.

> $\{\langle M\rangle \mid L(M) \cap \{a^{2^n} \mid n \geq 0\}$ is empty$\}$ Non RE

> $\{\langle M,k\rangle \mid |\{w \in L(M) : w \in a^*b^*\}| \geq k\}$ RE, not REC

> $\{\langle M\rangle \mid L(M) = L'$ for some undecidable $L'\}$ Rec
and M halts on all inputs $= \phi$

> $\{\langle M\rangle \mid M$ accepts at least 2 strings of different lengths$\}$ RE, not Rec.

> $\{\langle M\rangle \mid L(M)$ & $\overline{L(M)}$ both infinite$\}$ non RE.

> $\{\langle M\rangle \mid |L(M)|$ is prime$\}$ Non RE.

> $\{\langle M,x,k\rangle \mid M$ does not halt on x within k steps$\}$
Rec.

> $\{\langle M\rangle \mid \exists x \in \Sigma^*$ s.t. for every $y \in L(M)$, $xy \notin L(M)\}$
- non RE.

> $\{\langle M\rangle \mid \exists x, y \in \Sigma^*$ s.t. either $x \in L(M)$ or $y \notin L(M)\}$
Rec.

> $\{\langle M\rangle \mid \exists$ a TM M' s.t. $\langle M\rangle \neq \langle M'\rangle$ and $L(M) = L(M')\}$
(lang. of all TMs) - Rec.

> $\{\langle M_1, M_2\rangle \mid L(M_1) \leq_m L(M_2)\}$. non-RE.
↑ mapping reducibility.

> $\{\langle M\rangle \mid M$ doesn't accept any string w s.t. 001 is a prefix of $w\}$. non RE.

> $\{\langle M, x\rangle \mid M$ does not accept any string w s.t. x is a prefix of $w\}$ non-RE

> $\{\langle M, x\rangle \mid x$ 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

> $\{\langle M_1\rangle \mid \exists$ 2 TM's M_2, M_3 s.t. $L(M_1) \subseteq L(M_2) \cup L(M_3)$ Rec.

> $\{\langle M, w\rangle \mid M$ is a TM that accepts w using at most $2^{|w|}$ squares of its tape$\}$ Rec.

> L_D (Diagonalization lang.) $= \{\langle M\rangle \mid M$ is a TM & $\langle M\rangle \notin L(M)\}$ non-RE

[Set of descriptions of TMs that don't accept themselves]

> $A_{TM} = L(U_{TM}) = \{\langle M, w\rangle \mid M$ is a TM & M accepts $w\}$ - RE

[Lang. of UTM]

> $\overline{A_{TM}} = \overline{L(U_{TM})} = \{\langle M, w\rangle \mid M$ 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^n 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^n.) ∴
- Complement of non-regulare is always non-regulare.
- DCFL ∪ CFL = CFL

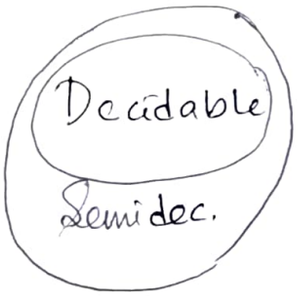
---

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

   REL but not RECL $\rightarrow$ non-REL.

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

   Semidecidable : RE, Turing recognizable, Turing acceptable, Turing computable, Enumerable, Partially decidable.
   (Only valid strings have logic.)



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

RE but not REC Semidecidable but not recursive, SD but not decidable

Non-RE Undec., Not semidecidable, has no TM, valid strings have no logic

Lang
- RE (SD)
  - D (REC) RL/CFL/CSL/RECL
  - undec.
- non-RE
  - undec

→ ~~CFG~~ $L = \{ CFG \mid CFG \text{ is ambiguous} \}$

has ~~Logic for yes~~, no logic for no.

— Semidecidable; RE, but not REC.

CFG, is ambig → there's a string for which ≥2 parse trees exist

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

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.

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

$\bar{L} = (\Sigma^* - L)$ → 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.

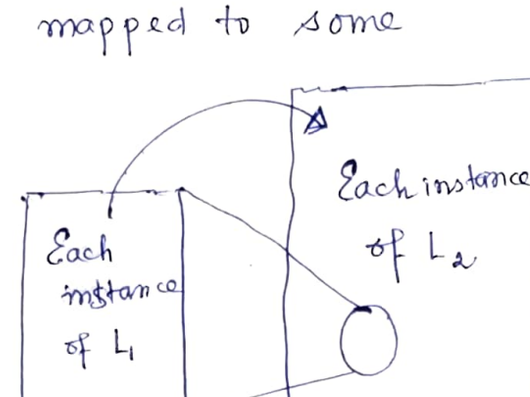
$P_1 \leq P_2$ or $P_1 \propto P_2$

$P_1 \leq_p P_2$ polynomially reducible

P_2 is at least as hard as P_1.

(mapping reducible)

$P_1 \leq P_2$ { 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. (contra +ve)
  - L_2 is REC $\Rightarrow$ L_1 is REC
  - L_1 not REC $\Rightarrow$ L_2 not REC.
  - L_2 or P_2 is Yes. $\Rightarrow$ P_1 is Yes.
    (P_2 is no $\Rightarrow$ X)
  - P_1 is no $\rightarrow$ P_2 is no.

Project (L_2) ↑ module (L_1)

- $L_1 \leq L_2$, $L_2 \leq L_3$
  - > If L_3 is decidable, L_2 & L_1 decidable.
  - > If L_1 is undec, L_2, L_3 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).

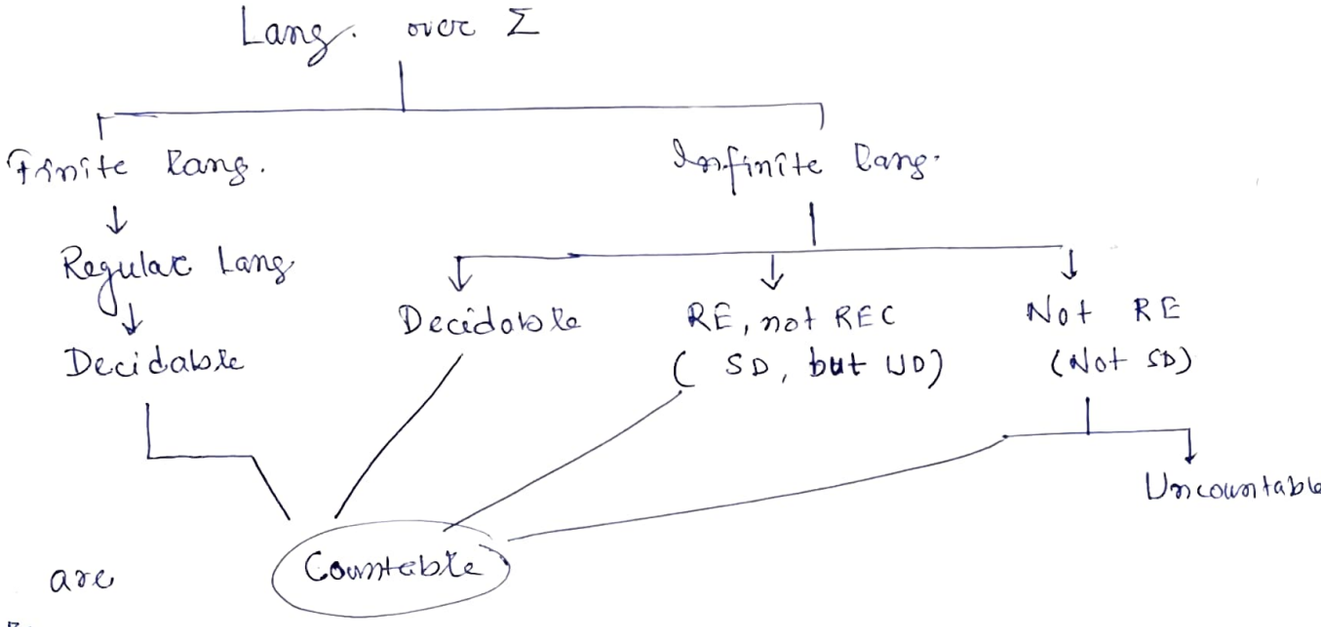
- $P_1 \leq P_2$ & $P_2 \leq P_3$. P_1 decidable, P_3 undec.

  $\Rightarrow P_2$ (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.



All REL are countable.

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

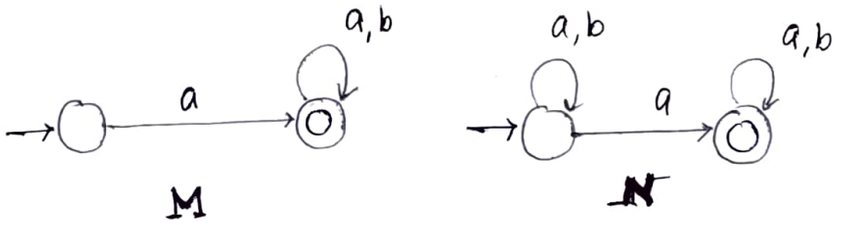
eg. $L = \{ \langle M \rangle \mid M$ is a TM and M does not accept a given string $w \}$

— non-RE and countable

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

— non-RE & uncountable.

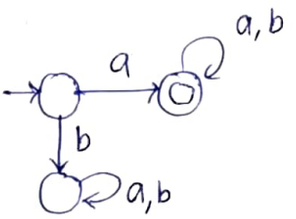
- Σ be a finite alphabet, then
  - > Σ, Σ^*, 2^Σ are countable.
  - > 2^{Σ^*} is uncountable.

Q. 

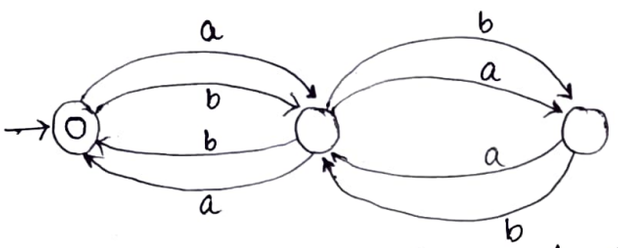
[Minimal DFA]

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

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

$L(H) \cap L(N) = L(M) = a(a+b)^*$ $\xrightarrow[\text{DFA}]{}$ 

Ans = 3.

Q ✓ 

regex ?

[FA → RE]

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

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

Q $L_1 = \{a^n b^n c^m \mid n, m > 0\}$

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

$L_1 \cap L_2 = a^n b^n c^n \mid n > 0$	CSL, ✗ CFL
$L_1 \cup L_2$ =	CFL, ✗ DCFL
$L_1 \cdot L_2$ =	CFL

[Closure properties of CFL]

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

$L_2 = a^n b^n c^m \mid n \neq m \wedge m, n > 0$ [Identificaⁿ of Langs]

→ More than 1 comparisons for both.

✗ CFL, both are CSL.

Q Decidable ? [Decidability] [Closure prop^y]

a) Whether intersec^n of a CFL & a finite regular lang is infinite. ✓

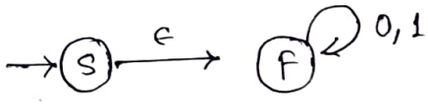
b) Whether CFL is regular. ✗

c) Whether 2 DPDAs accept same lang. ✓

d) Whether given grammar is Context free. ✓

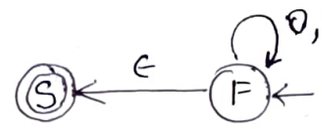
Q. $(a+b)^* b (a+b)^* b (a+b)^*$ [Regex desc^n]

→ Containing at least 2 b's.

Q 

[Reversal, Union, Int^n, Concat, Complement Cross prdt of FA]

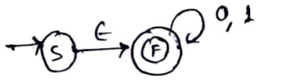
– Reversal of the FA :



both accepts same lang.

$(0+1)^*$

When f is only final



– 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. to minimal dfa has exactly 2 states.

Q Decidable ?

a) $\{\langle M \rangle \mid M$ is a TM & M has 21 states$\}$ ✓

b) $\{\langle M \rangle \mid M$ does not have a halt state (final)$\}$ ✓

c) $\{\langle M, w \rangle \mid M$ accepts w string$\}$ ✗

d) $\{\langle M \rangle \mid L(M)$ is finite$\}$ ✗

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.

✗ $\bar{L}$ must be CSL.

✓ $\bar{L}$ cannot be DCFL.

↳ If DCFL, then

$L = \bar{L}$ is also DCFL as closed under complement for DCFL.

⇒ $\bar{L}$ is not DCFL.

ww^r → complement → all non palindromes

We can make NPDA, so, CFL.

Every CFL → CSL ⇒ Complement- closed for CSL

So, $\bar{L}$ must be CSL.

Q PDA.

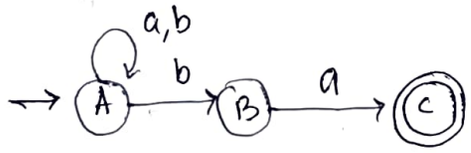
[PDA]

– 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.

→ Check with answers. ✓



$(a+b)^*ba$

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

Q $L_1 = a^n b^n \mid n \geq 0$

$L_2 = a^n b^{2n} \mid n \geq 0$

L^R reverse of L.

$L_1 \cdot L_1^R = a^n b^n \cdot b^m a^m$ ← DCFL [Closure prop^y]

$\bar{L}_2$ ← DCFL [Closure]

$L_1^3 = a^n b^n a^m b^m a^p b^p$ ← DCFL

$L_1 \cup L_2$ ← CFL, not DCFL

Ambiguous but has an eqv. unamb. grammar:

G_1: $S \to S_1 / S_2$

$S_1 \to aS_1b / \epsilon$

$S_2 \to aS_2bb / \epsilon$

grammar ambiguous

$L(G_1) = \underbrace{a^n b^n}_{\text{can't collapse}} \cup \underbrace{a^n b^{2n}}_{\text{can't collapse}}$

$a^n b^n \cap a^n b^{2n} = \phi \neq \infty$

So, we can have an eqv. unamb. grammar.

⇓

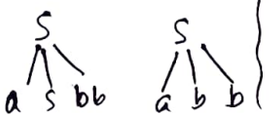
eqv unamb. grammar

$S \to S_1 / S_2 / \epsilon$

$S_1 \to aS_1b / ab$

$S_2 \to aS_2bb / abb.$

G_2: $S \to aSbb / abb / \epsilon$ (Ambiguous grammar)



$L(G_2) = a^n b^{2n}$ ← DCFL [can never have lang ambiguity]

Eqv. unamb. grammar: $S \to aSbb / \epsilon$

G_3:

$S \to S_1C / AS_2$ — $a^n b^n c^m$ / $a^p b^q c^q$

$S_1 \to aS_1b / \epsilon$ — $a^n b^n$

$C \to cC / \epsilon$ — c^m

$S_2 \to bS_2c / \epsilon$ — $b^q c^q$

$A \to aA / \epsilon$ — a^p

$L(G_3) = a^n b^n c^m \cup a^p b^q c^q$

→ 2 parts, can't collapse, ∩ is $a^n b^n c^n \to \infty$ lang

So, lang is ambiguous

⇓

all grammar for $L(G_3)$ is ambiguous.

= Can't have eqv. amb. grammar.

Checking ambiguity

Amb. if —

1. $- \cup -$ ≥ 2 parts
2. parts don't collapse
3. $- \cap - \neq \phi$ = may be ∞

eg $L = (\underbrace{a^n b^n c^m}_{①} \cup \underbrace{a^n b^m c^m}_{②})$

2 parts, can't collapse as ① has $a = b$ & ② has $b = c$. ∩ of them is $a^n b^n c^n \Rightarrow \infty$ strings in the lang.

So, L is ambiguous.

(CFL may or may not have lang ambiguity.)

Q [Regex] $\Sigma = \{0,1\}$ Every 0 is followed by exactly 1 or 2 one's.

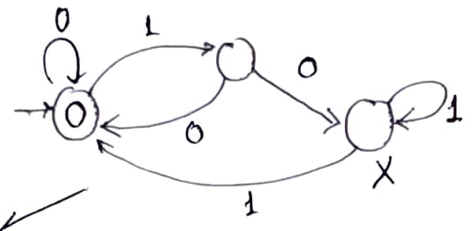
→ a) $(01+011)^*$ 11 can't be generated ✗

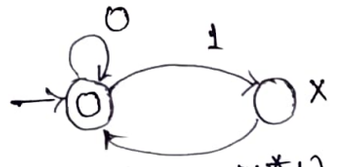
b) $(01+011)^*1^*$ 0111 gets generated ✗

c) $(01+011)^* + 1^*$ 101 doesn't get accepted ✗

d) $1^*(01+011)^*$

Q [Regex] from FA.





$(0+0\epsilon1^*1)$

$= 0(\epsilon + 1^*1)$

$= 01^*$

$(x^*y^*)^*$
$= (x+y)^*$

$0 + 1(0+0\epsilon1^*1)$

$(0 + 101^*)^*$ ← Ans.

also can be written as —

$(0^*(101^*)^*)^*$ **

Q CFL ?

a) Complement of $\{x \in \{a,b\}^* \mid x \text{ is not } ww \text{ for } w \in \{a,b\}^*\}$

XCFL ↳ becomes ww form when complemented

CSL ✓

b) $\{a^i b^{i+k} a^k \mid k \neq i\}$ Not CFL 2 cond^n checking.

c) Compl. of $\{a^n b^n c^m \mid m = n\}$

↳ becomes $\{a^m b^n c^p \mid m \neq n \text{ or } n \neq p\}$

CFL ✓

d) $\{a^n b^{2n} c^{3n} \mid n \geq 1\} - \{a,b,c\}^*$ → becomes ϕ

RL, so CFL ✓

RE but not REC

1. $\{\langle M\rangle \mid 1 \le |L(M)| \le 10\}$
non-trivial . non-RE. by $Rice_2$

2. $\{\langle M\rangle \mid L(M) \cup \overline{L(M)} \text{ is regular}\}$
trivial → Rec.

3. two $\{\langle M\rangle \mid |L(M)| \ge 10\}$
non-trivial RE by $Rice_2$

After deciding, undecidable

Every superset of $L(T_{yes})$ if satisfies the property then it is monotonic property ⇒ the lang is RE, not non-RE

If superset of $L(T_y)$ not satisfies the property then ⇒ non-monotonic .
So, non-RE, not RE.

monotonic → RE
non-mon. → non-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)

d) $L \subseteq \{0,1\}^*$ & L is the lang of all palindromes, then every infinite subset of L is non-regular. (F)
↳ $0^*, 1^*$ also in L. They are regular.

Q CFL ?

a) $\{a^m b^n \mid n = km, k = 0,1,2,\ldots\} = \{a^m b^m \cup a^m b^{2m} \cup \ldots\}$
= Infinite union ✗ CFL

b) $\{a^m b^n \mid n = 3m+2$ if m odd
& $n = 4m+5$ if m even$\}$ ← CFL ✓

c) $\{a^m b^n c^p \mid m < \min(n,p)\}$ to find min ← comparison + another ✗ CFL

d) $\{a^m b^n c^p \mid \gcd(m,n) = p\}$ ✗ CFL.
↳ (CSL)