Theory of Computation : By Reddy Sir ToC ≈ System Design

Syllabus:

*CSE branch famous for logics

1. Finite Automata & Regular Language → [6M] [60% - 70%]
2. Push Down Automata & Content Free Language [30%]
3. Turing Machine & REL → [10%]
4. Undecidability

[ToC ≈ Algorithm ≈ System Design]

[Donot read big stories]

*DM + Probability

ToC Definition:→

It is a mathematical study of computing machine and its capabilities

Formal Languages:→

Collection of strings where the format of string which matters. For example

$L_1 = \{ a^n b^n \mid n \geq 1 \}$

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

$L_3 = \{ a^m b^n \mid m, n \geq 1 \}$

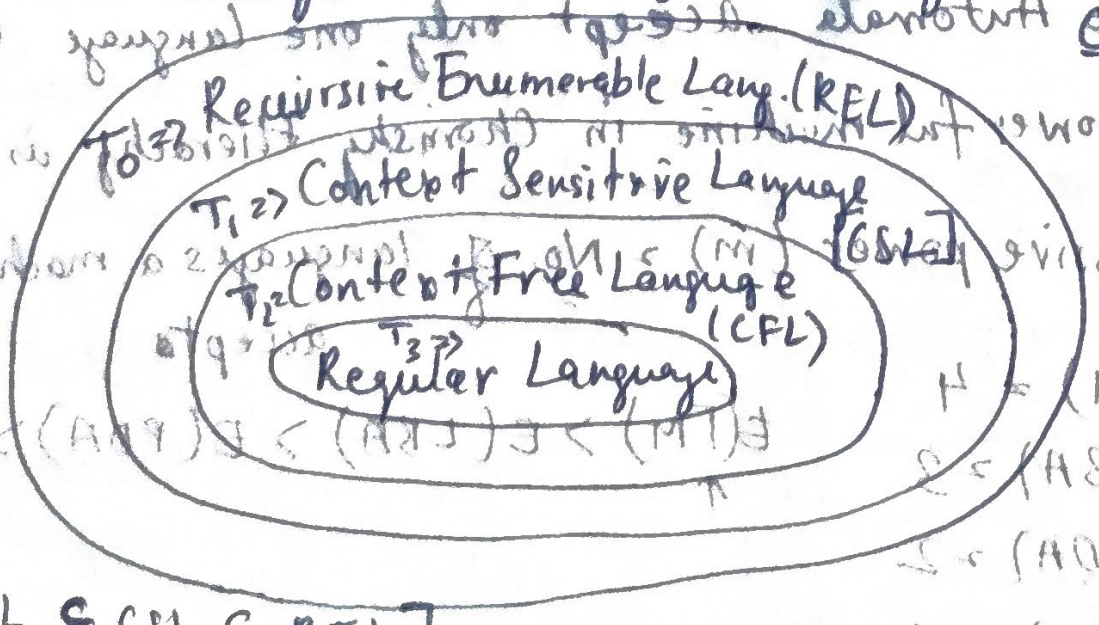
$L_1 = \{ ab, aabb, aaabbb, \ldots \}$

$L_2 = \{ aa, aaa, aaaa, \ldots \}$

In formal language, the meaning of string is not important.

Chomsky Hierarchy:

According to Chomsky, there are 4 types.



[R.L ⊂ CFL ⊂ CSL ⊂ REL]

C programming:
Is context Sensitive Lang.

```
int a;
int A;
```

↑ possible

```
int a;
int a;
```

↑ not possible.

① Regular Languages is accepted by Finite Automata Machine

② Context Free Grammar Language is accepted by Push Down Automata.

③ Context Sensitive Language is accepted by Linear Bounded Automata.

④ Recursive Enummerable Language is accepted by Turing Machine.

• Turing Machine will accept all four Languages.

① Every Regular Language is Context Free Language

② Every Context Free Language is Context Sensitive

③ Every Context Sensitive Language is Recursive Enummerable Language

$A \subseteq B$ [Every A is B] but Every B is A is not

Note: Every B.Tech is 12th Class.
But 12th Class may not be B.Tech.

$RL \subseteq CFL \subseteq CSL \subseteq REL$

10 20 30 40

① Turing Machine accept four language (T_0, T_1, T_2, T_3)

② Lower Bounded Automata accept three Lange (T_1, T_2, T_3)

③ Push Down Automata accept two language (T_2, T_3)

④ Finite Automata accept only one language (T_3)

⇒ Most powerful machine in Chomsky Hierachy is Turing Machine

Def: → Expressive power (m) = No. of languages a machine(m) can accept

E(TM) = 4
E(LBA) = 3
E(PDA) = 2
E(FA) = 1

E(TM) > E(LBA) > E(PDA) > E(FA)

Notes * Every Regular Language is CFL but not vice versa.

Q Why are these so many automatas?

Ans $L_1 = \{ a^n \mid n \geq 1 \}$

$\{ a, aa, aaa.. \}$

$a^1 = a$ → minimal length 1

$a^2 = a^2$

$a^3 = a^3$

$a^4 = a^4$

⋮

$a^n = \ldots$ → maximal (as n can take any value)

Hence → This is known as Infinite Language.

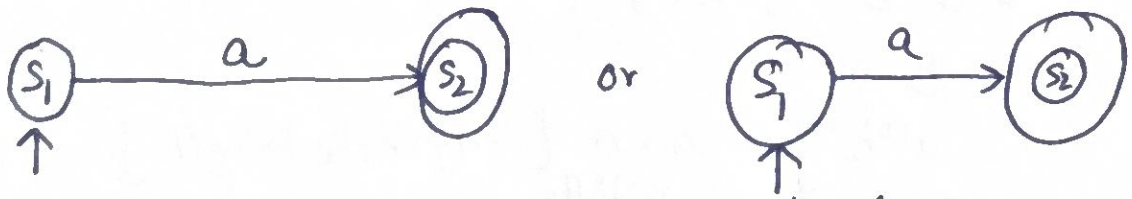
$L_1 = \{ a^n \mid 1 \leq n \leq 100 \}$

minimal string => a^1

maximal string => a^{100}

$L_1 = \{ a^n \mid n \geq 1 \}$ => a, aa, aaa, ----

↳ minimal string length is one.

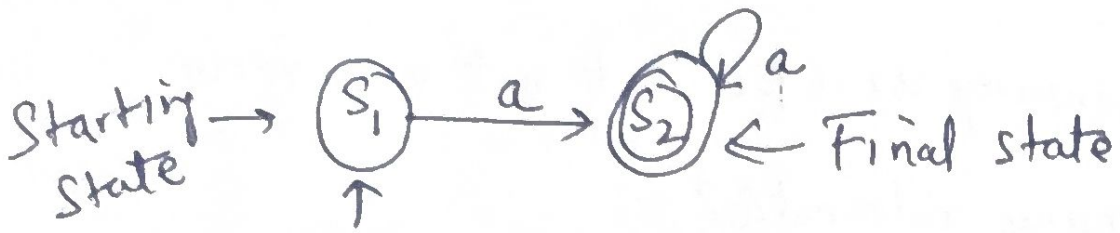


Steps : ① Total number of States will be :- The sum of length of minimal string and one.

a => 1

Total states => 1 + 1 => [2]

You can read it :- From starting state 1, by accepting one a you can reach the final state.



It will accept → {a, aa, aaa, ---- }

→ a ⟲ a => will accept all the strings starting with a, ~~after~~

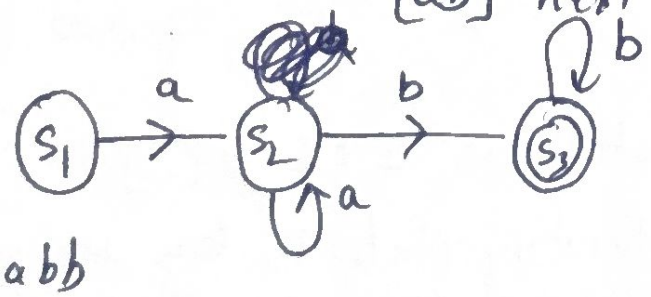
In final state : The string is accepted.
other than this, the string is not accepted.

① Every circle is a state or function in a program.
② From one state to another, we say transition.
③ The name of the diagram Finite State Machine or Finite Automata (FSM/FA)
④ Every circle is a function.

$L_2 = \{ a^n b^n \mid n \geq 1 \}$

ab ∈ ab, n = 1 [minimal string] [minimal length is 2]
a^2b^2 ∈ aabb n = 2
a^3b^3 ∈ aaabbb, n = 3
→ = a^4b^4, n = 4
⇓
$a^n b^n$ = n = n [infinite length]

Steps
① ~~Find~~ number of state initially 2) 2 + 1 => 3



How to do ① aabb

But it accept this also
① a^3b
② ab^3 m ≠ n also

Hence for this language, FA is not possible.

* $L_2 = \{a^n b^n \mid n > 1\}$

is not a regular language.

Hence, Finite State Machine not possible.

Imp Between two or more variables, then FSM is ~~would~~ for that language is not possible, as there is no way to keep track of number of occurence of each variable.

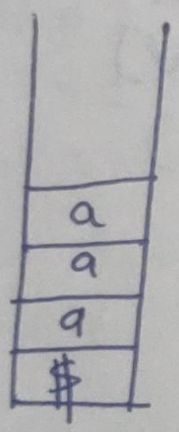
Solution: Check for Push Down Automata., In PDA, you add a stack.

$L_2 = \{a^n b^n \mid n > 1\}$

Let $n = 3$, aaabbb$ => Machine will know the string is over due $ sign.

↑ end will be $

$$\not{a}\not{a}\not{a}bbb\$$$



① aaabbb$

② aaabbb$

③ aaabbb$

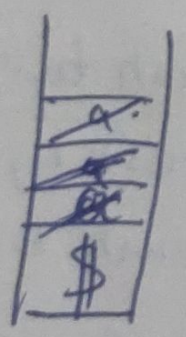
④ aaabbb$

a

① For every a, we push in the stack.

② For every b, we pop the stack.

Now, we started seeing b, we pop "a" from now onwards.



⑤ aaabbb$

⑥ aaabbb$

⑦ aaabbb$

in string ↑

① By seeing $ sign, the PDA knows that string is over.

② The stack is also empty.

For accepting a string
PDA :-
① The string must reach at the $ and the stack must be empty.

[redacted]p :

PDA :→ Is just a stack., with FSM where comparison between two different variables is needed..

$\{a^n b^n$ form$\}$ $[a=b]$

- $L_3 = \{a^n b^n c^n \mid n \geq 1\}$ => abc, $a^2b^2c^2$, $a^3b^3c^3$, ----

Ques To this language, when we say a machine is correct?
Ans The machine should only accept the strings mentioned in [redacted] formal language.

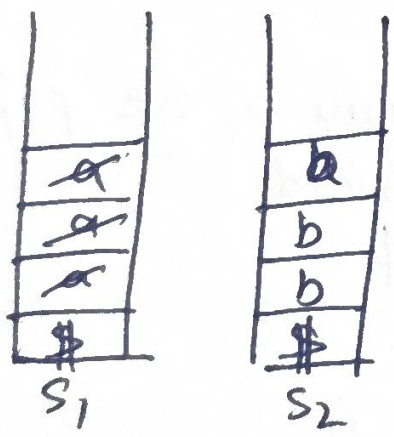
Comparison?, FAX

Number of a = Number of b = Number of c.

More than one comparison? [redacted] PDA X
✓ Linear Bounded Automaton (Theortical)
✓ Turing Machine (Practical purpose)

Turing Machine :→ [redacted] FA + two stacks.

a a a b b b c c c $
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ #



① a will in go stack 1
② b will be go in stack (2)

① push a in S_1 stack
② push a in S_1 stack
③ push a " " "
④ push b, pop a
⑤ push b, pop a
⑥ push b, pop a
⑦ see c, pop b
⑧ see c, pop b
⑨ see c, pop b
⑩ S_2 & S_1 empty, and string is empty

Hence, string is accepted

* FA + 2 stack or FA + 1 queue ⇒ Turing Machine

Lecture:2

Alphabet: Alphabet is a finite, non-empty set of symbols.

* Every language has a finite, non empty set of symbols.

Alphabet (Σ)

Σ of English Language = { a, b, c ..... z }

Σ of Hindi Language = { अ, आ, ... ज्ञ }

Σ of telegu Language = { అ, ఆ ..... ఔ }

String:→ Collection of characters over the given alphabet.

$\Sigma = \{a, b\}$ ⇒ ab, a, b, aaa, bbbaabb ✓

ab12ab ✗

|ab| = 2 , |aaa| = 3

|a| = 1 |bbbaabb| = 7

|b| = 1

Q How many strings possible for length 4?

Ans $2^4 = 16$

Q Max of length possible over $\Sigma = \{a, b\}$ is

Ans 2^n or ∞

Q Min length possible over $\Sigma = \{a, b\}$ is

Ans Epsilon (Special case in TOC), ~~zero~~ string of zero length.

"ϵ" Epsilon ⇒ It is a zero length string.

$|ab| = 2$ $\quad |cde| = 3$ $\quad \Rightarrow |ab| \cdot |\epsilon| \cdot |ab| = 2$ ($|ab| = 2$, $|\epsilon| = 0$, $|ab| = 2$)

$|ab \cdot cde| = 5$ $\quad |\epsilon| \cdot |ab| = |ab| = 2$ ($|\epsilon| = 0$, $|ab| = 2$)

↑ * concatenation (+)

$|\epsilon| \cdot |ab| \cdot |\epsilon| = |ab|$ ($0 \quad 2 \quad 0 \quad 2$)

Prefix :→ set of all leading symbols over the given string is known as pre-fix.

Be careful

ex: $\epsilon \cdot abc = abc$ ~~$\epsilon \cdot abc = abc$~~

↑ ϵ

If you want to take a => $\epsilon \cdot a = a$

" " " " " b ⇒ $\epsilon ab \Rightarrow ab$

" " " " " c ⇒ $\epsilon abc \Rightarrow abc$

Q How many prefixes are possible for a string of length 3?

Ans $1 + 3$ $\quad abc \Rightarrow \epsilon, a, ab, abc$

Suffix :→ Set of all tailing symbols over the given string.

Ex. $abc \cdot \epsilon =$

If you want to take c ⇒ $c \cdot \epsilon$ (ϵ)

" " " " b ⇒ $bc \cdot \epsilon$

" " " " a ⇒ $abc \cdot \epsilon$

Read from the right side.

4 possible $(1 + 3)$ (↑ ϵ)

* ⇒ Epsilon is the first symbol in suffix and prefix.

Substring :→

abc	abcd
ϵ => 1	ϵ 0 => 1
a, b, c => 3	a, b, c, d => 4
ab, bc => 2	ab, bc, cd => 3
abc => 1	~~a, b~~ abc, bcd => 2
	abcd => 1

Number of substring possible = $\left[\frac{n(n+1)}{2} + 1\right]$

No. of sub-strings for the given n-length string :→

=> $\underbrace{1+2+3+\ldots+n}_{} + 1$

=> $\frac{n(n+1)}{2} + 1$ or $\Sigma(n) + 1$

[Zero length ϵ]

How many prefixes possible for n-length => n+1

" " suffixes possible for n-length : n+1

Imp. statement

$\Sigma = \{a, b\}$ another way of writing => a + b

↑ Ex-OR operator

of two string.

① In concatenation, the order of strings matter and both are compulsory to construct a new string => a OR b

a and b can be any ~~diff~~ string.

a.b => a ✗

a.b => b ✗

a.b => a.b ✓

a.b => b.a ✗

② Union, or OR operator, only one of the string can be taken.

a + b => a ✓

a + b => b ✓

a + b => ab ✗

a + b => ba ✗

⊙ OR operator will only give the either one of the strings as output.

More example :→ ab+cd (either take one of the either)

ab+cd ⇒ ab ✓

ab+cd ⇒ cd ✓

ab+cd ⇒ ✗ abcd ✗

ab+cd ⇒ cd + ab ✗

Power of An Alphabet :→

Imp: → $\Sigma^2 = (a+b)^2 \Rightarrow (a+b) \cdot (a+b) \Rightarrow 2^2 \Rightarrow$ 4 possibility.

(· ↑ Concatenation)

$\Sigma^3 = (a+b)^3 \Rightarrow (a+b).(a+b).(a+b)$ or $\Sigma \cdot \Sigma \cdot \Sigma$

(↑ Concatenation, ↑ Concatenation)

Take either a or b: a . a . a ⇒ a a a

b . b . b ⇒ b b b

a . a . b ⇒ a a b

(a+b).(a+b).(a+b)

2.2.2 ⇒ 8 possible of string length 3.

(a+b).(a+b) ⇒ aa, ab, bb, ba } ⇒ 4. (concatenation)

(↑ In this)

* Remember epsilon is not possible, if concatenation is done.

$\Sigma^5 = (a+b)^5 \Rightarrow (a+b).(a+b).(a+b).(a+b).(a+b)$

aaaaa, bbbbb } $2^5 \Rightarrow$ 32 strings possible.

Important

$\Sigma^0 = (a+b)^0 \Rightarrow$ zero times, donot take any alphabet/string.

↳ "ϵ" Epsilon.

Σ^* ← star

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \dots\dots \Sigma^{100} \cup \Sigma^{101} \dots\dots$

Σ^0 ↓ ∈ ↑ zero length

Σ^1 ↓ a, b

Σ^2 aa, ab, ba, bb

Σ^3 aaa, bbb, ...

Σ^{101} d → a ... a (100 length)

⇒ set of all strings over Σ including ϵ.

Σ^* ⇒ "*" known "Kleene Closure".

These operators in TOC :→

1. Union or OR (+)
2. Concatenation (.)
3. Kleene Closure (*)

Σ^+ ⇒ positive closure [excluding ↙ important Σ^0]

$\Sigma^* = \Sigma^0 \cup \Sigma^+$

↑ positive closure over alphabet

Σ^4 ⇒ length of string will be 4.

Σ^{25} ⇒ " " " " " 25.

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

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

Can I get 75 "a" followed by 75 "b"s.

from first "75" (a+b) pick "a"s.

then the next 75 (a+b) pick "b".

a a ... a . b ... b

75 75

Imp:→

$L \subseteq \Sigma^* = \{ (a+b) \}$

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

or

$(a+b+c+\dots+z)$

$\Sigma^* = (a+b+c \dots +z)^*$

⇓

We can construct string of length.

"apple" => 5

the above will give invalid and valid english strings.

So to take only valid string from Σ^*, we take subset of Σ^*.

$L \subseteq \Sigma^*$

⇓

→ Language: A language is a subset of Kleene Closure of Σ (alphabet). or Σ^*.

OR

It is a subset of Σ^*.

$\Sigma = \{\epsilon\}$ Σ^* also known as Universal language over alphabet.

$\Sigma^{23} = (\Sigma)^{23} \Rightarrow \epsilon . \epsilon . \epsilon . \epsilon \dots \epsilon = \epsilon$

"Length of 0" not 23.

$\Rightarrow$

$\Sigma^0 = (\epsilon)^0 \Rightarrow \epsilon$

$\Sigma^5 = (\epsilon)^5 \Rightarrow \epsilon . \epsilon . \epsilon . \epsilon . \epsilon \Rightarrow \epsilon$

"length of 0" not 5.

Σ (sigma) will decide the length.

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

$\Sigma^5 = (\epsilon + a + b)^5 = (\epsilon + a + b) \cdot (\epsilon + a + b) \cdot (\epsilon + a + b) \cdot \ldots$

will give 3 choices because of epsilon.

First string $\Rightarrow \epsilon \cdot \epsilon \cdot \epsilon \cdot \epsilon \cdot \epsilon \Rightarrow \epsilon$

Single "a" $\Rightarrow a \cdot \epsilon \cdot \epsilon \cdot \epsilon \cdot \epsilon \cdot \epsilon \Rightarrow a$

Single "b" $\Rightarrow b \cdot \epsilon \cdot \epsilon \cdot \epsilon \cdot \epsilon \cdot \epsilon \Rightarrow b$

aa $\Rightarrow a \cdot a \cdot \epsilon \cdot \epsilon \cdot \epsilon \cdot \epsilon \Rightarrow aa$

aaa $\Rightarrow a \cdot a \cdot a \cdot \epsilon \cdot \epsilon \Rightarrow aaa$

five 'a's $\Rightarrow a \cdot a \cdot a \cdot a \cdot a \Rightarrow aaaaa$

From zero length to five length is possible.

Next Q. → $\Sigma = \{aa, bb, ab, ba\} = aa + bb + ab + ba$

$\Sigma^5 = (aa + bb + ab + ba)^5$ [observe, we have take length of 2.]

aa bb ab ba $\Rightarrow$ 10 length

aa.aa.aa.aa.aa $\Rightarrow a^{10}$

As we do not have epsilon, the length will be $2 * 5 \Rightarrow 10$ length

Note → You cannot say about the length of string without seeing alphabet.

Ques Σ^5 will generate length of zero ? without seeing alphabet.

Ans No, we need alphabets.

Chapter-1: Finite Automata & Regular Languages

Finite Automata is of two types :→

- Without o/p
  - further divided into two
    - DFA
    - NFA
- with output two types.
  - Moore Machine
  - Mealy Machine

Finite Automata (without output) :→

$$FA = M = (Q, \Sigma, \delta, S, F)$$

① Q → finite set of states

② Σ → set of characters which are taken as input.

③ δ → transition function : A function which take some state in Q and a alphabet to give some output which is a state in Q.
Or mapping from Q, Σ to some output.

loop

$$\delta : (Q, \Sigma) \rightarrow Q$$

Input Output

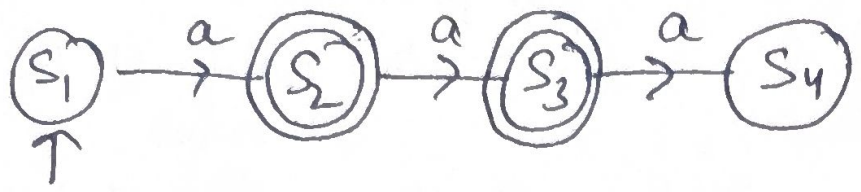
$$\delta(S_1, a) \rightarrow S_2$$

④ S (Start State)

⑤ F : → A set of final states. It can be empty, one or two or so on.

- No. of goals a human can achieve.

Finite Automata also known as Finite State Machines.



Q : Set of states $\{S_1, S_2, S_3, S_4\}$

or {set of final state, initial state and non final state}

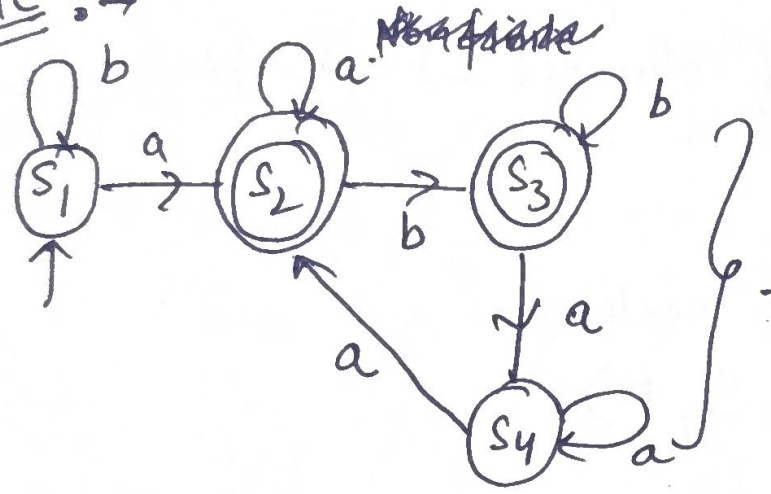
Using above: $\{ S_1 , \underbrace{S_2 , S_3}_{\text{Final State}} , S_4 \}$

S_1 : Initial or Starting State

S_4 : Non final State

[Remember this is a set, so duplicate states cannot be mentioned]

Example :→



→ * This is known as Transition Diagram

$Q \rightarrow \{S_1, S_2, S_3, S_4\}$ order doesn't matter

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

$\delta \rightarrow \delta(Q, \Sigma) \rightarrow Q$

$S \rightarrow S_1$ (only one element allowed)

$F \rightarrow S_2, S_3$ (set of final States)

$\delta(S_1, a) \rightarrow S_2$

$\delta(S_1, b) \rightarrow S_1$

$\delta(S_2, b) \rightarrow S_3$

$\delta(S_2, a) \rightarrow S_2$

$\delta(S_3, a) \rightarrow S_4$

$\delta(S_3, b) \rightarrow S_3$

$\delta(S_4, a) \rightarrow a S_4$

$\delta(S_4, b) \rightarrow S_2$

δ	a	b
→ S_1	S_2	S_1
* S_2	S_2	S_3
* S_3	S_4	S_3
S_4	S_2	S_4

For previous transition diagram

① How many states are there?
4 rows

② How many alphabet are then?
2 columns

A table of 4 × 2

[For initial state mark with "→"
final state mark with "*"

* This is known as transition table.
a type of Data structure.

Hence, there are two ways to represent transitions:→

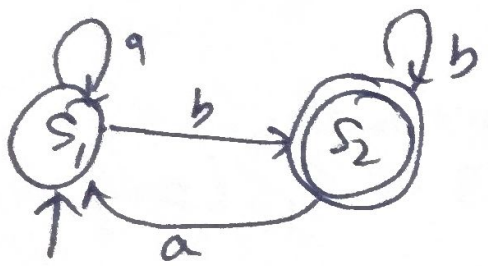
① Transition table → no. of columns (Σ) → {a, b} 2

② Transition diagram.

- A FA is a five tuple machine.

$$M = (Q, \Sigma, \delta, S, F)$$

M_1

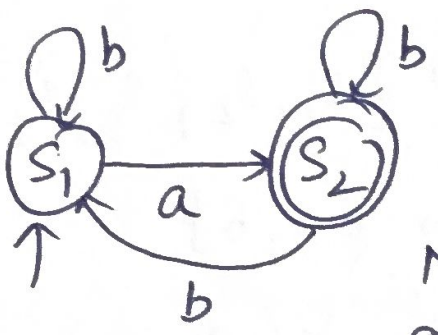


DFA

① One choice

δ	a	b
→ S_1	S_1	S_2
* S_2	S_1	S_2

M_2



NFA

① Dilemma is there or no.

δ	a	b
→ S_1	S_2	S_1
* S_2		(S_1, S_2)

↓

two states

Diagram is used by Computer Scientist
Tables is for Computer/Machine

* Deterministic Finite Automata : In a given Finite automata from each & every state and for every input symbol exactly one transition.

Check then it is DFA.

* Non-Deterministic FA → In a given FA, from each and every state and for every input symbol exactly one transition, or any number of transition.

Note : A special type of NFA is known as DFA. Check M_2 diagram (Pg 16) and table.

Every DFA is NFA but not vice versa.

For M_1 is DFA hence also NFA

For M_2 is NFA but not DFA.

* NFA is anything. Some NFA don't have special type means → In transition table, exactly one transitions.

δ	a	b
→ S_1	S_2	S_1
* S_2		(S_1, S_2)

NFA

=> For each input, one, two, null state.

δ	a	b
→ S_1	S_1	S_2
* S_2	S_1	S_2

DFA => For each input only one state.

Note : Construct Finite Automata mean generally mean NFA.

DFA construction :→

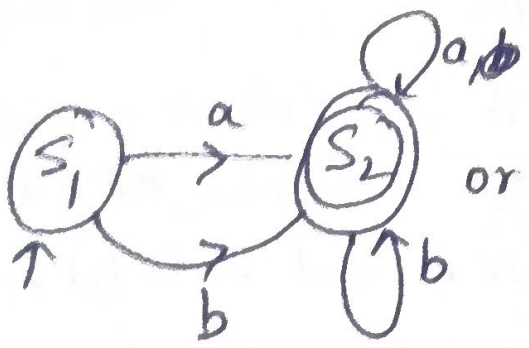
① Construct minimal DFA to accept all string of a's and b's. where ends including "ϵ".

→ L = { a, b, aa, ab, ba, bb, ... , ... }
1 1 2 2 3

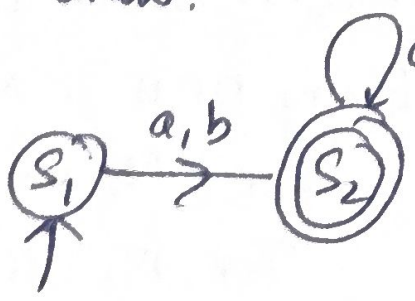
Length is not mentioned ∞.

① Construct the language first [what strcomes in the lang.], write the strings in increasing length.

② If minimal length is 1 then → 1+1 → 2 ↑ add just 1. [n+1] General formal then it given n+1 states to draw. ↑ length of minimal string.



or



	a	b
S_1	S_2	S_2
S_2	S_2	S_2

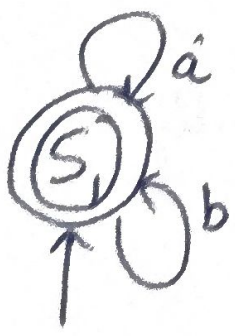
} table

Note $\delta(S_1, \epsilon) \rightarrow S_1$

Ques 2 : Construct minimal DFA that accepts all strings of a's and b's including ϵ.

Construct :→ L = { ϵ, a, b, aa, bb, ab, ba, ----, ----, ... }
0 1 2 4 5

Minimal length is zero → 0+1 → Number of states.



By default, ϵ is accepted. without reading anything.

Q How can you do that ?

And Just make the initial state as Final state to accept ϵ epsilon.

For previous problem:

$\Sigma^* = (a+b)^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots \Sigma^{100} \cup \ldots\ldots \Sigma^\infty$

$\Downarrow \quad \Downarrow \quad \Downarrow$

$\in \quad a, b \quad aa, ab, ba, bb$

$(a+b)^*$ → In CS it is known as Regular Expression.

Note → Gurrantee: If R.E is there then DFA possible AND DFA possible then RE possible

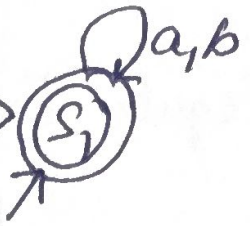
if R.E $\Leftrightarrow$ DFA

~~of RE~~

$L = \{\underset{0}{\epsilon}, \underset{1}{a, b}, \underset{2}{ab, aa, bb}, \ldots\}$ this is known as Regular Language.

$(a+b)^*$ is ~~also~~ Regular Expression.

is DFA $L_1 \Rightarrow (a+b)^* \Leftrightarrow$



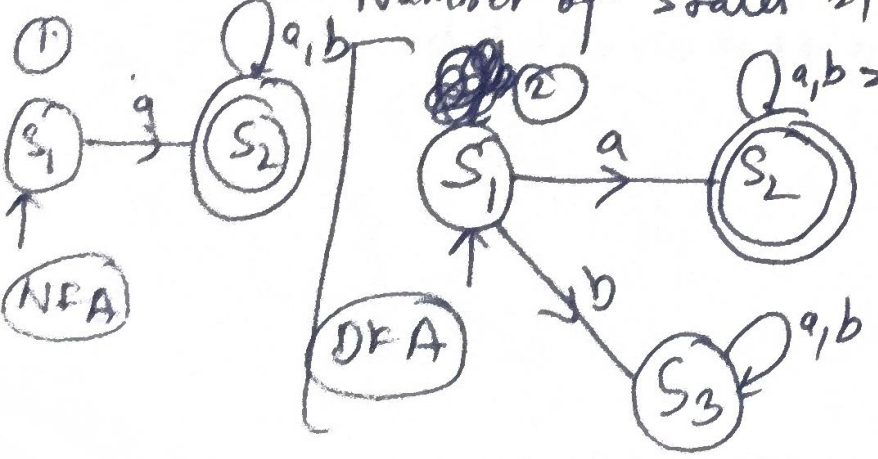
For first problem :- $(a+b)^+$ this +, exclude ϵ.

Imp : To prove L_1 is regular, ~~prove~~ construct DFA or NFA or R.E

3rd : DFA that accepts all strings a's and b's where each string starts with a. $\Sigma = \{a, b\}$

$L_1 = \{\underset{1}{a}, \underset{2}{ab, ab}, aab, aba, abb, \ldots\ldots\}$

Number of states = length of first minimal string + 1



1 + 1 → 2

δ	a	b
→ S_1	S_2	S_3
* S_2	S_2	S_2
S_3	S_3	S_3

NFA

δ	a	b
S_1	S_2	
S_2	S_2	S_2

- S_3 is known as Dead state. From this state, you cannot reach to final state.

In hospital, if there is no hope then take.

* when you got to know there is no hope.

⊗ Non-final state :→ S_1 and S_3 [rejected string]

For S_1, Difference is from S_3 is that there is a hope to accept string odd of length more than 1.

For this DFA, 3 state is req.

For this N-FA, 2 state is req.

For FA, 2 states is req. as NFA ≅ FA.

Regular Expression of above language :→

R.E(L_1) => $a \cdot (a+b)^*$ or $a(a+b)^*$ ↳ default concatenation

↳ this will generate all the strings of length from zero to ∞.

We need to start with a

$a \cdot (a+b)^0 = a \in = a$

$a \cdot (a+b)^1 = aa, ab$

$a \cdot (a+b)^2 = aaa, a$

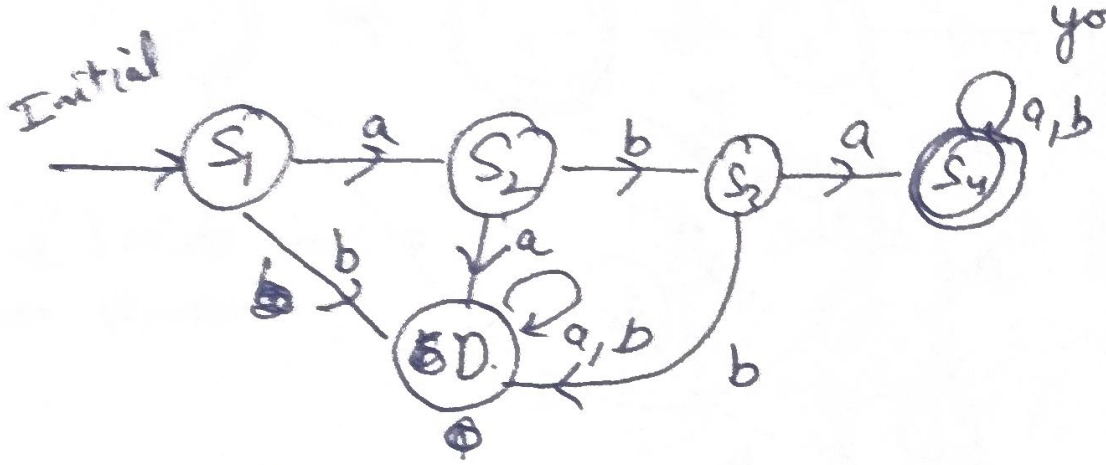
$a \cdot (a+b)^2 \cdot aab$

1 2 length 3

$a \cdot (a+b)^3 \Rightarrow a.bbb$

Construct minimal DFA that accepts all strings of a's and b's where every string starts with aba.

Note :→ For constructing DFA, the number of minimal state is $n+1$
minimal
↳ string of no length.
But for constructing further, you may required more states.



$L_1 = \{ \in, a, b, aa, ab, aba, aaa, abb \ldots \}$
null string is no accept

$\{ aba \ldots \}$

Minimal string => aba + 1
3 + 1 => 4 (start with four states)

Regular Expression => $aba \cdot (a+b)^*$

Temprory non-final → "S_1, S_2, S_3"

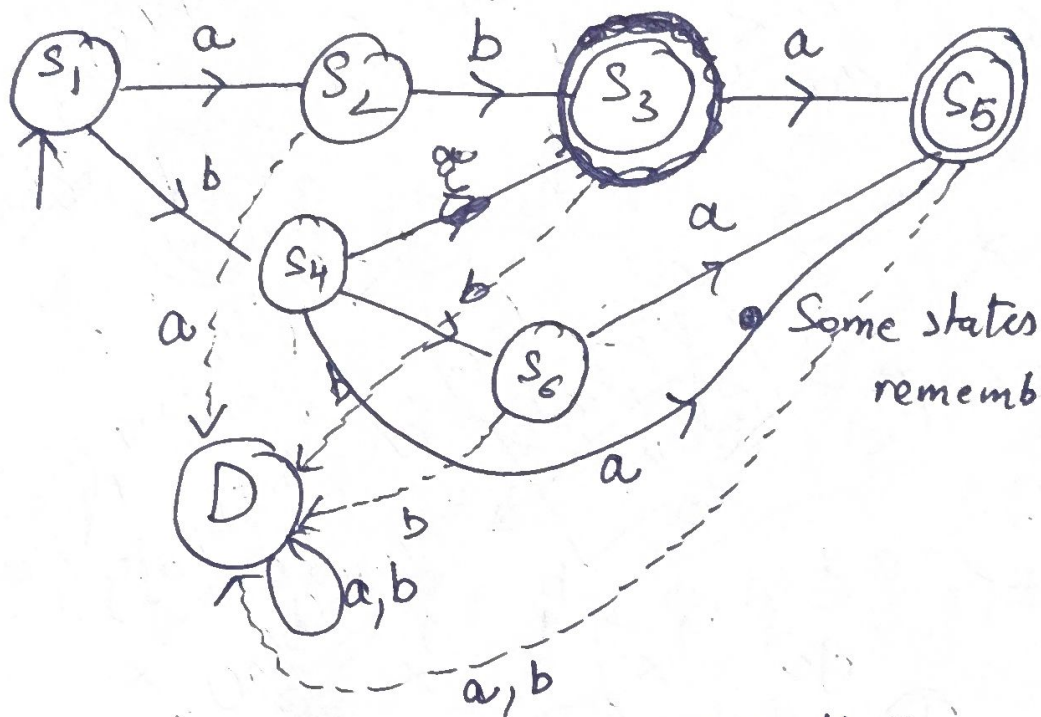
Permanent non-final → "D"

(*) Construct minimal DFA for the language $L = \{ab, aba, ba, bba\}$

Step1 → Construct language

Step2 → Check the minimal length string in here ab

Step 3 → At starting point, the number of initial states = 2+1 => 3



Step4 In Final, we have seven states. (Is their any algorithm)

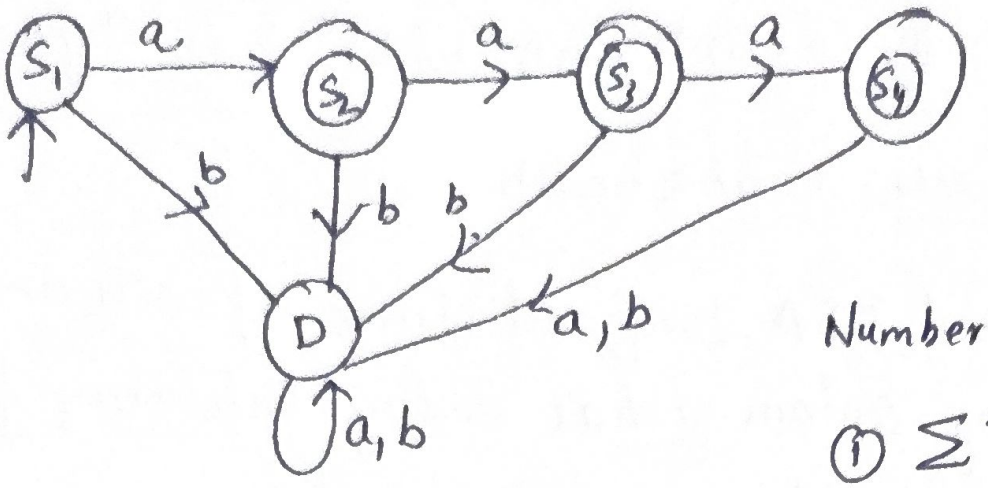
Note :→

For the given language, minimal DFA is unique. but DFA may not be unique.

For the finite language, DFA have to cover Σ^* or $(a+b)^*$ so dead state is compulsory. $\Sigma^* - \{ab, aba, ba, bba\}$, this many strings you have to reject. as DFA will have to accept Σ^*.

If you $\{aa, abb\}$.

Construct M-DFA $L = \{a, aa, aaa\}$ $\Sigma = \{a, b\}$



for DFA (5 states)

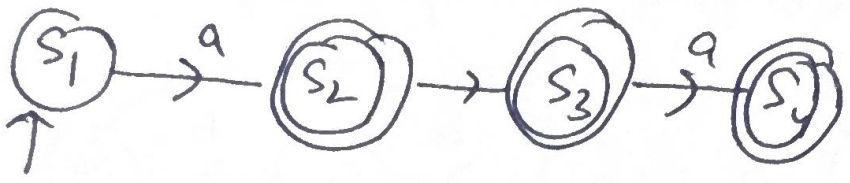
Number of strings rejected

① $\Sigma^* - \{a, aa, aaa\}$

or

② $\Sigma^* - L$

for NFA (4 states)



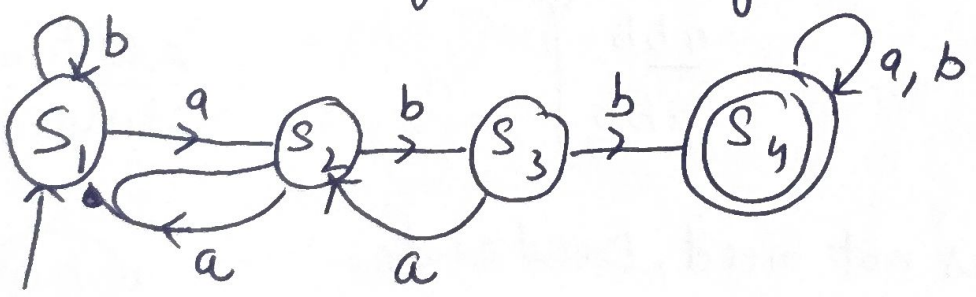
Construct minimal DFA for the language L = {all strings a's and b's where each string abb as substring.

① Construct language. $L = (a+b)^*.abb.(a+b)^*$

$\in .abb. \in \Rightarrow abb$

$a.abb. \in \Rightarrow aabb$

② minimum state required initially :> 3+1 => 4 states.



Check this:

{ aab, abb, aba } Edge cases

abb → pattern match.

ab|a ← pattern damaged where should we go?

As if we have seen once again 'a' you should go to S_2.

(3-1) => (2) to S_2.

~~{ (a+b)* abb (a+b)* }~~

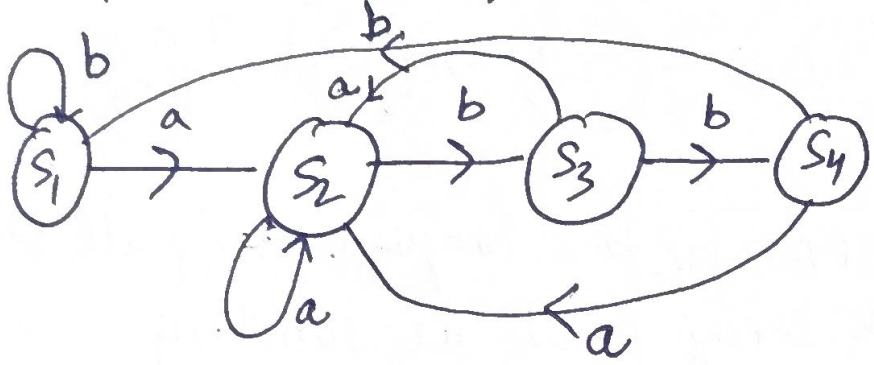
= (a+b)* − { (a+b)* . abb . (a+b)* } → Rejected strings.

Case :- aba babb.

Construct minimal DFA L = { all strings of a's and b's where each string contain abb as ending substring } or ends with abb.

L_1 = { abb , (a+b)* abb }

① Minimum states without initial/starting phase :- 3+1 = 4.



① abb
② abbb
additional b
abbb → useless.

Check usefulness :→ abb
abb
abb

* Ending does not need, Dead state.

Check strings
① abb
② aabb
③ abbabb
④ bbb abbb
⑤ aba abb

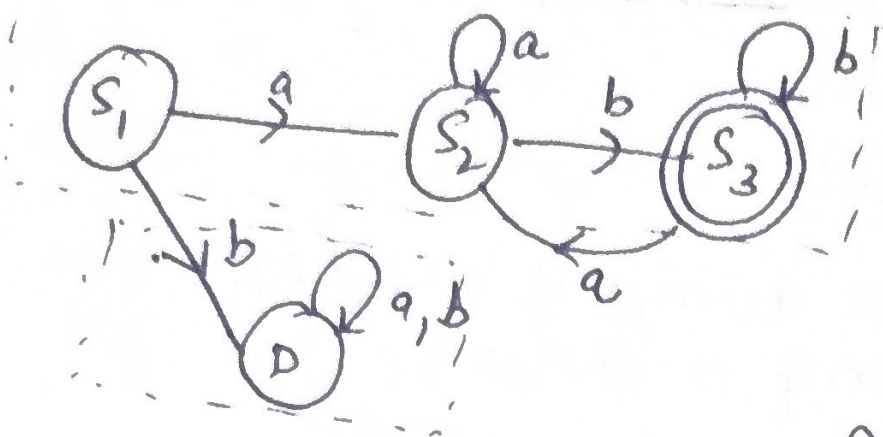
Rejected string
① a, aa, bb.
b, ab, ba
② aaa, bbb
aba, bba

Additional strings ③ abba, abbb

abbb
damaged as whole string is damaged

Ques Construct minimal DFA, L → of all strings of a's and b's where each string start with "a" and end with "b".

L_1 $L_1 = \{ ab, a.(a+b)^*.b \}$



$\Sigma^* - L$

① ab => S_3

② abb => S_3

③ abba => wait S_2

④ abbb F

Strings to watch e→

① a^*, b^*

② ab^*, aa^*, bb^*, ba^*

③ aab, aaabb, abb,

④ bbb, bba.

Cases

⑤ aba.

⑥ abababab

$a\,a^{10}b^{10}b$ => $a^{11}b^{11}$

Carefully :→

$aa^n.b^nb$

① abbb

② abbbba

Ques Construct minimal DFA that accepts all strings of a's and b's where each string of length atmost three.

$L_1 = \{ \underset{0}{\epsilon}, \underset{1}{a}, \underset{1}{b}, \underset{4}{aa}, \underset{8}{aab, bbb} \}$ or $(a+b)^0, (a+b)^1, (a+b)^2, (a+b)^3$

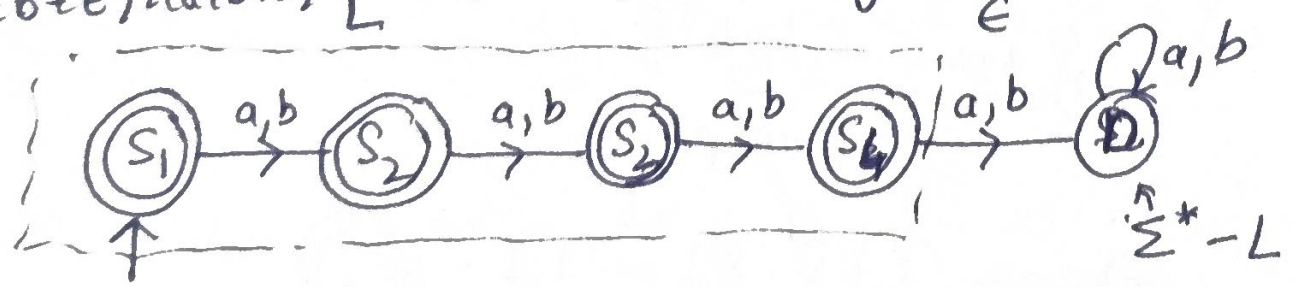
0+2+4+8 => 14+1 => 15 strings.

① As it is Finite language, dead state is compulsory.

$\Rightarrow \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3$

$\Sigma^0 = \epsilon$

RoE => $(a+b+\epsilon)^3 = (a+b+\epsilon).(a+b+\epsilon).(a+b+\epsilon)$



$\Sigma^* - L$

② Please, see, don't introduce loops as it is finite except dead state.

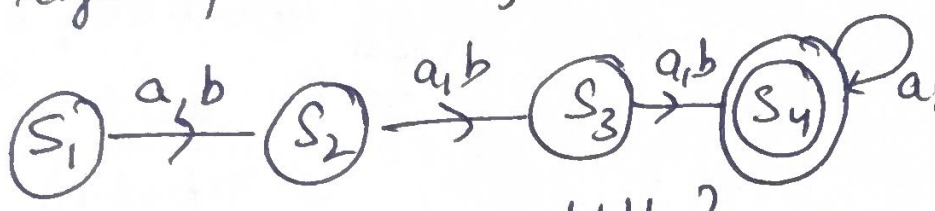
Total rejected strings → $\underbrace{\Sigma^*}_{\text{or } \Sigma^* - L_1} - \{ \underset{0}{\epsilon}, \underset{1\ 1}{a+b}, \underset{2}{aa}, \underset{8}{aaa \ldots bbb} \}$

Note: initiall minimal state => atmost + 1 => 4 state.
3

* On same note atmost 100 => 100+1 => 101 states initially.

Minimal DFA which accepts atmost n-length string contains n+2 states.
or (n+1+1) states
↑
Dead state

Ques Minimal DFA which accepts a strings of a's and b's where the length of the string is atleast 3.



RE = $(a+b)^3 \cdot (a+b+\epsilon)^*$

$L_1 = \{ \underbrace{aaa, bbb}_{8} + \underbrace{aaaa \cdots bbbb}_{16 \cdots \infty}, \}$ or $X = \Sigma^3 \cup \Sigma^4 \cup \Sigma^5 \cdots$

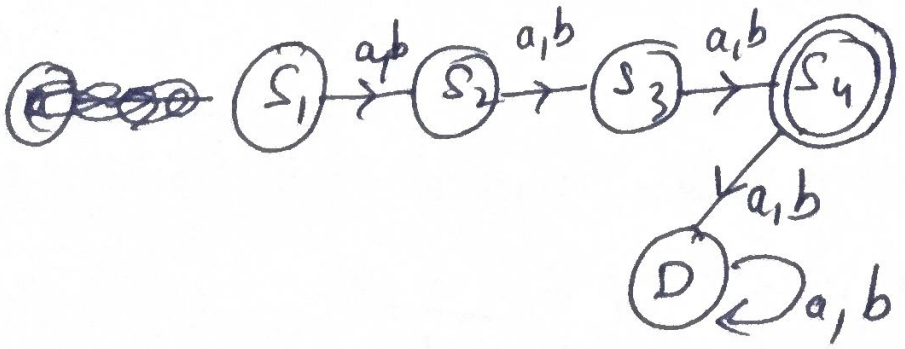
Minimum number of states initially => 3+1 => 4
As Language is infinite, atleast one of the states will have loop.
and no dead state will be there.

Not accepted strings = $\Sigma^* - X$

Note: Minimal DFA which accepts atleast n-length string needed (n+1) initially states.

Ques exactly three length.

$L_1 = \{ \underbrace{aaa \cdots bbb}_{8 \text{ strings only}} \}$



Minimal DFA which accepts exactly two length string, have (n+1+1) states
↑ dead state.

Exactly three => $(a+b)^3$

Atleast three => $(a+b)^3 . (a+b)^*$ [3 + anything]

Atmost three => $(a+b+\epsilon)^3$ => $(a+b+\underline{\epsilon})(a+b+\underline{\epsilon})(a+b+\underline{\epsilon})$

$\epsilon . \epsilon . \epsilon$ => zero length

.. or $(a+b)^* . (a+b)^3$ or $(a+b)^* . (a+b)^3 . (a+b)^*$

Note: For a particular language, R.E may not be unique.

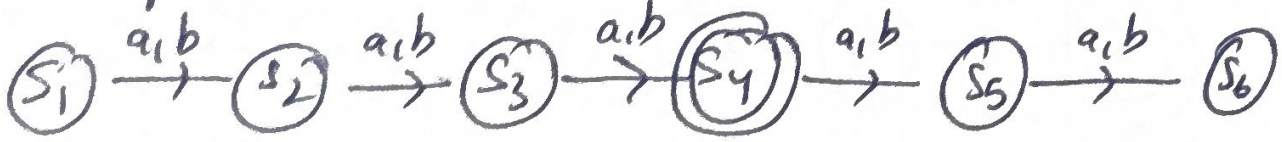
For a given a language, if a DFA and R.E accepts the given language. and also rejected ($\Sigma^* - X$) strings, then R.E and DFA are said to be equivalent.

Note: Every finite language is Regular, because R.E possible by 'inserting' '+' operator [Ex OR] symbol..

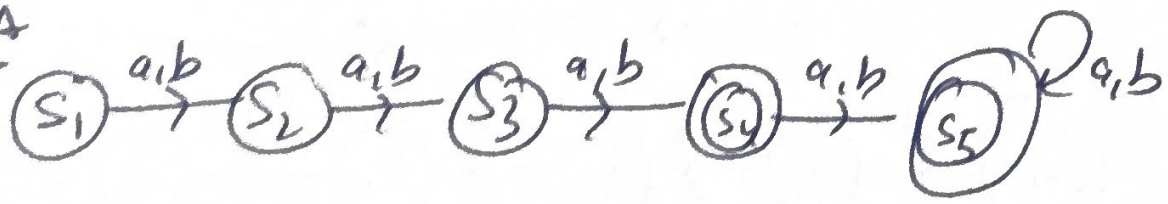
Note: Minimal DFA is unique but not minimum R.E.

Note: DFA may not be unique but minimal DFA is unique.

D.F.A



minimal DFA



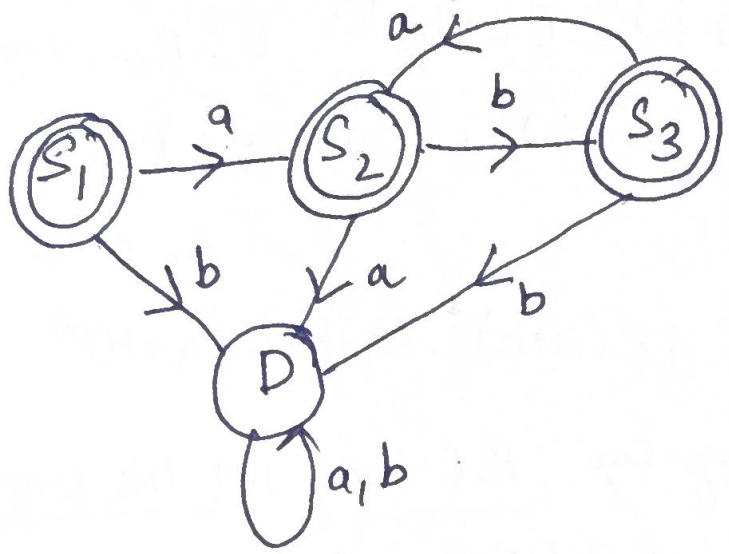
Q Construct Minimal DFA, that accepts all prefixes of string

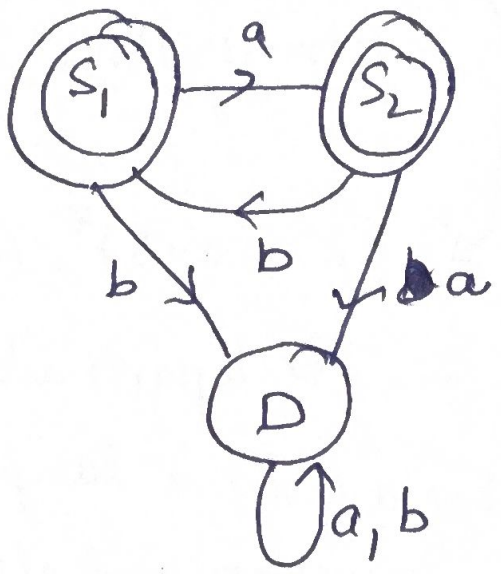
ababababab ---

$\in$ ababababab ------

$\therefore L_1 = \{a, ab, aba, abab, ababa ----, \in\}$

Total $\Rightarrow n+1$ prefixes, $\therefore$ infinite language



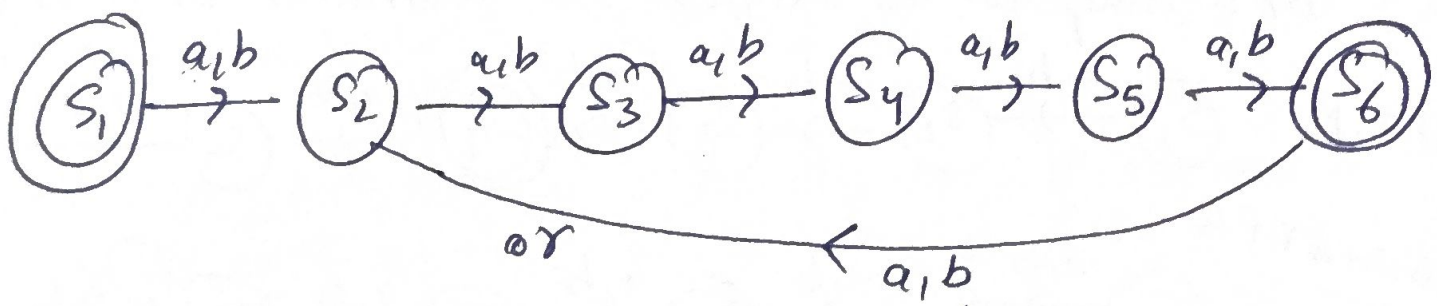


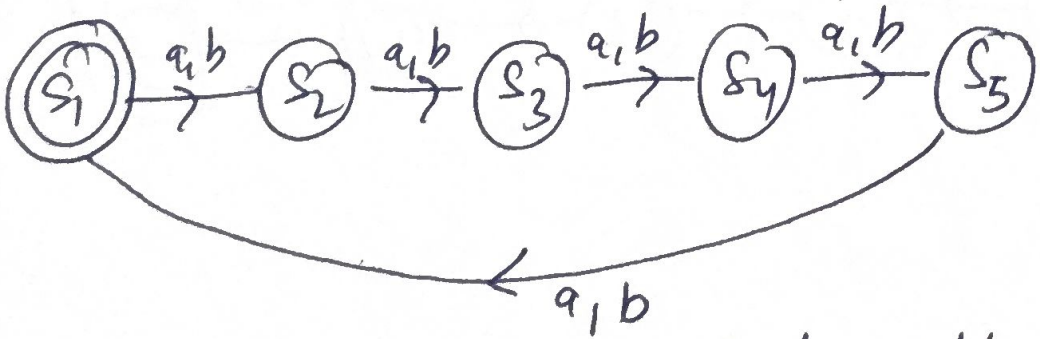
Ques Construct minimal DFA that accept all strings of a's and b's where length of each string is divisible by 5.

$L_1 = \{\in, \underbrace{aa\ a}_{5 \text{ length}}, \underbrace{b\ \ b}_{5 \text{length}}, \underbrace{a\ \ \ \ a}_{10 \text{length}}, \}$

$\in$, 2^5 , 2^{10} , 2^{15} , 2^{20} , infinite length.

DFA





minimal DFA

Note: minimal DFA that accepts string of a's and b's, where each string length is divisible by n contain, needed n states.

Modified previous question, string start with a, then n+1 state.

* Regular Exp. for previous problem? $(a+b)^5$

⇓

generate only string of length 5 only

Additional : $((a+b)^5)^*$ → this is extra and req.

⇙

$(a+b)\ ||\ [(a+b)^5]^1\ ||\ [(a+b)^5]^2$

R.E = $(a+b),\ (a+b)^5,\ (a+b)^{10}$

Note :- $L = \{0, 5, 10, 15, 20, \ldots\}$ ⇒ If AP series is there then DFA is there and loop is there.

$5-0 = 10-5 = 20-15 = 15-10 \Rightarrow \boxed{5}$

① $[(a+b)^3]^*$ multiple of 3.

② $((a+b)^4)^*$ multiple of 4.

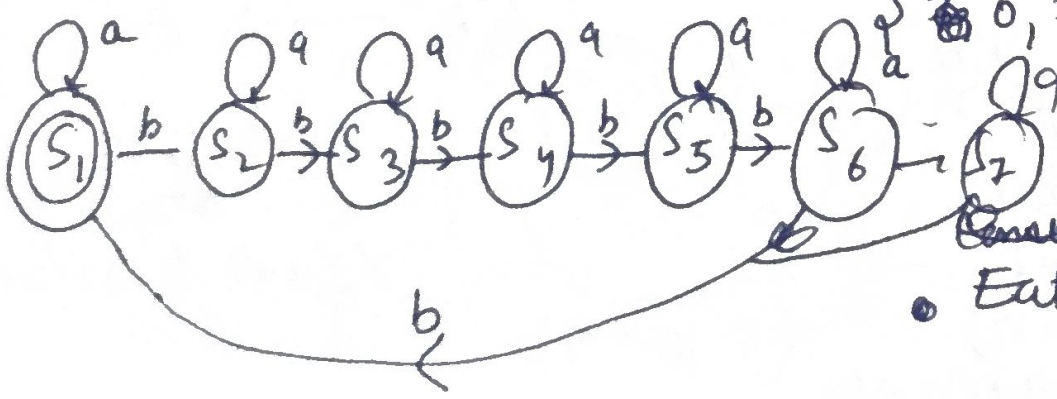
[* Having infinite language, generates AP series, then it is Regular Language. (There is algorithm to check]

* For finite language,

Ques Construct min. DFA that accepts all strings of a's and b's where 'a' is every string number of b's divisible by 7.

$L = \{\in, \underbrace{bbb\ldots b}_{7}, \underbrace{abb\ldots b}_{7+1}, \ldots, a, aa, aaaa, \ldots\}$ ← unique.

$\{0, 7, 14, 21, \ldots\}$



Length of number of b's only

• Eat 'a', but seeing b move to next state.

Tricky Problem

$$\_b\_b\_b\_b\_b\_b\_b\_$$

$$\Downarrow$$

$$(a^*ba^*ba^*ba^*ba^*ba^*ba^*)^*$$

$L_1 = \{\epsilon, b^7, b^{14}, b^{21}, b^{28} \ldots\}$ $\{a^1, a^2, a^3, \ldots a^n\}$

only b what I care.

Only a also be considered in this.

$a^*.(a^*ba^*ba^*ba^*ba^*ba^*ba^*)$ ← a^* can also be here.

↓

$\epsilon . \epsilon = \epsilon$

$a . \epsilon = a$

↓

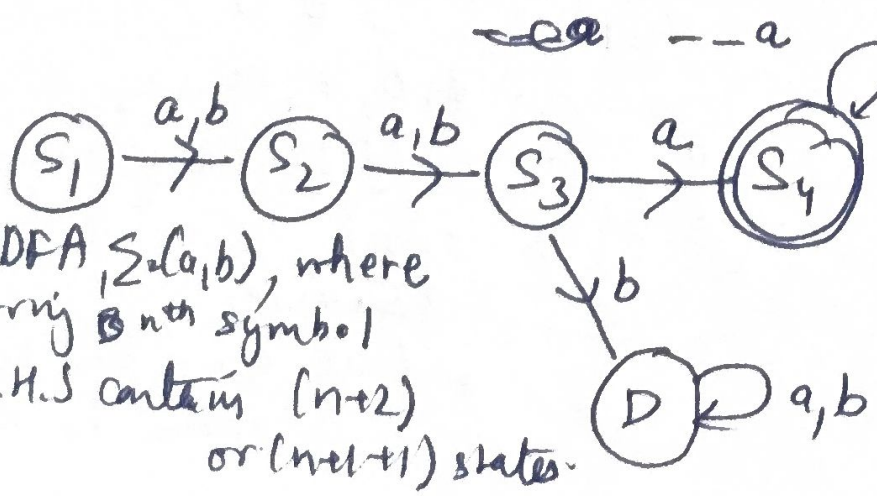
any number of a

Remember ~~$(a^*b^7a^*)^*$~~ => this will give multiple of 7 of b's only which are continuous but alternate b's are also there

Timestamp: 25 : 2:02:01 [time]

Construct minimal DFA, $\Sigma = \{a,b\}$, where in every string 3rd symbol from left hand side is "a".

$L_1 = \{aaa, baa, aba, bba, bbab, aaaaa, aaab, \ldots$



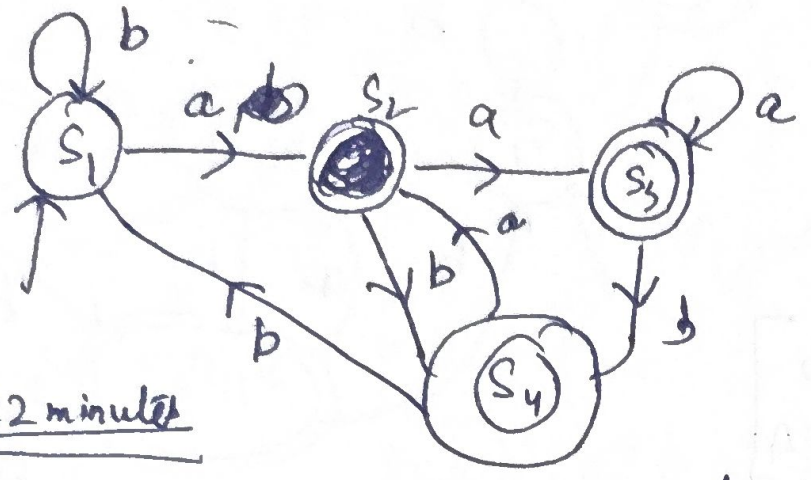
Note: min DFA $\Sigma = (a,b)$, where in every string nth symbol from L.H.S contains (n+2) or (n+1+1) states.

$(a+b)^2 . a . (a+b)^*$

$\times (a+b)^* . a . (a+b)^{*2}$

Ex: Const. min-DFA, $\Sigma = \{a, b\}$, where every string ~~string~~ ~~contains~~ from R.H.S, then 2nd symbol is always a.

$L_1 = \{ \underline{aa}, ab, aab, baa, bab, \dots \}$ $\{2, 3, 4, 5, 6, \dots\}$



$\{\Sigma^* a - \}$

Lecture-6

from 00-22 minutes

Cases to check

① aba
② abb
③ ab
④ aa.... a
⑤ bb.... b
⑥ bb ← pattern broken go to the initial state
⑦ abab
⑧ aa
⑨ babab

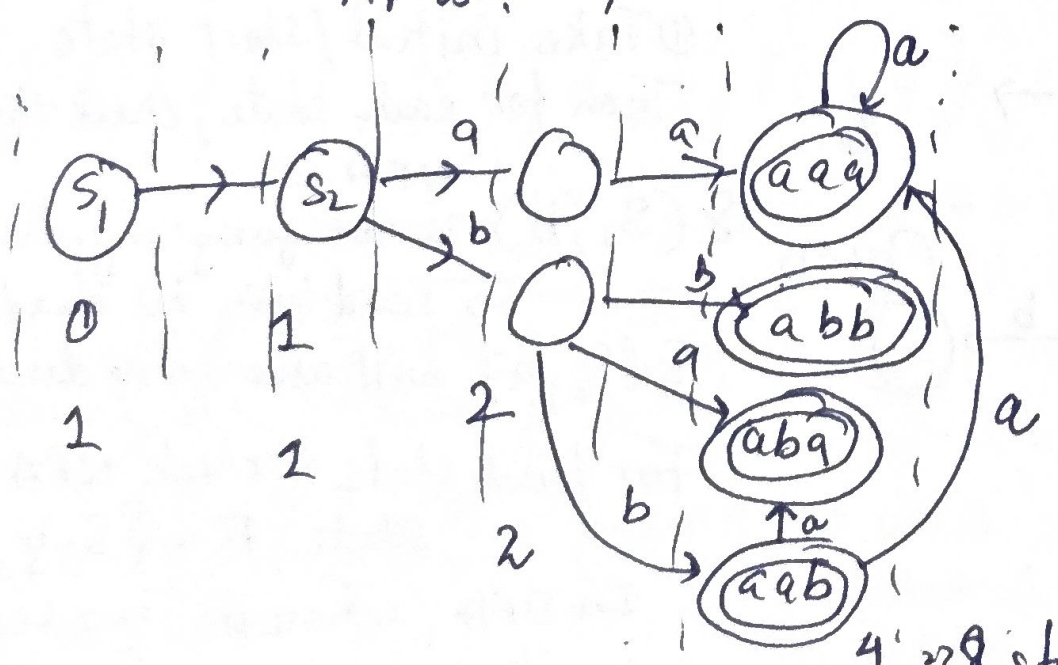
Note: Donot create state which are dead, and ending can be changed. So wait in the non-final state.

Formula

Note: The minimal DFA, that accepts all strings of a's and b's where nth symbol from right hand side is a, contains 2^n states., last n symbols you have to remember. From R.H.S, 3rd symbol is a :) 2^3 :) 8 states.

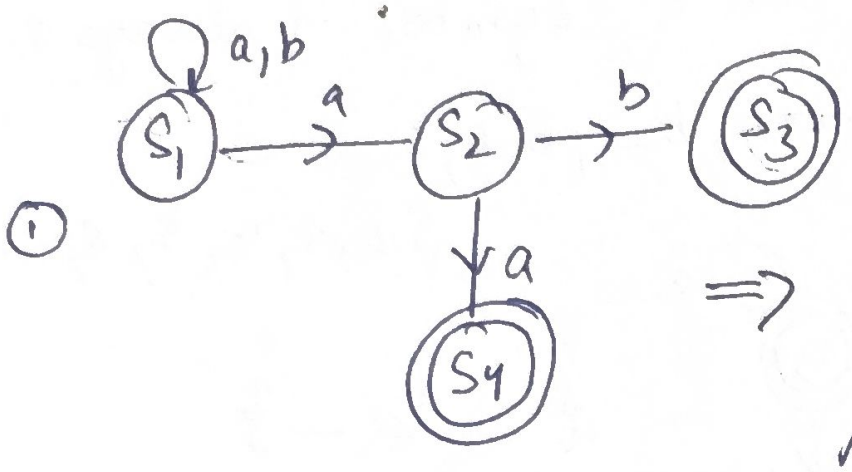
⑩ $L_1 = \{2, 3, 4, 5, 6 \dots\}$ AP series, Hence DFA.

bababb ← accepted

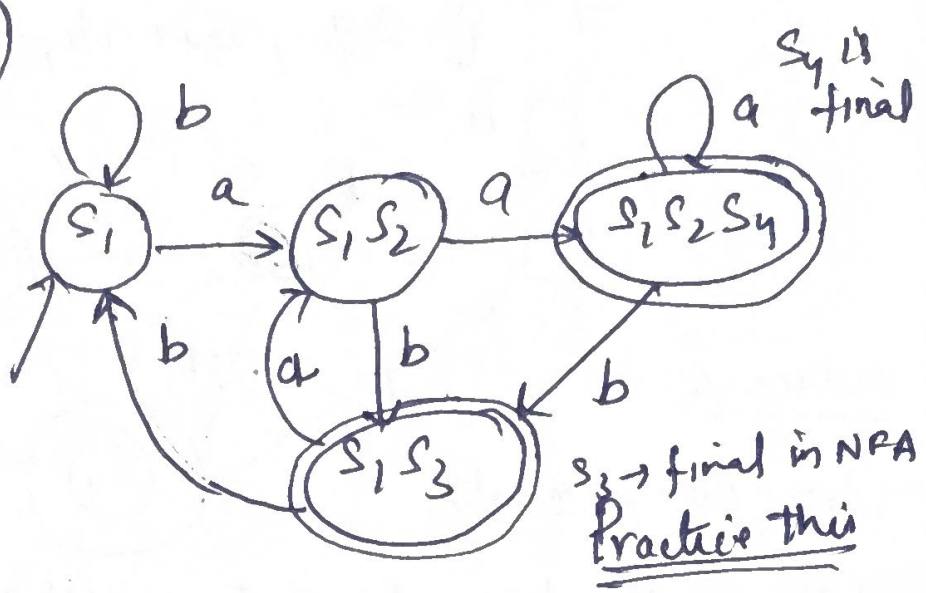


Lecture 6 :→ 1:01:28

2nd Symbol - RHS - "A". NFA to DFA Construction (Indirect approach)



[NFA to DFA]



Practice this

Power of NFA ≠ Power of DFA

① One is easy.

② One is difficult.

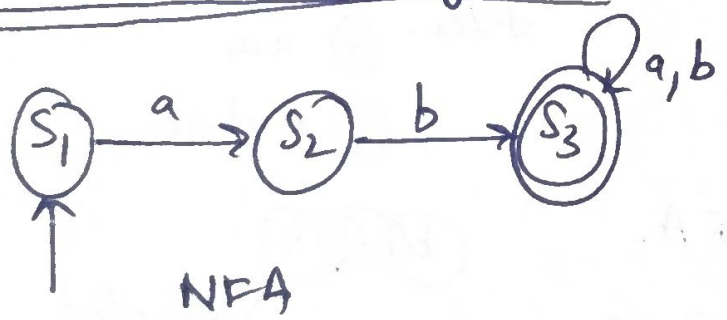
For final states in DFA

① Check NFA, where the final states are named

$F = \{S_3, S_4\}$

② Check DFA, if the states include S_3 or S_4 // S_3 and S_4

Consider the following NFA:→



NFA

$L_1 = \{ab, ab\Sigma^*\}$

$R.E = ab.(a+b)^*$

Steps

① Take initial / start state.

Then for each state, check the NFA.

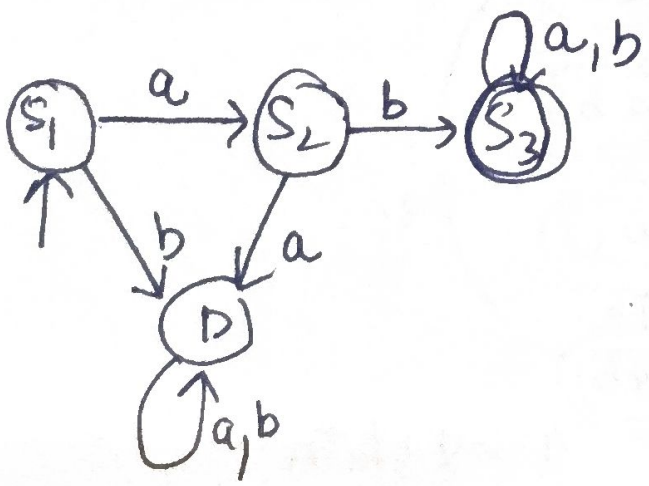
$\delta(S_1, b)$ is not going anywhere, So send him to dead state.

$\delta(S_2, a)$ will also go to dead state.

For final state: check NFA final state $F = \{S_3\}$,

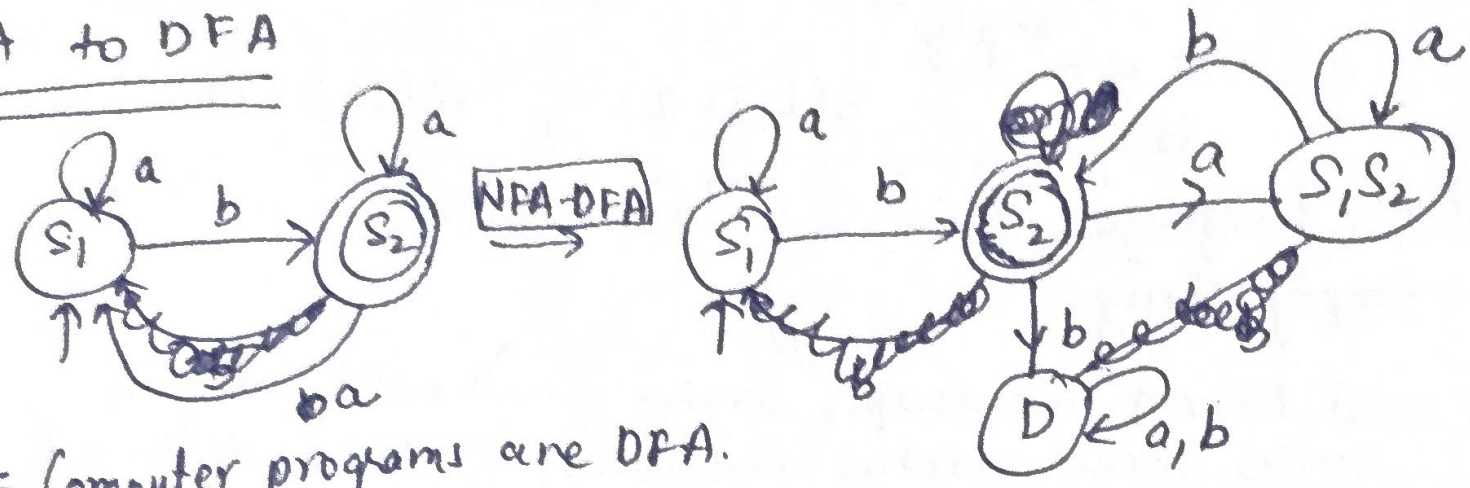
In DFA, wherever you see S_3 as its state, then make it final.

→



- Power of NPA is equivalent to DFA.
  Power here refers to number of Language it can accept.

NFA to DFA



#. Computer programs are DFA.

$L_1 = \{ a^* b a^* \}$

Atleast one b in between

① abab ✓
② ab ✓
③ baaa... ✓
④ bab ✗
⑤ bb ✗

S_1S_2
$(S_1, b) \Rightarrow S_2$ $(S_2$, no state)
$(S_2, b) \Rightarrow$ no state (S_2)
$S_0, [(S_1, S_2), b] \Rightarrow S_2$

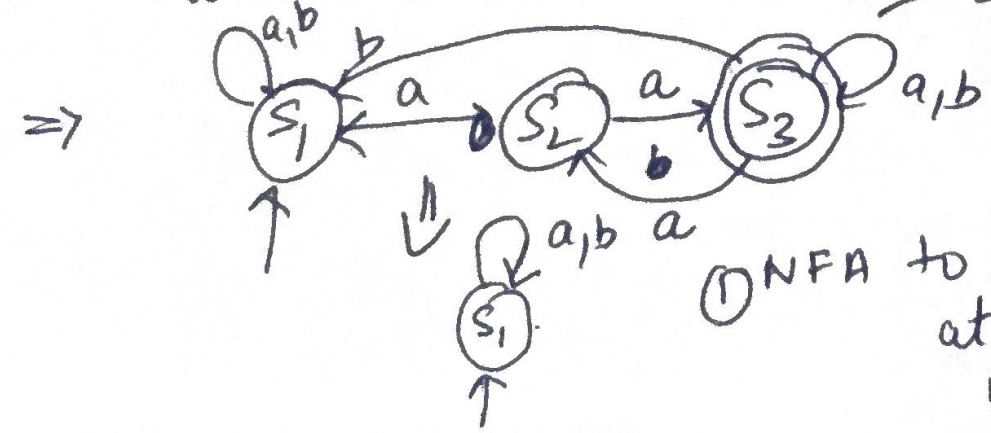
Note:
- If NFA contains n-states, equivalent DFA contains atmost 2^n states.

Note

$Q = \{S_1, S_2\}$

$P(Q) = \{ \{\}, \{S_1\}, \{S_2\}, \{S_1, S_2\} \}$ sets

The above contains 2^n subsets → **Note:** from start you Cannot reach to S_3.



NFA: { } or ϕ

DFA: $L_1 = \{ \}$ or ϕ empty language

① NFA to DFA atleast 1 to atmost 2^n states in DFA.

$L_1 = \{\in\}$, $L_2 = \{\}$, recursive language is empty

~~empty~~ string is empty but not language.

$L_1 \neq L_2$ are not equivalent.

$n(L_1) = 1$, $n(L_2) = 0$

$n(L_1) \neq n(L_2)$

In short, Language contains empty string.

Ques :→ const. M-DFA that accepts all strings of a's and b's where every string starting and ending symbol is different.

Ques starting and ending symbol is same.

Solution: $L = \{ab, ba, a\Sigma^*b, b\Sigma^*a\}$

NFA to DFA, min(DFA) → to get RE $= a(a+b)^*b + b(a+b)^*a$

* May not be useful to GATE! min(DFA)

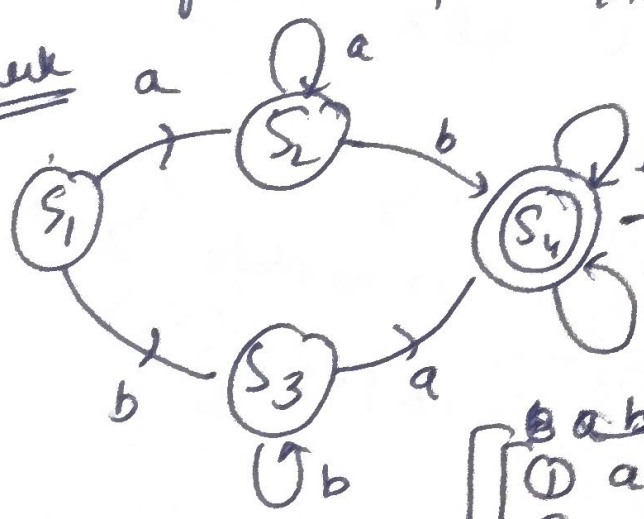
(You can take either 1st or 2nd but not both.

aaaab :→ $a(a+b)^2b$ $b^{10}a^{10}$

aaaab $b\,b^9a^9\,a$

$\Rightarrow b(a+b)^9(a+b)^9a$

Check



→ common final

$L = \{2, 3, 4, 5, \ldots\}$

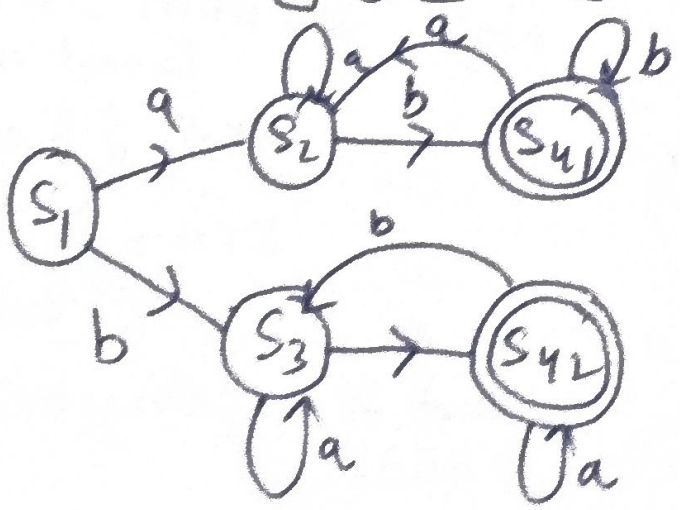
An AP series, Loop will be there.

This implies, atleast two final State will be there

aabbaa

① aabb ↵
② abaX
combined Hence false

③ baa ↵
④ babX
⇒ combined Hence false



Solution:2 Strings ends with same symbol.

$L = \{a, b, aa, bb, a\Sigma^*a, b\Sigma^*b, \epsilon\}$ (null string)

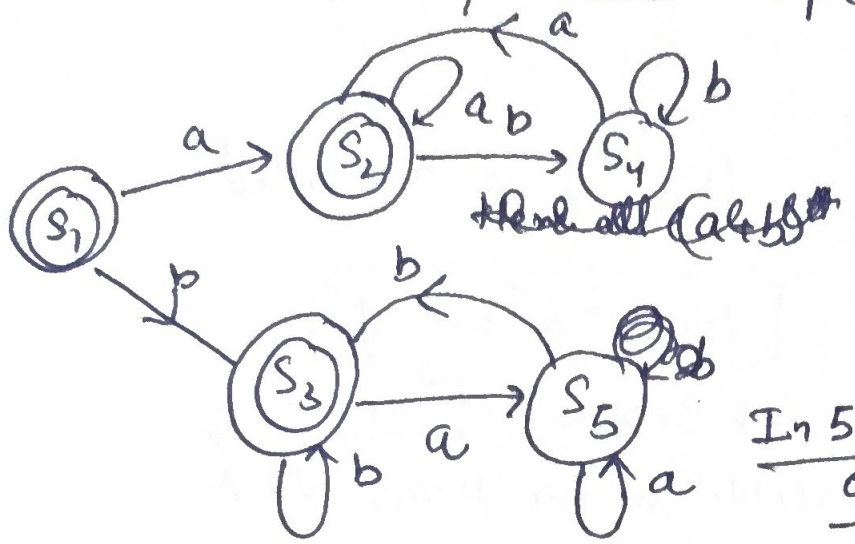
aa ✓
bb ✓
ε ✓
but ab, ba also accepted

R.E ⇒ $a + b + a(a+b)^*a + b(a+b)^*a + \epsilon$

$L = \{a, 1, 2, 3, 4, 5 \ldots\}$

AP is there, Hence DFA is possible with loops

Remember in R.E, "+" sign indicate you can take only one out of all.



In 5 states only

Valid for DFA:

In solution "1" and "2", they are complimentary in nature. For every Yes in "1" the second one will say invalid.

For every No in first solution, the second one will say valid.

Hence:

$U = \{1, 2, 3, 4, \ldots 8\}$

$A = \{1, 3, 5, 7\}$

$A^C = U - A$

$\Rightarrow \{2, 4, 6, 8\}$

$L^C = \Sigma^* - L \Rightarrow$ {complement of L}

Two automata's are complimient of each other.

① $L \cup L^C = \Sigma^*$

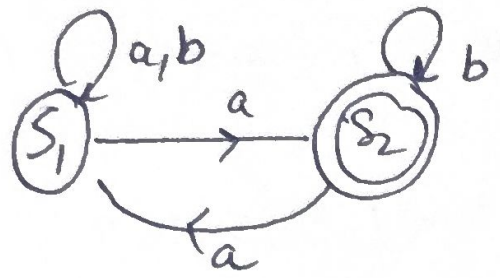
② $L \cap L^C = \phi$

$L^C = \Sigma^* - L$

To get the L^C, make the non-final state to final state and final state to non-final state.

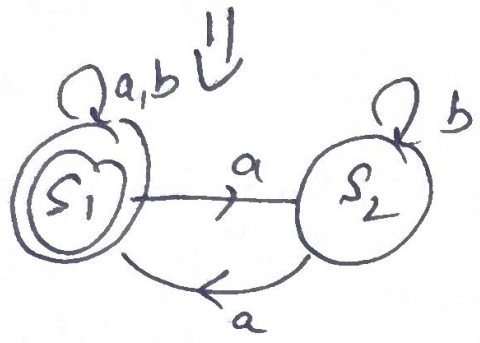
Remember, the complement property doesnot work for NFA. may be because there is ambiguity.

L



$(S_1, a) \Rightarrow$ a is accepted

L^c

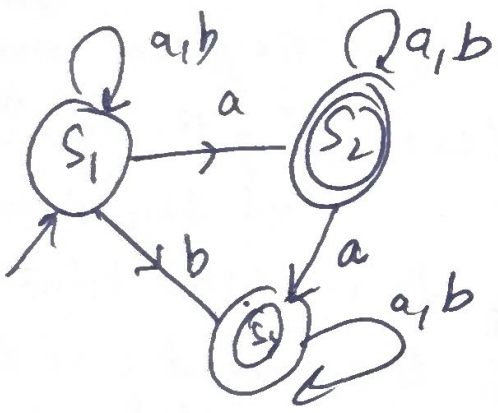


(S_1, a) is accepted

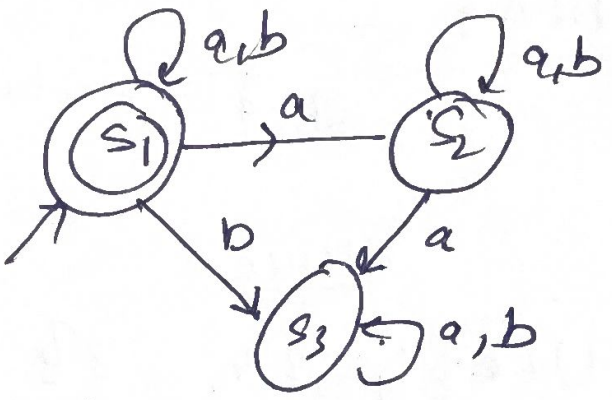
$L \cap L^c$ = "not ϕ"

For NFA, "compliment" may be not possible for some NFA.

* Break the 2nd property.



L_1 =



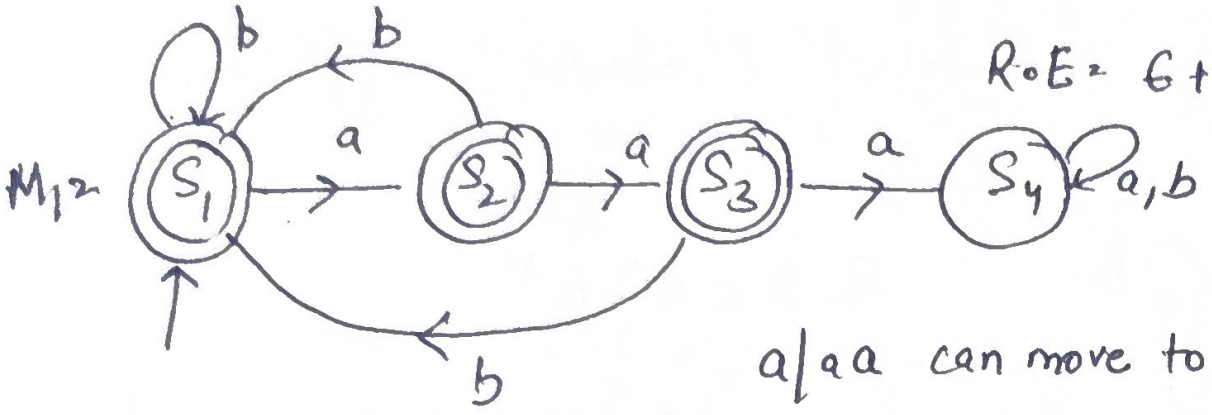
L_2 = ?

$L_2 = (a+b)^*$ * There may be relation

* NFA :→ to compliment of NFA generally comes in GATE

Construct minimal DFAs $\Sigma = \{a, b\}$, in which every string does not contain three consecutive a's.

$L = \{ \underbrace{\epsilon}_{1}, \underbrace{a}_{2}, b, \underbrace{ab, ba, bb, aa}_{4}, aab\ \ bba, aba\ \ bbb, baa\ \ bab, \ldots \}$



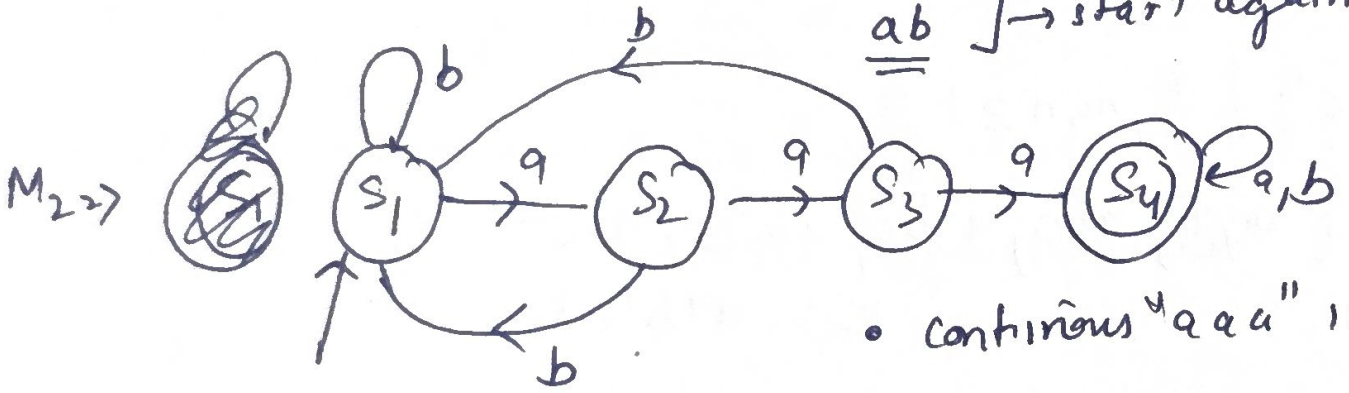
R.E = $\epsilon + a + aa + b +$

a/aa can move to S_4

ab => construction broken, go to S_1

aab, broke the construction of "aaa"

aab } start again.
ab }



- Continuous "aaa" in a string.

Note :→

- If not is there in the problem, without "not" you solve the problem. Then convert Non-final to final and final to non-final.

$(L_1^c)^c \Rightarrow L_2$

Remember both machines must be DFA.

- Construct minimal DFA, $L = \{ a^m b^n \mid m, n \geq 0 \}$

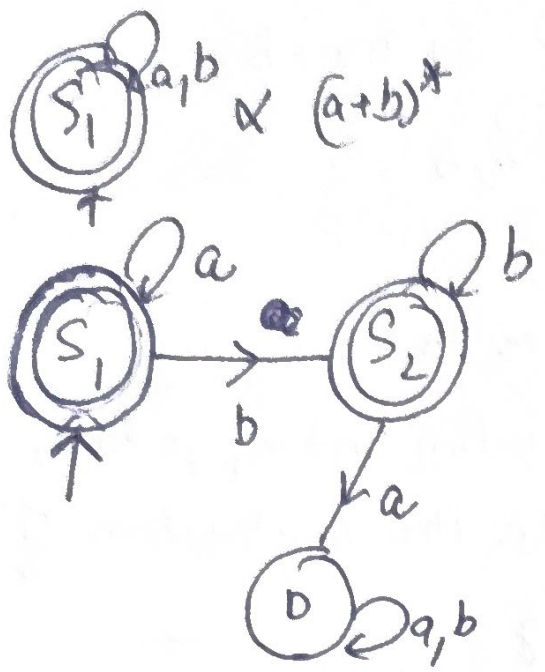
$L = \{ \epsilon, a, b, \underline{ab, ba, bb, aa}, aaa, bbb, aab, abb \}$

1,2
2,1

abaX

b → aX

a → b ✓



R.E = $a^* . b^*$

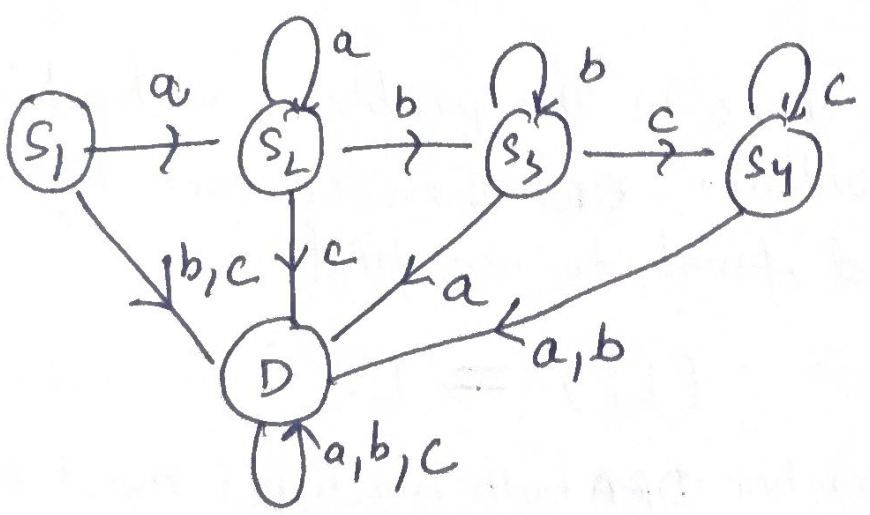
$L = \{ 0, 1, 2, 3, 4 \ldots \}$

* AP series hence, DFA possible.

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

$L = \{$ ~~abc~~, abc, a^2bc, ab^2c, abc^2 $\}$

R.E = $a^+ b^+ c^+$



abc

b ← start b

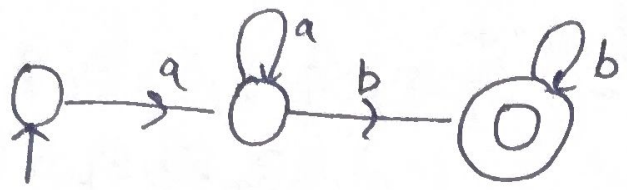
c ← start c

ac ← a after a

Solution

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

$L = \{ ab, a^2b^2, a^3b^3, \ldots \}$



This will accept all the L

$ab, a^2b^2 \ldots$

but

$a^2b, ab^2, a^3b \ldots$

Hence,

Drawback for DFA is, it cannot reject invalid string.

For eg:- $a^2b, ab^2, a^3b \ldots$

The R.E for above Language is not possible.

* Reason is comparison for this

Construct minimal DFA $L = \{ w \mid w \in (a+b)^* \}$, $n_a(w)$ divisible -2 and $n_b(w)$ divisible -3 $\}$

$L = \{ \epsilon, ab, aabb, \ldots \}$ $\frac{5!}{2!3!}$ $\frac{7!}{4!3!}$

$L = \{ \underset{0}{\epsilon}, \underset{2a's}{aa}, \underset{3bs}{bbb}, \underset{2a's\ 3bs}{aabbb}, \underset{4a's}{aaaa} \underset{3b's}{bbb} \}$

ababb abababa

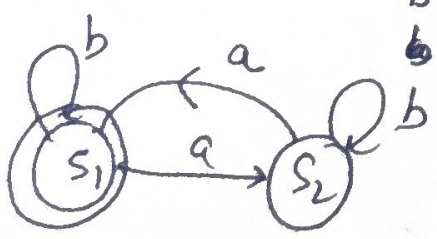
abbba

① $n_a(w)$ divisible by 7. 7 states.

② $n_a(w)$ divisible by k, k states

(P1) $n_a(w)$ divisible by 2.

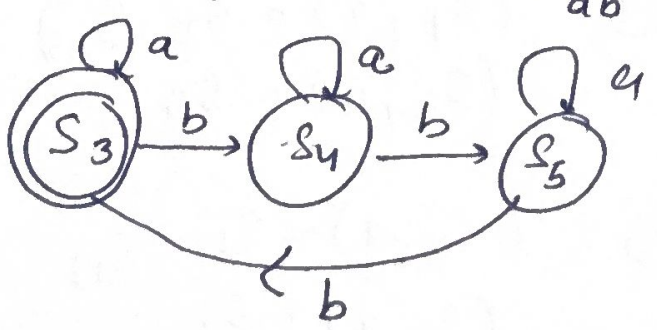
$L_1 = \{ \epsilon, b, bb, aab, baa, \ldots \}$



$b^*(b^*ab^*ab^*)^*$

$n_b(w)$ divisible by 3

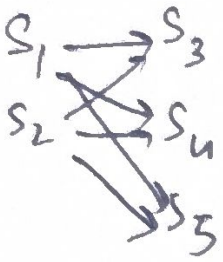
$L_1 = \{ \epsilon, aa, aaa, \ldots, baaa, ab \}$

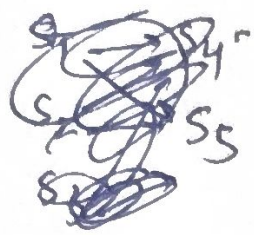


$a^*(a^*ba^*ba^*ba^*)^*$

* To get AND product, as we know the solution of the two subproblem. Follow this :→ Number of state in P1 X no. of state in P2

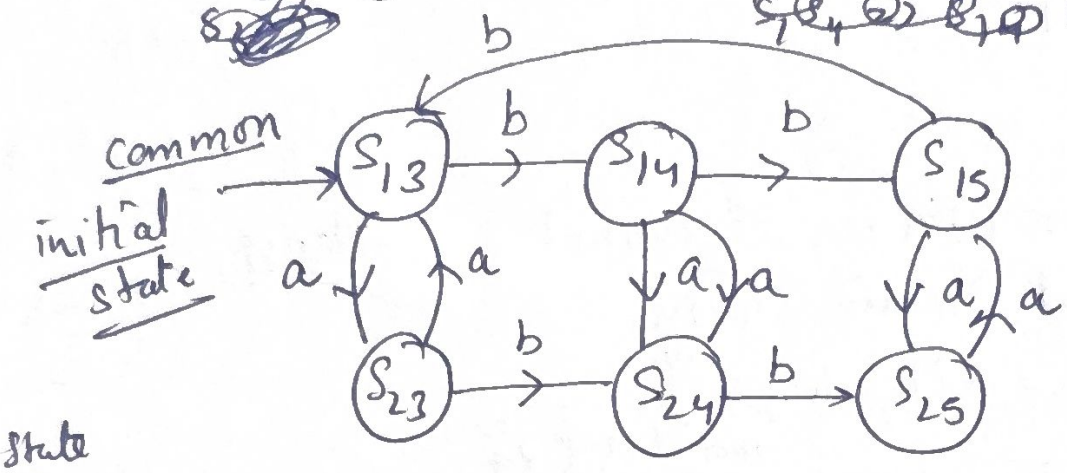
2 X 3 → 6 state





Rows :→ ~~S1S3, S1S4, S1S5, S2S3, S2S4, S2S5, S3S4, S3S5~~

~~S1S4 S2 S1S5~~ (shortcut)



For Initial state

For initial state, where P_1 and P_2 ~~have same~~ S_1 and S_4, That this known as initial state. S_{14}.

Check the P_1 P_2

① $(S_1, a) \Rightarrow S_2$ He will go to S_2S_3 or S_{23}
$(S_3, a) \Rightarrow S_3$

② $(S_1, b) \Rightarrow S_1$ go to S_1S_4 or S_{14}
$(S_3, b) \Rightarrow S_4$

③ $(S_1, b) = S_2$ ... S_2S_4
$(S_4, a) = S_4$

④ $(S_1, a) = S_2$ (S_{25}) $(S_1, b) = S_1$ (S_{13})
$(S_5, a) = S_5$ $(S_5, b) = S_3$

⑤ $(S_2, a) = S_1$ S_{13} $(S_2, b) = S_2$ (S_2S_4) S_{24}
$(S_3, a) = S_3$ $(S_3, b) = S_4$

⑥ $(S_2, a) = S_1$ S_{14} $(S_2, b) = S_2$
$(S_4, a) = S_4$ $(S_4, b) = S_5$

For combining two or problems set of

① Take the cartisian product of states from set 1 to 2

$P_1 = \{S_1, S_2, S_3 \ldots S_4\}$

$P_2 = \{S_3, S_4, S_5\}$

→ S_1 S_2 S_3 S_4 S_5

S_1S_3 S_1S_4 S_1S_5

S_2S_3 S_2S_4 S_2S_5

For initial and final states, check the set in where both initial state is there.

For final state, ---

① AND operator we both final in previous problem $\Rightarrow S_{13}$

② OR operator, any final in S_KS_L.

$S_{14}, S_{15}, S_{23}, S_{25}$

③ $P_1 - P_2$ P_1 should satisfy but not P_2.

where S_1 should be there but not S_3

S_{14}, S_{15}.

④ $P_2 - P_1$, S_3 should be there but not S_1.

S_{23}

Purpose of this model :→ Divide into subproblems and solve them.

Construct n-DFA. $\Sigma = \{a, b\}$, where in each string number a's divisble by 2 or 4.

$L_1 = \{\epsilon, b, bb, b^* \ldots, aa, aaaa,$

$aab, aaaab \ldots$

$aab \ldots,$

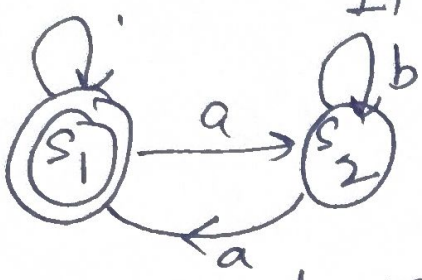
$ababb \ldots,$

$n_4(\omega)$ by 2 .
$n_4(\omega)$ by 4.] Using cartisian product $2 * 4 = 8$ states later

$L = \{\epsilon, b, bb..., 2's, 4's, 6's, 8's, 10's, 12's ....\}$

It is an AP series, 2 difference.

Anything divisible by 4

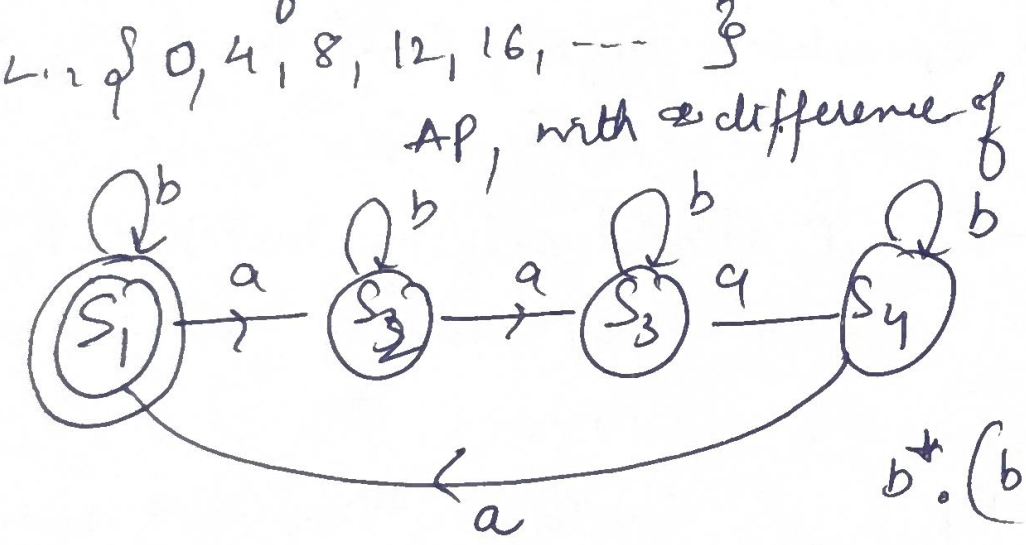


Note:→ For same symbol, you can minimize. observe the pattern But for different subproblem, do product. This will be minimal DFA also.

Problem Same symbol number of a's divisible by 2 and 4 also.

$L = \{0, 4, 8, 12, 16, ...\}$

AP, with a difference of 4, loops will be there.



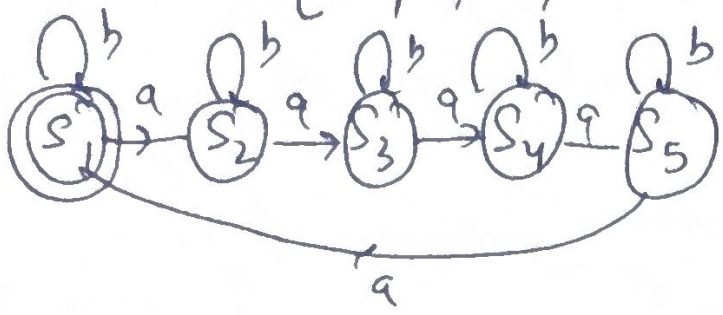
AND ⇒ $b^*.(b^*ab^*ab^*ab^*ab^*)^*$

OR ⇒ LCM of (2,4) on a's symbol ⇒ 4

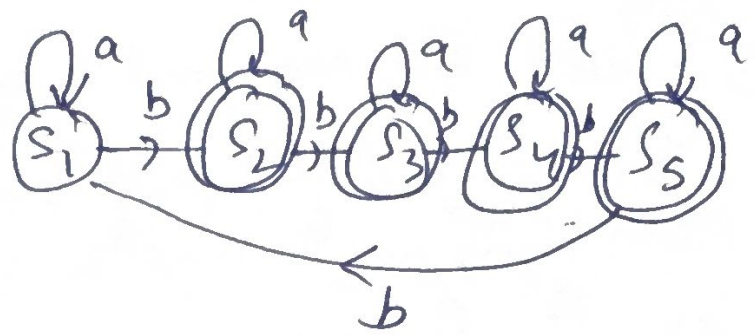
Problem number of a's, divisible by 2 and number of b's divisible by 4.

Construct m-DFA for all strings of a's and b's where no. of a's divisible by 5 or number of b's not divisible by 5.

$L_1 = \{\epsilon, a, aa, aaa \ldots\}$



$L_2 = \{b, bb, bbb, bbbb, ab, \ldots\}$



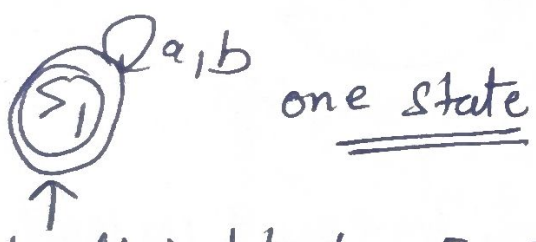
a. $5 \times 5 \Rightarrow$ 25 states [as it is no minimization]

Problem

No. of a's divisible by 5 or no. of a's not divisible by 5.

$L_1 = \{0, 1, 2, 3, 4 \ldots\}$ $\quad \Sigma = (a, b)$

A.P. $(a+b)^*$



one state

no. of a's divisible by 5 and no's of a's divisible by 5

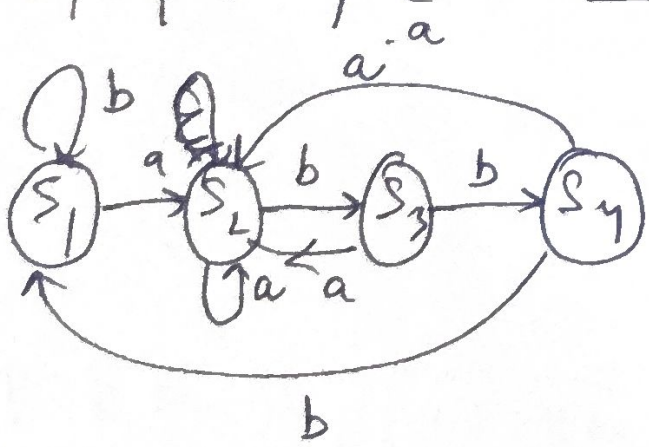
$L_1 = \{\}$

no final state

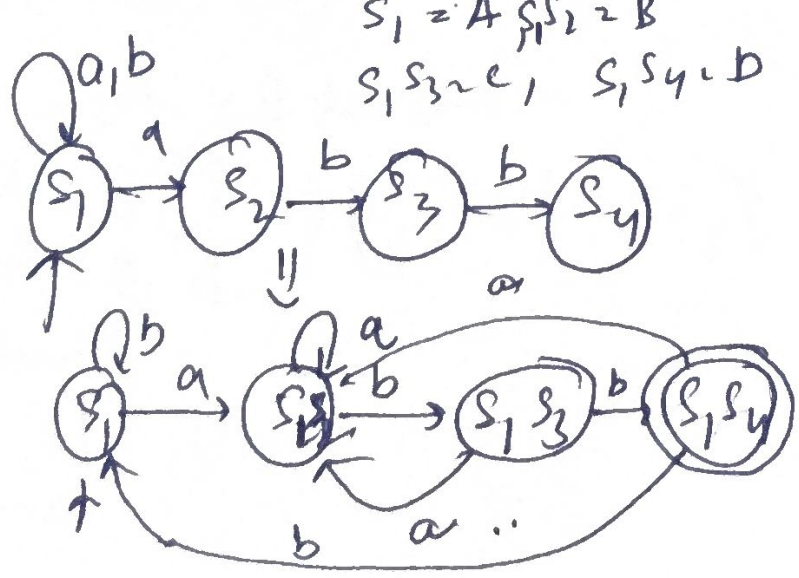
Problem

Const minimal DFA, that accepts all string of a's and b's, where in every string last three symbols are abb.

$L_1 = \{abb, aabb, baabb \ldots\}$ NFA



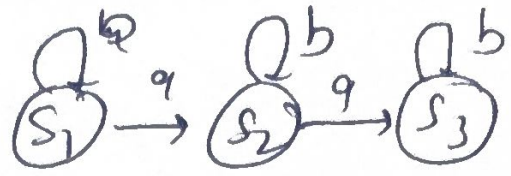
DFA



Step Convert

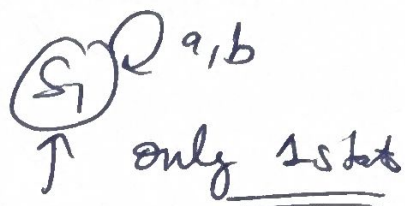
③ ~~Minimization~~ DFA to minimal DFA as there is ~~guarantee~~ surity NFA to DFA will give ~~A~~ min DFA.

Partition Algorithm : (Page 78) see)



Initially

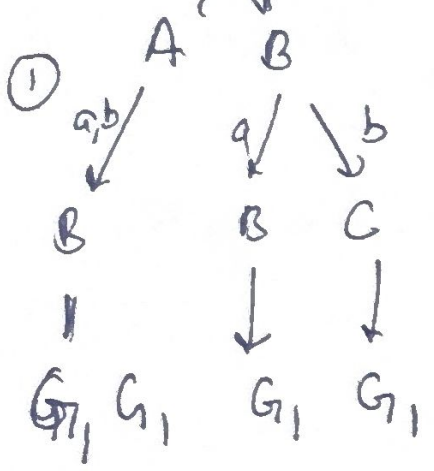
① ~~Equal~~ All non-final is equal. and final is equal. ⇓ minimizals

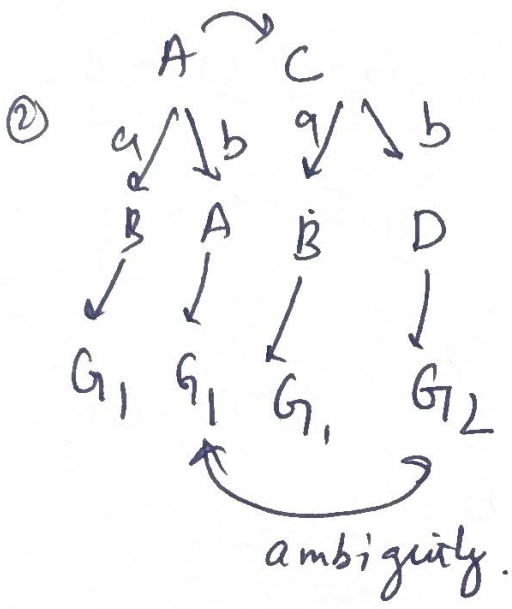


↑ only 1 state

I_0 => (ABC) (D)

ask G_1 G_2



② 

ask

~~A~~ A B

① => A ask B, and they belong to group 1 initially and checking transition on a and b, they still be in same group.

② => A ask C

Here A, b → G_1] → different group) give seperate room.
C, b → G_2

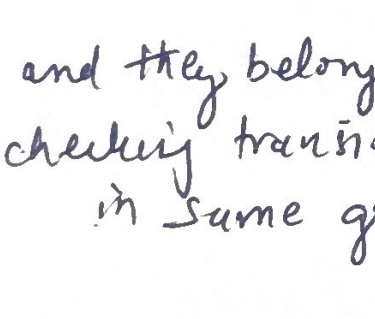
I_1 => (AB) (C) (D) => After 1st round.

G_1 G_2 G_3

(A (C) (D)

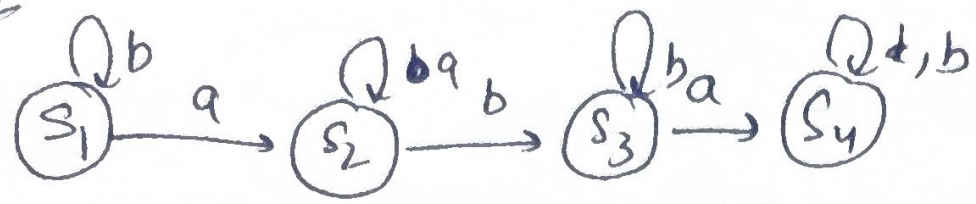
A ask B

I_2 => (A) (B) (C) (D)



G_1 G_2 moving as they belong to different group on ambiguity, hence must seeing "B", so be seperated

Problem



① $I_0 = (S_1, S_2, S_3, S_4)$

$I_1 = (S_1 S_2 S_3 S_4)$

As everyone belongs to same group, no minimization can happen as there is no final state.

$[S_1 \quad S_2 \quad S_3]$ same group.

$[S_1 \quad S_4]$ same group.

$[S_1 \quad S_2 \quad S_3]$

a/ b, a/ b

$S_2 \quad S_1 \quad S_2 \quad S_3$

$G_1 \quad G_1 \quad G_1 \quad G_1$

$I_0 = I_1 \Rightarrow (S_1 S_2 S_3 S_4)$ ✓

only one state.

Redo Problem ⑯

Construct DFA all binary numbers which are divisible by 5.

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

0 ⇒ 0

101 ⇒ 5

1010 ⇒ 10 (8 4 2 1)

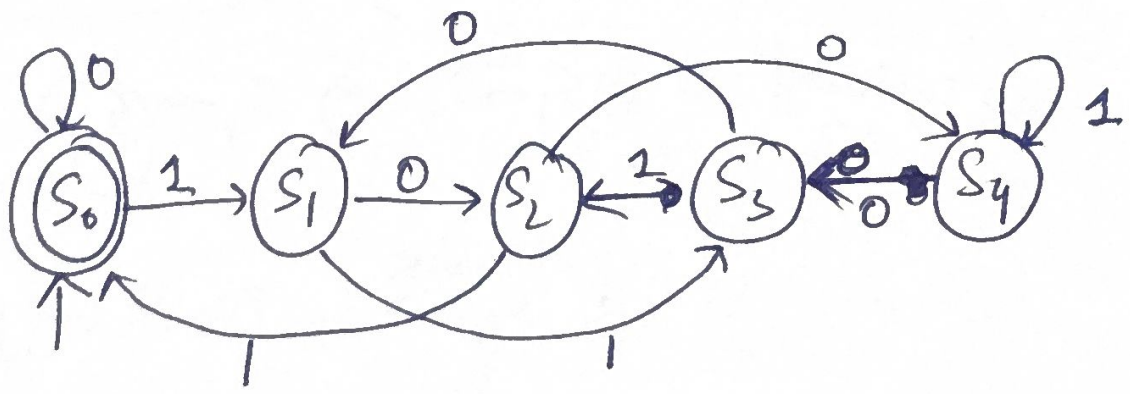
1111 ⇒ 15 (8 4 2 1)

10100 ⇒ 20 (16 8 4 2 1)

11001 ⇒ 25

11110 ⇒ 30

100011 ⇒ 35 (32 16 8 4 2 1)



S_0 ⇒ n mod 5 = 0 ⇒ 0, 101, 1010, ...

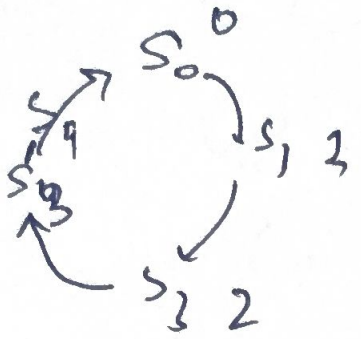
S_1 ⇒ n%5 = 1 ⇒ 1, 110, 1001

S_2 ⇒ n%5 = 2 ⇒ 10,

S_3 ⇒ n%5 = 3

S_4 ⇒ n%5 = 4 ⇒ 100,

δ	0	1
$\rightarrow^* S_0$	S_0	S_1
S_1	S_2	S_3
S_2	S_4	S_0
S_3	S_1	S_2
S_4	S_3	S_4



Equivalent to Quick sort partition.

Lecture - 9
44:01

Apply minimization algorithm :⇒ (Partition Algorithm)

$I_0 \Rightarrow (S_0\ S_1\ S_2\ S_3\ S_4)$ (S_0)
G_1 G_2

$(S_1 S_2 S_3 S_4) \Rightarrow$ zero equivalent

$S_1 \rightarrow S_2$

0/ \1 0/ \1

S_2 S_3 S_4 S_0
= = = =
G_1 G_1

G_1 G_2

use this

I_0
1
$I_1 \Rightarrow$ I_0 will be used, don't use the current one.

S_1 ask S_2, then S_2 will go.

$I_1 \Rightarrow$ (S_0) $(S_1 S_3 S_4)$ (S_2)
G_{11} G_{12} G_{13}

use this

$S_1 \xrightarrow{\text{ask}} S_3$

I_1 indicate length of string will be ≤ 1
Another same one equivalent.

$I_2 \Rightarrow$ (S_0) (S_1) (S_3) (S_2) $(S_1 S_3 S_4) \rightarrow$ one equivalent

If two equivalent ≤ 2

① $S_1 \xrightarrow{\text{ask}} S_3$ ② $S_1 \rightarrow S_4 \Rightarrow I_1$ (use I_1)
0/ $\Rightarrow$ Goes out

③ $S_3 \xrightarrow{\text{ask}} S_4$ (So it can join S_3 group)

(S_0) (S_1) (S_3) (S_4) (S_2)
G_{21} G_{21} G_{22} G_{22} G_{23}

Please don't ask S_0 group or S_2 group.

"K-equivalent":→ By reading less than or equal to k-length strings if two states are equal then ~~they~~ we can say that ~~that~~ those two states are k-equivalent.

$(S_1 S_2 S_3 S_4)$ are in same group until I_0 ∘ zero equivalent

$(S_1 S_2 S_4)$ are in same group until I_1 ∘ one equivalent.

$(S_1 S_3 S_4)$ is in same group in I_0 I_1

Hint ~~Last~~ 2 student said to be 20 equivalent.
until 20 days they are both are friends.
or in same group.

(From top to bottom)

S_0 S_1 are not even zero equivalent also

S_3 S_4 are I_1 equivalent or one-equivalent.

S_1 S_4 are I_1 equivalent or one-equivalent.

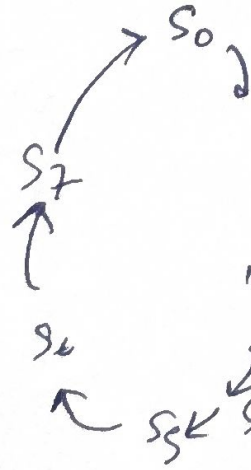
S_0 S_4, ~~a~~ S_0 S_1 not even zero equivalent.

Lecture 9 (1:28:00)

Construct minimal DFA that accepts all binary numbers divisible by 8.

Remainders → 0, 1, 2, 3, 4, 5, 6, 7.

	0	1
→ * S_0	S_0	S_1
S_1	S_2	S_3
S_2	S_4	S_5
S_3	S_6	S_7
S_4	S_0	S_1
S_5	S_2	S_3
S_6	S_4	S_5
S_7	S_6	S_7



S_1, S_3 divided will not be Stopping

minimization

$I_0 = \underbrace{(S_0)}_{G_1} \underbrace{(S_1 S_2 S_3 S_4 S_5 S_6 S_7)}_{G_2}$

$I_1 = \underbrace{(S_0)}_{G_1} \underbrace{(S_1 S_2 S_3 S_5 S_6 S_7)}_{G_{21}} \underbrace{(S_4)}_{G_{22}}$

$I_2 = \underbrace{(S_0)}_{G_1} \underbrace{(S_1 S_3 S_5 S_7)}_{G_{211}} \underbrace{(S_2 S_6)}_{G_{212}} \underbrace{(S_4)}_{G_{22}}$

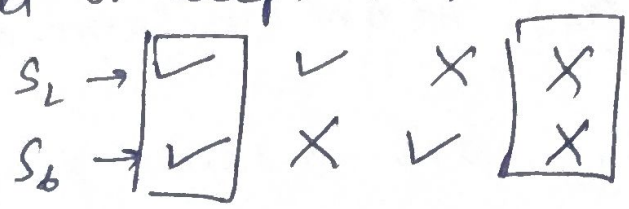
$I_3 = (S_0)\ (S_1 S_3 S_5)\ (S_2 S_6)\ (S_4)$

S_7

$I_3 = (S_0)\ (S_1 S_3 S_5)\ (S_2 S_6)\ (S_4)$

(S_2, S_4) => forever equal then they are equal.
Two people are equal. ~~upto~~ in each iteration,
if they are in same group upto infinite rounds.

They both reject or accept ~~both~~ the string.



both reject
or both accept
the string.

~~Hence~~
After applying ~~on~~ partition algorithm, you got

only $[S_0]$ $[S_1, S_3, S_5 S_7)$ $(S_2 S_6)$ (S_4)

K_0 K_1 ⇓ keep one only K_2 keep one K_3

(S_0) (S_1) (S_2) (S_4)

Imp. Property :→

Shortcut :→

$\underset{z}{8}$ => even => 2^3

3+1 => 4 states

All binary number

5 => odd => 5 states

Modified table after ~~their~~ minimization

		0	1	
→ *	S_0	S_0	S_1	
	S_1	S_2	~~S_3~~ S_1	
~~Delete~~	~~S_2~~	S_4	~~S_5~~ S_1	
X	S_3	S_0	S_7 $S_3 = S_1$	
	S_4	S_0	S_1	
X	S_5	S_2	S_3 $S_5 = S_1$	
X	S_6	S_4	S_5	$S_6 = S_2$
X	S_7	S_6	S_7	$S_7 = S_1$

* Construct m-DFA that accepts all binary numbers divisible by 6.

	0	1
→* S_0	S_0	S_1
S_1	S_2	S_3
S_2	S_4	S_5
S_3	S_0	S_1
S_4	S_2	S_3
S_5	S_4	S_5

$I_0 = (S_0)\ (S_1 S_2 S_3 S_4 S_5)$
$\quad\quad G_1 \quad\quad G_2$

$I_1 = (S_0)\ (S_1 S_2 S_4 S_5)\ (S_3)$
$\quad\quad G_1 \quad\quad S_{21} \quad\quad G_{22}$

$I_2 = (S_0)\ (S_1 S_4)\ (S_2 S_5)\ (S_3)$
$\quad\quad G_1 \quad\quad G_{211} \quad\quad G_{212} \quad\quad G_{22}$

$I_3 = (S_0)\ (S_1 S_4)\ (S_1 S_5)\ (S_3)$
$\quad\quad G_1$

Some shortcut :→

6 ⇒ odd × even exactly power of 2 ⇒ ① * ③ ⇒ 1+3 ⇒ 4 states

12 → even ⇒ $2^2 \times 3$ ⇒ 2+3 ⇒ 5 states

All binary numbers divisible by n
⇓ if you construct minimal DFA

if n is odd
⇒ n states

if n is even
n = 8
eg ⇒ 12 ⇒

exactly power of 2^k
$n = 2^k$
so (k+1) states

not exactly power of 2^k.
2^k * some odd number
⇒ $2^k * m$
⇒ m + k states

Some problem :→

32 → 25 ⇒ 5+1 ⇒ 6 states

30 → 2 * 15 ⇒ 1+15 ⇒ 16 states
(2: even, 15: odd)

31 → 31 → 31 states
(31: prime odd)

50 → 2*25 ⇒ 2*25 ⇒ 2+25 ⇒ 27 states.

Hard

Construct min DFA which accepts all binary numbers divisible 7 and starts with 11.... .

$L = \{ 11, 0, \}$

0
7
14 => 1 1 1 0
8 4 2 1

21 => 1 0 1 0 0 1
16 8 4 2 1

28 =>

δ	0	1
→ * S_0	S_0	S_1
S_1	S_2	S_3
S_2	S_4	S_5
S_3	S_6	~~S_0~~
S_4	~~S_1~~	S_2
S_5	S_3	~~S_4~~
S_6	S_5	S_6
~~S_7~~	~~S_7~~	~~S_7~~

Don't touch the existing machine.
And accept the string which starts with 11 only and rest 00, 01, ~~11~~ 10 must go to dead state.

=>

	0	1
→ q_0	D	q_1
q_1	D	S_3
D	D	D
* S_0	S_0	S_1
S_1	S_1	S_2
S_2	S_3	S_4
S_3	S_5	S_6
S_4	S_1	S_2
S_5	S_3	S_4
S_6	S_5	S_6

↙ all binary number ~~state~~ start with 11

All binary Starts with anything

Hence :- 7 + 3 => 10 states

① Initial state will change to q_0

② Final state will remain unchanged.

* Donot messup the existing existing table

(NFA or DFA)

Decidable problems of Finite Automata :→

Definition :→ for any problem, an algorithm exist then it is known as decidable problem.

If algorithm doesn't exist, then undecidable problem.

Decidable problems of Finite Automata are :→

① Finitness of Finite Automata.

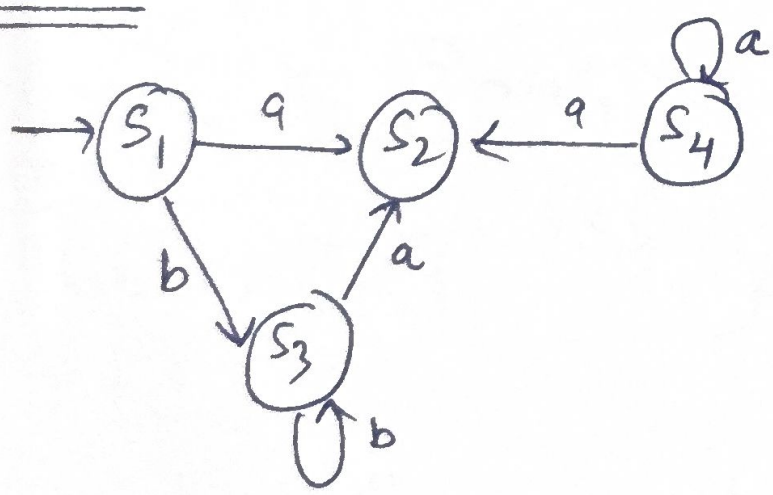
② Emptiness of Finite Automata.

③ Equality of Finite Automata. and few more.

Definition

Finiteness :→ Test verify

Emptiness :→



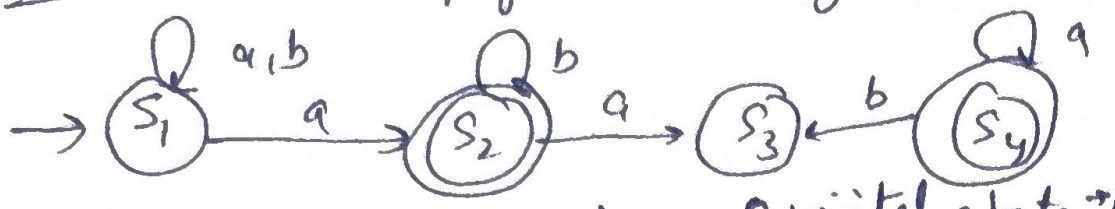
Algorithm :→

① eliminate those states not reachable from initial state.

② In the remaining Finite Automata, if atleast one final state is present, then non-empty language is accepted.

otherwise it is known as empty.

Problem: Check empty or non empty.



from 0, initial state → (S_1) here.

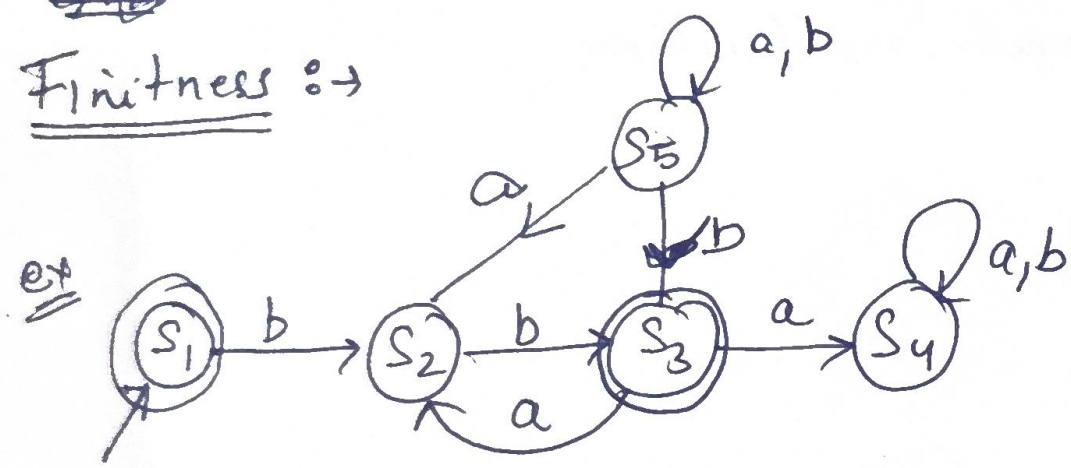
Delete non-reachable. [S_4 is final but not reachable]

(1)

(2) ~~Finding~~ Atleast one reachable state, hence non-empty language.

~~Note~~

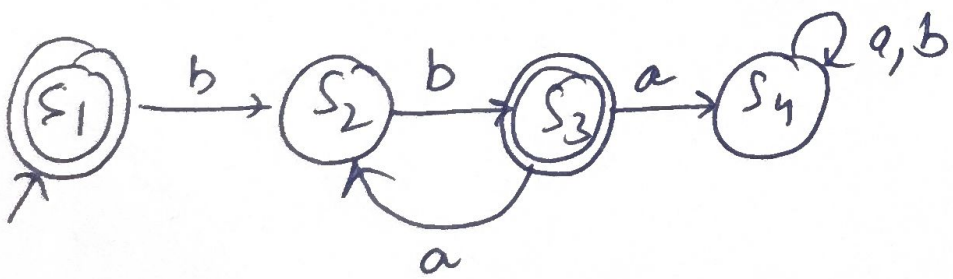
Finitness :→



ex

Algorithm

① Delete unreachable state first :→ (S_1) ooo (S_5) X please remove also transitions.



[S_5] is deleted

② Now, check dead state, in this S_4 is there, because after reaching S_4 you cannot return to S_1 after seeing a or b.

$S_1 \rightarrow [S_2]$ can you go to final.

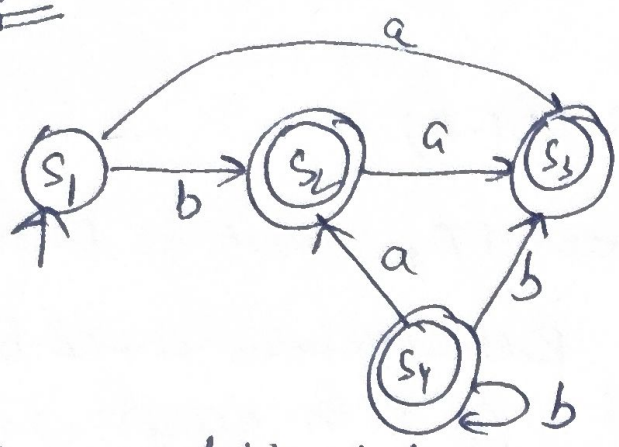
③ Atleast one final is there, accepting something and loop is there, then it is accepting infinite language otherwise, finite language

Algorithm

① Delete unreachable state first

② Check Dead state.

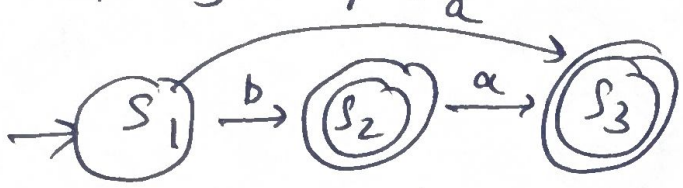
Example



Delete unreachable state

① $S_1 \xrightarrow{b} S_2$ ✓ , $S_1 \to S_3$ ✗ , $S_1 \to S_4$ ✗

Delete dead state



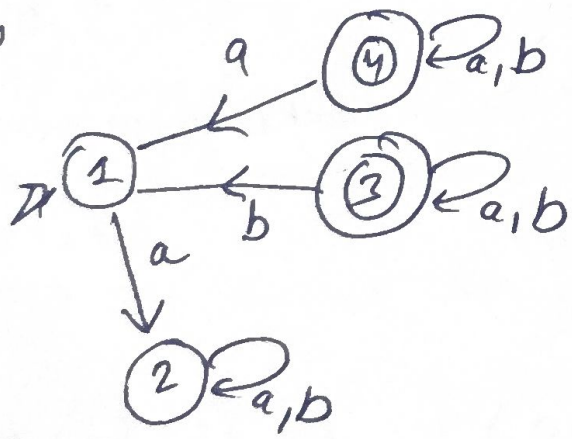
✗ ② ~~$S_1 \to S_2 \to S_3$~~ you can't reach to S_1, S_2 in any way.

③

This has one final state, no loops is there and accept only "ba" and "a" as string. Hence, finite.

$L_1 = \{ ba, a \}$

Example :→

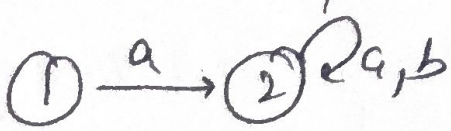


if Empty Language, finite lang.

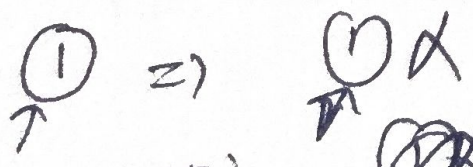
if non-empty language, may be finite or infinite language.

Delete unreacable

$1 \to 4$ ✗ , $1 \to 3$ ✗ , $1 \to 2$ ✓



Remove dead state, (2) → you cannot reach final state.



② In this, no state is there, as no final state.

③ no empty language, hence finite language.

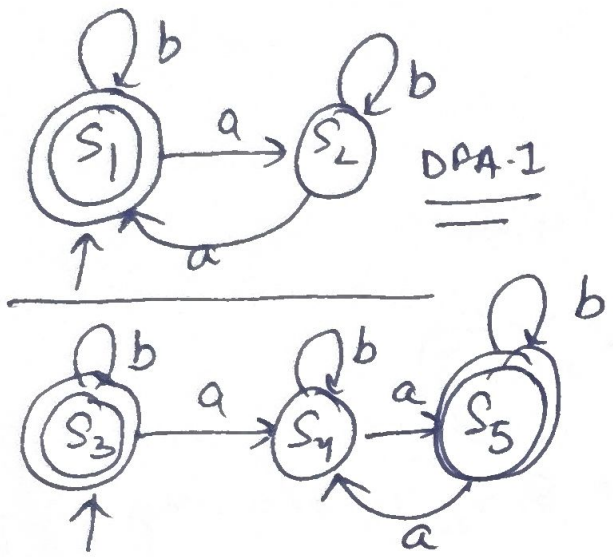
Equality of FA :- (Lecture 10, 1:20:00)

Input for equality

① two finite automata (DFA)

If either of them were given NFA, convert to DFA.

Both accepts same language. It doesn't matter if they have the same number of states or final state.



① Take initial state S_1 and S_3 in transition table,
Then input on "a" (S_1, a) (S_3, a)

N → non-final
F → final.

	a	b
$(S_1\ S_3)$	(S_2, S_4) N N	(S_1, S_3) F F
(S_2, S_4)	$(S_1\ S_5)$ F F	(S_2, S_4) N N
$(S_1\ S_5)$	(S_2, S_4) N N	(S_1, S_5) F F

take new person,

① (S_2, S_4)

② $(S_1\ S_5)$ is new

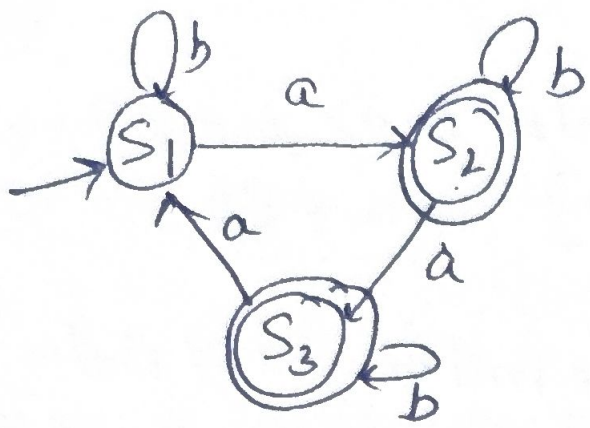
① Further, no new members.

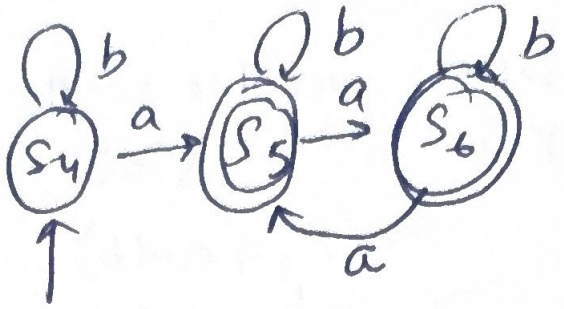
Indication is that DFA-1 and DFA-2 pair must have either final states or non final states but not accepts both.

You have to check for some input a or b,

same language.

Ex

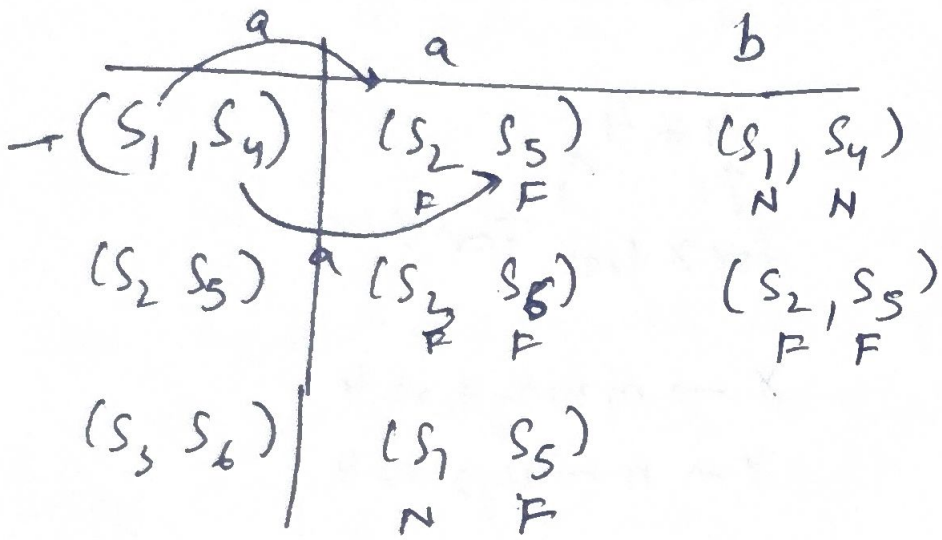




$(S_1, a) \Rightarrow (S_2)$

$(S_4, a) \Rightarrow (S_5)$

(S_2, S_5)



~~S_1 S_5~~ • $S_3 \rightarrow S_1$ (non final) & they are behaving differently
Hence we can stop.

• $S_5 \rightarrow S_6$ (final)

To DFA are equal :→ $L_1 = L_2$ (language must be same)

(#) Isomorphic DFA e→
Two DFA are equal, states are also same, Language accepted by both are same. but labels are different.

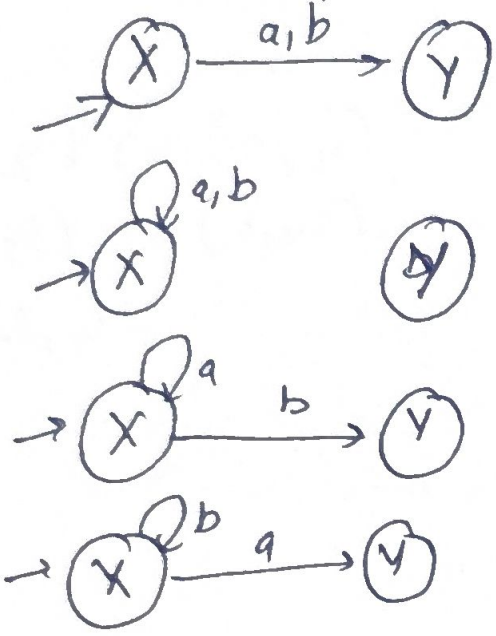
① Two DFA are equal, may not be isomorphic automatas.

② But Isomorphic DFA should be equal.

◉ Isomorphic graphs = Isomorphic DFA

How many DFAs possible with two states X and Y?
where X is initial state $\Sigma = \{a, b\}$ Ans: 64

for (X, a and b)



Same will be for Y states.
Just interchange X and Y.

4 * 4
↑ ↑
for X tran For Y tran

X → m ways => 4
Y → n ways => 4

Now, we have to take final states set.

for final state : { { }, {X}, {Y}, {X, Y} }
: 4 final state possibles.

4 * 4 * 4 => 64 possible.

0 - final → 16
X → => 16
Y → => 16
X, Y → => 16

4 × 16 => 64 DFA.

(X, a) => X / Y => 2 choices
(X, b) => X / Y => 2 choices
(X_2, a) => X / Y => 2 choices
(X_2, b) => X / Y => 2 choices

Atleast 1 final state => 48
" 2 states => 16
Atmost 2 states => 32

Final states as same as above

=> (2 * 2 * 2 * 2) * 4

=> 64 DFA possible

00
01
10
11

0 → non-final
1 → final

* If initial states is not mentioned, then you have a choice but remember don't choose both the states as initial.

	δ	a	b	
2	X	2	2	$\Rightarrow 2^2$
2	Y	2	2	$\Rightarrow 2^2$

$2^2 * 2^4 \Rightarrow 2^6 \Rightarrow \underline{64}$

↑ For final states set ↑ from transition table

Problem: How many DFA are possible for X, Y and 2 states, where X is initial state over alphabet a, b?

	a	b	
m ways	X	3	3	$\Rightarrow 3^2$
n ways	Y	3	3	$\Rightarrow 3^2$
l way	Z	3	3	$\Rightarrow 3^2$

transitions possibles ↓6 Final states initial state is fix

$\Rightarrow 3^6 * 2^3 * \boxed{1}$

$\Rightarrow$

① X —a,b→ Y

② X —a,b→ Z

③ X —a→ X, X —b→ Y

④ X —b→ X

On(X, a or b) $\Rightarrow$ 9 ways

Atleast one final state $\Rightarrow 7 * 3^6 * 1$

0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

Atmost one final state $\Rightarrow \boxed{3^6 * 1 * 2}$ ways

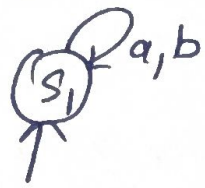
Exactly two states as final :→ $3 * 729$

$\underline{X, Y}$ → ${}^3C_2 * 729$

X, Z

~~How many DFA~~

How many DFA is possible over alphabet a, b where it accept no language is empty. $L = \phi$, with two states.



$n(L) = 0$

For non empty language, but whose length is zero / ~~string~~

$L = \{\epsilon\}$ $n(L) = 1$

length of ϵ = "0"

As we do not to have to accept any language :→

	a	b
X (pa)	X/Y	X/Y
Y	X/Y	X/Y

→ 2^2

→ 2^2

→ $2^2 * 2^2 * 1$ ← zero final states

→ 16 DFA possible.

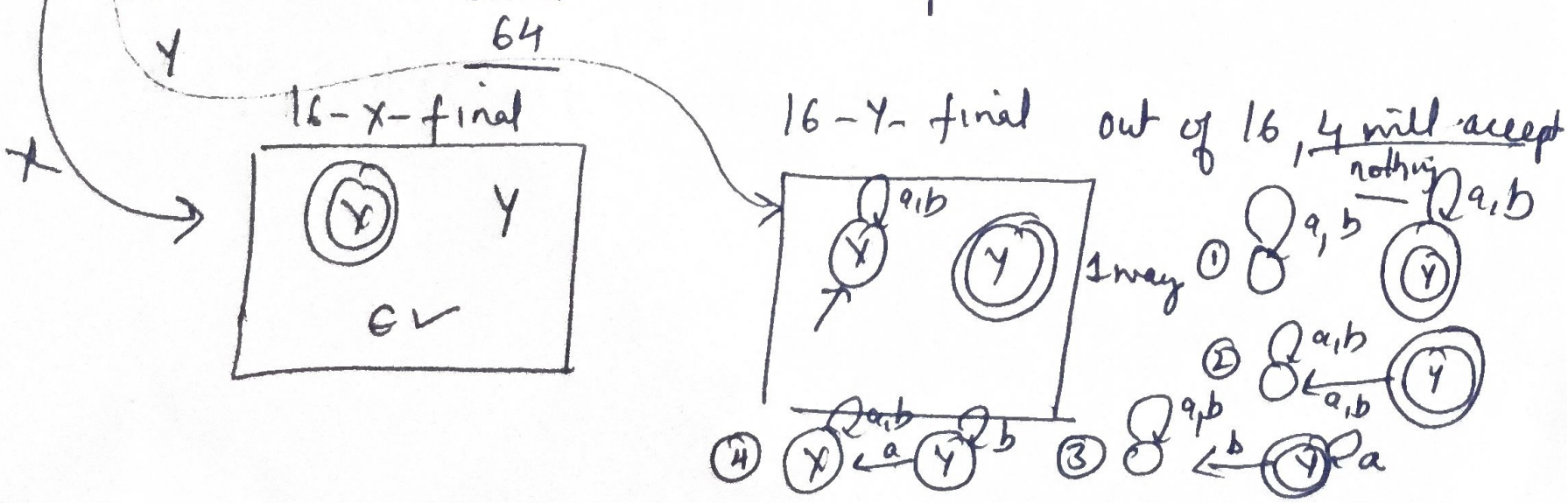
① 0 - S - F → 16 → ϕ [empty language]

② 1 → S - F → X → 32, Y

③ 2 → S - F → 16 → Σ^* accept or $(a+b)^*$

Final is there but not reachable

64

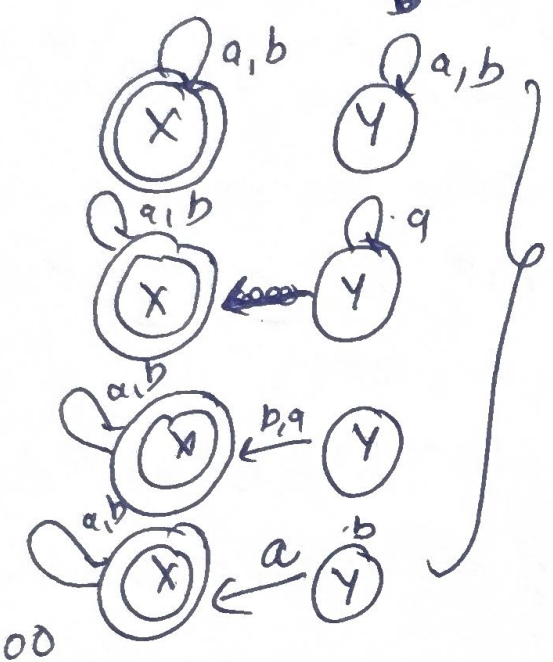


Wrong

How many DFA's with two state X and Y where X is initial state that accepts ~~everything~~ everything over alphabet {x, y}.

① Both final => 16 states.

② One of the final :→ X



4 states also there.

16+4 => 20 DFAs possible

Lecture 11
1:52:00

Regular Expressions

① All strings contains exactly two a's.

L = { aa, baa, aba, aab, bbaa .... }
(baa, aba, aab: 3)

=> [(a+b)* a (a+b)* a (a+b)*]* minimal

ϵ a ϵ a ϵ => aa

② All strings contains atmost two a's.

L = { ϵ, a, b, ab, ba, bb, aa, bbb.... }

b* (a+ϵ) b* (a+ϵ) b*

(3) Atleast two a's.

$L = \{ \underbrace{aa}_{2}, \underbrace{aba, baa, aab}_{3} \ldots \}$

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

minimum two a's $b^* a b^* a b^* (a+b)^*$

Remember R.E's are not unique.

or

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

or

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

Give R.E, that generate all strings of a's and b's in which no two a's are ~~side~~ together.

$L = \{ \epsilon, a, b, ab, ba, bb, abbb, \ldots \}$

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

$(b+ab)^*$

$(b+ab)^0 \rightarrow \epsilon$

$(b+ab) \Rightarrow b, ab$

$(b+ab)^2 \Rightarrow (b+ab)(b+ab)$

$bb, bab, abb, abab$

$(a+\epsilon)(b+ba)^* \rightarrow \epsilon.\epsilon \Rightarrow \epsilon$

$a.\epsilon \Rightarrow a$

$\epsilon.b \Rightarrow b$

$\epsilon.ba \Rightarrow ba$

$a.b \Rightarrow ab$

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

$\epsilon.a \Rightarrow a$

$b.\epsilon \Rightarrow b$

$ab.\epsilon \Rightarrow ab$

$b.a \Rightarrow ba$

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

$b . ab . \epsilon \Rightarrow bab.$

Problem where two a's ~~are not allowed~~ together are not allowed and starting with a.

$L = \{ a, ab, abb, aba, abbb, abab, abba, \}$

$\underline{a}(b+ba)^* \Rightarrow a.\epsilon \Rightarrow a$

$a.b \Rightarrow b$

$a.b.b \Rightarrow abb$

$\underline{a}.\underline{b}.a \Rightarrow aba$

Lecture 11 2:20:00

Problem where two a's are not together and two b's are not together.

$L = \{ \epsilon, a, b, ab, ba, aba, bab, \ldots \}$

$(ab+ba)^* \Rightarrow \epsilon \Rightarrow \epsilon$

$aba \Rightarrow ab$

$ba \Rightarrow ba$

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

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

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

$\epsilon . \epsilon . \epsilon \Rightarrow \epsilon$

$\epsilon . \epsilon . a \Rightarrow a$

$b . \epsilon . \epsilon \Rightarrow b$

$\epsilon \quad ab \quad \epsilon \Rightarrow ab$

$b.\epsilon.a \Rightarrow ba$

$b.(ab)^2.a \Rightarrow bababa$

$(ab)(ba)^*$ Wrong

Problem :→ $0^*(10^*)^*$ is same as :→

a.) $(1^*0)^*1^*$

b.) $(0+1)^*10(0+1)^*$

c.) $0+(0+10)^*$

d.) NOA

a.) $0^*(10^*)^*$ L: { ∈, 0, 0....0, [illegible], 1, 11 [illegible]

b.) $(1^*0)^*1^*$ L: { ∈, 1,

$(\underbrace{\in}_{\in}.0).1$ Σ^*: $(0+1)^*$

Σ^*: $\Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3$ .....

∈ 0 00 8 strings
1 01
10
11

$0^*(10^*)^*$

only zero ← one followed by zero

but only .0 is not possible ~~10 is not possible~~

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

First check first five strings. of the questions.

2) ∈, 0, 1, 10, 11,

2) $\Sigma^0, \Sigma^1, \Sigma^2, \Sigma^3$, ⇐ this is sufficient

① $(a+b)^*$ = $(a+b)^{10}$ Invent 10 a's

$(a+b)(a+b)^{10}$ 5 times

⇒ a ... a ⇐ 10 times

② $(a+b)^* = (a+b^*)^*$

⇓

10 b's ⇒ $(a+b)^{*1} \Rightarrow (a+b^1)^1, (a+b^2)^1, (a+b^3)$

only b's, b, bb, bbb ...

$(a^*+b^*)^*$ ⇒

Important Expressions :→

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

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

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

⇒ $(a^*b^*)^* = (a^*b)^*a^*$

X

X	XX
a^*b^*	$(a^*b^*)(a^*b^*)$
only a's ✓, only b's ✓	↓ ↓
	∈ ∈

Not contain "100" as substring :→ $L_2 = \{\epsilon, 0, 1, 01, 10, 101, 11, 00 - \}$

① $0^*(0+1)^*$ ⇒ $(0+1)^*$ will generate 1, 11, ... all the string

② 0^*1^*01 ⇒ It will generate without "00" but ∈ is not there.

③ 0^*1010^* ⇒ ⊗ 10100 invalid.

④ $0^*(10+1)^*$ ⇒

∈, is not possible $L_4 = \{\epsilon, 0, 1, 00$.

$L_2 = \{$ 01, 001, 101, zeroes possible, ...

$L_3 = \{$

0^*1010^*, 010100

∈ ⇒ ∈, minimum three 3 length.

For Regular Expression, valid string should be generated and invalid string must not be generated.

Problem

substring 1101 does not belong to Regular Expression?

① $110^*(0+1)$

$L_1 = \{1101\}$ ✓

② $1(0+1)^*101$

$1(0+1)^0101$

$\Rightarrow 1101$ ✓

③ $(10)^*(01)^*(00+11)^*$

$\epsilon \quad \epsilon \quad (00+11)(00+11)$

1100
0011 but
1101 not possible

④ $(00+(11)^*0)^*$

a

1101101

$(00+(11)^*0)$

a b ✓

11.1.0 11 0

Problem

a.) $(00)^*(\epsilon+0)$ ⓑ $(00)^*$ ⓒ 0^* ⓓ $0(00)^*$

which are equal.

$L_a = \{\epsilon, 0, \quad \approx 0^*$

even length ⇒ 00, 0000

odd length ⇒ 000, 00000

$L_a = L_c$ are equivalent

$L_b = \{$ only even length possible $\}$

$L_c = \{\epsilon, 0, 00, 000 \ldots\}$

even and odd possible

$L_d = \{0, \epsilon \quad\}$ ϵ not possible.

Which are equivalent?

(a) $1(01)^*$ and $(10)^*1$

(b) $x(xx)^*$ and $(xx)^*x$

(c) $(ab)^*$ and a^*b^*

(d) $x^+ \cdot \phi \; x^* x^+$

a.) $L_1 = \{1, 101, 10101$
$= \{1, 101, 10101, \ldots\}$

$1(01)^* = (10)^*1$

$(ar)^*a = a(ra)^*$

b.) $L_2 = \{x, xxx, xxxxx$
(even & odd is generate)

$2^{nd} \Rightarrow \{x, xxx, \ldots\}$ string of two length

c.) $L_3 = \{\epsilon, ab, abab$
length of even string only

$\Rightarrow \{\epsilon, a, b, aa, ab \ldots\}$

20-R.E
$20 \rightarrow$

d.) $L_4 = \{x, xx, xxx \ldots\}$

$L_5 = \{x, xx, xxx \ldots\}$

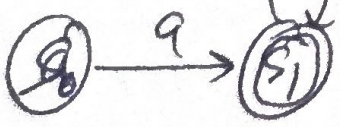
* Formulae :→ ara, starting and ending letters must be same.

Then, $(ar)^*a = a(ra)^*$

Problem $L = \{a^{nk} \mid k>0, n>0\}$

$L = \{a, a^4, a^9, \ldots\}$

$a \; a^+$



m-DFA

n	k
1	→ (1 .. n)
2	→ (

$n * n \Rightarrow n^2$

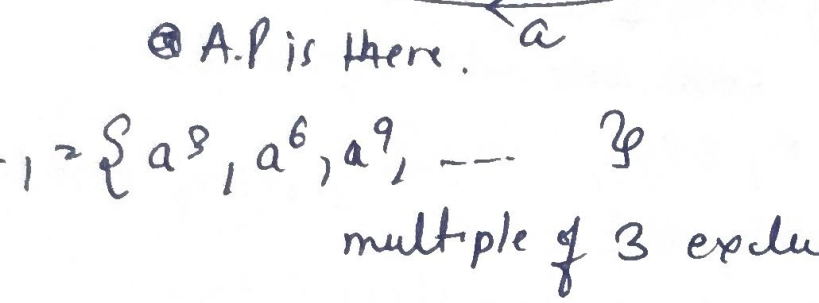
Problem

$L = \{ a^{nk} \mid n>0, k>0 \}$

k is constant

$L_1 = \{ a^5, a^{10}, a^{15}, a^{20} \ldots \}$ let $k=5$

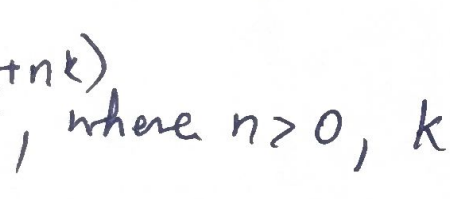
$\Rightarrow a$ multiple of 5



A.P is there.

For $k=3$, $L_1 = \{ a^3, a^6, a^9, \ldots \}$

multiple of 3 excluding 0



* (k+1) states for above problem.

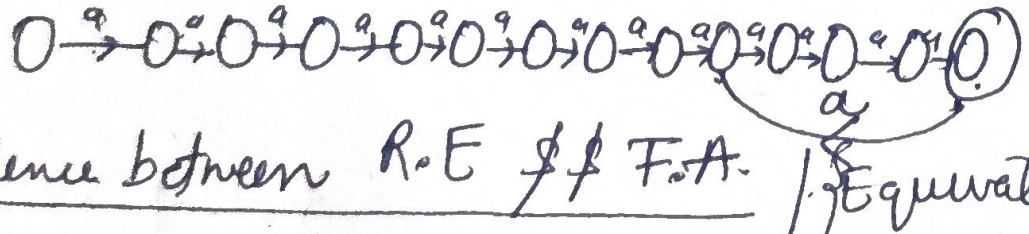
Problem :→ $L = \{ a^{(8+nk)}$, where $n>0$, $k=5 \}$

$L = \{ a^{8+5}, a^{(8+10)}, a^{(8+15)}, a^{(8+20)}$

$a^{13}, a^{18}, a^{23}, a^{28}$

difference of 5, AP series.

14 states



Equivalence between R.E & F.A. | Equivalence between NFA and DFA → meaning for every NFA there exist a DFA

Regular Expression to Finite Automata :→

(Used by Compiler)

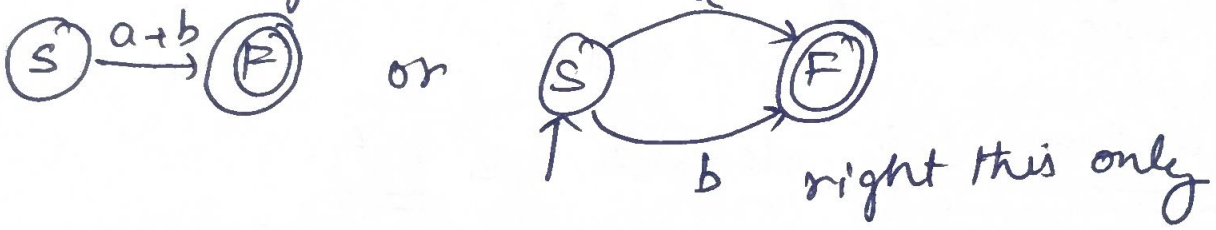
① Lexical → F.A
② Syntax Analyzer ⇒ PDA
③ Semantic ⇒ Turing Machine

* Manual Checking is possible., to do that.
If worst case, convert R.E to F.A.

Input R.E ⟶ Output F.A → DFA
→ NFA ⟶ DFA.

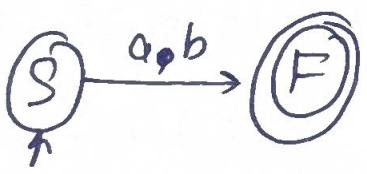
Ex. 1 ① $(a+b)$, $L_1 = \{a, b\}$

By read a , you go to final state
By reading b, you go to final state.



right this only

② $a.b$ = concatenation

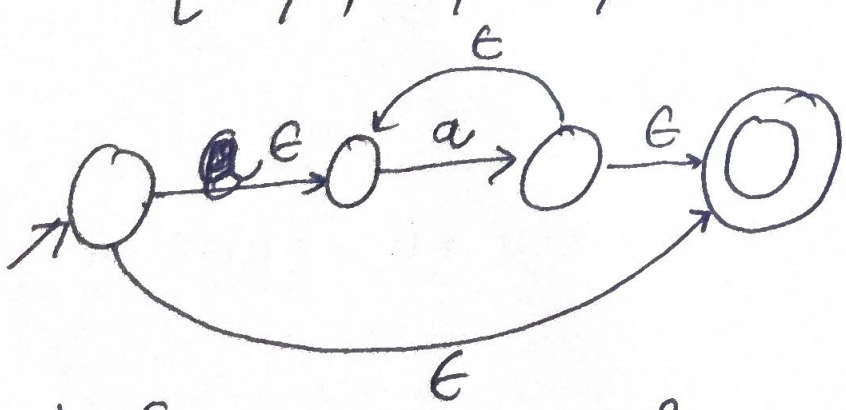


both are necessary and order is important.

$a.b = a$ ✗
$a.b = b$ ✗
$a.b = a.b$ ✓
$a.b = b.a$ ✗

③ Kleene Closure

$a^* = \{\epsilon, a, aa, aaa, aaaa, \ldots\}$



This is known as epsilon-NFA as ϵ is also allowed

$a^+ = \{a, aa, aaa \ldots\}$, just remove ϵ below.

$\delta : Q \times \Sigma \to Q$ (DFA) (only go to one state)

$\delta : Q \times \Sigma \to P(Q)$ NFA (Anywhere it can go)

State $Q = \{S_1, S_2\}$

Power set of Q, $P(Q) = \{\{\}, \{S_1\}, \{S_1, S_2\}, \{S_2\}\}$

$\Rightarrow 2^Q$

On some state seeing a alphabet, in DFA, we will go to exactly one state $Q = \{S_1, S_2\}$

In Power set, elements are sets

In $Q \times \Sigma \to Q$, elements are single states only.

For $\{S_1, S_2\}$

$Q \Rightarrow \{S_1, S_2\}$, $P(Q) = \{\{\}, \{S_1\}, \{S_1, S_2\}, \{S_2\}\}$ (set)

$\delta : Q \times \Sigma \to P(Q)$ or 2^Q (NFA)

Epsilon-NFA :→

$\delta : Q \times \{\Sigma \cup \in\} \to P(Q)$ epsilon-NFA

or

$\delta : Q \times \{\Sigma \cup \in\} \to 2^Q$ [remember not QQ]

For every $\in$-NFA, a R.E exists.

DFA

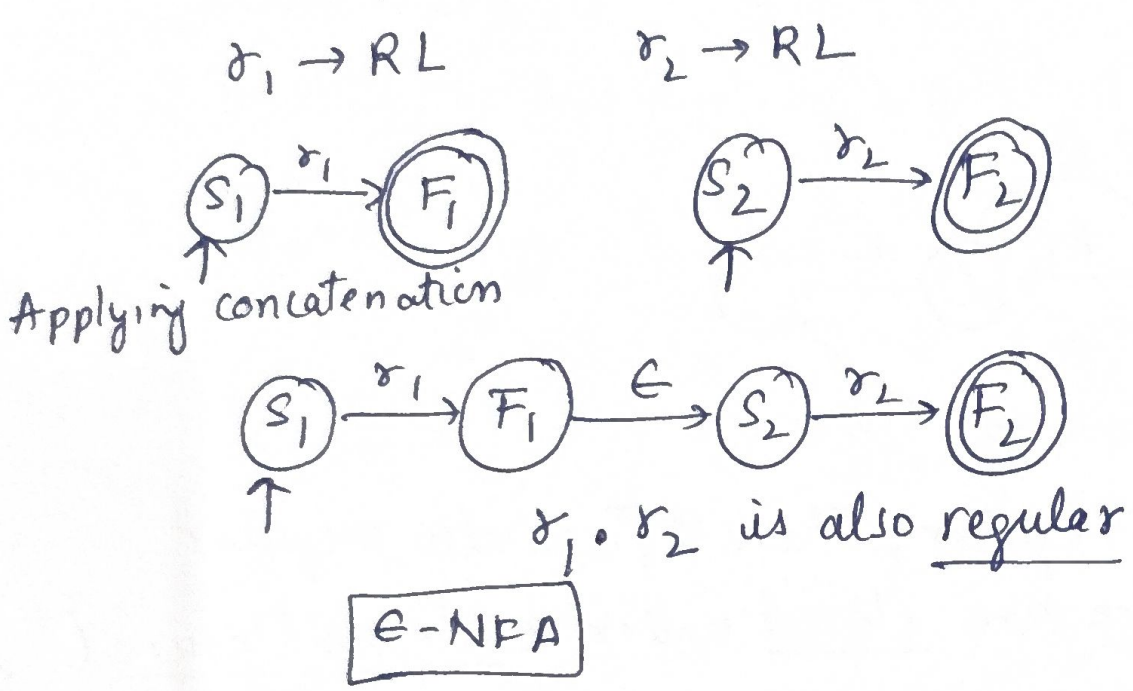
NFA

* Note :→
Power of $\in$NFA, DFA, NFA, R.E are all equivalent.

Constructing Difficulty :- ϵ-NFA < NFA < DFA

Regular Language properties :-

(1) If r_1 is regular and r_2 is regular, then $r_1 \cdot r_2$ (concatenation) is also regular.
So regular language are closed under concatenation operator (.)

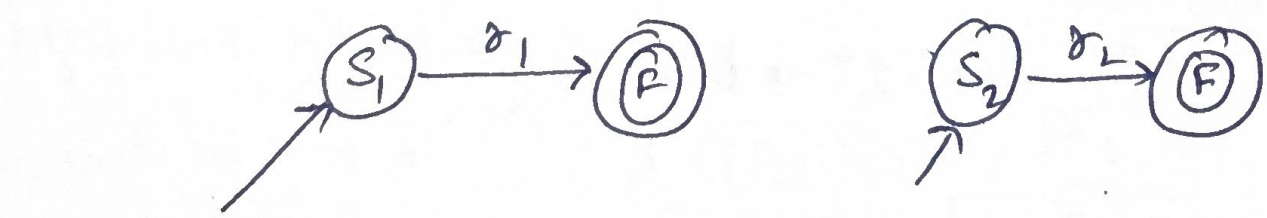


How to get closed under concatenation

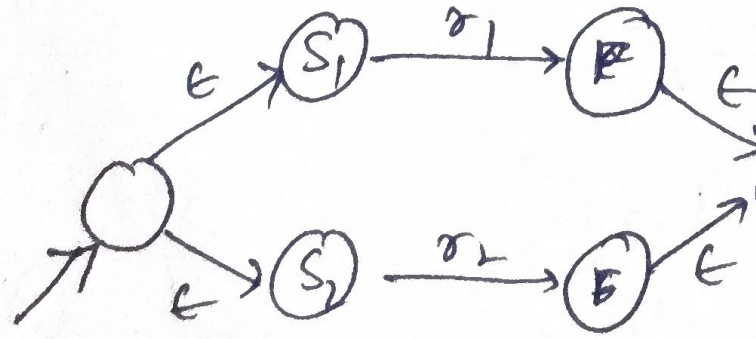
(1) remove r_1's, F_1 final.
(2) Add ϵ in between.
(3) Remove, S_2 as initial state.
(4) Make S_1 is start state.
(5) F_2 as final.

(2) If r_1 and r_2 are regular, then $r_1 + r_2$ is also regular.
So, regular languages are closed under union operation (+).

$r_1 \to RL$ $r_2 \to RL$



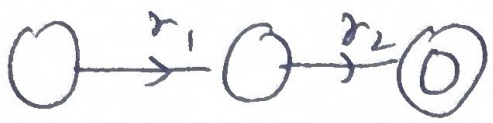
Applying union operator

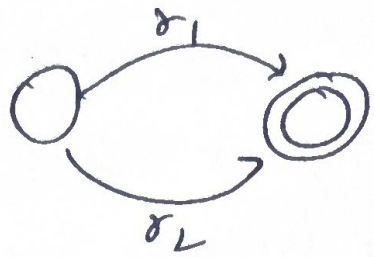


ϵ-NFA

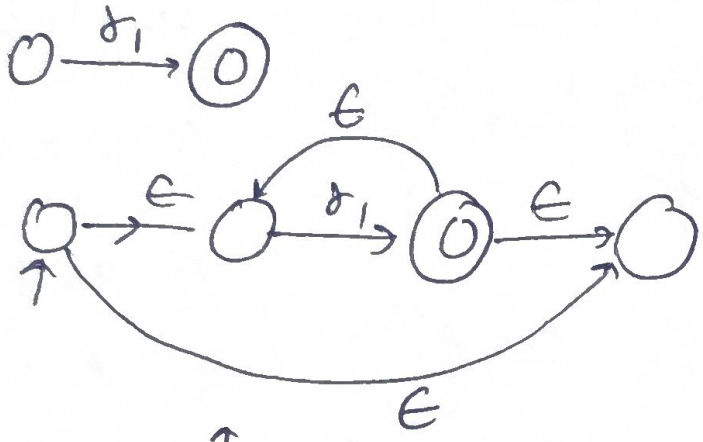
(1) Remove initial states S_1 and S_2
(2) remove finals
(3) Add new initial state
(4) Add new final state.

In NFA :→ ~~r~~ $r_1 \cdot r_2$ $r_1 + r_2$

① 



Constructing $r_2^* = \{ \epsilon, r_1, r_1 r_1, r_1 r_1 r_1 \dots \}$



III

Property :→ If r_1 is regular, then r_1^* is also regular.

So, regular language are closed under ⊛ Kleene Closure (*).

41:00 (L-13)

Example :→

Priority order :→ $\overset{\text{winner}}{*}$, . , +

$* \rightarrow H$

$\cdot \rightarrow M$

$+ \rightarrow L$

$r_1 = 01^* + 1$

$= \boxed{0 \cdot \boxed{1^*} + 1}$

$*$ → will be done first

then

$\cdot$ and $+$ will fight for 1^*.

then

$\cdot$ will be done second

then last union operator.

order of evaluation

$\boxed{1^*}$ or $1^* = a$

$\boxed{0 \cdot 1^*}$ $\boxed{0 \cdot a} = b$

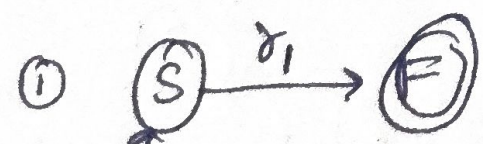
$\boxed{0 \cdot 1^* + 1}$ $\boxed{b + 1}$

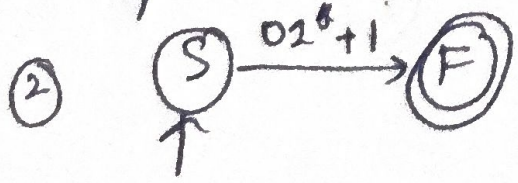
$L = \{0, 1, 01, 011, \dots\}$

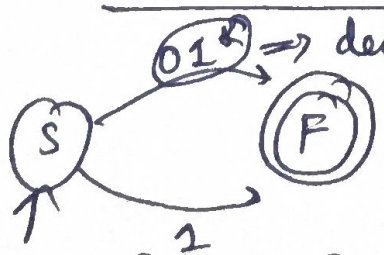
* Diagram is available in exam

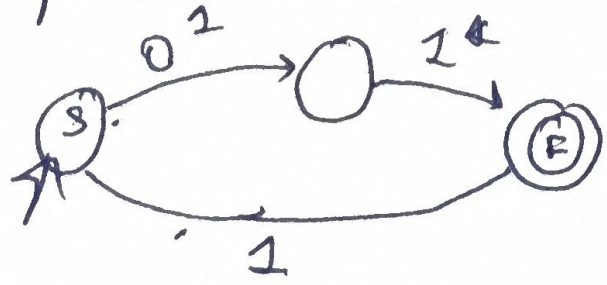
01^* → decompose more

Concatenation

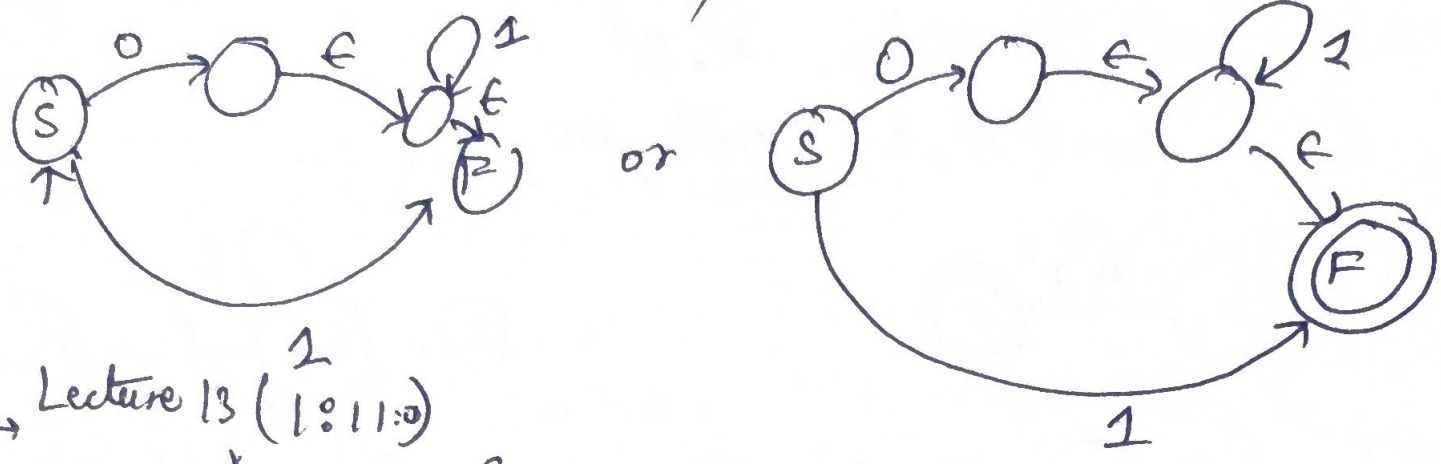
① 

② 

③ 

④ 

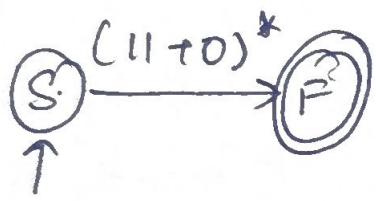
(5) Transition edges still has operator, so

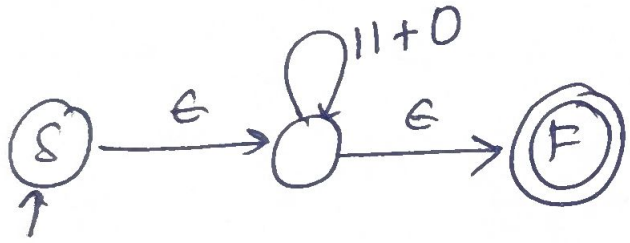


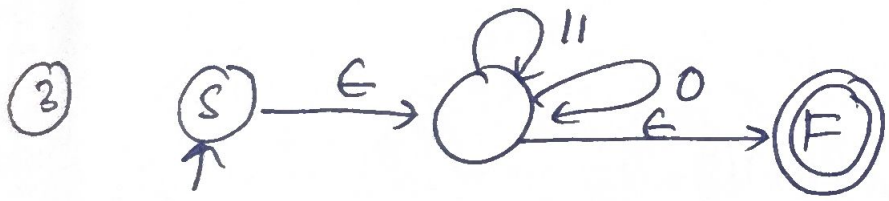
Example :→ Lecture 13 (1:11:0)

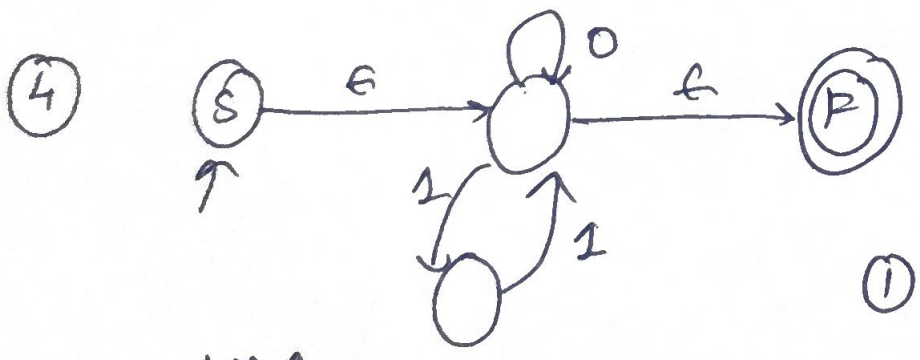
$\gamma = (1.1 + 0)^*$ $L_1 = \{ \epsilon, 0, 11, 110, 011, \ldots \}$

$a = 11$, $b = 0$

① 

② 

③ 

④ 

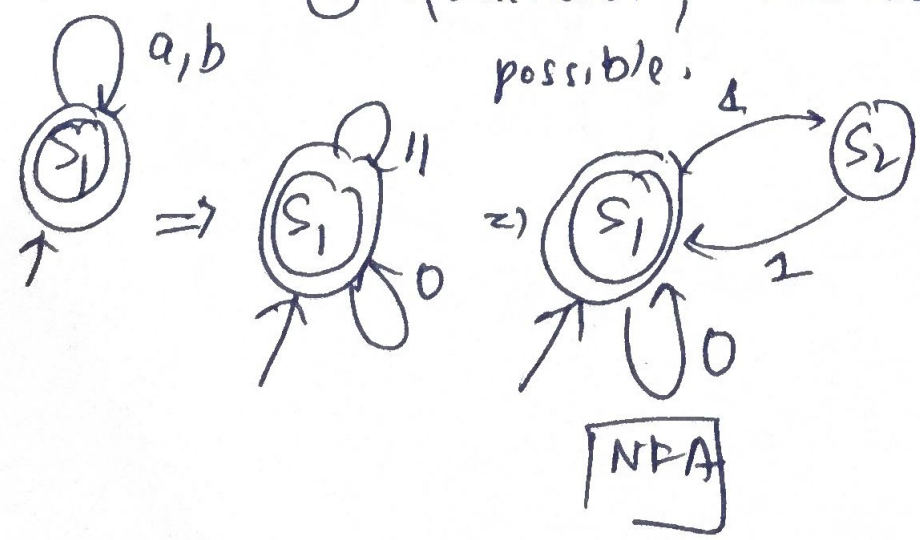
ϵ-NFA

① for * (Kleene closure)
② add ϵ ~~around~~ before and after ϵ.
③ Inside ( )*, add the loop.

Remember it is 1.1 not eleven.

① Check first five minimal string to verify.

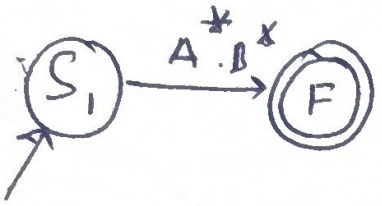
② Remember, more are also possible.

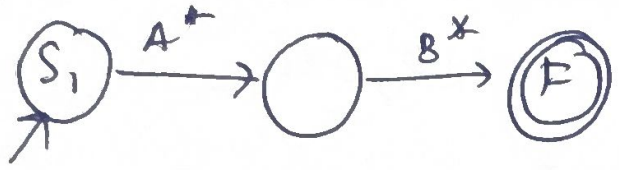
$(a+b)^* \Rightarrow$ 

Example:→

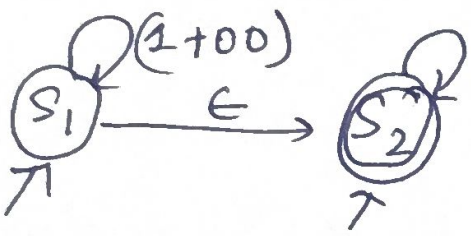
$\gamma = (1+00)^* . (0+11)^*$, $A^* . B^*$

$L_1 = \{ \epsilon, 1, 11, 00, 11, \ldots \}$

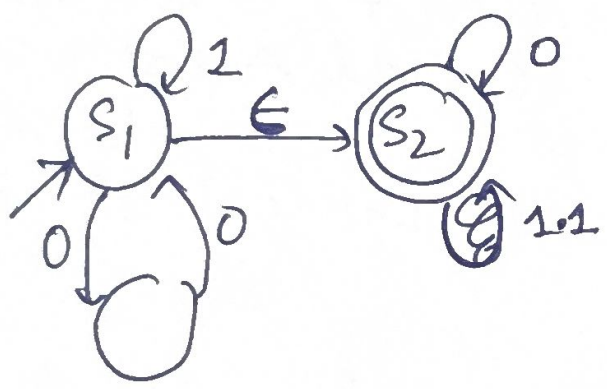
① 

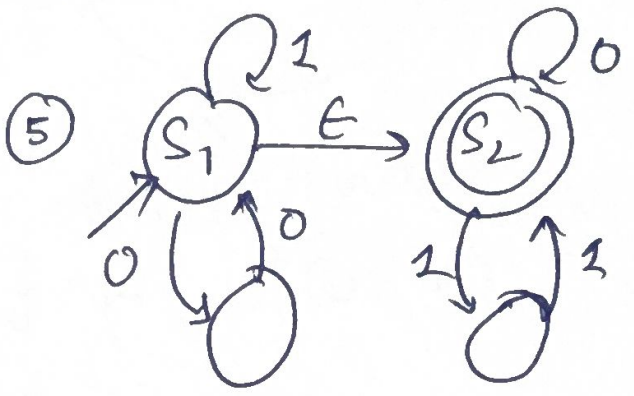
② 

Alternative

③ 

(Using previous example)

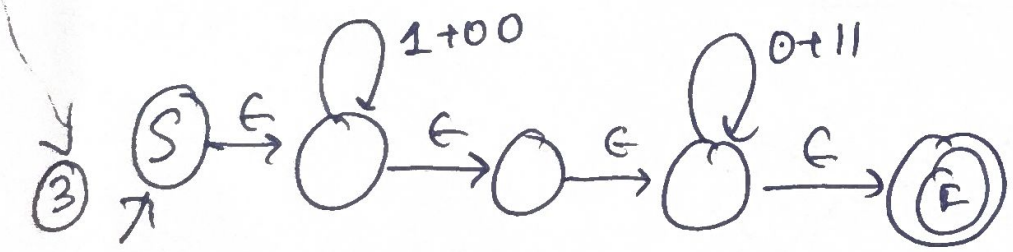
④ 

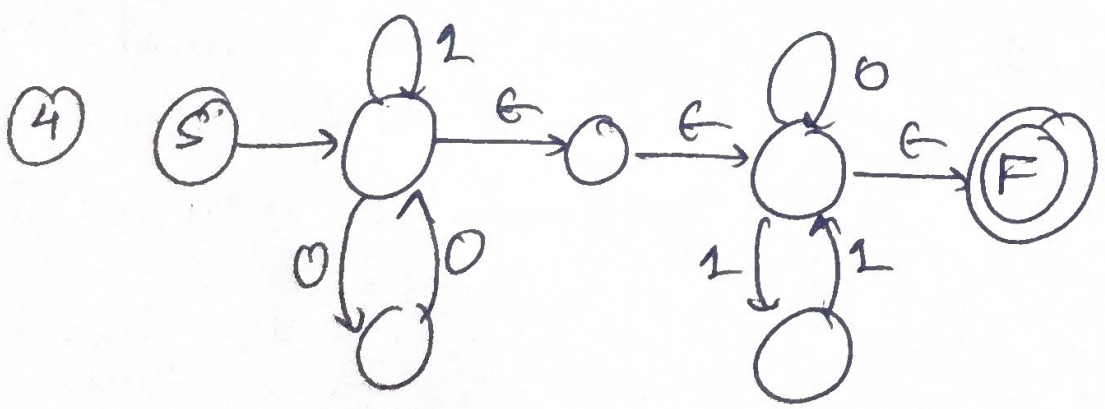
⑤ 

$L_1 = \{ \epsilon, 1, 11, 00, 10, 011, \ldots$

0 followed by 1

0 then 1 not possible

③ 

④ 

Example :→

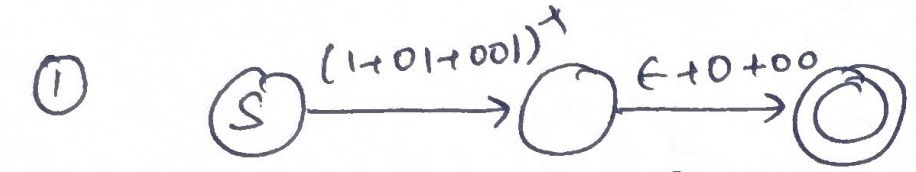
$r = (1+01+001)^* (\epsilon+0+00)$

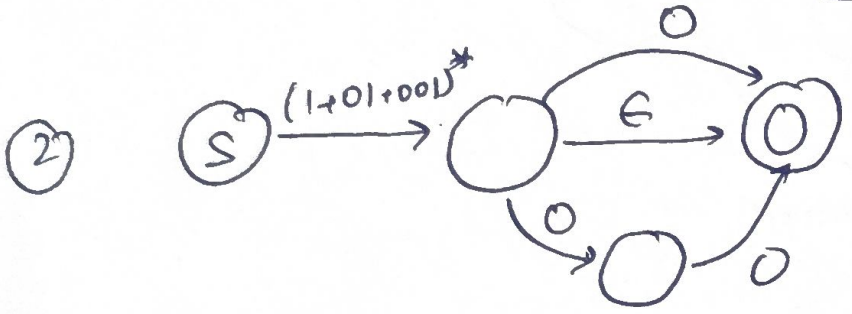
$L_1 = \{\epsilon, 0, 1, 00, 01, \ldots\}$

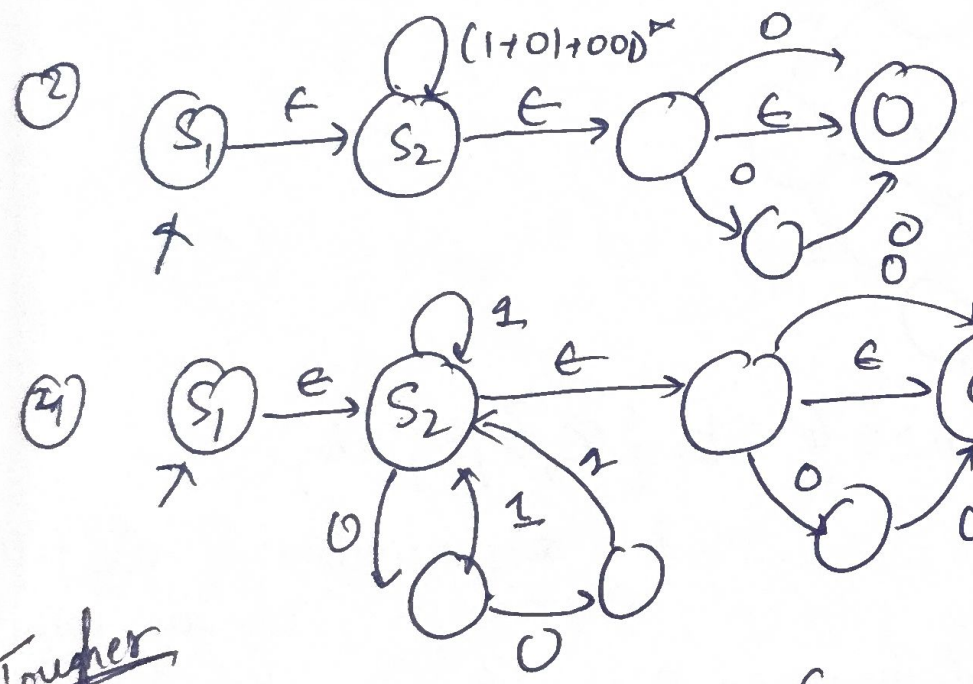
$11\alpha, 10\alpha, 000\alpha, \alpha$

000

000







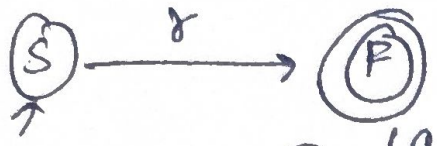
Tougher

Example :→ $r = [\overset{A}{00} + \overset{B}{11} + (01+10).\overset{C}{(00+11)^*}.(01+10)]^*$

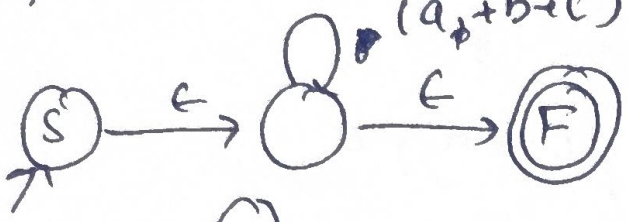
$L_1 = \{\epsilon, 00, 11, 0101, 1001$

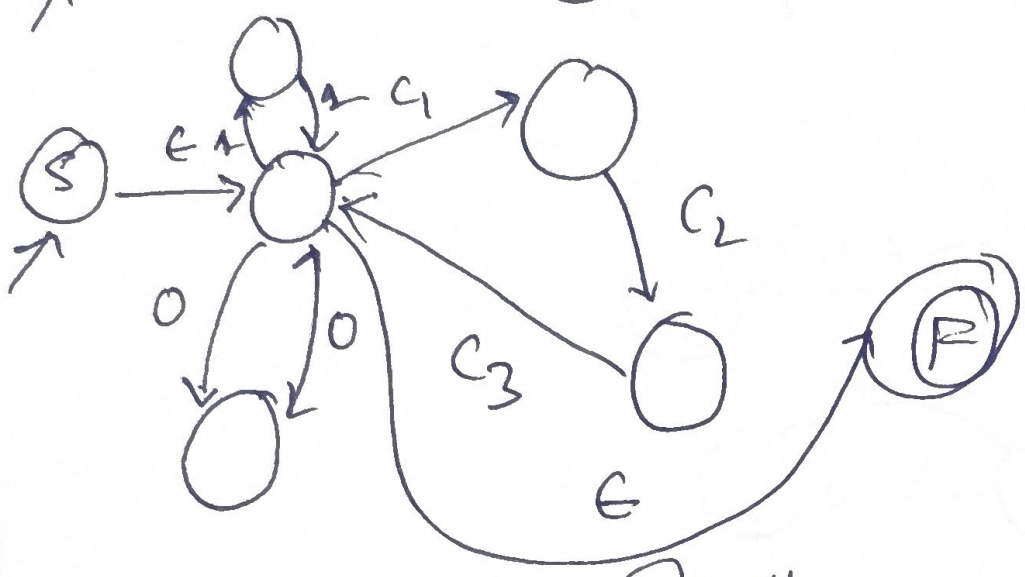
10 possible → ✓

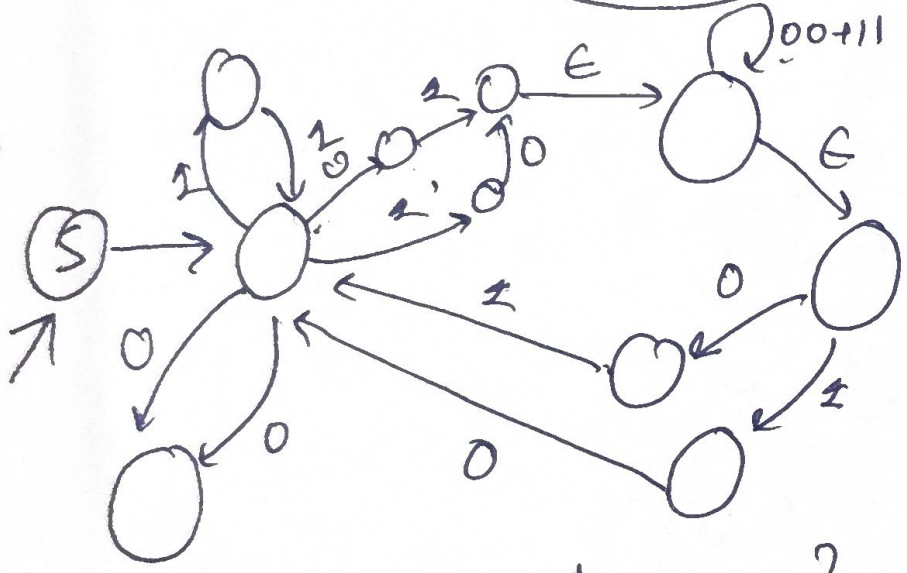
1111 possible → ✗

1. 

γ → ∅∅

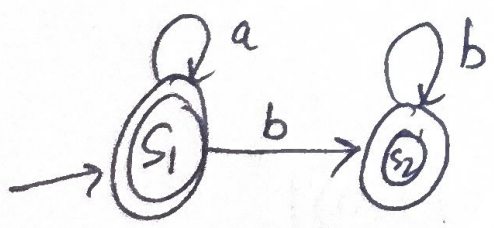
2. 

3. 

4. 

Lecture 14 :→ $L = \{ a^m b^n \mid m, n \geq 0 \}$ $L = \{ \epsilon, a, b, ab, \}$

. / any number of a's followed by any number of b's.



only necessary is covered.

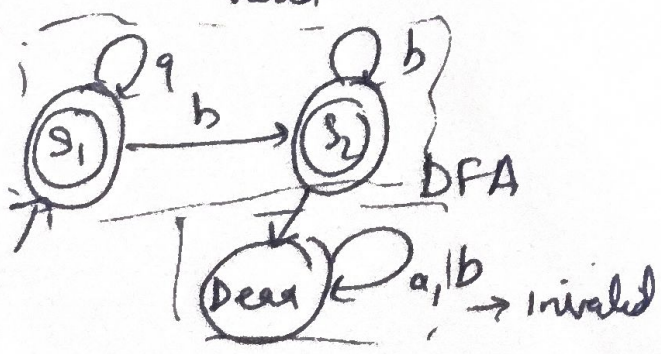
NFA

b.a✗

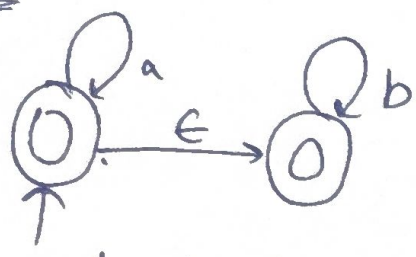
a.b✓

~~@ is Role Overed.~~

valid



Continue :→



ϵ-NFA $\quad \delta : Q \times \{\Sigma \cup \epsilon\} \longrightarrow 2^Q$

Notes

① If order is important, go to ϵ-NFA.

② epsilon-NFA

③ epsilon free-NFA = NFA

④ Finite Automata → All will come, DFA, NFA, ϵ-NFA

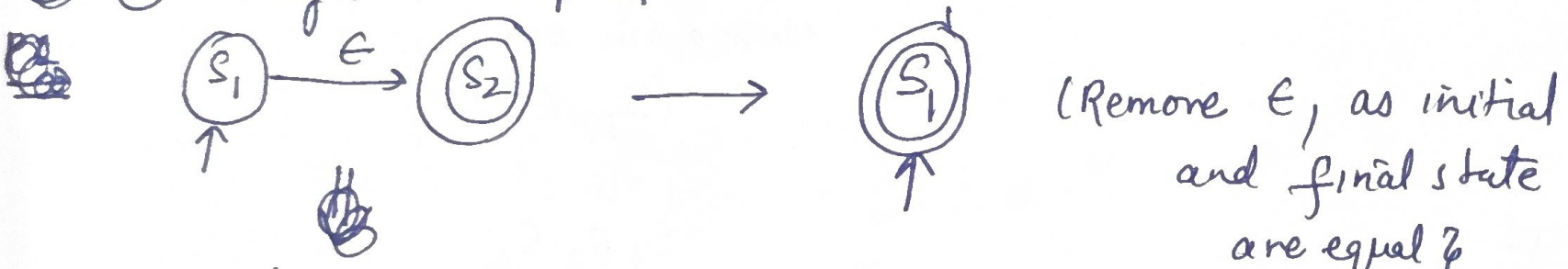
⑤ NFA => NFA, ϵ-NFA

⑥ Special type of NFA is DFA.

Conversion :→ (only for observation) Lec(14, 29:20)

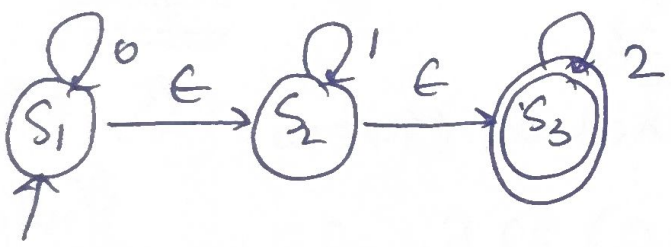
ϵ-NFA to NFA (No conversion will give min-NFA or min-DFA.)

① Only ϵ is accepted



(Remove ϵ, as initial and final state are equal)

②



Steps

① Write all states as it is. with initial state.

② Whoever is final in ϵ-NFA, same final state must be taken but more can be added, by checking ϵ transition edge.

$(S_1, \epsilon) \Rightarrow S_3 \Rightarrow$ final state

$(S_2, \epsilon) \Rightarrow S_3 \Rightarrow$ final state

S_1 and S_2, By reading ϵ, if we can go to the final state, then the state is also final state.

① $(S_1, 0)$ where we can we go.

Zero can be rewritten as $\epsilon^* . 0 . \epsilon^*$.

$\epsilon . 0 \Rightarrow 0$

$0 . \epsilon \Rightarrow 0$

$\epsilon^{25} . 0 . \epsilon^{25} \Rightarrow 0$.

wherever you can go, please check

$S_1, \epsilon^* \Rightarrow$ $(S_1, \text{nothing})$ (S_1, ϵ^1) (S_1, ϵ^2)

$(S_1, \epsilon) \Rightarrow S_1, S_2, S_3$

$\Downarrow$ $(S_1, 0)$, $(S_2, 0)$, $(S_3, 0)$

$(S_1, 0), (S_2, 0), (S_3 0) \Rightarrow S_1$ only

Now again on

(S_1, ϵ^*)

$\Downarrow$

$S_1 S_2 S_3$

Steps

① $(S_1, 0) \Rightarrow (S_1, \epsilon^*) \Rightarrow S_1, S_2 S_3$

$(S_1, \epsilon) \rightarrow S_1$

$(S_1, \epsilon) \rightarrow S_2$

$(S_1, \epsilon^2) \rightarrow S_3$

Check on zero

$(S_1, 0) \Rightarrow S_1$

$(S_2, 0) \Rightarrow$ —

$(S_3, 0) \Rightarrow$ —

Sequence

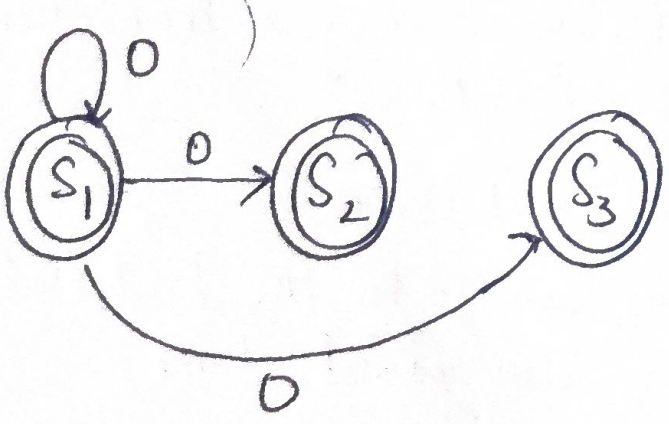
(S_1, ϵ^*)

$\Downarrow$ (output, 0)

$\Downarrow$

(output, ϵ^*)

$(S_1, \epsilon^*) \Rightarrow S_1 S_2 S_3$



② $(S_1, \epsilon^* 1 \epsilon^*)$

$(S_1, \epsilon^*) \Rightarrow S_1 S_2 S_3$

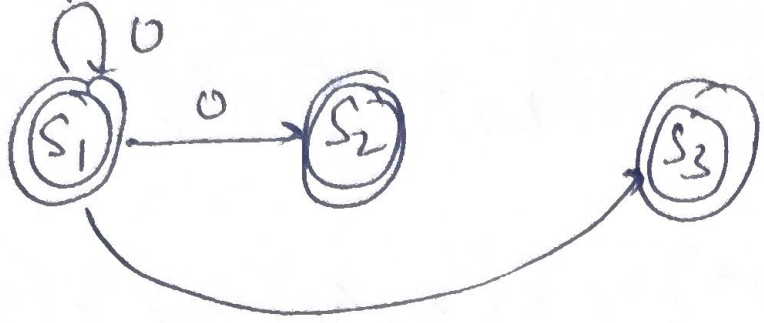
$\Downarrow$ $(S_1, 1), (S_2, 1), (S_3, 1)$

$(S_1, 1)$ $(S_2, 1)$ $(S_3, 1) \Rightarrow S$.

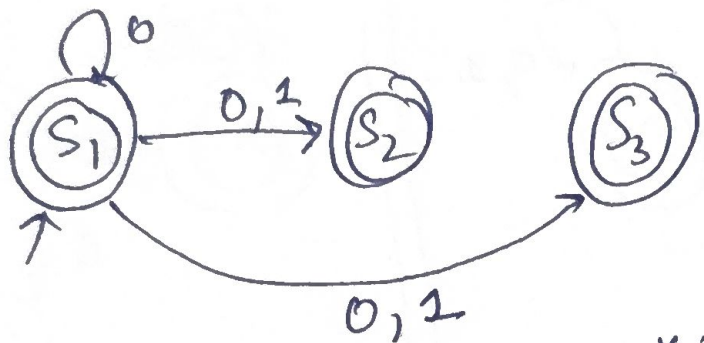
① $(S_1, 0) \Rightarrow (S_1, \epsilon^*) \xrightarrow{\text{transition on } \epsilon} S_1, S_2, S_3 \xRightarrow{0} (S_1, \epsilon^*) \Rightarrow (S_1 S_2 S_3)$ ← transition on 0

Finally $(S_1, \epsilon^* 0 \epsilon^*) \rightarrow S_1, S_2 S_3$

↓ transition on ϵ^*



② $(S_1, 2) \Rightarrow (S_1, \epsilon^*) \xRightarrow{0} S_1 S_2 S_3 \xRightarrow{1} (S_2, \epsilon^*) \xRightarrow{\epsilon^*} S_2, S_3$



$\epsilon^* 2 * e^*$

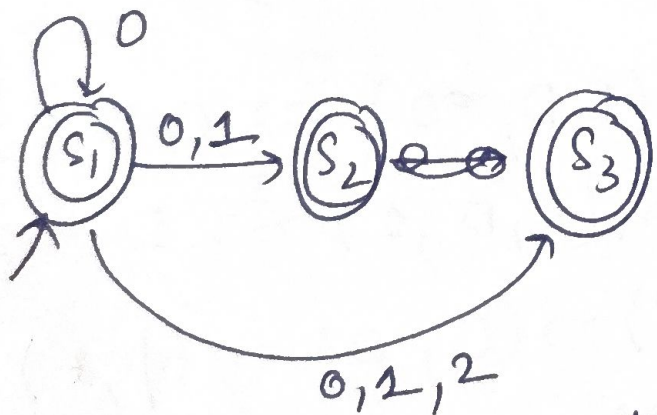
③ $(S_1, 2) \Rightarrow (S_1, \epsilon^*) \Rightarrow$ ~~$S_1 S_2 S_3$~~ $(S_1 S_2 S_3)$

↓↓ 2 transition

(S_3, ϵ^*)

↓↓ ϵ'

S_3

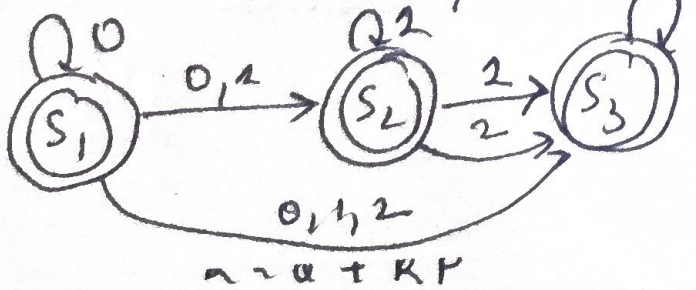


no state is there

④ $(S_2, 0) \Rightarrow (S_2, \epsilon^*) \xRightarrow{\epsilon^*} S_2 S_3 \xRightarrow{0} (\phi, \epsilon^*) \Rightarrow \phi$

⑤ $(S_2, 1) \Rightarrow (S_2, \epsilon^*) \xRightarrow{\epsilon^*} \phi S_2 S_3 \xRightarrow{1} S_2 \xRightarrow{\epsilon^*} S_2 S_3$

⑥ $(S_2, 2) \Rightarrow (S_2, \epsilon^*) \xRightarrow{\epsilon} S_2 S_3 \xRightarrow{2} S_3 \xRightarrow{\epsilon^* \epsilon^*} S_3$



Using next page

① $(S_3, 0) \Rightarrow (S_3, \epsilon^*) \overset{\epsilon^*}{\Rightarrow} S_3 \Rightarrow (S_3, 1) \overset{1}{\Rightarrow} (\phi, \epsilon^*) \Rightarrow \phi$

② $(S_3, 1) \Rightarrow (S_3, \epsilon^*) \Rightarrow S_3 \Rightarrow (S_3, 1) \Rightarrow (\phi, \epsilon^*) \Rightarrow \phi$

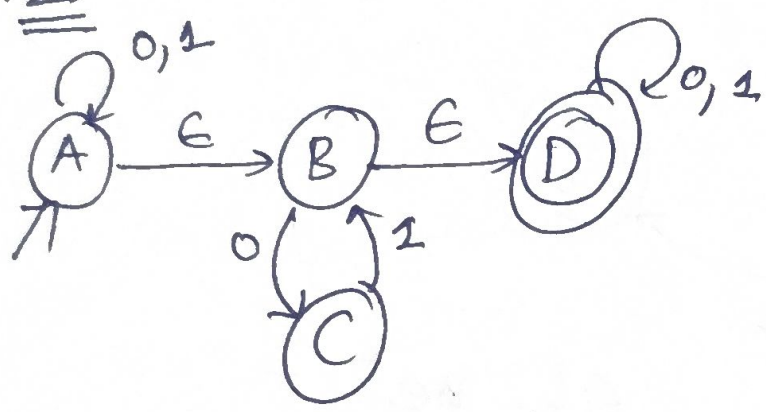
③ $(S_3, 2) \Rightarrow (S_3, \epsilon^*) \Rightarrow S_3 \Rightarrow (S_3, 2) \Rightarrow (S_3, \epsilon^*) \Rightarrow S_3$

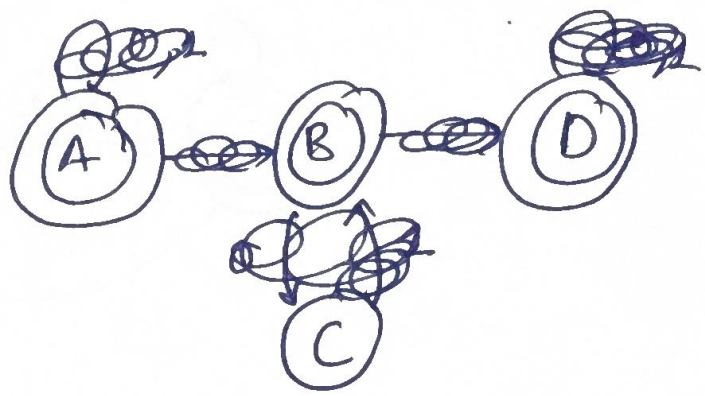
$\delta(S_0, \epsilon^* 1 \epsilon^*) \otimes (S_1, \epsilon^* 0 \epsilon^*), (S_2, \epsilon^* 1, \epsilon^*)$

(

$\delta(S_1, 0) = \delta(S_1, \epsilon^*) \Rightarrow \delta(x, 0) \Rightarrow \delta(y, \epsilon^*) \Rightarrow$ final

Example: 2





① $(A, \epsilon^* 0 \epsilon^*) \rightarrow (A, \epsilon^*) \overset{\epsilon^*}{\Rightarrow} (A, B, D, 0) \overset{0}{\Rightarrow}$ ~~A B~~ A C D $\overset{\epsilon^*}{\Rightarrow}$ ~~A B D~~

② $(A, \epsilon^* 1 \epsilon^*) \rightarrow (A, \epsilon^*) \overset{\epsilon^*}{\Rightarrow} (A B D, 1) \overset{1}{\Rightarrow} A D \overset{\epsilon^*}{\Rightarrow}$ A B ~~D~~ C D
A B D

③ $(B, \epsilon^* 0 \epsilon^*) \rightarrow (B, \epsilon^*) \Rightarrow (B, D, 0) \overset{0}{\Rightarrow} C D \overset{\epsilon^*}{\Rightarrow} C D$

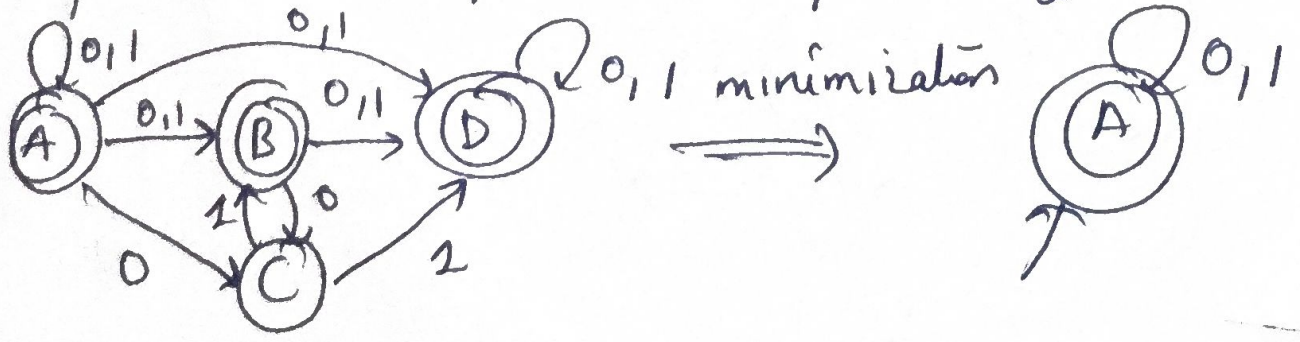
④ $(B, \epsilon^* 1 \epsilon^*) \rightarrow (B, \epsilon^*) \Rightarrow (B, D, 1) \overset{1}{\Rightarrow} D \overset{\epsilon}{\Rightarrow} D$

⑤ $(C, \epsilon^* 0 \cdot \epsilon^*) \rightarrow (C, \epsilon^*) \Rightarrow (C, 0) \Rightarrow (\phi, \epsilon) \Rightarrow \phi$

⑥ $(C, \epsilon^* 1 \epsilon^*) \rightarrow (C, \epsilon^*) \Rightarrow (C, 1) \Rightarrow (B, \epsilon^*) \Rightarrow B, D$

⑦ $(D, \epsilon^* 0 \epsilon^*) \rightarrow (D, \epsilon^*) \Rightarrow (D, 0) \Rightarrow (D, \epsilon^*) \Rightarrow D$

⑧ $(D, \epsilon^* 1, \epsilon^*) \rightarrow (D, \epsilon^*) \Rightarrow (D, 1) \Rightarrow D$

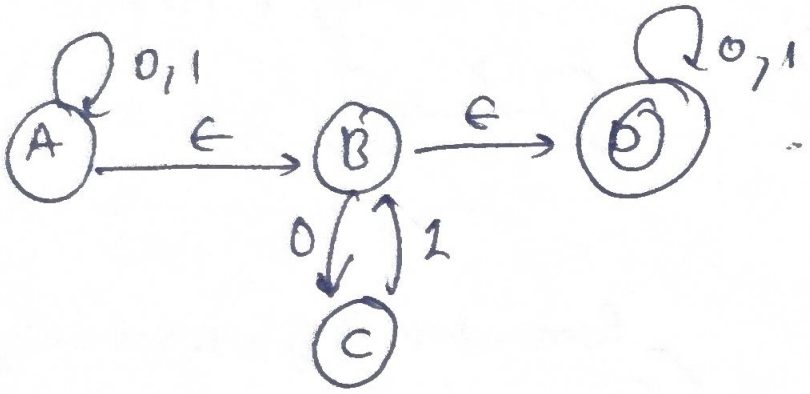


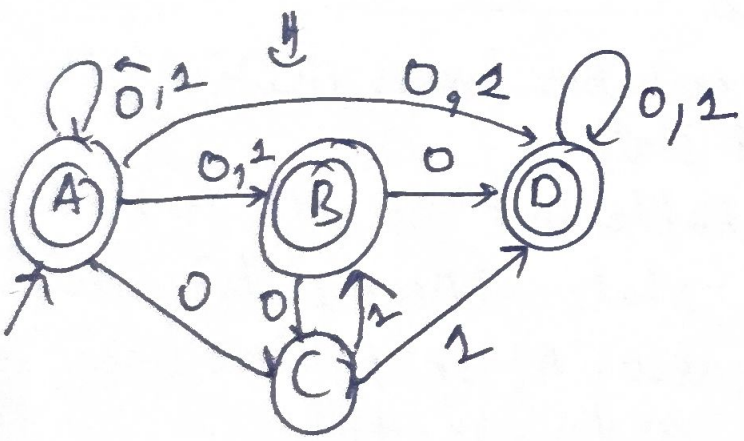
**Note: On any given DFA, if you want to apply minimization algorithm.

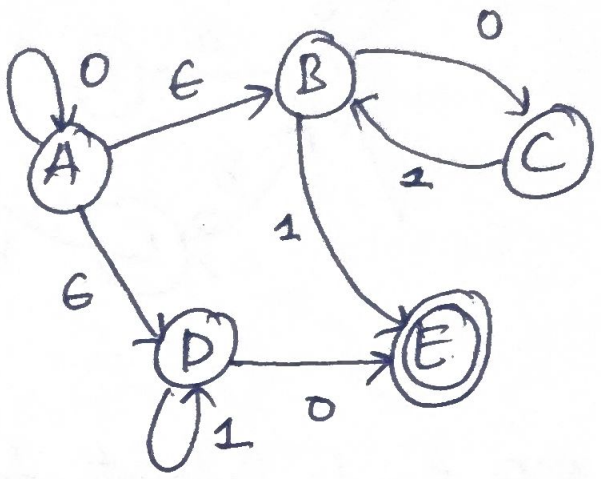
① Eliminate those states which are not reachable from initial state.

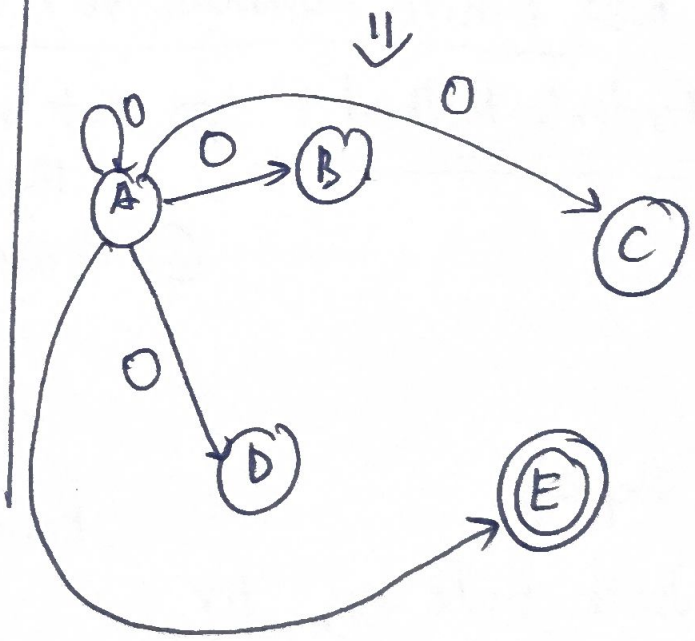
② Apply partition algorithm.

Check L-14 (2:09):00) [Shortcut]







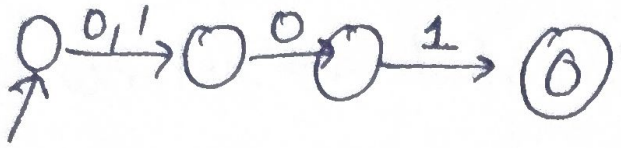


0,1

	0	1
→ A	A, B, C, D, E	D E
B	C	E
C	ϕ	B
D	E	D
E	ϕ	ϕ

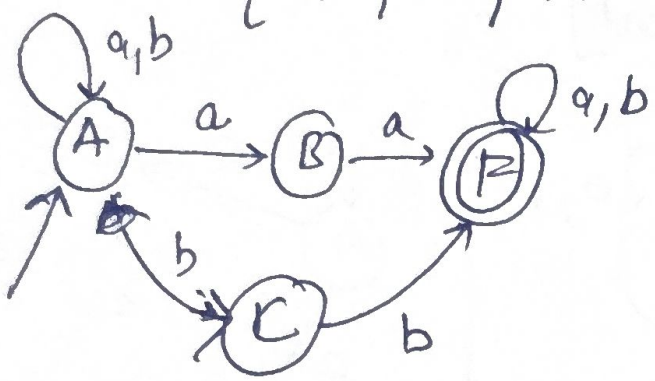
Lecture-15

(P1) ⇒ (0+1)01

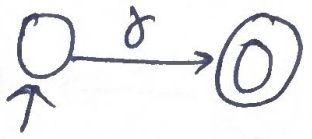


(P2) Construct NFA $(a+b)^*(aa+bb)(a+b)^*$

$L = \{ aa, bb, \ldots \}$ substring aa or bb containing.



(Using ε-NFA)



Remember they are not unique.

Finite Automata to Regular Expression :→

Arden's Method :

① not reachable from initial, then delete that state.

② Reachable but no able to reach the final state. (Dead state), delete them also. After for above steps, follow the arden's method.

Remember dead state or unreachable, will not be part of the regular expression.

(Step 1)

① Write state eqn for each state.

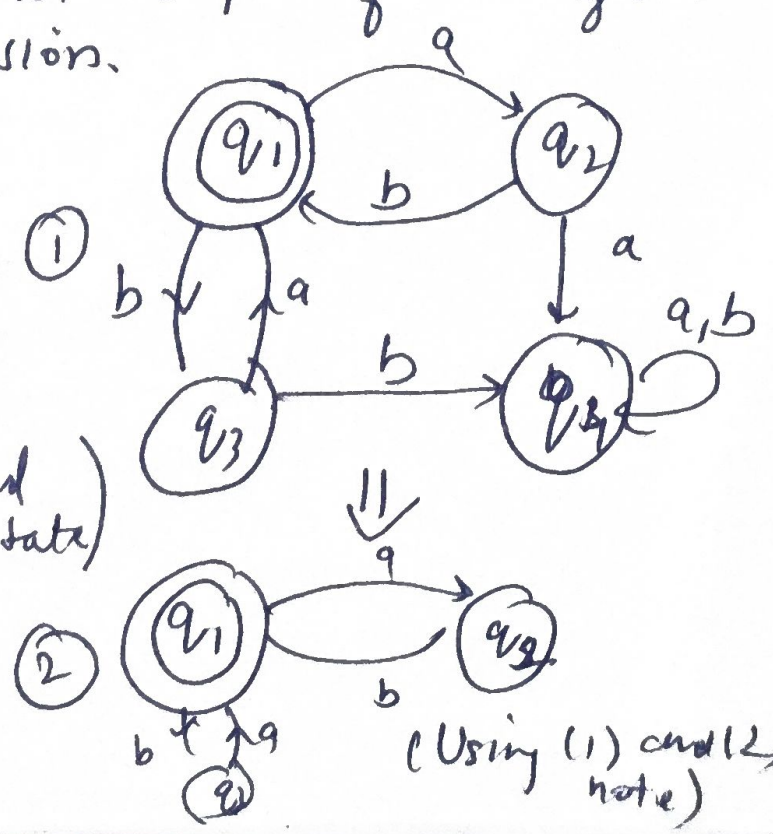
$q_n = q_k \cdot$ (seeing some alphabet)

$q_3 = q_1 \cdot b \rightarrow (1)$

$q_2 = q_1 \cdot a \rightarrow (2)$

$q_4 = q_3 b + q_2 a + q_4 a + q_4 b$ (Dead State)

$* q_1 = \epsilon + q_2 b + q_3 \cdot a \rightarrow (3)$



Step (2)

I want only, final state equations. So, Substitute the value of q_2 and q_3 in terms of q_1.

* $q_1 = \epsilon + q_1 \cdot a\, b + (q_1 b)\, a$ P

$q_1 = \epsilon + q_1 (ab + ba) \rightarrow$

*According to Arden's :→

$R = Q + RP$ ⇒ *R places must match

$R = QP^*$

Q

$q_1 = \underset{Q}{\epsilon} + \underset{R}{q_1} \underset{P}{(ab+ba)}$

$q_1 = Q \cdot P^*$

$q_1 = \epsilon\,(ab+ba)^*$

or

$q_1 = (ab+ba)^*$ (as R.E doesn't include ϵ)

Note

(1) aab + abba

concatenation is there

$ab \neq b.a$

ab(a+a) ✗

only a is common.

⇒ a(ab+ba)

(2) aab + bbb

1 2 3 4 5

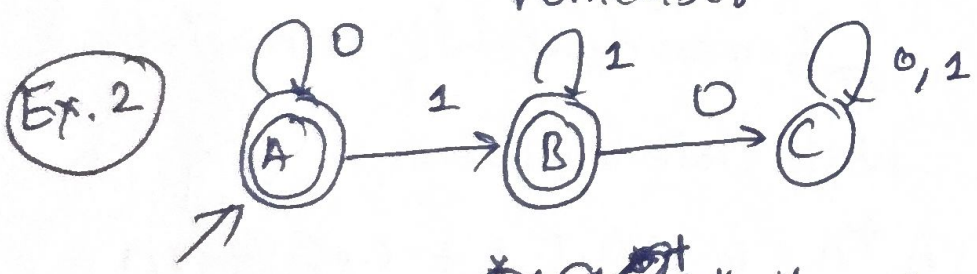
a ✗ common

(a, b) is not possible.

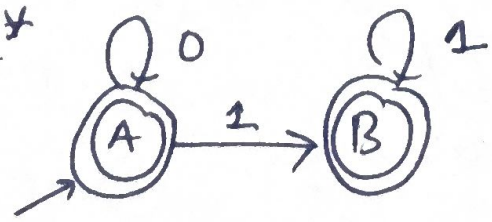
1 5 $2*3*5 \neq 3*2*5$

remember

Ex. 2



⇒ $0^* 1^*$



(1) Remove unreachable.

(2) Reachable but are dead

A = ∈ + A.0 (Starting is final add ∈)

*B = A.1 + B.1 → "Ardenl format"

R = Q + RP

B = A.1 1^* → (2)

A = C.⊥ A.0 . A = ∈ 0^* → (1) In (2), Substitute (1)

R = Q + RP

$\Rightarrow 0^* 1 1^*$

$\Rightarrow 0^* 1^+$

Now, both final state A and B, add both the equation.

$\Rightarrow 0^* + 0^* 1^+$

$\Rightarrow 0^*(\epsilon + 1^+)$ $\quad \epsilon + 1^* \Rightarrow 1^*$

$\Rightarrow 0^* 1^*$

Properties :→

① $a.a^* = a(\epsilon, aa, aa, aaa, \dots)$

$\Rightarrow a, aa, aaa, \dots \Rightarrow a^+$

② $a.\epsilon = a$

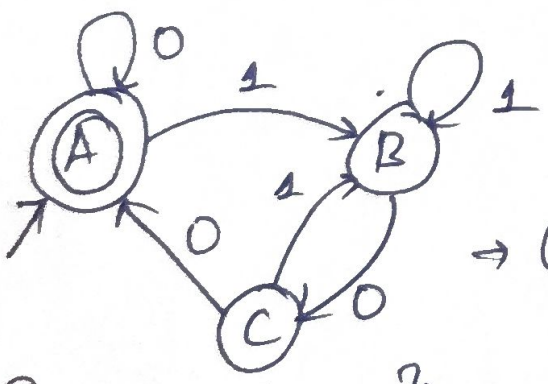
$a+\epsilon = a+\epsilon$

but $a+\epsilon = a$ ✗ (not possible)

③ $a + \phi = a$

$a.\phi = \phi$

Example:



$L_1 = \{\epsilon, 0, 100, \dots\}$

Remove dead state, unreachable

Write state equations

① $A = \epsilon + A.0 + C.0 \rightarrow (1)$

② $B = A.1 + B.1 + C.1 \rightarrow (2)$

③ $C = B.0 \rightarrow (3)$

Using (2) and (3)

$B = A.1 + B.1 + B.01$

$B = A.1 + B(1+01)$

R Q RP

$B = A.1(1+01)^* \rightarrow$

Using (1) and (3)

$C = (A.1(1+01)^*.0$

$$A = \epsilon + A.0 + A1(1+01)^*00$$

$$A = \epsilon + A.0 + A.1(1+01)^*00$$

$$R = A.\epsilon + A[0 + 1(1+01)^*00]$$

$$R = Qp^*$$

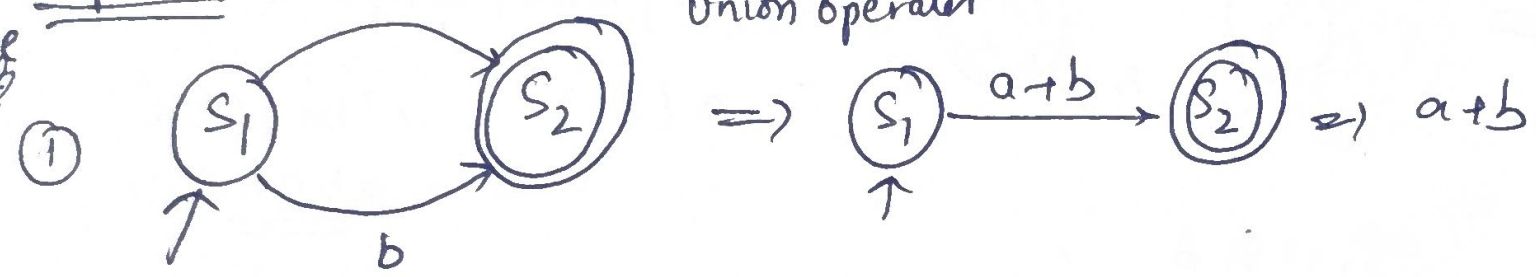
$$A = \epsilon(0+1(1+01)^*00)^*$$

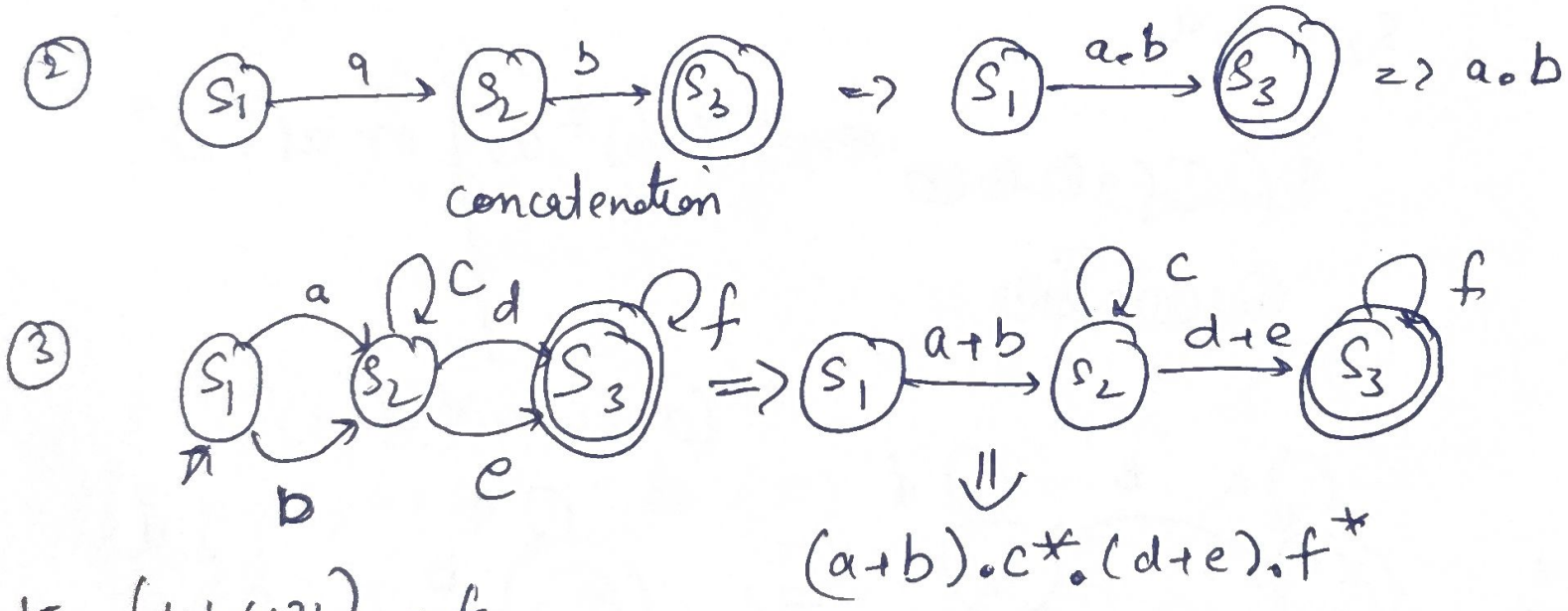
$$A = (0+1(1+01)^*00)^*$$

Method-2 (State elimination method)

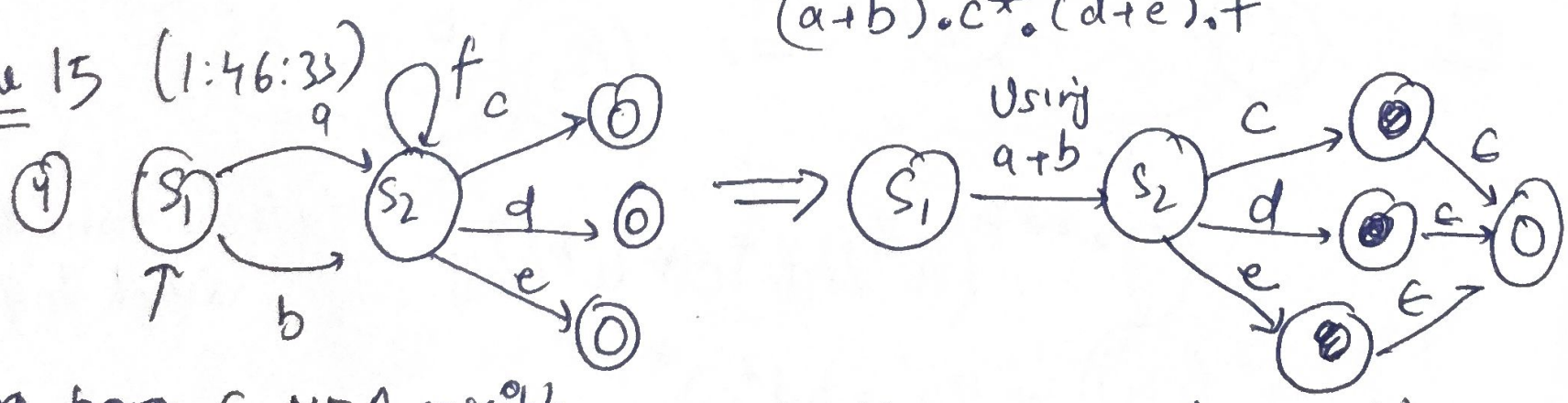
Properties :→ a

Example





Lecture 15 (1:46:33)

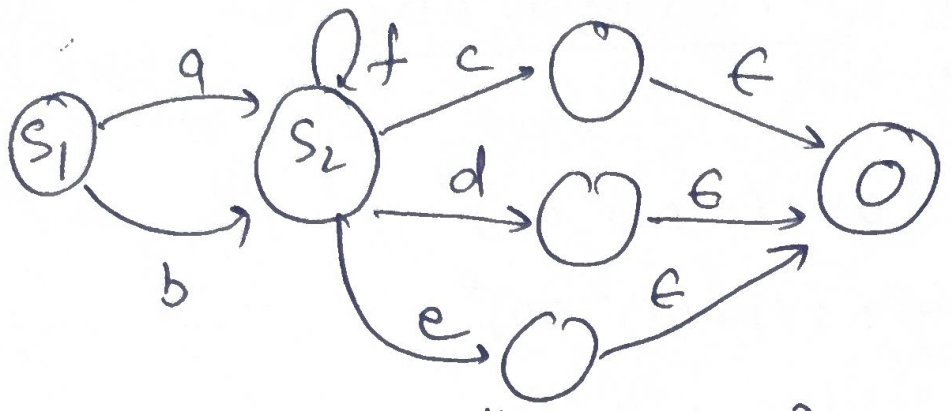


ϵ-NFA possible

For DFA, I have multiple final in DFA, then is it possible to construct DFA with one Final?

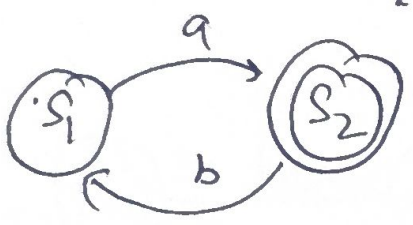
Ans Not always [

For ϵ-NFA, from multiple final state, single final is possible with the help of ϵ.



$\Rightarrow (a+b)\, f^*\, (c+d+e)$

ex



$L = \{ a, aba, ababa \ldots \}$

$a(ba)^*$ $a(ba)(ba)$

ababa

$S_1 = S_2 b$

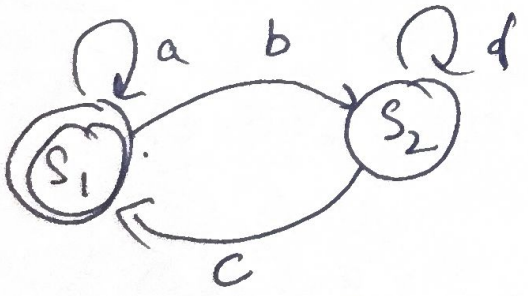
$S_2 = S_1 a$

~~$S_1 = S_1 ab + \epsilon$~~ or $\Rightarrow (ab)^* a$ / or $a(ba)^*$

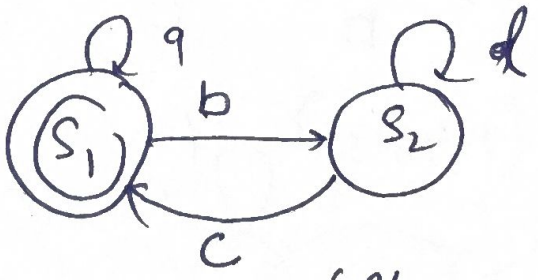
~~$S_2 = S_2 ab + Sab$~~

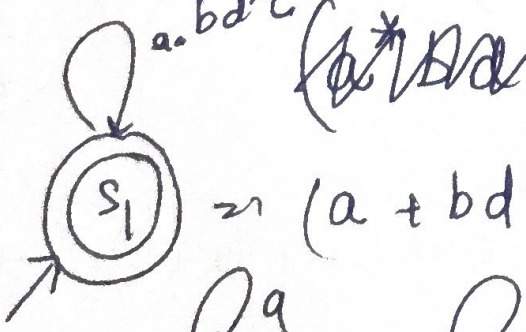
$(a+\epsilon)a^* = \epsilon + a(\epsilon + a)^*$

ex



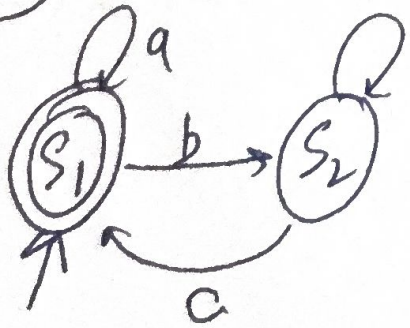
$\Rightarrow$





$= (a + bd^*c)^*$

(Please come to back to final)



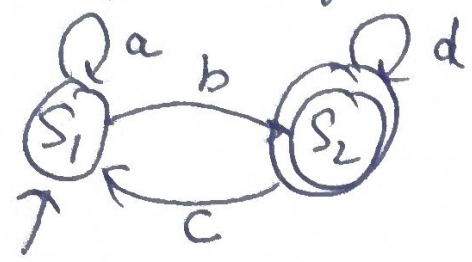
$a^* b [d + ca^* b]^* c$

minimal DC, ϵ is missing you have to merge

Q. $(\underbrace{b(d+(a^*b)^*c}_{x}+a)^{y}$

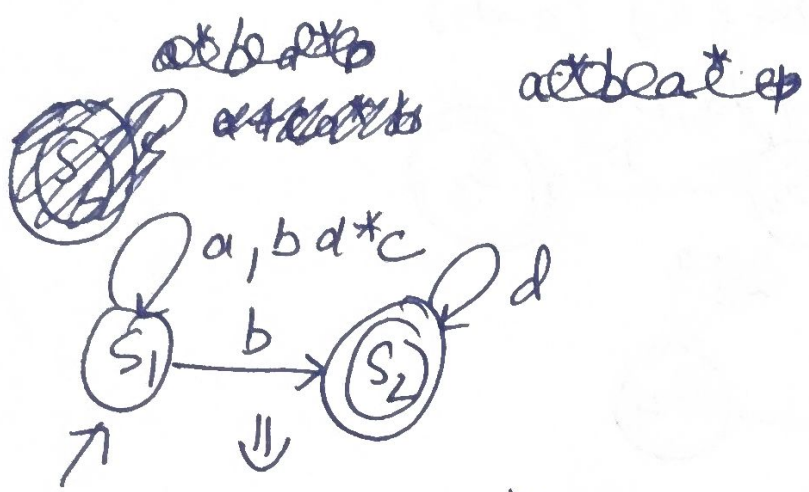
* Always manage at final state. (2:13:03)(L15)

Problem:

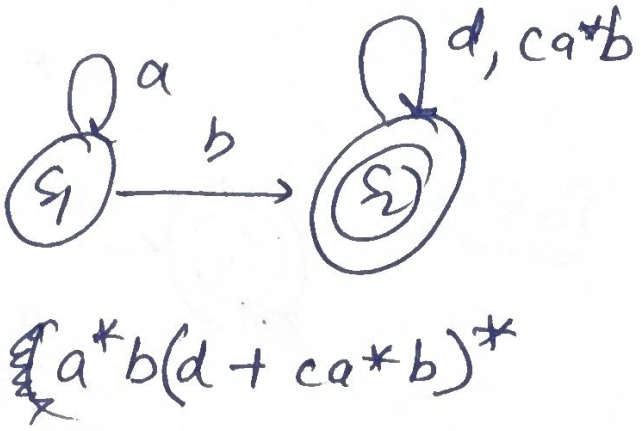


Manage: at S_2

S_2 is final state, manage here.



$(a+bd^*c)^*.bd^*$



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

$S_1 = S_1a + S_2.c$

$S_2 = S_2b + S_2d + S_2b$ (F)

$S_1 = S_1a + S_2.c$

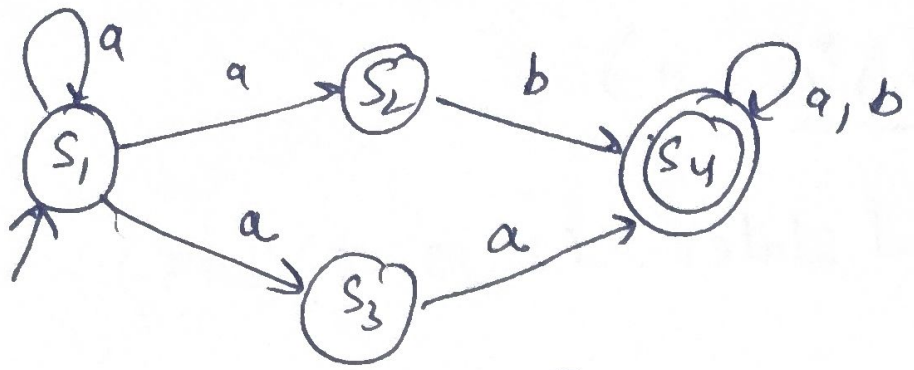
$S_1 = S_2.ca^*$

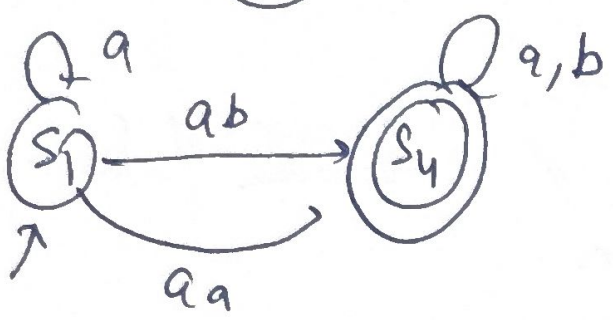
$S_2 = S_2(b+d) + S_2.ca^*b$

$S_2 = S_2(b+d) + ca^*b$

$S_2 = ca^*b(b+d)^*$

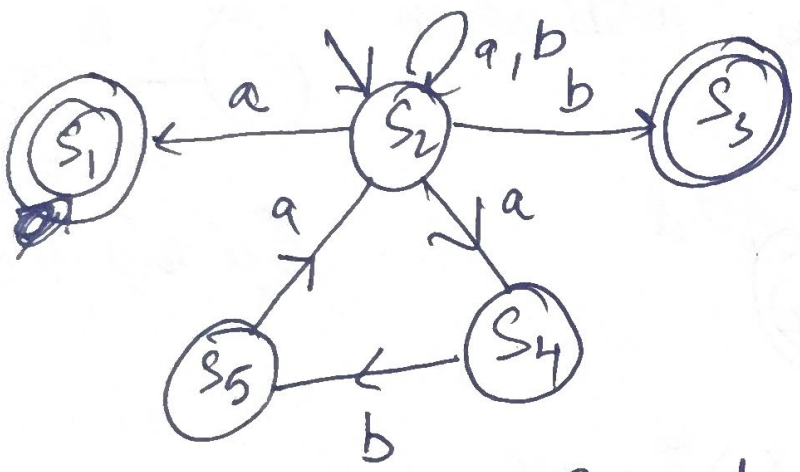
P1





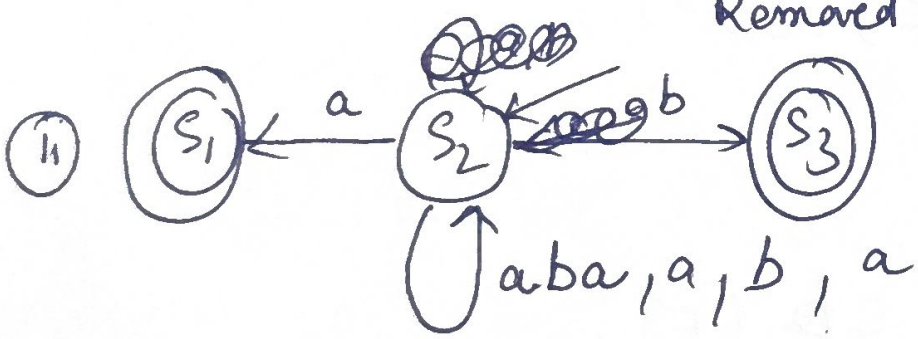
$a^*(ab+aa)(a+b)^*$

Problem



Removed ~~reachable~~ unreachable but dead state.

(ii)



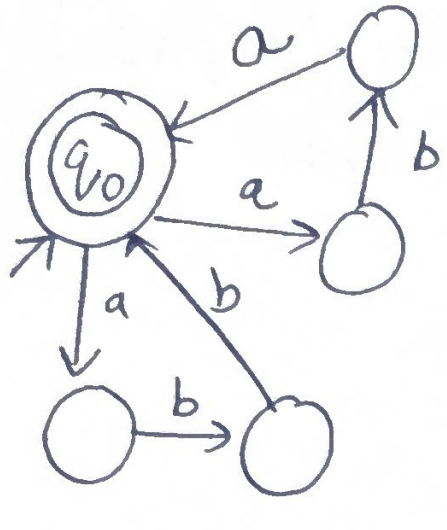
(iii) $(aba+a+b)^*a$

$L = \{a, aa, aba\,a, b, bb\}$

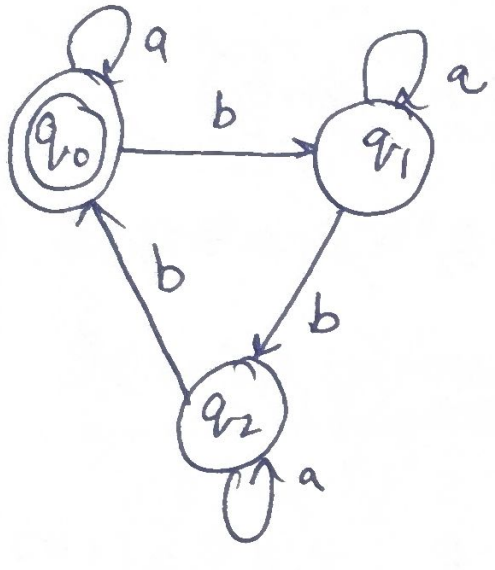
ends with a only.
or with b only

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

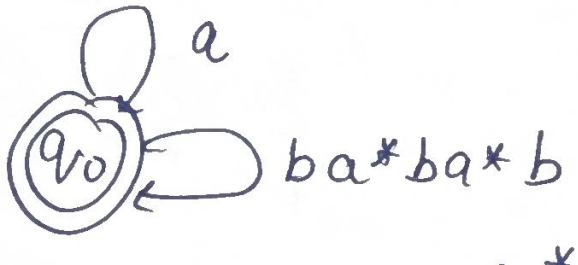
Lecture-16 :→



$\Rightarrow (aba + abb)^*$



$\Rightarrow$

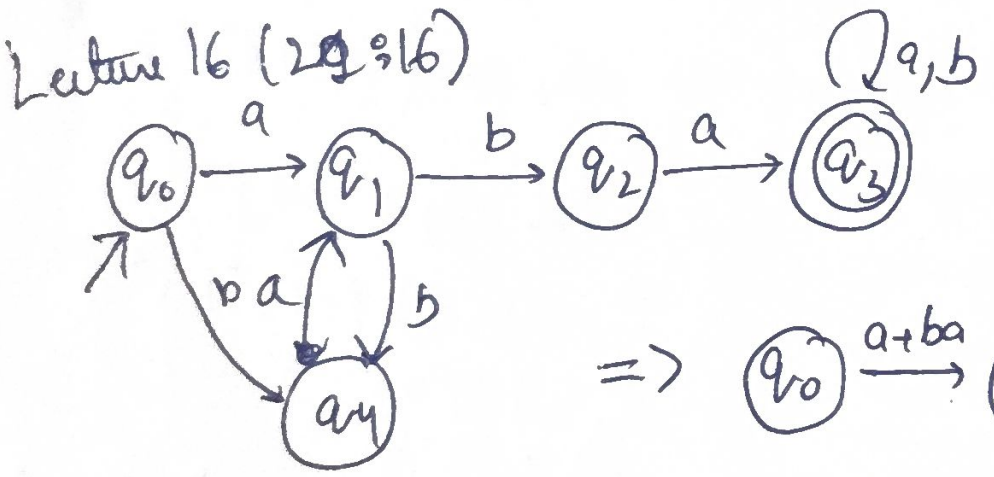


$(a + ba^*ba^*b)^*$

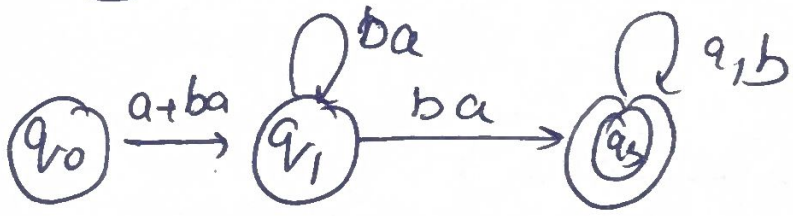
or

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

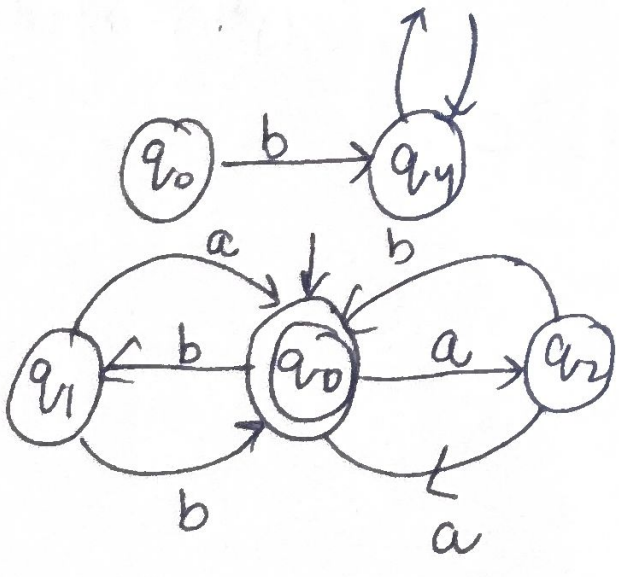
Lecture 16 (29:16)

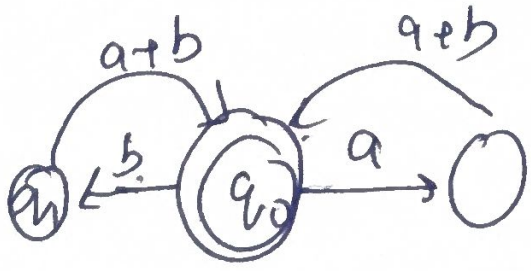


$\Rightarrow$



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

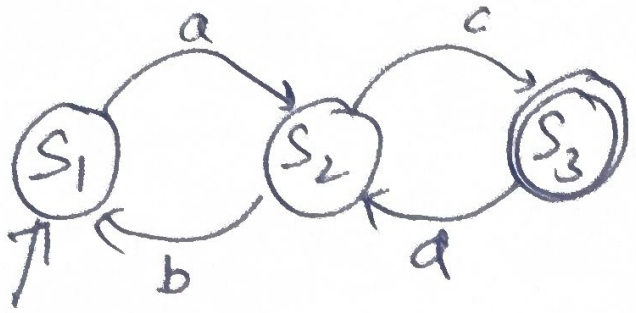




$= (b(a+b) + a(a+b))^*$

$= (b+a)\cdot(a+b)$ (Order is imp)

$= (a+b)(b+a)$ X Wrong



Three loops (I prefer Arden's method)

① Arden's method.

$S_1 = S_2 b + \epsilon$

$S_2 = S_1 a + S_3 d$

$\rightarrow S_3 = S_2 c$

~~$S_2 = S_2 b + S_3 d$~~ (Arden property)

R P Q

$S_2 = S_1 a + S_2 cd$

~~$R(S_2 = S_3 cd(cd)^*$~~

$S_2 = S_3 d b^*$

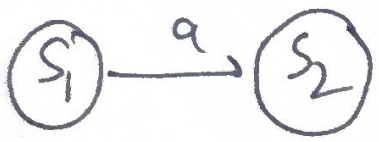
~~$S_2 = S_3 d b^* a$~~

$S_1 = \epsilon + S_1 a(cd)^* \cdot b$

$S_1 = (a(cd)^* b)^*$

$S_2 = S_1 a(cd)^*$

$= (a(cd)^* b)^* a$

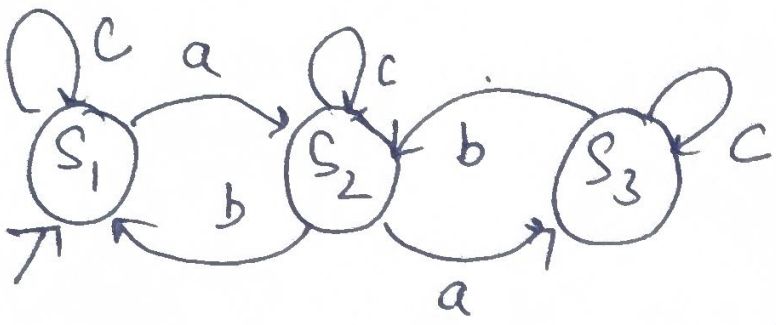


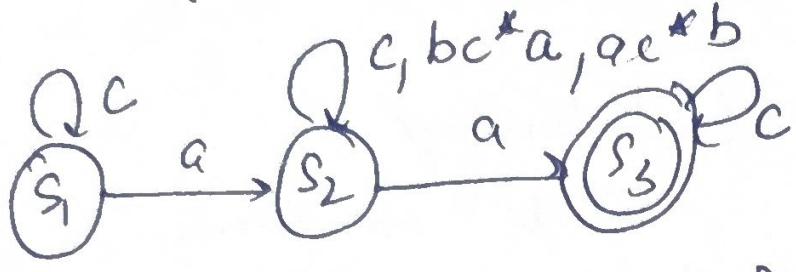
$S_3 = S_2 \cdot c$

$= (a(cd)^* b)^* a(cd)^* c$

② $a(ba + cd)^* c$

values to enter.





$$c^*a(c+bc^*a+ac^*b).ac^*$$

① Track loops, how many are there.

② On S_2, we have three loops. ©.

③ On S_1, one.

Imp

How to check given Language Regular :→

Steps 1 ① Verify, a language is either ~~Regular~~ infinite or finite.

↳ finite is all regular language

Infinite Language :→ $(a^*$,

a^* → Regular Language.

a^p → prime number [you can not make this machine]

Infinite Language. → NRL → $|\Sigma| = 1$

$|\Sigma| = 1$

Check AP series

If this is

① Regular Language

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

ϵ, aa

Non-AP series

$\{a^p \mid p$ is prime $p > 1\}$

but

$\{a^p \mid 2 < p < 100\}$ is regular. as we have finite values to check.

Be careful for this

No-Comparison

① Check AP or not

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

comparison required

Non-Regular

① $\{a^n b^m \mid m,n \geq 1\}$

Problems (Lecture 16, 1:35:00)

① $L_1 = a^n b^n \mid n \leq 100$ → finite language.

$a^{100}b^{100}$ max string,

$L = \{\epsilon, ab, a^2b^2 \ldots a^{100}b^{100}\}$ with the help of 100+1 states

② $L_2 = a^n b^n \mid n \leq 2^{1000^{1000^{10^5}}}$

→ finite language (Regular)

③ $L_3 = a^n b^n c^k d^k \mid n < 1000, k \geq 1$

↑ infinite

comparison is required.

④ $\{L_4 = w \mid w \in (a+b)^*, n_a(w) = n_b(w)\}$ $a^n b^n$ ✓ but $c^k d^k$ ✗

infinite language, comparison req.

Non-Reg.

⑤ $L_5 = w \mid w \in (a+b)^*, n_a(w) = n_b(w) + 7$

infinite $a > b$, comparison

⑥ $L_6 = \{a^p \mid p \text{ is prime } p > 1\}$ Non Regular

{ infinite $|\Sigma| = 1$, non AP } Non-Regular

⑦ $L_7 = a^p \mid p$ is not prime $p > 1$

{ infinite $|\Sigma| = 1$, non AP, as you have to check whether it is prime or not }

Note :→ $L_6 = U - L_7$ (they don't exist)

If L_7 exist $\Leftrightarrow$ L_6 exist but their not happening

⑧ $L_8 = \{a^{n!} \mid n \geq 1\}$ infinite, Non-Reg.

$\{1, 2, 6, 24, 120, 720\}$ not AP

⑨ $L_9 = \{a^{n^2} \mid n \geq 1\}$ infinite $\{1, 4, 9, 16, \ldots\}$ not AP

⑩ $L_{10} = \{a^m a^n \mid m, n \geq 1, m = n\}$ Comparison, ⑪ $L_{11} = \{a^m a^n \mid m, n \geq 1, m \neq n\}$

Using 6 and 7 same Comparism

(10) $L_{12} = \{ a^{2^n} b^m \mid m, n \geq 1 \}$

a^{2^n} => doesn't form AP series

(13) $L_{13} = \{ a^{2^n} a^m \mid m, n > 1 \}$

$\{ a^2 a^2, a^2 a^3, a^2 a^4 \ldots \ldots$ ... AP series $\}$

$\{ a^4 a^2, a^4 a^3 \ldots \ldots$ AP series $\}$

$\{ a^8 a^2, a^8 a^3 \ldots \ldots$ AP series $\}$

∴ so on

$\{ 2, 3, 4, 5, 6 \ldots$ so on $\}$ AP series.

(14) $L_{14} = \{ a^{m^n} \mid m, n \geq 2 \}$

$\{ a^{2^2}, a^{2^3}, a^{2^4}, a^{2^5} \ldots \}$

$\{ 4, 8, 16, 32 \ldots \}$ not AP.

Imp

(15) $L_{15} = \{ a^{m^n} \mid m, n \geq 1 \}$

$\Rightarrow \{ a^{1^2}, a^{1^2}, a^{1^3}, a^{1^4}, a^{1^5} \ldots \}$ AP series

$\{ a^{2^1}, a, a \ldots \} \supseteq \{ a \}$

Everyone will come

$\{ a^{3^1}, a^{2^2} \ldots \} \Rightarrow$

$\{ a^{3^1}, a^{3^2}, \ldots \}$

$\Rightarrow a^{3^3}$

L-16 $L_{16} = \{ ww \mid w \in (a+b)^* \}$

string companion

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

L-17 $= \{ ww^R \mid w \in (a+b)^* \}$

$ww^R \Rightarrow$ pallindrome

$w = abb$

$w_R = bba$

abbbba companion is there. abbbba

L-18 $= \{ ww \mid w \in (a+b)^* \}$ Regular lang.

$|w| \leq 1000$

↑ finite

Lecture 17 :→

Important

CLOSURE PROPERTIES :→

① L_1 is Regular, L_2 is regular.

Conclusions :→

① $L_1 \cup L_2$ is also regular

② $L_1 \cap L_2$ is also regular

③ $L_1 \cdot L_2$ is also regular.

Ex1 $L_1 = \Sigma^*$, $L_1 \cup L_2 = L_1$

$L_1 \cap L_2 = L_2$

(q.35)

Ex2 $L_1 = \phi$ $L_1 \cup L_2 = L_2$

$L_1 \cap L_2 = \phi$

$L_2 \subseteq L_1$

Ex. $L_1 = a^*b^*$, $L_2 = a^n b^n \mid n \geq 0$

$L_1 \cup L_2 = L_1$ { $L_1 \supset L_2$ }

$L_1 \cap L_2 = L_2$ common things

Imp

Ex $L_1 = (a+b)^* b (a+b)^*$ 5'a, 5'b

$L_2 = (a+b)^* a (a+b)^*$

L_1 contains ab substring $L_1 \cup L_2 = \Sigma^+$

L_2 contains ba as substring Imp $L_1 \cap L_2 =$

Ex 4 (Previous Eg)

$L_1 \cap L_2 =$ $(a+b)^* a (a+b)^* b (a+b)^* + (a+b)^* b (a+b)^* a (a+b)^*$

Ex.5

$L_1 = a^*b^*$, $L_2 = (a+b)^*$

$L_1 \cup L_2 = L_2$

$L_1 \cap L_2 = L_1 (a^*b^*)$

$L_1 = \{ \epsilon, a, ab, aa, ab \}$

$L_2 = \{ \epsilon, a, b, aa \}$

$L_1 \subseteq L_2$, Hence L_2 is bigger.

Questions in Gate

Regular Language ∪ Non Regular Language

$(a+b)^* \cup \underbrace{a^n b^n}_{\text{Non Regular}} = (a+b)^*$ → (Regular Language) also. as it covers $a^n b^n$

Regular → $\phi \cup \underbrace{a^n b^n}_{\text{Non Regular}} = a^n b^n$ → (Non regular language)

↑ take DFA, with no final state

* Regular Language union(∪) Non Regular Language, may be regular or may not be regular.

$a^p / p \text{ is prime} \cup a^{ip} / p \text{ is not prime} = a^*$

* Non-regular Language Union(∪) non-regular language; may be regular.

NRL ∪ NRL gives RL/NRL

① $L_1 \cup L_2 = RL$

RL/NRL ↓ RL/NRL

② $L_1 \cup L_2$ is Regular, but L_1 or L_2 may be regular or non regular

③ $L_1 \cup L_2$ is non regular, what about L_1, L_2 individual.

$\phi \cup a^n b^n = a^n b^n$ (NR)

(NR)

Atleast one person be non-regular.

Imp

Regular Language ∩ Regular Language = Regular Language (This is Standard property)

① Product Automata, (It is already done)

② $(a+b)^* \cap a^n b^n = a^n b^n$
↑ R.L → NRL → NRL

$\phi \cap a^n b^n = \phi$
↑ RL ↑ NRL → RL

Note ✗ Non standard: Intersection of one regular and other one is non-regular, may be regular or may not regular.

* If both non-regular, then may be regular or non regular.

Non Standard

$a^n b^n \cap b^n a^n = \phi$
↑ NRL ↑ NRL → R.L

Do carefully

$a^n b^n \cap a^n b^n c^m \mid m, n \geq 0 = a^n b^n \mid n \geq 1$
↑ NRL ↑ NRL NRL

$m=0$
$a^n b^n \cap a^n b^n c^0 = a^n b^n$ Hence $a^n b^n / n \geq 1$
NRL

Example

① $a^* \cap (aa)^* = (aa)^*$

② $a^* b \cap \{ab^*\} = \{ab\}$ → R.L

③ $a^* b^* \cap b^* a^* =$ $\{\in, b, bb, \ldots\}$ i.e. $a^* b^* = \{\in, a, b, \ldots\}$ infinite R.L $b^* a^* = \{\in, b, a, \ldots\}$

④ $a^+ b^+ \cap b^+ a^+ = \phi$

⑤ $ab^+ \cap ba^+ = \phi$ $\{ab, abb \ldots$
$ba, baaa \ldots\}$

(L17 18/7/23)

Concatenation :→

① $a^* \cdot b^* = a^* b^*$

② $\phi \cdot L_1 = \phi$ } Confusing { $L_1 \cdot \phi = \phi$
$\epsilon \cdot L_1 = L_1$ } { $L_1 \cdot \epsilon = L_1$

③ $\Sigma^* \cdot a = (a+b)^* \cdot a$ ending with a

④ $\Sigma^* \cdot (a+b)^* \Rightarrow (a+b)^* \cdot (a+b)^* = (a+b)^*$

⑤ $a^* \cdot (a+b)^* \Rightarrow (a+b)^*$

⑥ $L_1 = a^n \mid n \geq 1$ (R.L), $L_2 = b^n \mid n \geq 1$ (R.L) both n are not same

$L_1 \cdot L_2 = a^m b^n$ $m, n \geq 1$

↑ with the help of ϵ production → regular language.

both are sitting by side by side, does not necessarily means

$\phi \cdot a^n b^n = \phi$

Regular NR Regular

⑦ $(a^p \mid p \text{ is prime}) \cdot (a^p \mid p \text{ is not prime}) =$ ~ RL

$(a^p \mid p \text{ is prime})$ ⇓ NRL — NRL

NRL

Note *

$ab \cup a^2b^2 \cup a^3b^3 \cup a^4b^4 \cup \ldots \Rightarrow a^n b^n \mid n \geq 1$

(ab: regular) — NRL

* Infinite union of regular lang. may be non-regular language

$RL_1 \cap RL_2 \cap RL_3 \cap \ldots \infty$ = need not to be regular.

$RL_1 \cdot RL_2 \cdot RL_3 \ldots \infty$ = need not to be regular.

$a \cup a^2 \cup a^3 \cup a^4 \ldots \infty$ = regular [this time pass]

(3) **Complimentation :→**

If L_0 is regular then L^C is also regular., so Regular

$L^C = U - L$

language are close under complementment.

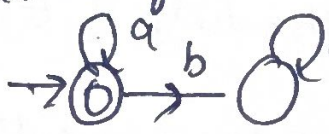
① $L = \phi \Rightarrow L^C = U \Rightarrow (a+b)^*$ Regular

② $L = \Sigma^* \Rightarrow L^C \Rightarrow \phi \Rightarrow \phi$ is regular

③ $L = a(a+b)^* \Rightarrow L^C \Rightarrow$ Regular making ϕ non-final state - final. $b(a+b)^* + \in$

④ $L = (a+b)^* b(a+b)^* \Rightarrow L^C$, only a^*, $\in + a^+ = a^*$

↓ contains b

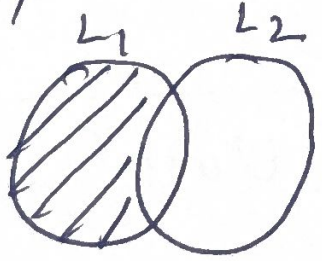


Note | If L is regular, then L^C is regular
If L is non-regular, then L^C is also non-regular.

(4) **Difference :→**

$L_1 - L_2$ = L_1 but not L_2

① $L_1 - L_2$



Only shaded Area

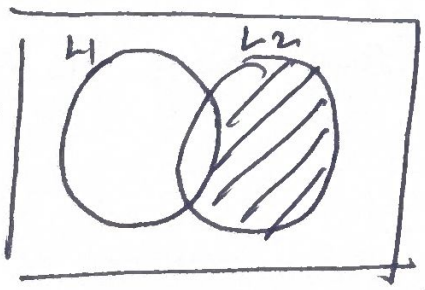
A = {1,2,3}

B = {3}

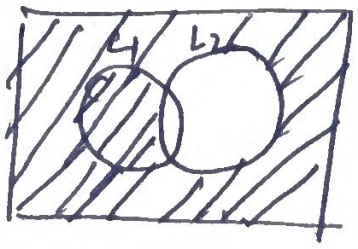
A − B = {1,2}

{1,2,3} − {3}

② $L_2 - L_1$ =



③ L_2^C =



Note

$L_1 - L_2 = L_1 \cap L_2^C$ Another way of writing

$L_1 - L_2 = L_1 \cap L_2^c$ $L_2^c \Rightarrow$ complement of L_c is Regular

L_1 — Regular. L_2 — Regular

* Hence, $L_1 - L_2$ is also regular. as $L_1 \cap L_2^c$

Remember this, it is not fundamental operator.

① If L_1 and L_2 is regular, then $L_1 - L_2$ also is also regular.

$L_1 \cap L_2^c$ is regular, but L_1 or L_2 may be regular or not.

Remember, this is not fundamental operation.

① $\phi - L = \phi$

② $L - \phi = L$

③ $\Sigma^* - L = L^c$

④ $L - \Sigma^* = \phi$ some − all ⇒ nothing will remain

⑤ $a^* - b^* = a^+$ [common thing will be cancelled]

⑥ $b^* - a^* = b^+$ ϵ is cancelled

⑦ $(a^*b^*) - (a^* + b^*) = a^+b^+$ a followed by b.

⑧ $(a^* + b^*) - (a^*b^*) = \phi$

$L_1 = \{\phi, \epsilon, a, b, ab^*, ab^*\}$

$L_2 = \{\epsilon, a \ldots\}$

Reverse each string in the language. → $(ab)^R = ba$, $(abc)^R = cba$

$L_1 = a(a+b)$, $L_2 = (a+b)a$

everything reverse, everything.

Reversal Operator :→

① $L = \Sigma^* \Rightarrow L^R = \Sigma^*$

② $L = a^* \Rightarrow L^R = a^*$

③ $L = \phi \Rightarrow L^R = \phi$

④ $a(a+b)^* = L^R = (a+b)^* a$

⑤ $a^* b^* = L^R = b^* a^*$

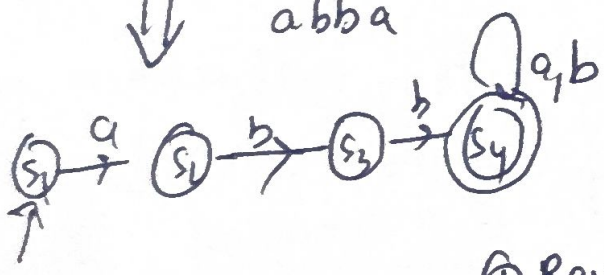
⑥ $(a+b)^* ab = ba(a+b)^*$

⑦ $(a+b)^* ab(a+b)^* = (a+b)^* ba (a+b)^*$

$L = abb(a+b)^* \longrightarrow (a+b)^* bba$ (L^R)

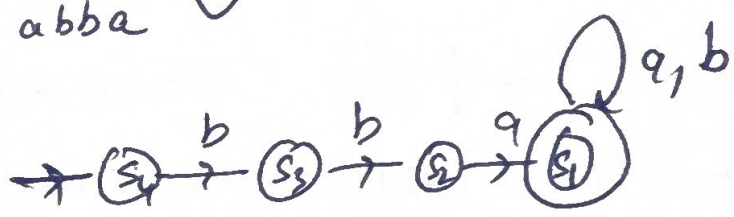
abb

abba



bba

abba



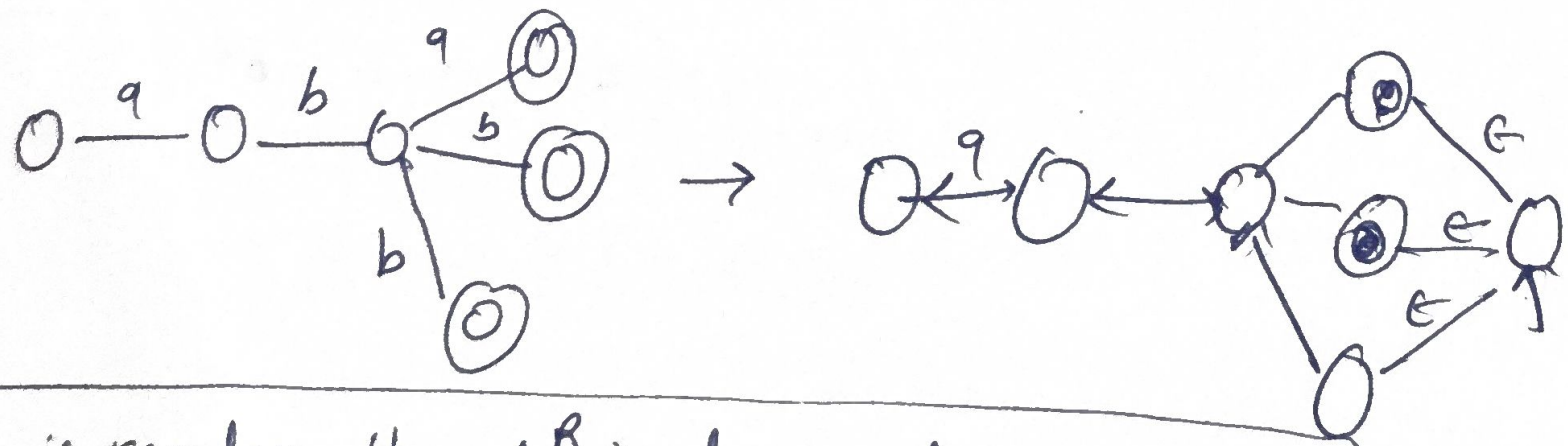
① Reverse the direction, keep self loop as it is.

② Change final to initial state.

③ Change initial to the final state

①

②



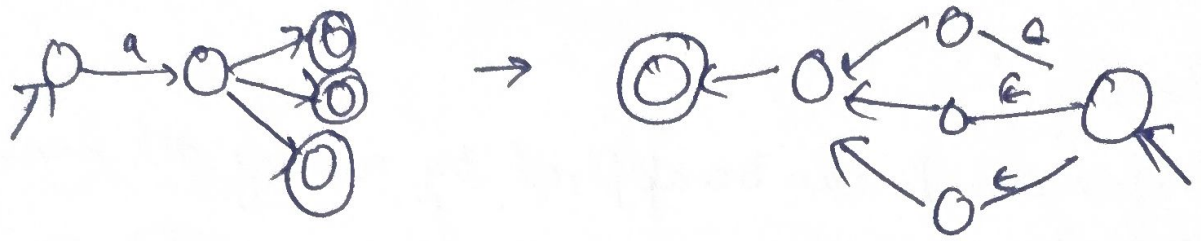
If BL is regular, then L^R is also regular,

Hence, L^R is ~~regular~~ closed under operation reversal.

Note

While creating L^R, if you get multiple initial state, then

① Using ∈ transition, you can make, one initial



Lecture 18

(Only this operation is not closed)

Subset Operation :→

$L = (a+b)^*$ => Regular Language

$S_1 \subset L$

$\Downarrow$

$a^n \mid n \geq 1$ => regular lang.

a^*b^* => Regular lang.

$a^n b^n \mid n \geq 1$ => not regular language.

PTR

① Regular language not closed under subset. Means,

② the regular language may or may not be regular.

③ Subset and finite language is regular.

④ Every finite subset of a language is regular.

L is non-regular

$S_3 \subset L = a^n b^n \mid n \geq 1$

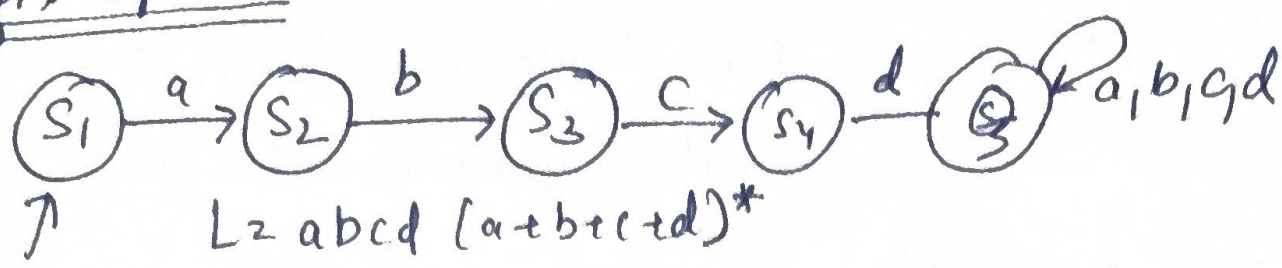
$\Downarrow$

$S_3 = \{ab\}$ → finite, Hence regular.

① Finite set of a non-regular is regular, Hence closed.

② Infite set of a non-regular may be regular.

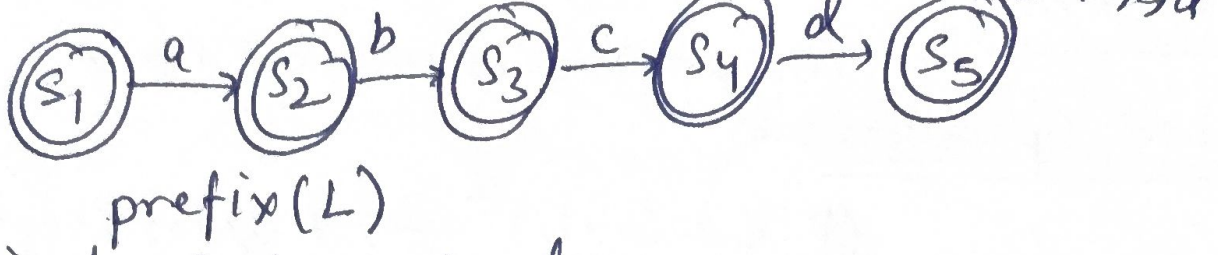
Prefix Operation



L = abcd (a+b+c+d)*

~~Prefix of~~

① Prefix on L can be applied by making all non-final to final.



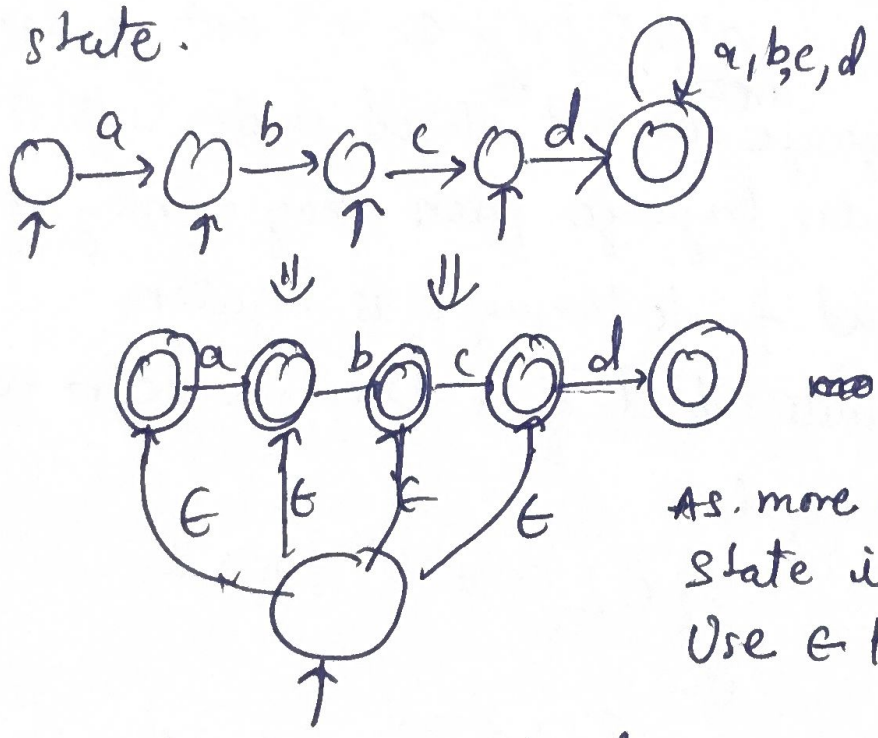
prefix(L)

② Prefix on L_1 is closed, hence regular.

prefix: a, ab, abc, abcd, ∈.

Suffix operation :→

① Suffix operation on L can be applied by making non-final as initial state.



As more than one initial state is not allowed, Use ∈ transitions.

② Suffix of L is closed, hence regular.

(Next substring)

$L = \{abcd\}$

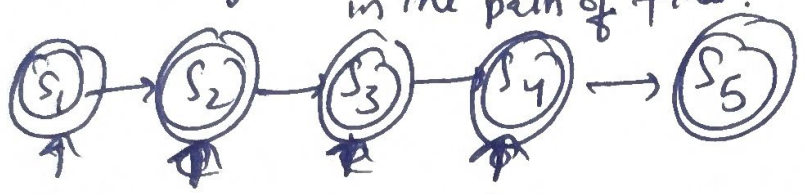
Substring :- ϵ, a, b, c, d
ab, bc, cd
abc, bcd
abcd

$\frac{n(n+1)}{2} + 1 \Rightarrow \frac{4\times5}{2} + 1$
$\Rightarrow 11$

~~Every~~ substring

① 

② Make every state as final ~~not~~ initial state as well as final in the path of final. ~~give prefixes~~ and make them suffix using ϵ-transition.



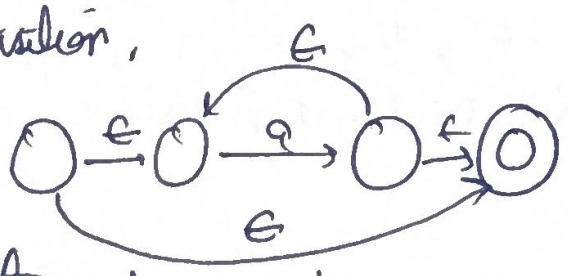
③ Then make all state as '

① Every prefix is substring.
② Every substring may not be substring.

* Kleene Closure :→

$L = \{a\} \Rightarrow$ 

L^* :- using null transition,

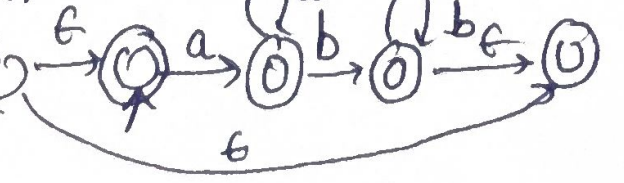


~~Under Kleene Closure is closed~~ and

① If L is regular then L^* is regular under ~~of~~ kleene closure hence closed.

$(\phi)^* = \{\epsilon\}$ as we don't have any final state

NOT ① $L = \phi$ $L^* = \phi$ Regular

② $L = a^*b^* \Rightarrow L^* = (a^*b^*)^* = (a+b)^*$ 

Regular

③

③ $L = (a^*b^*)^+ \Rightarrow L^* = [(a^*b^*)^+]^* \Rightarrow \{\epsilon, \ldots\}$
$(a^*b^*)^*$ Regular Expression

④ $\epsilon + a^+ \Rightarrow L^* = (\epsilon + a^+)^* = \{\epsilon, a, aa\}$
a^* Regular Expression

⑤ $L = a.a^+ \Rightarrow L^* = (a.a^+)^* = \{\epsilon, a^* \ldots\}$ regular.
$(a^+)^*$

⑥ $(L^*)^* = L^*$.

$(a^*)^* \Rightarrow a^*$

⑦ $L = \epsilon$, $L^* = (\epsilon)^* \Rightarrow \{\epsilon\}$

Quotient Operation :→ (New operation)

Note: If L_1 & L_2 are regular then L_1/L_2 is also regular.

$L_1/L_2 = \{x \mid \forall x, y \in L_1, \forall y \in L_2\}$

Note : ① From right side, the symbols will be cancelled in numerator.
② ~~Below must be all~~ The denominator must be cancelled
, if not, the answer is ϕ.

L_1 = 2 string
L_2 = 3 string
2X3 combination will be checked.

Example:1

① $\frac{abc}{b} \Rightarrow \phi$

② $\frac{abc}{bc} \Rightarrow a$

③ $\frac{abc}{\epsilon} \Rightarrow \frac{abc}{\epsilon}.\epsilon \Rightarrow abc$

④ $\frac{a}{a} \Rightarrow \epsilon$

⑤ $\frac{abc}{ac} \Rightarrow \phi$

⑥ $\{a, ab\}/\{\epsilon, b, aa\}$

(6) $\{a, ab\} / \{\epsilon, b, aa\}$ 2×3

$\frac{a}{\epsilon}, \frac{a}{b}, \frac{a}{aa} \Rightarrow a, \phi, \phi$.

$\frac{ab}{\epsilon}, \frac{ab}{b}, \frac{ab}{aa} \Rightarrow ab, a, \phi$

Hence $= \{a, ab\}$

(7) $\frac{aaaaa}{\{a, aa, aaa\}} \Rightarrow \frac{a^5}{a} \Rightarrow a^4$ $\frac{a^5}{a^3} \Rightarrow a^2$

$\frac{a^5}{a^2} \Rightarrow a^3$ $L = \{a^4, a^3, a^2\}$

(8) $\frac{a^*}{a} \Rightarrow \{\epsilon, aa, aaa\} \Rightarrow a^*$ $\frac{\infty}{2} \Rightarrow \{\frac{\epsilon}{a}, \frac{a}{a}, \frac{a^2}{a}, \frac{a^3}{a}, \dots\} \Rightarrow a^*$

(9) $\frac{a}{a^*} = \frac{a}{\epsilon}, \frac{a}{a}, \epsilon, \Rightarrow \{a, \epsilon\}$

(10) $\{f, a\} / \{\epsilon, a\} \Rightarrow \frac{\epsilon}{\epsilon}, \frac{a}{\epsilon} \Rightarrow \epsilon, a \Rightarrow \{a, \epsilon\}$

$\frac{\epsilon}{a}, \frac{a}{a} \Rightarrow \phi \quad \phi$

(11) $L / \phi \Rightarrow \phi$ nothing is divided (by Heart ♡)

(12) $\phi / L \Rightarrow \phi$

(13) $\frac{a^*b}{ab} \Rightarrow \frac{ab}{ab}, \frac{ab}{ab}, \frac{aab}{ab}, \phi, \epsilon, aa, \dots a^*$

(14) $\frac{\epsilon}{a} \Rightarrow \phi$

Imp Properties

① If L_2 including $-\epsilon$, then L_1 / L_2 ⇒ minimum L_1 will contain as dividing by ϵ,

L_1 Let {ab, cd, ef}

L_2 {e, ----}

② If L is non-empty then $\frac{\Sigma^*}{L}$ ⇒ $\frac{(a+b)^*}{L}$ ⇒ * as it is infinite long, infinite

$(a+b)^*$ ⇒ $(a+b)^*$ Σ^*.

③ If L is non-empty then $\frac{L}{\Sigma^*}$ ⇒ $\frac{\epsilon}{\epsilon}$, ϵ minimum, ϵ is there only and only ϵ.

$L = \{abb\}$

Σ^* ⇒ everything is possible Hence only one.

$\frac{L}{\Sigma^*}$ will give, all prefixes of L. and prefixes of abb L

New Convention :→

Homo-morphism :→ Δ is output alphabet assume {a, b}

Δ^* is strings over Δ.

$P(\Delta^*)$ = will be regular over Δ.

Power set of Δ^* ⇒ Collections of strings over Δ, which a is a Regular Language.

$H: \Sigma \to \Delta^*$ (Mapping of of alphabet to output alphabet)

Homomorphism

(1:56:00 L18)

Example : Homo morphism $H: \Sigma \to \Delta^*$

① $\Sigma = \{0,1\}$ $\Delta = \{a,b,c\}$

$H(0) = ab$ then find $H(010)$

$H(1) = bbc$ mapping

Formula

① $H(0^*) = [H(0)]^*$

② $H(\epsilon) = \epsilon$

③ $H(\phi) = \phi$

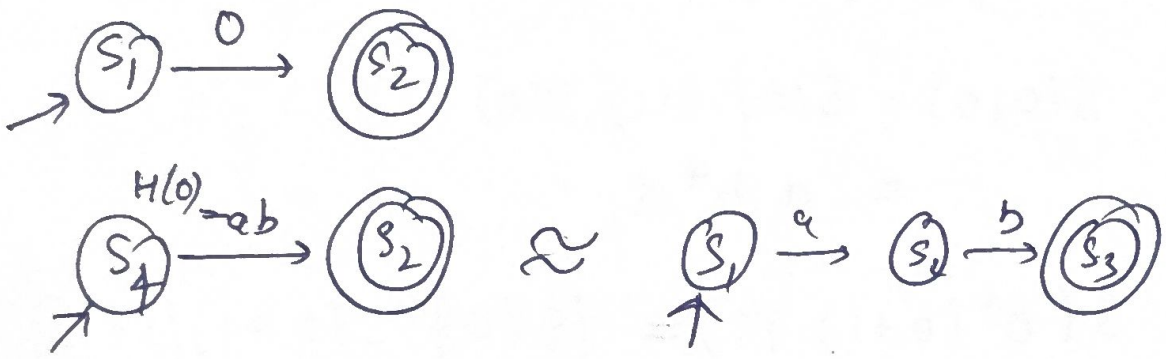
Formulae :→

① $H(010)$ can be written as $H(0) \cdot H(1) \cdot H(0)$

② Then replace the string corresponding to $H(0), H(1)$

$H(0) = H(1) \cdot H(0)$

$ab \cdot bbc \cdot ab \Rightarrow abbbcab$.



② $L = (01)^* 11$

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

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

$H(0) = abb$, $H(1) = \epsilon$

Then $H(L)$

① $H(01)^* \cdot H(1) \cdot H(1)$

② $[H(01)]^* \cdot H(1) \cdot H(1)$

③ $[H(0) \cdot H(1)]^* \cdot H(1) \cdot H(1)$

④ $[abb \cdot \epsilon]^* \cdot \epsilon \cdot \epsilon$

⑤ $(abb)^*$

③

Lecture 19

It will substitute with regular expression

Substitution :→ $S : \Sigma \to P(\Delta^*)$

Property :→

① $S(ab) = S(a) \cdot S(b)$

② $S(0^*) = (S(0))^*$

③ $(0 \cdot \phi)^* = \epsilon$

④ $\phi \cdot$ anything $= \phi$
but $(\phi)^* = \epsilon$

$\Sigma = \{a, b\}$ $\Delta = \{0, 1\}$

$S(a) = 0^*$, $S(b) = 0^*1$

Property $S(ab) = S(a) \cdot S(b)$

$= 0^* \cdot 0^*1$

$\Rightarrow 0^*1$

② $\Sigma = \{0, 1\}$ $\Delta = \{0, 1\}$

$S(0) = a$, $S(1) = b^*$

$S(010) = S(0) \cdot S(1) \cdot S(0)$

$= a\, b^* a$

$S(0^*(0+1)\,1^*) = [S(0)]^* S(0+1) [S(1)]^*$

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

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

In option, this may be not there

$L = \{\epsilon, a, b, ab, \ldots\}$
after b a is not allowed

5 string must be checked $\Rightarrow a^* b^*$

③ $\Sigma = \{0, 1\}$, $\Delta = \{0, 1\}$ $\Sigma \to P(\Delta^*)$

$L = (ab)^*$ $S(a) = 0^*$ $S(b) = \phi$

$S(L) = (S(a) \cdot S(b))^*$

$= (0^* \cdot \phi)^*$

$\Rightarrow (\phi)^* = \epsilon$

Inverse Homomorphism :→

$h^{-1}(w) = \{x \mid h(x) = w\}$

$H^{-1}(\epsilon) = \epsilon$

$H^{-1}(\phi) = \phi$

① H $\quad h(a) = 0$

② I-H $\quad h^{-1}(0) = 1$ reverse mapping

Example $\quad \Sigma = \{0, 1\} \quad \Delta = \{a, b\}$

$H(0) = aa \quad H(1) = aba$, $H^{-1}(aa) = 0$, → (1)

$L = (ab + ba)^* a$ $\quad H^{-1}(aba) = 1$ → (2)

$H^{-1}(L) = h^{-1}((ab + ba)^* a)$

$L = \{a, aba, baa, \ldots\}$

$\Rightarrow h^{-1}((ab + ba)^*) \circ h^{-1}(a)$

$\Rightarrow [h^{-1}(ab + ba)]^* \circ h^{-1}(a)$

$\Rightarrow [h^{-1}(ab) + h^{-1}(ba)]^* \cdot h^{-1}(a)$

Only (1), (2) is possible, check next minimal.

$h^{-1}(aa) = 0$

$h^{-1}(aba) = 1$

$aba\,ba = X$

$\Rightarrow 1$

As you are mapping reverse, some mapping may not be possible, Then whatever possible results.

Ex

$\Sigma = \{0,1,2\}$ $\Delta = \{a,b\}$

$H(0) = a$ $H(1) = ab$ $H(2) = ba$

$H^{-1}(a) = 0$
$H^{-1}(ab) = 1$
$H^{-1}(ba) = 2$

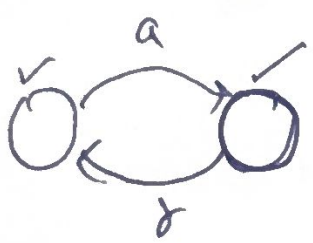
① $h^{-1}(ababa) = h^{-1}(a)\, h^{-1}(ba)\, h^{-1}(ba)$
$= 0 \cdot 2 \cdot 1$
$= 021$

other possibility
ab@ba
102

② $h^{-1}(a(ba)^*) = h^{-1}(a(ba)^*)$
$= h^{-1}(a)\, h^{-1}((ba)^*)$
$= h^{-1}(a)(h^{-1}(ba))^*$
$= 0\,2^*$ (Ans not in option)

$a(ba)^* = (ab)^*a$ or

$a(ra)^* = (ar)^*a$



$a.(\gamma a)^* . o\gamma$

$\Rightarrow h^{-1}((ab)^* a)$
$\Rightarrow (h^{-1}(ab))^* . h^{-1}(a))$
$\Rightarrow 1^* . 0$.

Ex $\Sigma = \{a,b\}$ $\Delta = \{0,1\}$

$H(a) = 01$, $h(b) = 0$
$H^{-1}(01) = a$ $H^{-1}(0) = b$
$L_1 = (10+1)^*$, $L_2 = \phi = (a+b)^*$ L_3 = equal a's & b's

these are not possible

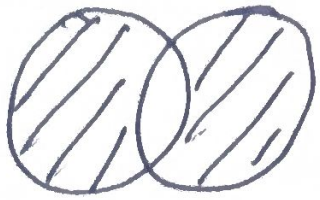
Checking.
? ϕ $H^{-1}(L_1) = H^{-1}(10+1)^*$
$L_1 = \{\epsilon, 1, 10, 101, 1\ldots\}$
$\Rightarrow \phi$

② $H(L_2) = H(a+b)^*$
$= H(\epsilon)$,
$L_4 = \{\epsilon, a, \ldots\}$
$\epsilon = \epsilon$

Imp
⑤ $H^{-1}(L_3)$
$= (01)^*, (10)^*$
$0^n 1^n, 1^n 0^n$
$\Rightarrow a^*$

④ $H^{-1}((10+1)^+) \Rightarrow \phi$

Symmetric Difference (Δ) (unusual symbol.) (Never asked in GATE)

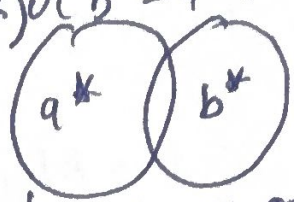
$A \Delta B =$ 

Other than common part, we need.

$\Rightarrow (A-B) \cup (B-A)$ two times you take difference.

Symmetri Difference is closed.

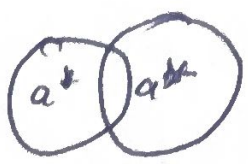
Problem 1: $L_1 = a^*$ $L_2 = b^*$

$L_1 \Delta L_2 = (a^*-b^*) \cup (b^*-a^*)$ 

only ϵ is common.

$\Rightarrow a^+ b^+$ or ~~Ba~~

Problem $L_1 = a^*$, $L_2 = a^*$

$L_1 \Delta L_2 = \phi$ 

Ex · $L_1 = a^*$, $L_2 = a^*b$ $L_1 = \{\epsilon, a, aa, aaa \ldots\}$

$L_1 \Delta L_2 = (a^* + a^*b)$ $L_2 = \{b, ab, a^2b, a^3b\}$

$\Rightarrow (L_1 - L_2) \cup (L_2 - L_1)$

$\Rightarrow L_1 \cup L_2$ ✓

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

$\Sigma = \{a, b, c\} \Rightarrow a^*(\epsilon + b)$

Problem $L = \{\epsilon, a, ab, abc, aabbbcc\}$

$\frac{1}{2} L = \{a, aab, \epsilon\}$, if $\frac{1}{2}$ is possible.

$\frac{1}{2} e = \epsilon$, as it can $\epsilon \cdot \epsilon$ or $\epsilon \cdot \epsilon \cdot \epsilon$ ...

* $\frac{1}{3}$ L = { aa, a, ∈ } (Tricky)

Context Free Language And Push Down Automata (10P+10P)

Grammar

TOC has its own grammar.

$G_1(V, T, P, S)$

V → Variables, T → Terminal, P → Production, S → Start Symbol (single)

① Except Start Symbol, rest are sets.

Ex $G_1 = (V, T, P, S \Rightarrow S)$

$V \Rightarrow \{S, A, B\}$ $T \Rightarrow \{a, b\}$ $P \Rightarrow \{S \to AB, A \to a, B \to b\}$

(set of production arrows are there)

*No uppercase, No lowercase are fixed. $T \to \{A, B\}$, $V \to \{a, b, k\}$

① Variables → { S, A, B } ⇒ three variables
② " { S, AB } ⇒ two variables
③ " { a, b, c } ⇒ three variable.

epsilon does not belong to variables or terminal.

Ex
① S → AB
② A → a
③ B → bc

Variables : { S, A, B }
Tamind : { a, b, c }

By default if in question it is not given. ① 8 first person is start Symbol
② whoever capital letters, variables
③ small letters, terminal

(ex) $S \to AB$
$A \to ab$
$B \to bc$
$AB \to C$

By default :→
Start → S
Variables → { S, A, B }
terminals { a, b, c }

(ex) · $G = (V, T, P, S)$ If everything is given

$V = \{ AB, CD, EF, S \}$

$T = \{ a, De, Fg \}$

$P = \{ S \to CD$, → one person

$AB \to a$

$CD \to De\ Fg$.

↑ ↑ terminals

Types of Grammars :→ (Check Page-1)

① Type-0 (Non-Restricted Grammar)
accepted by Turing Machine

① $\alpha \longrightarrow \beta$

② $\alpha, \beta \in (V+T)^*$
Variable and Terminal can be used in both α and β, Hence Unrestricted grammar.

② Type-1
⇓
Condition :→
① Type-0 (Must be T-0)

② $\alpha \longrightarrow \beta$
$\alpha, \beta \in (V+T)^*$

③ $|\alpha| \leq |\beta|$
number of α is less than or equal to β.

✗ $|AB| \to |C|$
Size of 2 — Size of 1 not allowed in Type 1

Type-2 Condition

① Must be Type-1

② Production is in the form

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

every production

① the left is single variable

② right side anything.

Left side is single variable

Type-3 Conditions

① Must be Type-2

② $V \rightarrow VT^* \mid T^*$

only one variable is allowed.

$V \rightarrow T^*V \mid T^*$

V is single variable, T is also single

VT^* ⇒ Left linear Grammar

variable is in

T^*V ⇒ Right Linear Grammar

Grammer :→

P1

① $S \rightarrow ABC$

② $AB \rightarrow DE$

③ $E \rightarrow F$

④ $F \rightarrow g$

More to Type→3 (Use by default conditions)

Left side in (2), two variables

T_0 ⇒

T_1 ⇒ T_1 $|AB| \leq |DE|$ ✓

T_2 ✗ 2 2

T_3 ✗ $|S| \leq |ABC|$ ✓

$|E| \leq |F|$ ✓

$|F| \leq |G|$ ✓

Type→2

P2

① $S \rightarrow ABC$

② $A \rightarrow BC$

③ $C \rightarrow D$

④ $D \rightarrow e$

~~Type-3~~

T_0 ✓

T_1 ✓

T_2 ✓

T_3 ✗ As only one variable is allowed in left side (1), (2)

T_3

(P3) $S \to AB$

$A \to a$

$B \to \epsilon$

T_0 ✓

T_1 ✗

T_2 ✓

(1) $|S| \leq |AB|$ ✓

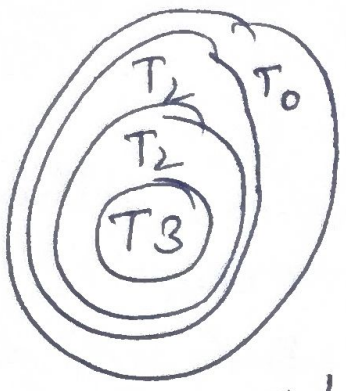
(2) $|A| \leq |a|$ ✓

(3) $|B| \leq |\epsilon|$ donot hold true.

Imp

T_3 ⇒ right side single variable allowed (×)

* Keep ϵ, to fail T_1.



True for non-null production.

(P4) $S \to Aa$ → LLG

$A \to aA \mid b$ → RLG

$B \to bB \mid a$

T_0 ✓

T_1 ✓

T_2 ✓

T_3 ✗ ($V \to VT^* \mid T^*$)

single variable followed by terminals

The grammar in T_3 must be either LLG or RLG but not both

Hence T_3 failed.

(P5) (1) $S \to BC$

(2) $BCD \to e$

(3) $F \to g$

T_0 ✓

T_1 ✗ $|S| \leq |BC|$, $|BCD| \leq |E|$ failed

T_2 ✗

T_3 ✗ ✓ T_0

(P6) (1) $S \to Aa \mid b$

(2) $A \to Ab \mid b$

(3) $B \to Bc \mid C$

(1) Check ϵ transition ✗

(2) T_0 ✓

(3) T_1 ✓

(4) T_2 ✓

(5) T_3 ✓ "T_3"

Procedure :→

① Check the left

(Pb) $S \to a$ → LLG, RLG → T_0 ✓
$A \to b$ → T_1 ✓
↑ → T_2 ✓
LLG, RLG → T_3 ✓ ~~✗~~ both production is either
① LLG RLG
② LLG RLG

* Contextfree Grammar generates Context Free Language.
Similar to RoG generates Regular Language.

Context Free Grammar :→ To generate strings.

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

Left side are single variable Hence T_2.

Expansion or Derivation

S
⇓
AB
⇓ ⇓
a b

strings → a collection of terminals.

Terminals →

$L(G) = \{ab\}$ only one string is possible.

$S \to aS$ ⇝ recursion as S is replaced.
this will generate infinite strings.

Purpose of Grammar :→
① Generating strings.

Ex

$S \to aS|b$ rewrite $\begin{cases} S \to aS \\ S \to b \end{cases}$ two production is there.

1st Minimal strings :→

① S ⇓ b → b

② S ↙↓ a S ⇓ b → ab

③ S ↙↓ a S ⇓ a S ⇓ b → aab

..... Any number of a's followed by b.

⇒ a*b. R.E → R.L → CFL

Note CFG does not gurrantee R.E. (as restriction)

Note

① Every R.G generate Regular language. [R.L means CFL, but vice versa ~~be~~ not true.]

② Every CFG generate CFL.

Every recursion has ~~termination~~ to be terminated. by using terminal.

Ex $S \to aSbS|bSaS|\epsilon$

$S \to aSbS$

$S \to bSaS$

$S \to \epsilon$

this

Generate ⇒ abab

① aSbS, I can use this

Check Type

① Type 0, Type 1

② Type 2 ⇒ ✓

③ Type 3 ⇒ X as we two product

Note

① Derivation Tree (parse tree): Visual representation of the process by which CFG generates a particular string.

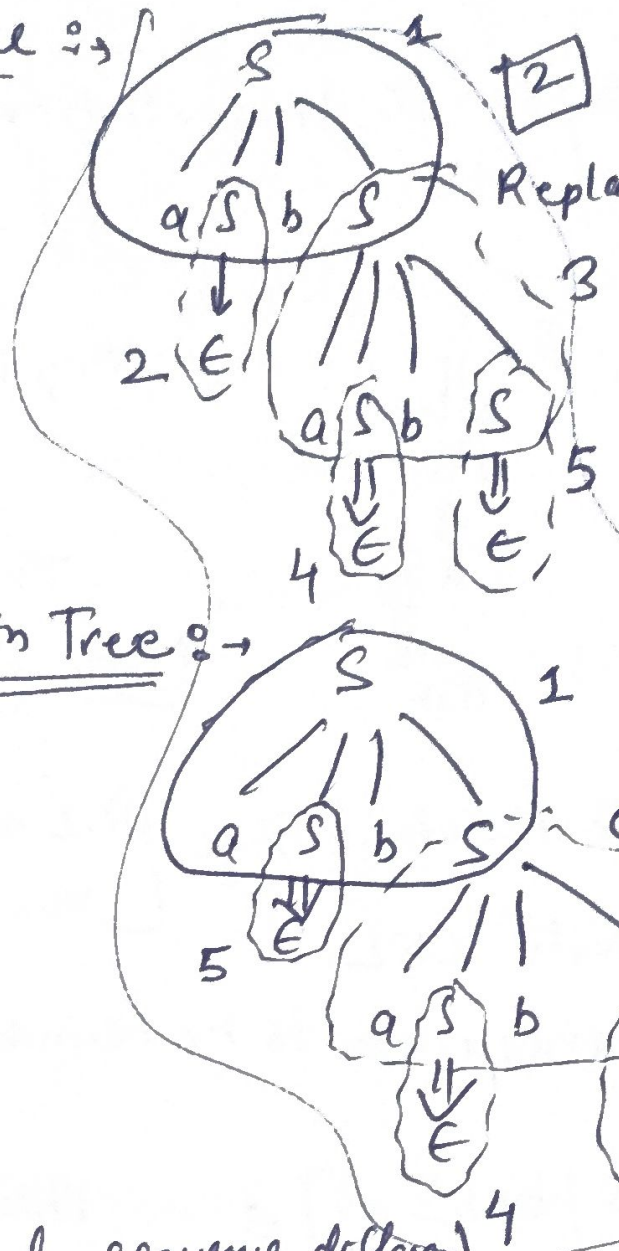
Elements of DT

① Root Vertex

② Vertices

③ Leaves.

Left Derivation Tree :→



Right Most Derivation Tree :→

LMDT = RMDT (only sequence differ)

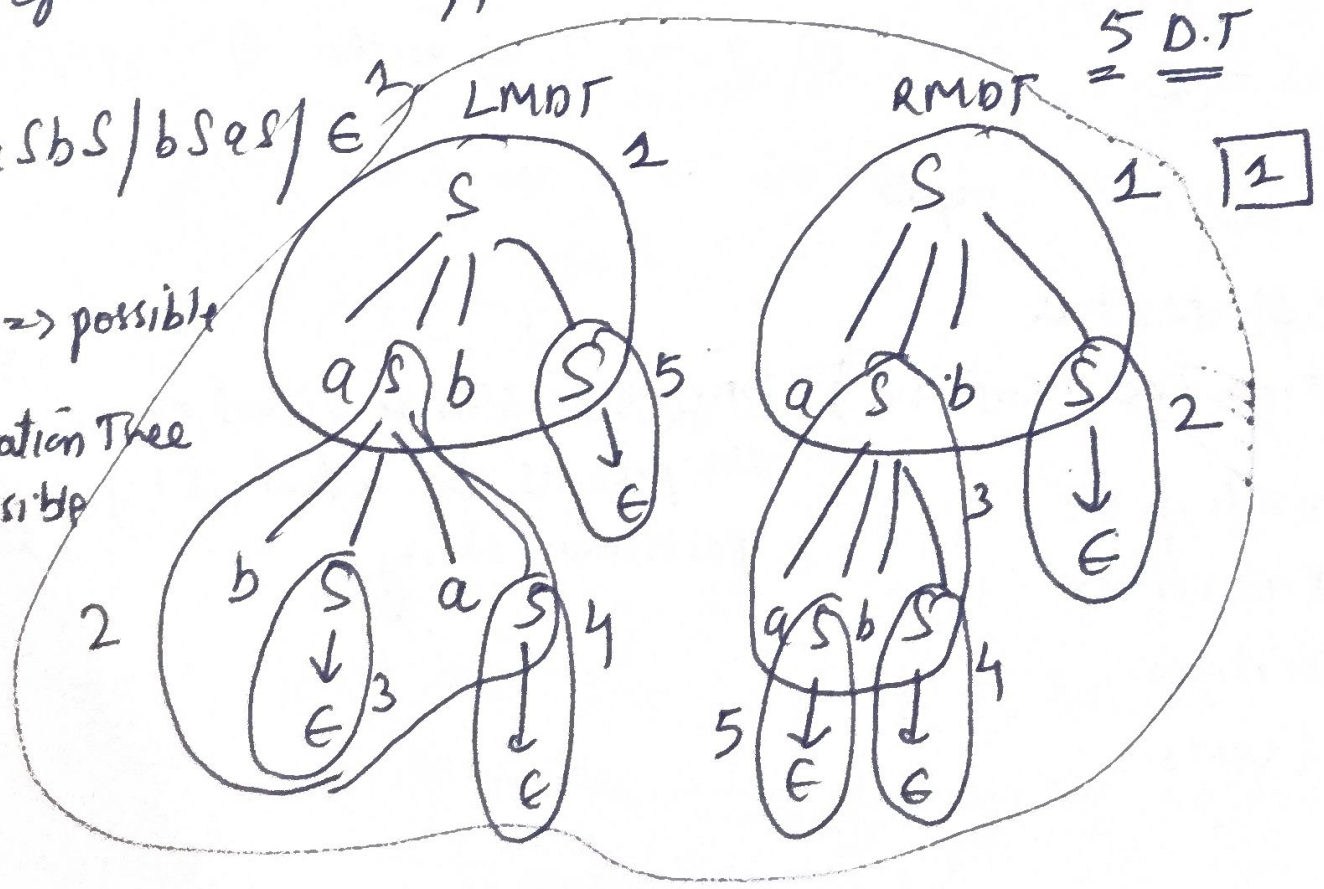
~~Here both are same~~ 5 Different L.M. D.T = 5 RMDT, but = 5 D.T

$S \rightarrow aSbS/bSaS/\epsilon$

I/p abab → possible

2 → Derivation Tree is possible

[1] [2]



① **Ambigious Grammar** :→

For atleast one string, for which ~~multiple~~ atleast 2 derivation tree is possible is known as ambigious grammar.

② Number derivation Tree is equal to number LMDT and equal to RMDT.

DT = LMDT = RMDT

Ex

* $S \to Aabb$
  $A \to aA \mid bA \mid \epsilon$

Recursion ?
Yes. $A \to aA \mid bA \mid \epsilon$. (Use this)

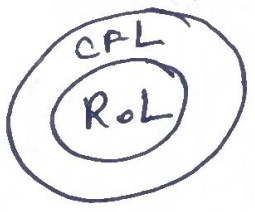
① S
↓
Aabb
↓
ϵ

② $A \to aA \mid bA \mid \epsilon$

$\Rightarrow \epsilon, a, b, aa, bb, ab, ba, \ldots \Rightarrow (a+b)^*$

$\Rightarrow (a+b)^*abb \Rightarrow$ RoE $\Rightarrow$ RoL $\Rightarrow$ CFL.

* You created a bigger machine.



Ex

$S \to AaBaC$
$A \to aA \mid bA \mid \epsilon$
$B \to bB \mid aB \mid \epsilon$
$C \to aC \mid bC \mid \epsilon$

① Guarantee CFL $|S| \leq$ | Three Variables |
~~NOT RE expressible~~.

Check every variable

$A \to aA \mid abA \mid \epsilon \to (a+b)^*$
$B \to aB \mid aB \mid \epsilon \to (a+b)^*$
$C \to aC \mid bC \mid \epsilon \to (a+b)^*$

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

① Atleast two a's.

② It is only the side effect, as you made a bigger mach.

If question asked is,

① Gramer ? Type 2

② what is the language ? CFG, CFL

③ Check Regular ? if possible

If (3) is true, the type language will generate is Regular.

① CFL guranttee

② for RL check.

Ex
Write CFG, for the lang $L = \{ab(a+b)^*\}$.

$$S \to abS \mid aS \mid bS \mid \in$$

or

$$S \to abA$$
$$A \to aA \mid bA \mid \in \quad (a+b)^*$$

I have done R.L.G.

Ex Write CFG $L = \{a^*b^*\}$

$$S \to AB$$
$$A \to aA \mid \in \longrightarrow a^*$$
$$B \to bB \mid \in \longrightarrow b^*$$

$L = \{\in, a, ab, aab, \ldots$
a follows by b.

Eg
Write CFG $L \{a^+b^+\}$

$$S \to AB$$
$$A \to aA \mid a \Rightarrow a^+$$
$$B \to bB \mid b \Rightarrow b^+$$

$L = \{ab, aab, \quad\}$

Actual purpose :→

Ex: Give CFG? $\{a^n b^n \mid n \geq 1\}$. Regular Language not possible

Hence, R.G is not possible.

* Comparison is required.

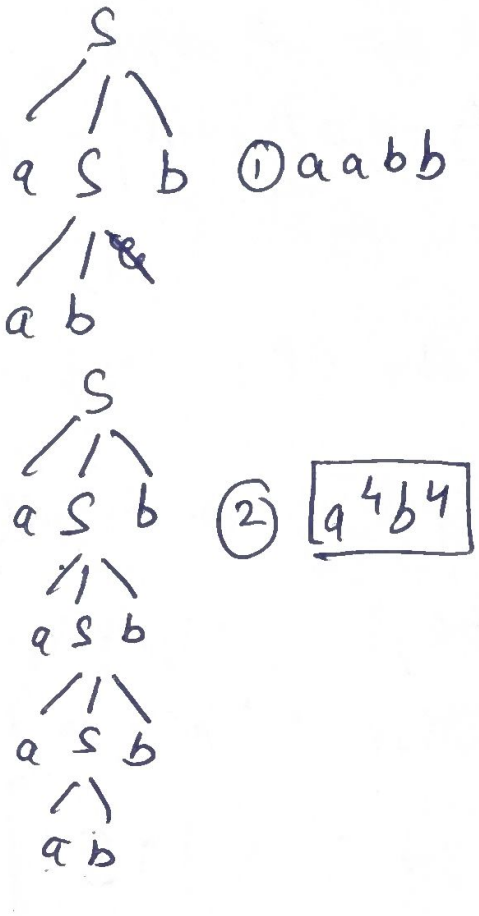
① Relation is required, donot divide.

$S \rightarrow ab \mid aSb$

Note :- Relation is there, donot use different variable

$S \rightarrow AB$ false

this will generate aab, abb etc.



① CFL is there, CFG is also there.

In this problem, R.L failed at comparison.

Imp

Ex Give CFG $L = \{a^m b^n c^n \mid m, n \geq 1\}$

① b, c are in relation.

② a, bc, a, c

$S \rightarrow AB$

$A \rightarrow a \mid aA$ $\{a^n \mid n \geq 1\}$

$B \rightarrow bc \mid bBc$ → as relation is there, we write b c in same production.

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

Ques 1 If CFG possible ✓, then Regular may be.

Ques 2

Give CFG, $L = \{ \underbrace{(a+b)^*}_{A} abb \underbrace{(a+b)^*}_{A} \}$ contains "abb."

$S \rightarrow AabbA$

$A \rightarrow aA \mid bA \mid \in \Rightarrow (a+b)^*$

$\Downarrow$

CFL

This fake CFL, originally Roh.

Ex Give CFG $L = \{$ set of all pallindromes over the alphabet $0,1\}$

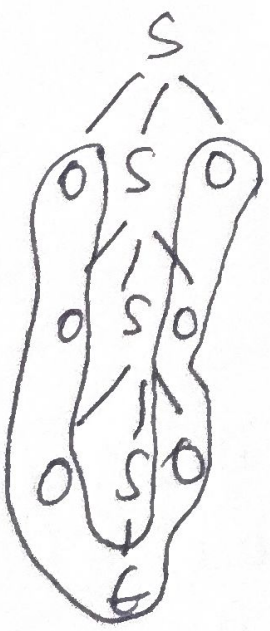
$L = \{\in, 00, 010, 101, 11011, 10101, 11, 111, 000, \ldots\}$

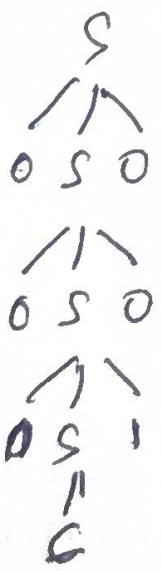
Relation 1st and last one relation.

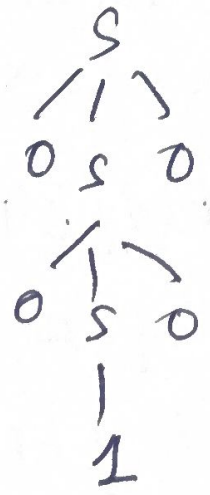
$S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \in$

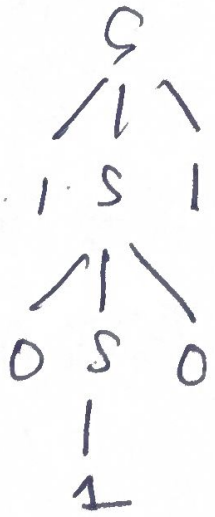
$\in$ will give even length pallendrom

$0 \mid 1 \rightarrow$ will give odd length pallendrome.









① - 000000 even length

② 001100

③ 00100 odd length

④ 10101 odd length

Previous Problem :->

① Fake CFL or not? No, original CFL., as comparision is required.

Problem Give CFG, $L = \{ a^i b^{i+j} c^j \mid i,j \geq 1 \}$ L-20 (1:56:00)

Minimal string $a^i b^i b^j c^j$ $i=1, j=1, i+j=2$

① abbc ② abbbcc $i=1$ $j=2$ $i+j=3$

③ abbbbccc $i=2$ $j=3$ $i+j=4$

① a, b relation
② b, c relation

b is the common friend

$S \rightarrow a^i b^i b^j c^j$ a going b also come.

$A \rightarrow ab \mid aAb$ $a^i b^i \mid i \geq 1$

$B \rightarrow bc \mid bBc$ $b^j c^j \mid j \geq 1$

Problem $L = \{ a^m b^n c^n \cup a^m b^m c^n \mid m,n \geq 1 \}$ L-20 (2:06:00)

$a^m b^n c^n$: A, Relation B, S_1

$a^m b^m c^n$: D, DC, Relation

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow AB$
$A \rightarrow a \mid aA$
$B \rightarrow bBc \mid bc$

$S_2 \rightarrow DC$
$C \rightarrow cC \mid c$
$D \rightarrow aDb \mid ab$

Properties :->

$S_1 \rightarrow$ CFL

$S_2 \rightarrow$ CFL

Union of two CFL is also CFL

(#) CFL are closed under Union.

① UNION operator, one after another., as we have required only stack, at a time in union,

$L_1 = a^m b^n c^n \mid m, n \geq 1 \Rightarrow$ CFL

$L_2 = a^m b^m c^n \mid m, n \geq 1 \Rightarrow$ CFL

Property :→ $L_1 \cap L_2 = a^n b^n c^n \mid n \geq 1 \Rightarrow$ non-CFL

why?, two stacks are required., Hence CFL not possible for this language.

① More than one comparsion will required be req, CFL is not possible.

② More than on comparsion about in Union, then only one stack required.

Imp points

① $S \rightarrow aS \mid \epsilon$

① One Grammer will generate only one language.

Reverse.

$L = a^*$

$S \rightarrow aS \mid \epsilon$ (RLG) $\qquad S \rightarrow Sa \mid \epsilon$ (LLG) $\qquad S \rightarrow aS \mid Sa \mid \epsilon$

Note: A single Language, may contain multiple grammar.

Write CFG $L_2 \{ a^n b^n c^n \mid n \geq 1 \}$

X $S \rightarrow abc \mid aSbScS$

a b c c come before b

X $S \rightarrow abc \mid abcS$

X If you try without seperation, concatenation will be broken.

Context Free Grammar is not possible.

① If the relation is broken.

② more than 1 comparison is required.

Properties

CFLs are closed under concatenation.

$L_1 \cdot L_2 = L_1 L_2$

$S \rightarrow A.B$

$A \rightarrow aA \mid a$ this will generate a^*, a CFL (L_1)

$B \rightarrow bB \mid b$ this will generate b^* a CFL (L_2)

For concatenation, just write a new production.

$S \rightarrow A.B$ — B production.

↳ A production

CFG $L = \{ a^i b^j \mid i \neq j \quad i, j \geq 1 \}$ (L2L: 31:00)

How many comparison ① one

Possibility :→

① $i < j$

② $i > j$

Not equal, equal

i, j $1 \neq 2$ $2 \neq 1$

$1 \neq 3$ $2 \neq 3$

$2 \neq 4$

↳ comparison is required

① Before generating a language :→

① Try bottom up approach.. for production

No this first ② Generate the actual Language.

$i > j$

$S \rightarrow \overbrace{CA}^{a^+ a^i b^i} \mid \overbrace{AB}^{a^i b^j b^+} \quad i < j$

① $A \rightarrow ab \mid aAb \quad a^i b^i \mid i = j$

② $B \rightarrow b \mid bB \Rightarrow b^+$

③ $C \rightarrow a \mid aC \Rightarrow a^+$

$a^5 b \mid a^6 b^5$

Approach :→

① Generate $i = j$ language.

② Then to ~~increase~~ make unequal, generate atleast one extra.

Point to Remember :- For each string, you have to make a ~~Derivation~~ Tree

Derivation Tree / Parse Tree.

Give CFG L = { set of all valid balanced parenthesis }

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

① ()✓

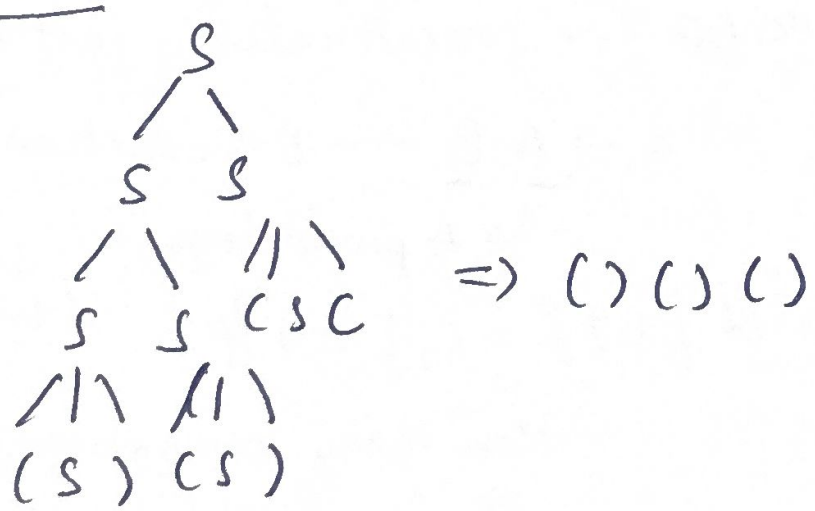
②)(X

③ (())

④ ()()()()

⑤ ((( )()))

Imp: For getting fourth ~~pro~~ string :→ just add SS, which is recursion.



$\Rightarrow ()()()$

Chat [()S, S()]

(25) Give CFG for

the language = $L = \{ WW \mid W \in (a+b)^* \}$ even length pallindrom WWR

L-21 (1:22:00)

Famous problem

Imp

$S \longrightarrow aSa \mid bSb \mid a \mid b \mid \epsilon$

Odd length pallindrom WCWR

C can be a or b

Whole :→ Set of all pallindrome

$\begin{bmatrix} e+e=e \\ o+o=e \end{bmatrix}$

$\begin{bmatrix} e+o=o \\ o+e=o \end{bmatrix}$

How to solve?

① Construct CFG

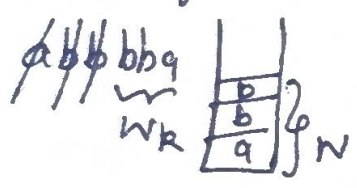
Or/and

② Using single stack.

$\epsilon.\epsilon = \epsilon$
$a.a = aa$
$b.b = bb$
$ab.ab = abab$
$ba.ba = baba$

aa/aa

(Read multiple times)



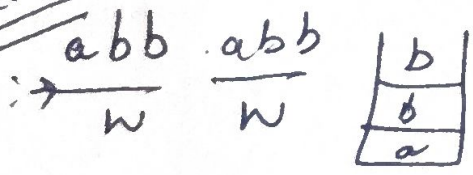
Compare top of stack and W_R first symbol

① if a symbol is from W, push in the stack.

- push over,

② W_R first will be compared to the top of the stack, which is the last symbol of W.

Observation



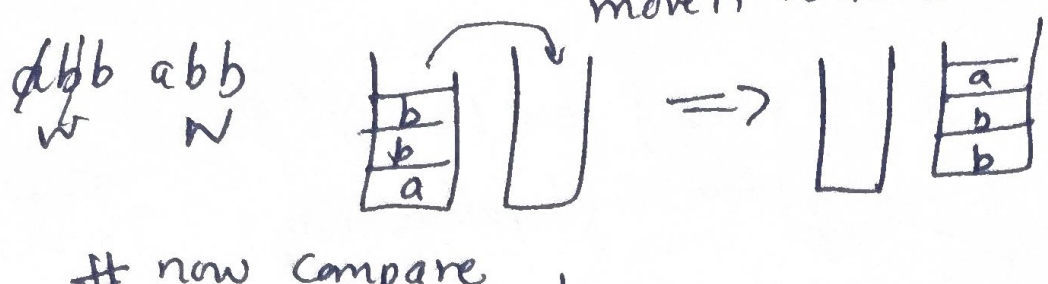
I have to compare first symbol of W must be compared to first symbol of W which is in the stack.

But it is in the bottom.

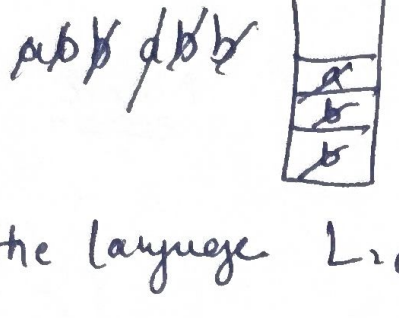
So to solve above problem, you need atleast two stacks or queue which will become Queue.

$L = \{ WW \mid W \in (a+b)^* \}$ or Queue.

* It is not CFG, as we required two stacks.



now compare



⇒ Hence possible with two stacks

Give CFG for the language $L = \{ \overline{WW} \mid W \in (a+b)^* \}$

L^C → complement of language.

$\overline{WW}$ complement.

Note: every WW is not $\overline{WW}$ but a $\overline{WW}$ may be WW.

4 length string

aaab → complement of WW

WW	$\overline{WW}$
$\epsilon \cdot \epsilon = \epsilon$	$a\alpha$
$a \cdot a = a$	α
$b \cdot b = b$	α
$a\alpha$	a, b

all even

Imp

even length
ab, ba
* aba, bab
* aaa, bbb
a^5, b^5
all odd length
will be either complement + some even

aabb
abba

Hard

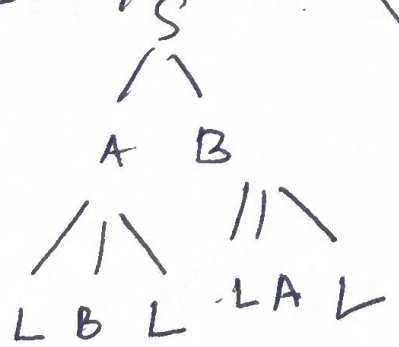
(4) $S \to AB \mid BA \mid D$ (even, even)

(3) $B \to LBL \mid b$ → middle symbol is b of odd length

(2) $A \to LAL \mid a$ → middle symbol is "a" of odd length

(1) $L \to a \mid b$

$D \to LLD \mid a \mid b$ → all odd length



→ a, b, a/b a/b a/b
D

CFL is not closed under complement operator..

$L = a^n b^n c^n$ / $\overline{L} = a^n b^n c^n$ ⇒ non-CFL ⇒ CSL / ⇒ CFL

$L = WW$ ⇒ non CFL

$L = \overline{WW}$ ⇒ CFL

Hence, need not be closed under. or no guarantee

& complement.

$L = \{ a^n b^n / n \geq 1 \}$ ⇒ CFL possible

$S \rightarrow aSb / ab$

$L = \{ (a^n b^n)^* / n \geq 1 \}$, $L = \{ \epsilon, a^n b^n, (a^n b^n)^2 \ldots \}$

$S \rightarrow aSb / ab / SS / \epsilon$ $a^n b^n a^n b^n \ldots\ldots$

take care of $(x.y)^*$

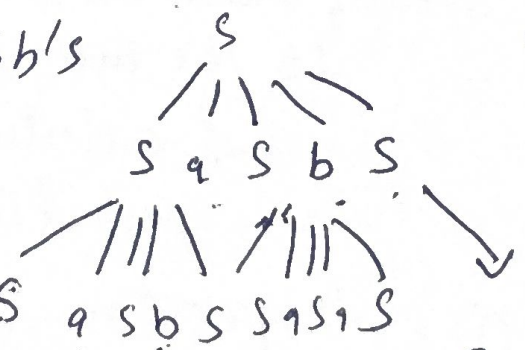
Problem

Give CFG for the language $L = \{ w / w \in (a+b)^* \; n_a(w) = n_b(w) \}$

$L = \{ aabb, abab, \epsilon, abab \}$

$S \rightarrow \epsilon / SaSbS / SbSaS$ ab, ba, ϵ order doesn't matter

5 a's, 5 b's



abaabab

Give CFG :- $L = \{ w / w \in (a+b)^* \; n_a w = n_b(w) + 2 \}$

$L = \{ aa, aaab, aaaabb \}$

can be anywhere

$S \rightarrow aa / S_1$

$S_1 \rightarrow SaSaSaSbS / aaab / baaa / abaa / baaa / aaba$

Header

$S \rightarrow AaAaA$

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

or

$S \rightarrow AaAaA$

$A \rightarrow aAb \mid bAa \mid \epsilon$

Lecture 22 :→

Push Down Automata :→ (PDA)

Push → push().

Down → pop().

PDA = Finite Automata and one stack.

PDA Machine = $(Q, \Sigma, \delta, S, F, \Delta, Z)$

Q, Σ, δ, S, F → FA

Δ, Z → Stack

$\delta : Q \times \Sigma \times T \rightarrow Q \times T^*$

(transition function)

Takeany state q in Q, applied some symbol and T (tao), top of the stack, based on these, will go to the some state and collection of symbol pushed in stack.

① Δ → stack alphabet

② Z → initial top element of a stack.

③ $Z \in \Delta$.

① push operation. $b \rightarrow$ pushed x on b xb

② $b \rightarrow \epsilon$ (pop operation)

③ $b \rightarrow b$ (nothing changes)

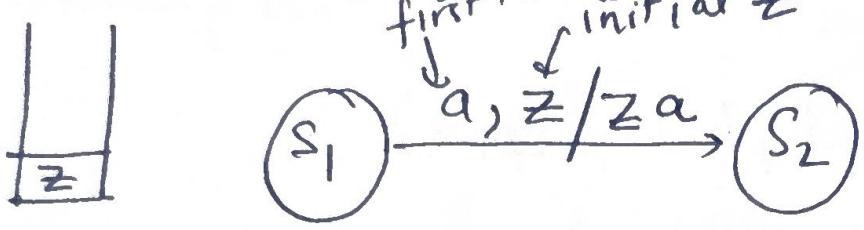
increase → push

decrease → pop

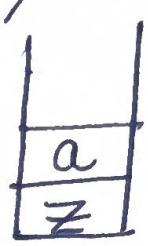
$\delta : Q \times \Sigma \times \Gamma \longrightarrow P(Q \times \Gamma^*)$ Non-Deterministic PDA as power set is there.

Operations :- push, pop, skip

Push :-> i/p aaabbb # → # denotes end of the string.



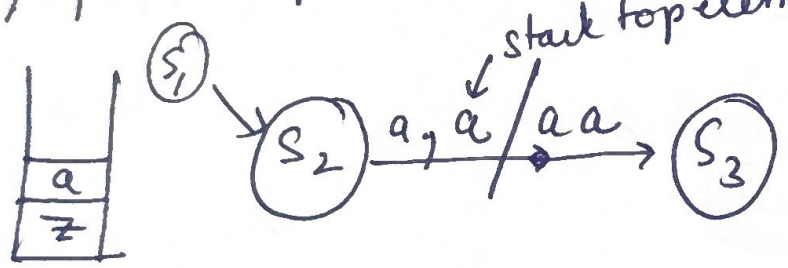
i/p : aaabbb #



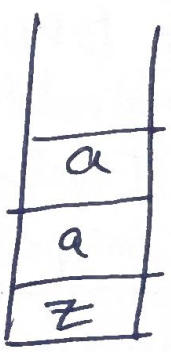
$\delta(S_1, a, Z) \longrightarrow (S_2, Za)$

Input Output

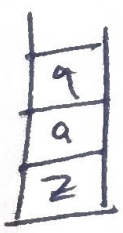
i/p : aaabbb #

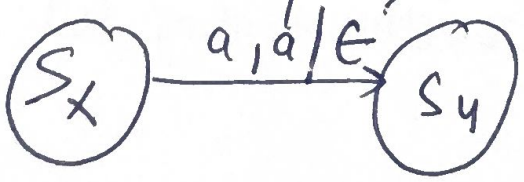


i/p : aabbb

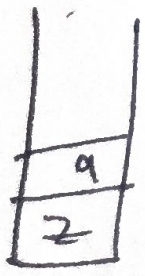


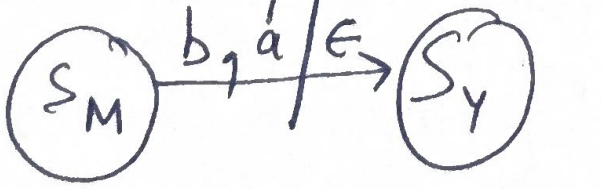
Pop operation :->

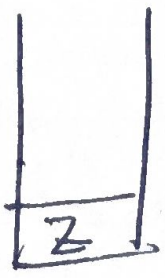
① 



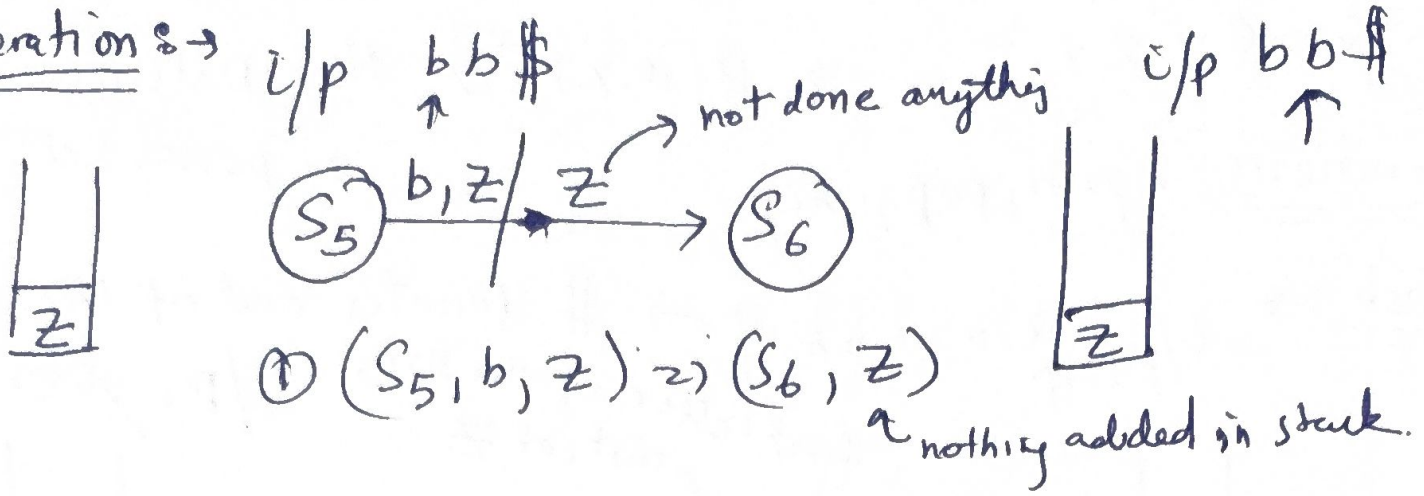
② bbb $$







Skip operation :→



① $(S_5, b, Z) \rightarrow (S_6, Z)$ → nothing added in stack.

② $(S_6, b, b) \rightarrow (S_7, Z)$

③ $(S_7, \phi\$, Z)$ end of the string.

Skip operation :→ nothing added to stack.

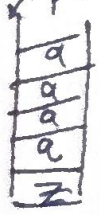
Problem 1

Construct DPDA $L = \{ a^n b^n \mid n \geq 1 \}$ ① CFG ✓ ② PDA ✓

① No valid is missing
② Invalid must be rejected.
* ③ Comparesion required

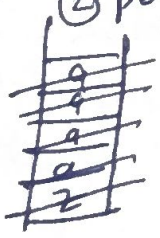
a a a a b b b b $ Then pop start

① push



→

② pop (a).



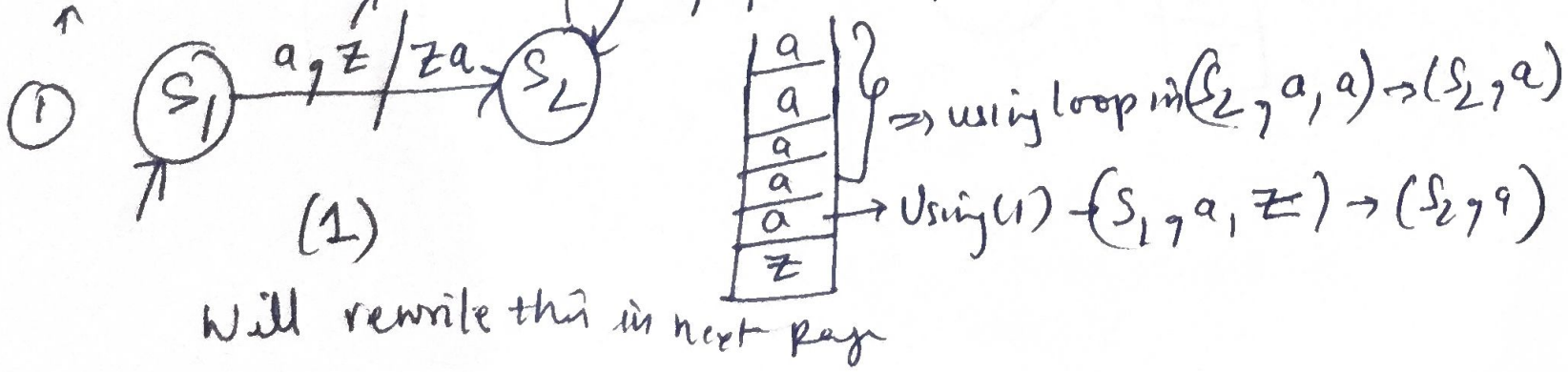
If the reading symbol
① If the input symbol

Valid :→

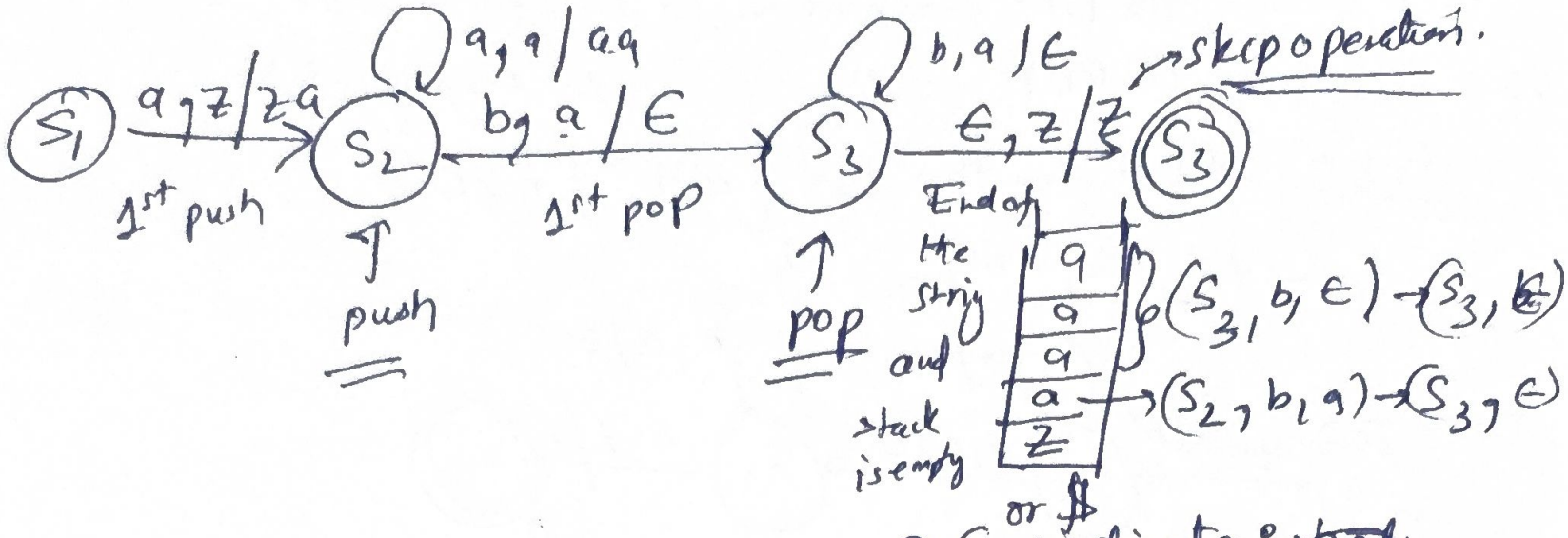
① b is over and a are all poped.
② the last symbol in stack is Z.

If empty. then on a ⇒ self loop

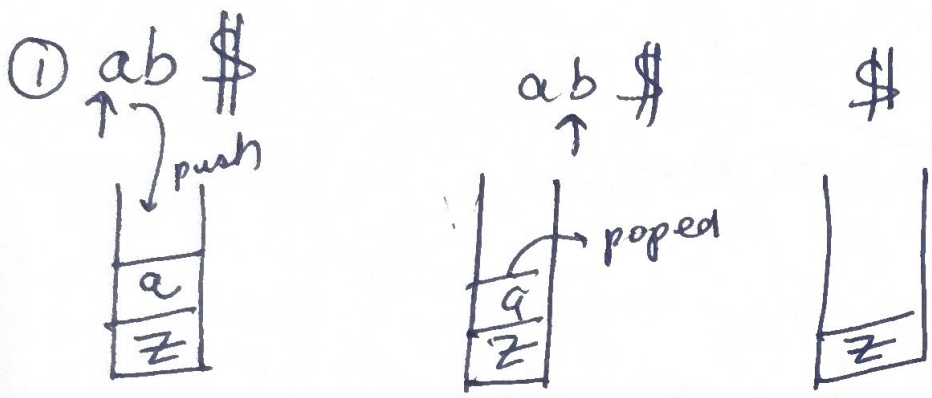
a a a b b b b



Will rewrite this in next Page



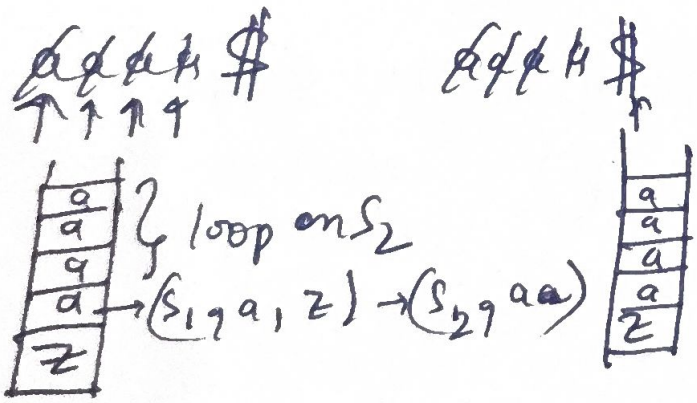
① ∈ indicates all b's are read.



② $, Z, moved to final state.

② , in this case, ~~th~~ In bigger machine automatically go to dead state and you need not to mention.

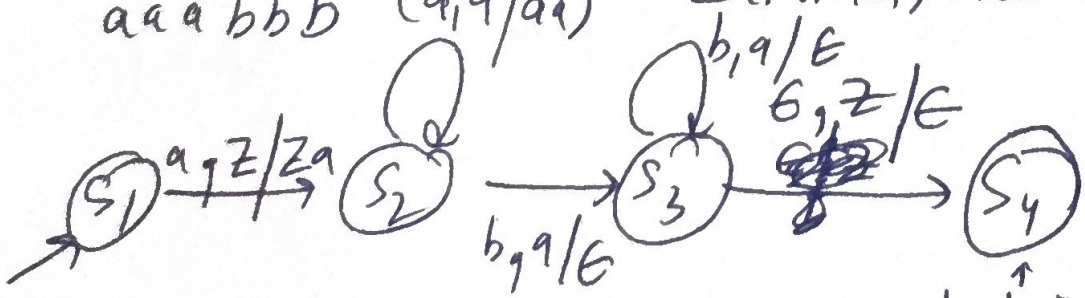
(S1, ∈, Z) → string is rejected.

② 

stack is not empty. by default go to dead state.

③ PDA accepted by ~~dead state~~ empty stack. In this, Z (initial) must also be removed.

aaabbb



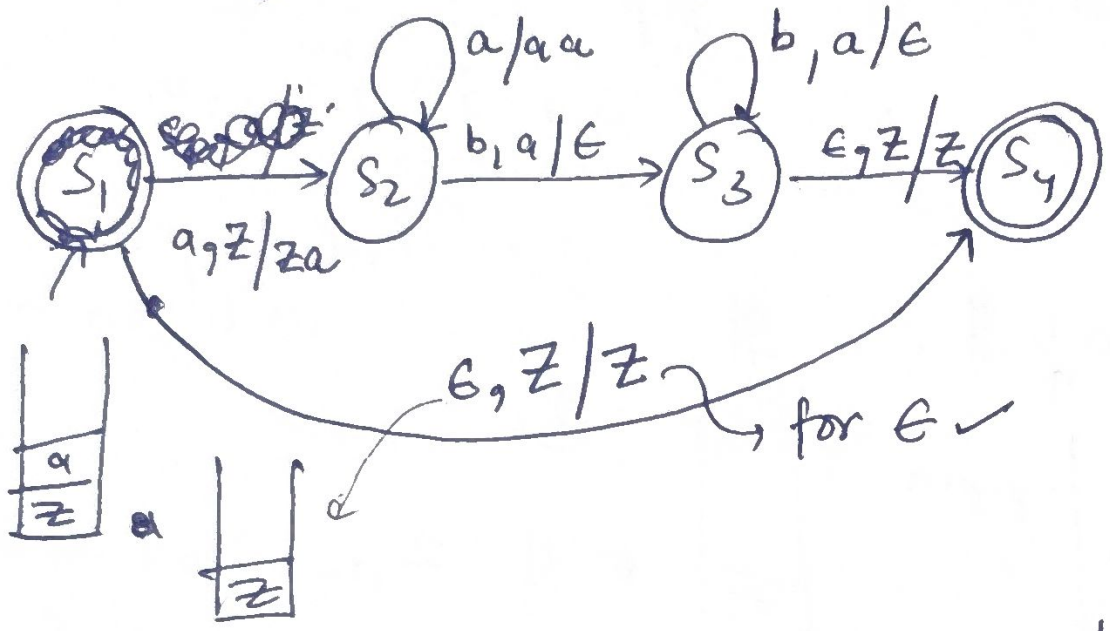
• Stack is empty.

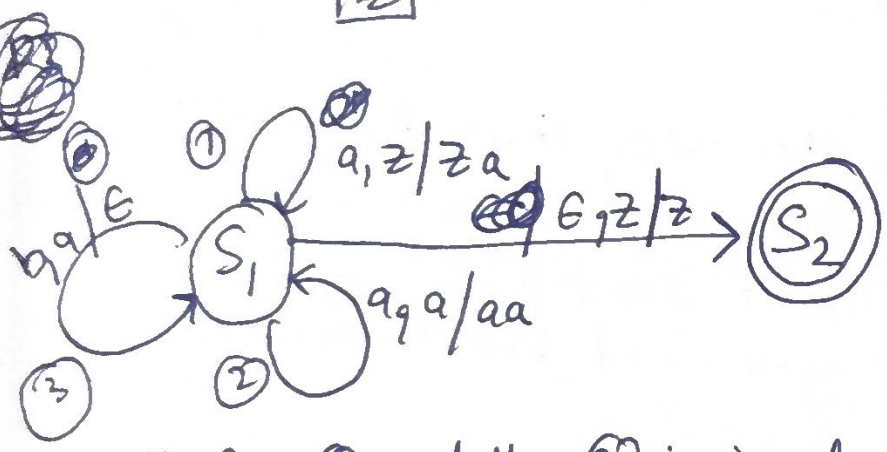
Acceptance :→ ① PDA acceptance in the final state

(P2) ② PDA " " " empty state

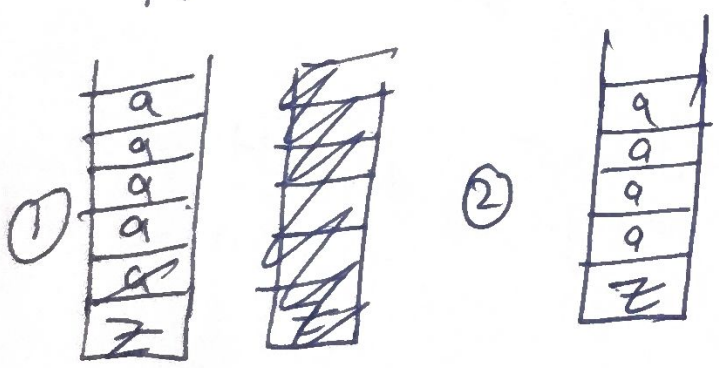
Construct DPDA for the $L = \{a^n b^n \mid n \geq 0\}$

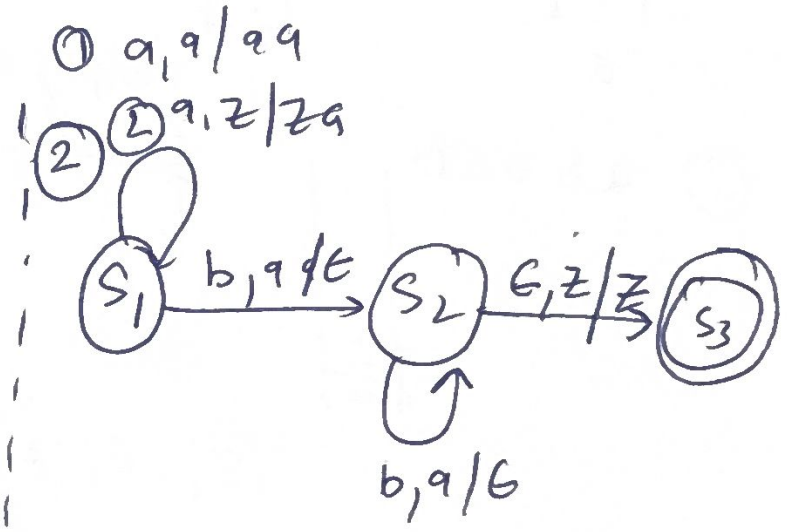
$L = \{\epsilon, ab, a^2b^2, \dots\}$





In S_1, ① and Item ② is in order.





① This will accept

②

① bbaa

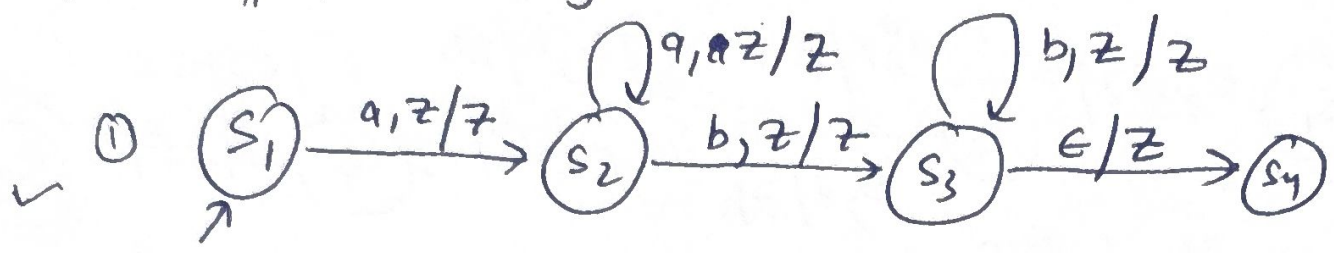
② abab → this will also attempted. but should not be accepted

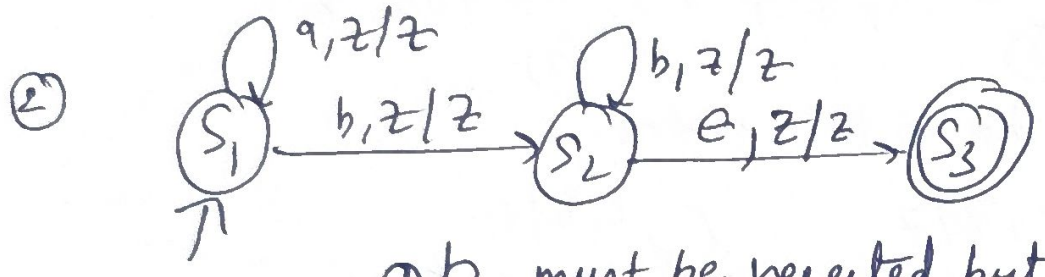
Hence, the above is wrong

~~Wrong accepted in this case.~~

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

$\{ab, aab, \ldots\}$





① b_1 must be rejected but here it is accepted.

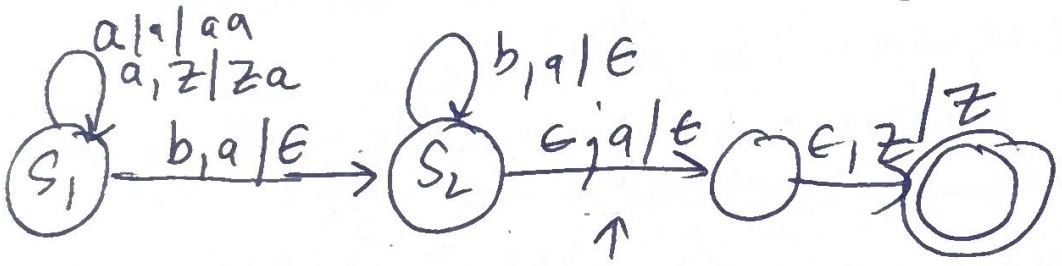
#. Construct DPDA $L = \{a^m b^n \mid m, n \geq 1 \text{ \& } m = n+1\}$

$a^5 b^5$

aaaaabbbb ϵϵϵϵ → can be rewritten as.

① a is one extra in comparision to b.

② not 20 extra.



atleast one a is extra

one a is only extra.

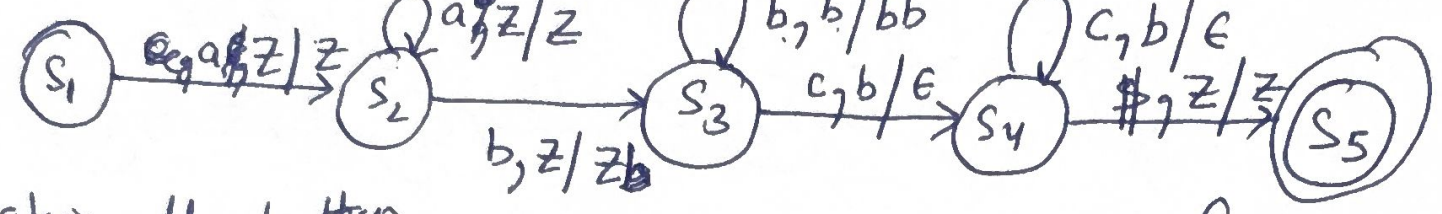
Cases: → ① aaa
② bbb
③ ab
④ ϵ
⑤ aab

$L = \{a^m b^n c^n \mid m, n \geq 2\}$

Lecture-23

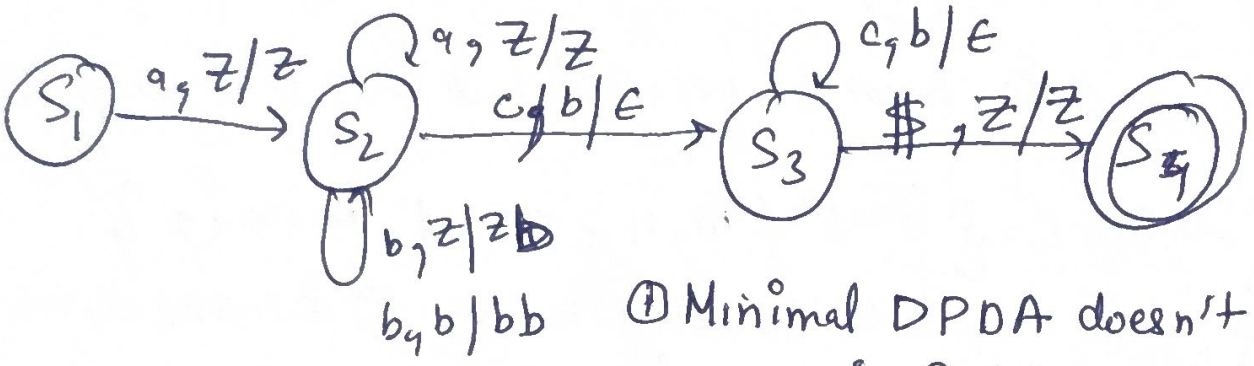
Draw DPDA : $L = \{ a^m b^n c^n \mid m, n \geq 1 \}$

aaa bbbbb ccccc $



① skip all a's ①, then push b's ②, pop b's for each b's ③.

Must be in this order



① Minimal DPDA doesn't exist in Push Down Automata. So, both are correct.

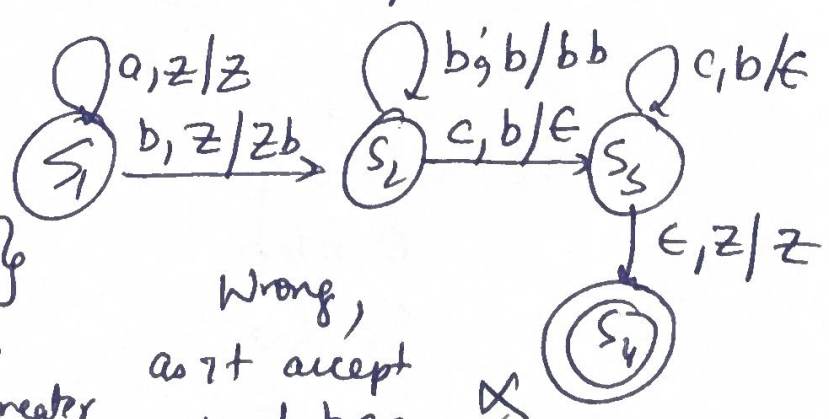
- Problems Type :→ Language will be given,
  * DPDA is already given, no need reconstruct in problem.

For abcm Cases : abc ✓, aabc ✗, $a^3b^3c^3$, $a^2b^3c^3$, abcc ✗

Draw DPDA $L = \{ a^m b^n / m, n \geq 1 \}$

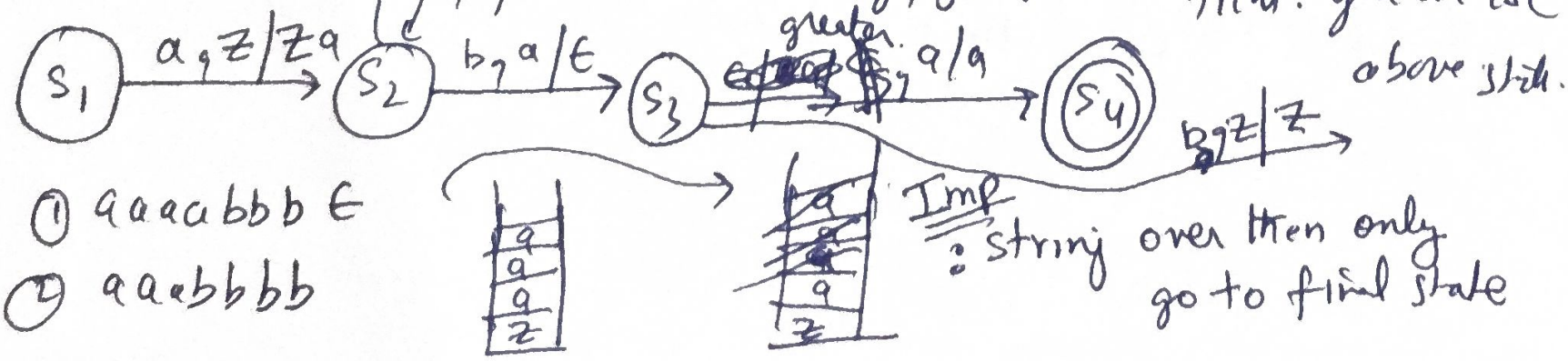
$\$ \{ m \neq n \}$

Do not a>b or a<b less than, greater than

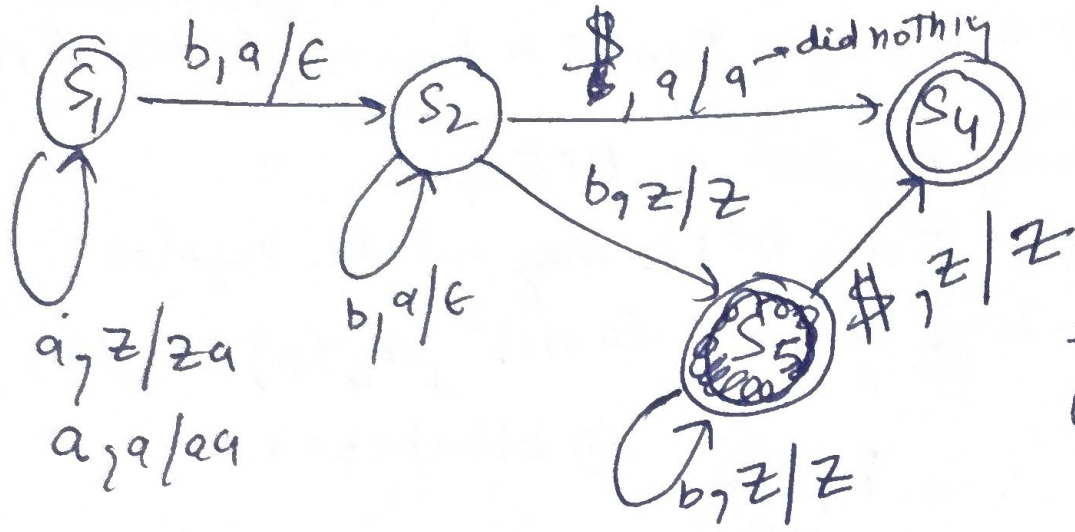


Wrong, as it accept * bbcc

* Remember, you don't have to count exactly, just prove greater



Hint: you can use above sth.



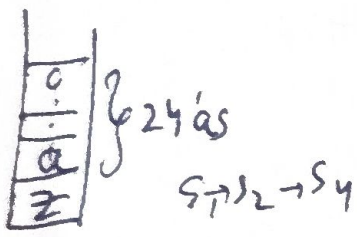
Down path related to

Remember

① You have read all the characters of a string.

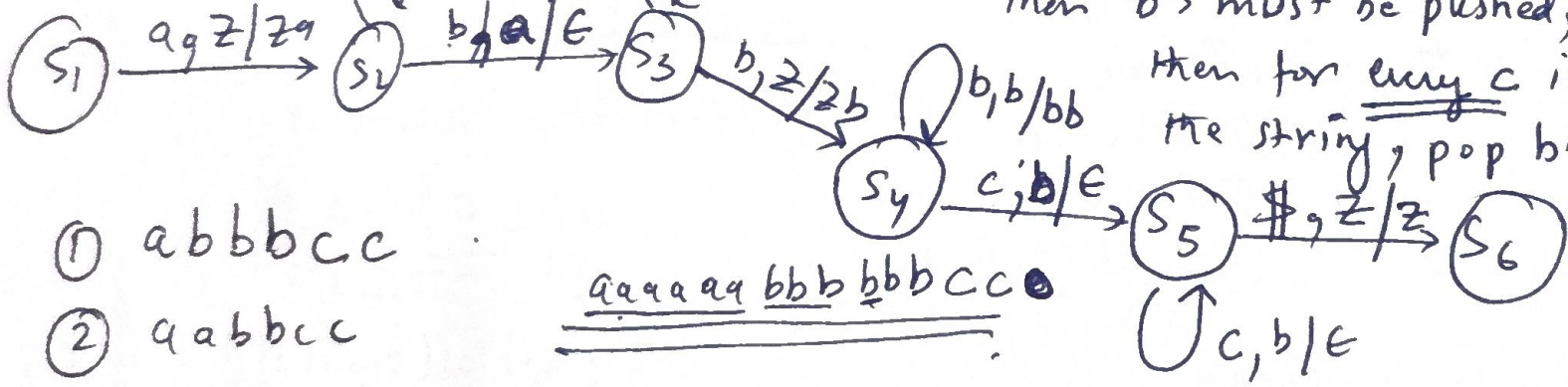
② Not covered will automatically go to dead state. [No need to Draw]

$a^{25}b$, ab^{25}



Draw DPDA $L = \{a^m b^{m+n} c^n \mid m, n \geq 1\}$ (L → 23 | 07 : 00)

Logic :→ a' push, then for each b, pop a's. Then you reach empty stack, then b's must be pushed, then for every c in the string, pop b's.

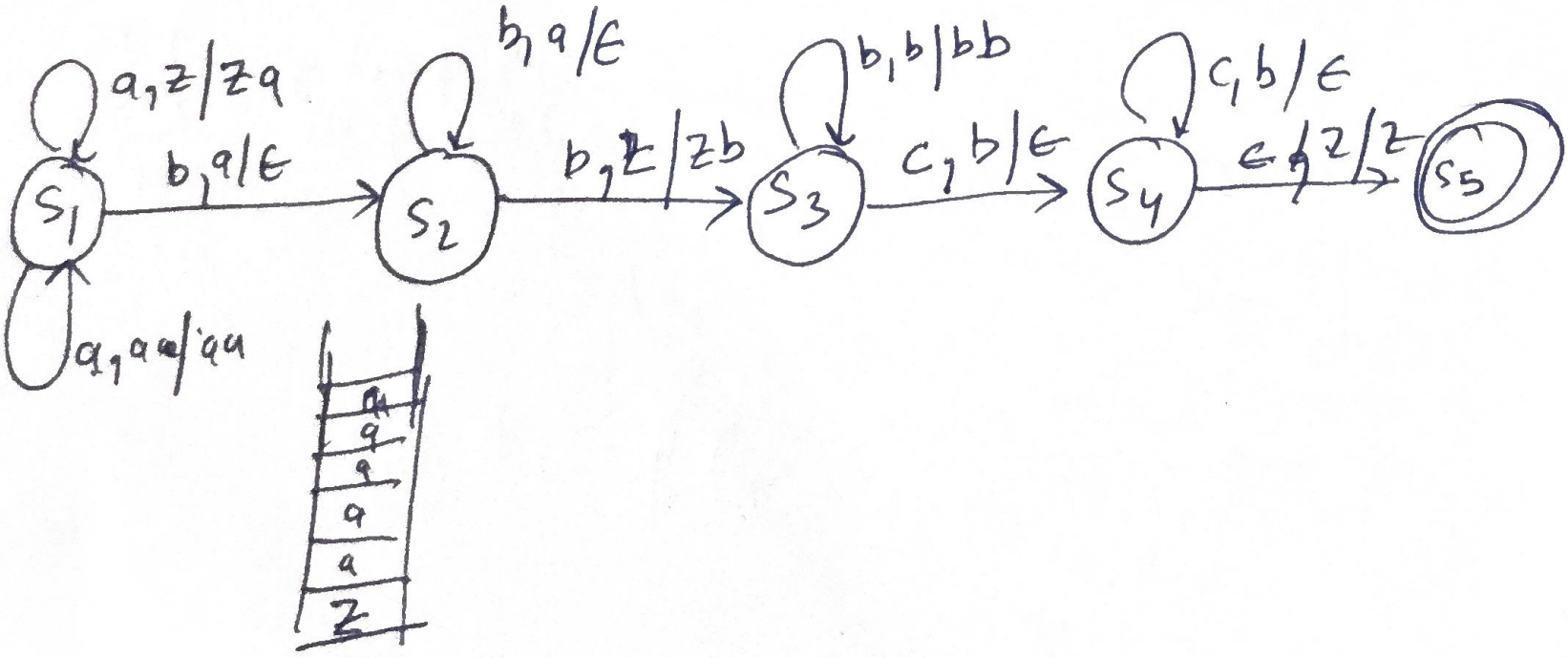


① abbbcc

② aabbcc

③ aaabbcc

aaaaaa bbb bbb cc



Deterministic.
#If DPDA possible for a language, then it is known as Context Free Language. (DCFL)

① Every Regular is DCFL

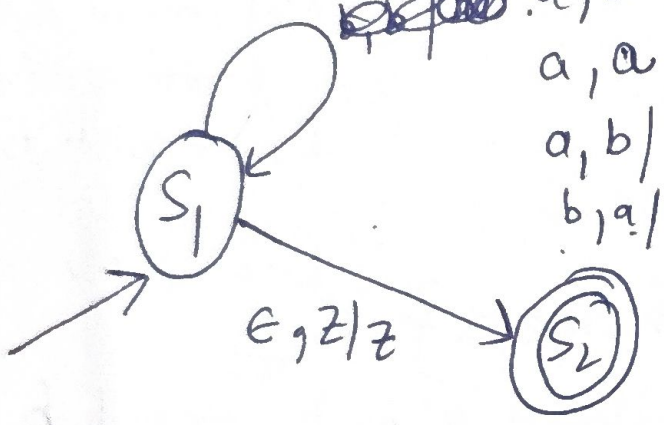
② But Every DCFL may not be regular.

Draw DPDA for the $L = \{ w \mid w \in (a+b)^*, n_a(w) < n_b(w) \}$

① abab ab
② aaabbb
③ abbaabbaa.
④ bbbabab aa

b, b/bb
b, Z/bZb
a, Z/Za
a, a/aa
a, b/ε
b, a/ε

If top remains non empty, then,-

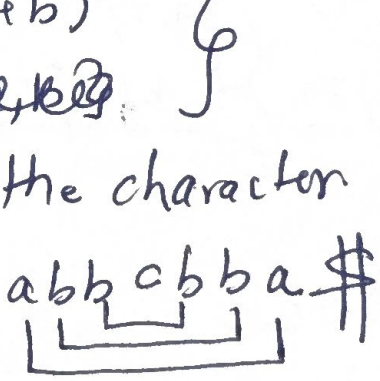


Draw DPDA $L = \{ wcw^R \mid w \in (a+b)^* \}$

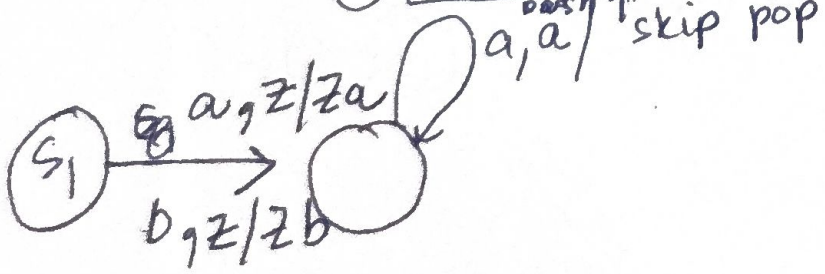
odd length pallindrome

① abbcbb
② bacab
③ abbabcbba

- c is the character

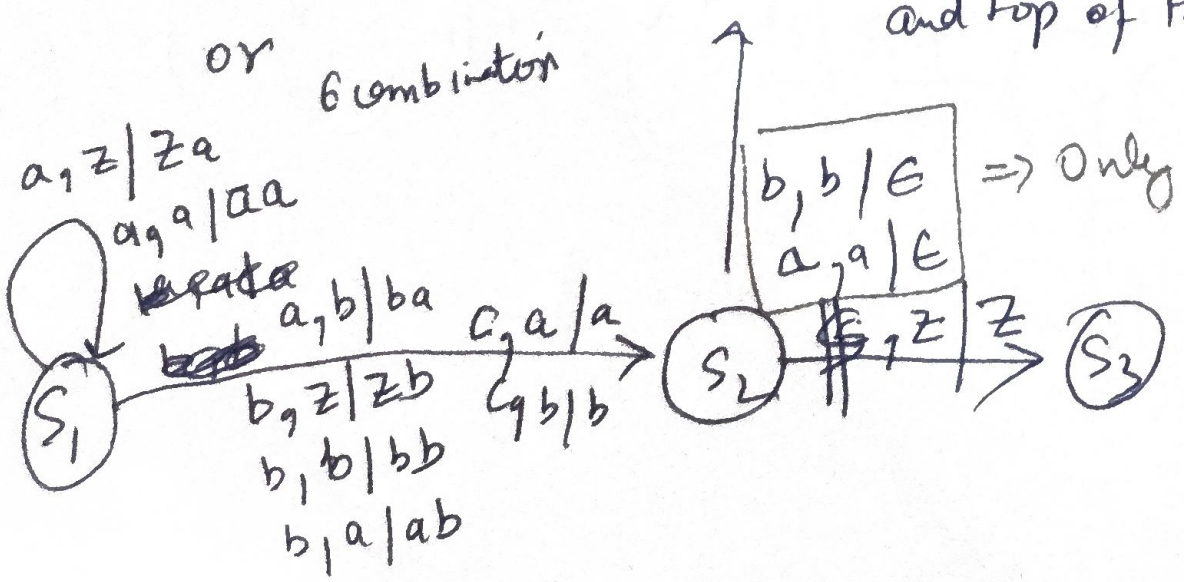


Easy push all symbol before c,
① pop start.



or combination

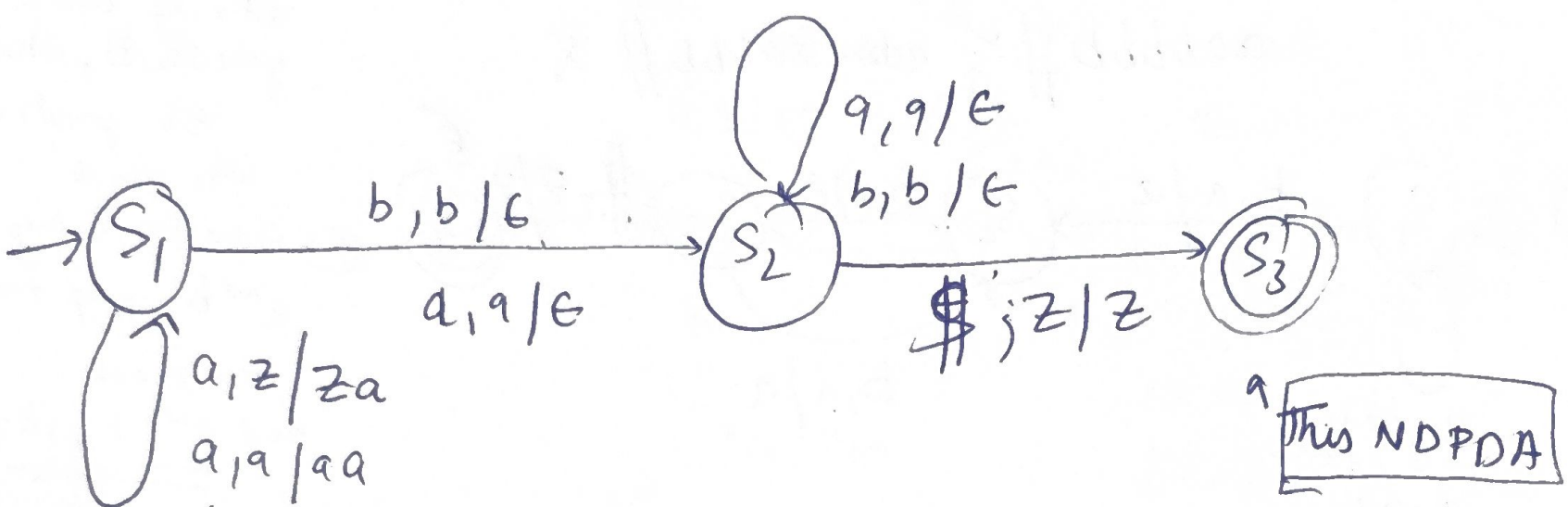
indication w^R, first symbol and top of the stack, w last symbol



=> Only 2 combination.

Draw DPDA $L = \{ WW^R \mid W \in (a+b)^* \}$

$L = \{ \epsilon, ab, aabb, ba, bbaa \}$ Two consecutive



① DPDA is not possible for this.

② N-DPDA is possible.

$On\ (S_1, b, b) \longrightarrow (S_2, \epsilon)$

$(S_2, b, b) \longrightarrow (S_1, b)$ ← N-DPDA

① ab ba → two consecutive symbol are same
W W^R

② abb bba → *Guess, if a or b ∈ W, push
W W^R if a or b ∈ W^R, pop.
but in this case, you have to guess.

Properties :→

For any language, N-DPDA possible, CFL is also.

① CFL doesnot mean DCFL.

② DCFL means CFL.

Power of NDPA and DPDA.

P(NDPA) > P(DPDA)

Lecture-24 :→

Draw Push Down Automata $L = \{ a^n b^{2n} / n \geq 1 \}$

↳ You can draw D-PDA

↳ You can also draw N-D-PDA

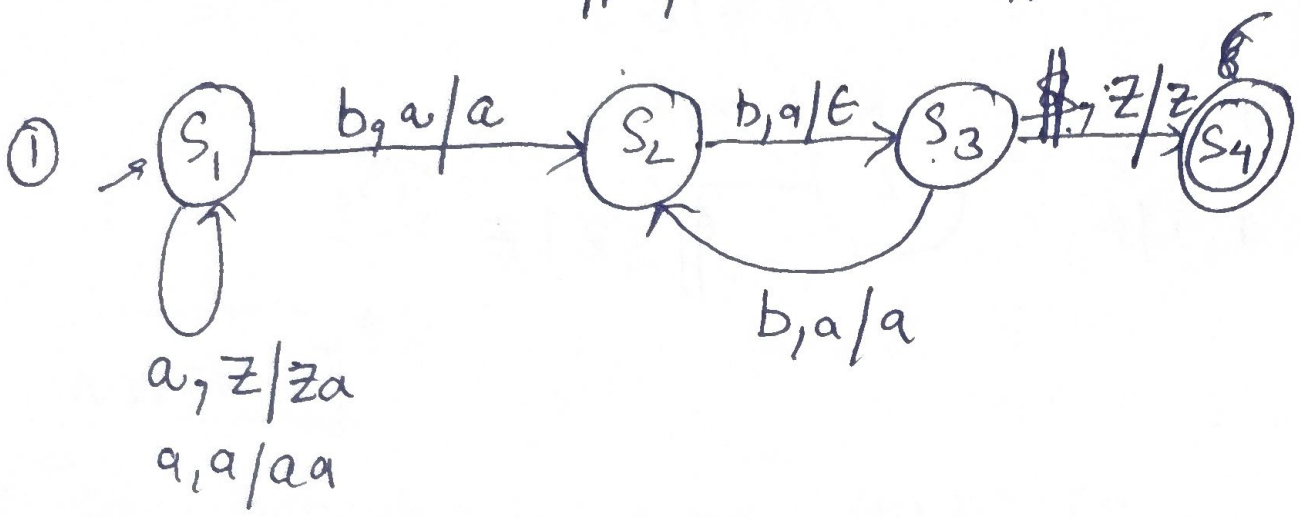
Hard

Observations :→

① For every a in string until you see b, should be push in the stack.

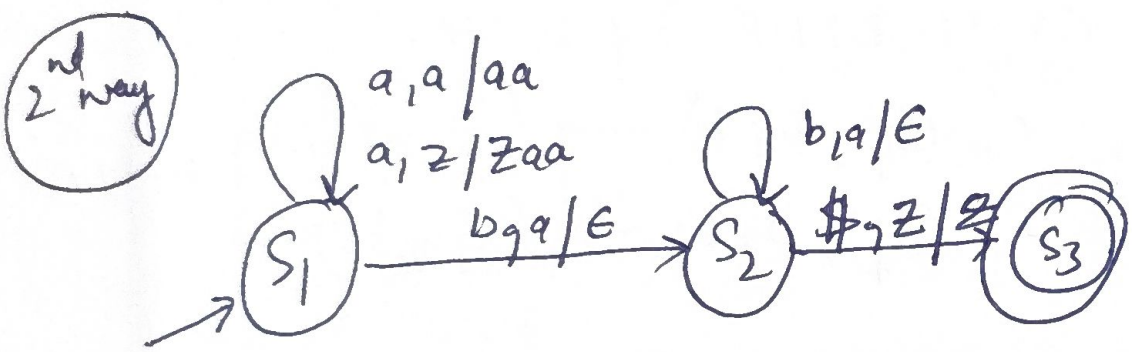
② Now for every 2nd b, pop from the stack. and 1st b, skip.

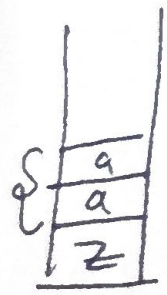
aabbbb$ ✓ aabbb$ X



push → storing

pop → Comparison





aaaa For each a, we will push 2 a's.

↑ Then, we will simply pop all a's for every b, because for every a in the stack we have b in the string.

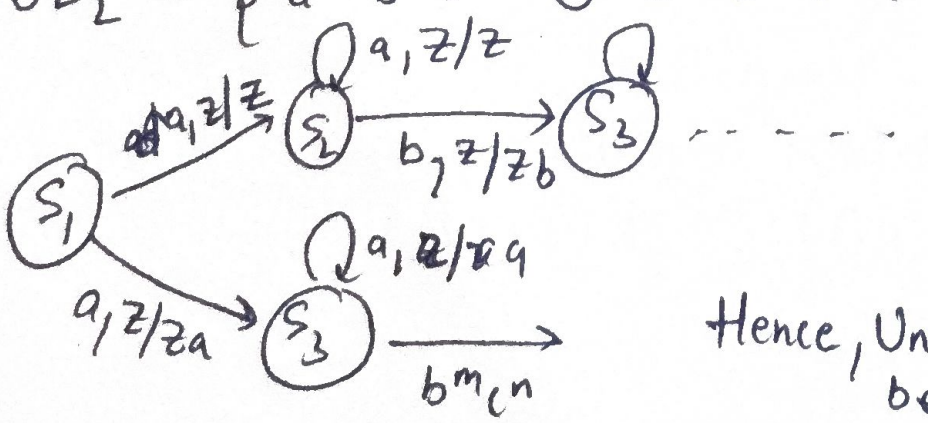
aaaabbbbbbbb

3rd way

$L_1 = \{ a^m b^n c^n / m, n \geq 1 \}$ → D-PDA → DCFL → CFL

$L_2 = \{ a^n b^m c^n / m, n \geq 1 \}$ → D-PDA → DCFL → CFL

$L_1 \cup L_2 = \{ a^m b^n c^n \cup a^n b^m c^n / m, n \geq 1 \}$



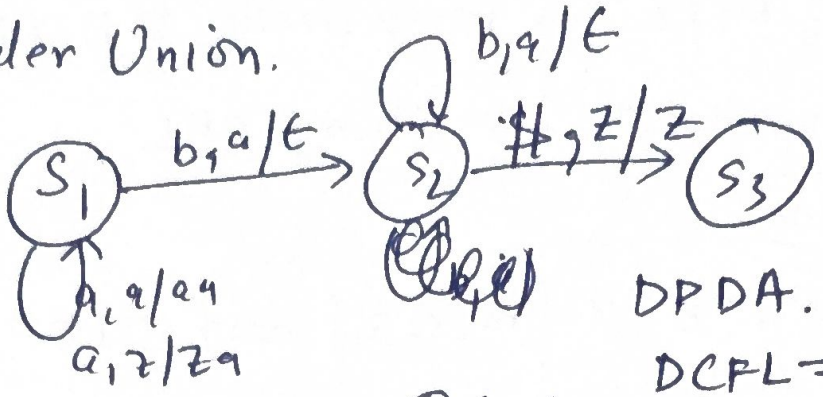
For (S_1, a, Z) → (S_2, Z), (S_3, a)

Two possibility

Hence, Union of two DCFL may not be closed under Union.

But CFL is closed under Union.

$L_1 = \{a^n b^n \mid n \geq 1\}$,



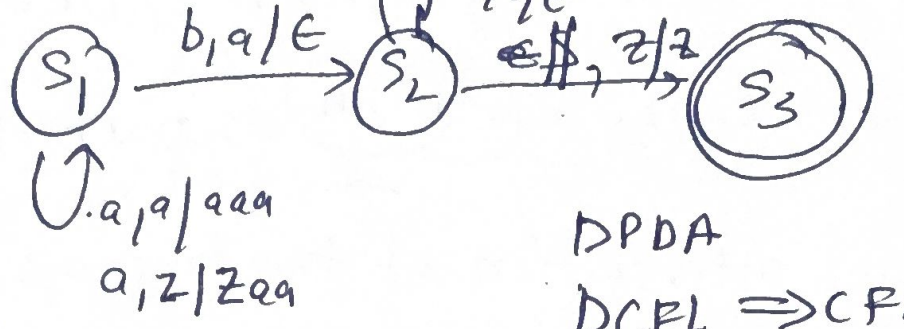
→ DCFL

CFLL

DPDA.

DCFL ⇒ CFL.

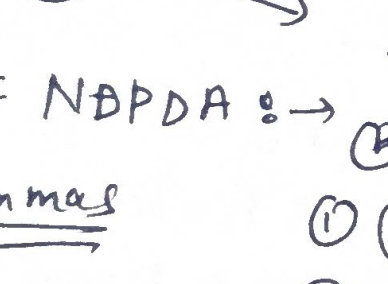
$L_2 = \{a^n b^{2n} \mid n \geq 1\}$,



DPDA

DCFL ⇒ CFL

$L = L_1 \cup L_2 \cdot a^n b^n \cup a^n b^{2n}$



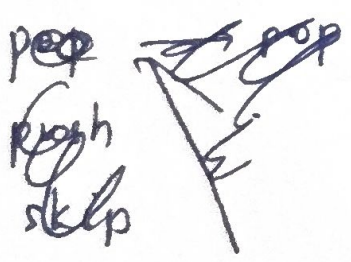
Hence, CFL of language is closed under Union.

~~NOT~~ NDPDA :→ Type

Dilemmas

① (push, dilemma) → either push two $ symbol or one ($*) for a symbol in string in case.

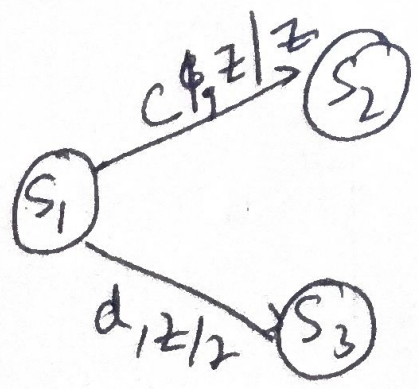
② ~~pop~~ (pop, push) dilemma.



$L_1 = \{c a^n b^n \mid n \geq 1\} \longrightarrow$ DCFL

$L_2 = \{d a^n b^{2n} \mid n \geq 1\} \longrightarrow$ DCFL

$L_0 = L_1 \cup L_2 = \{c a^n b^n \cup d a^n b^{2n} \cup d a^n b^{2n}$



Note → c and d ~~will~~ are will act as marker.

Hence, DCFL DCFL

$L_1 \cup L_2 \Rightarrow L \rightarrow$ CFL

① Union of DCFL → no gurrantte of DCFL but CFL.

② Union of CFL → always CFL

Intersection

① $L_1 = a^m b^n c^n \mid m,n \geq 1$ ⇒ DPDA ⇒ DCFL ⇒ CFL

② $L_2 = a^m b^m c^n \mid m,n \geq 1$ ⇒ DPDA ⇒ DCFL ⇒ CFL

$L_1 = L_1 \cap L_2 = a^n b^n c^n$ ⇒ D-PDA ✗, NPDA ✓ ⇒ CFL ✓

Hence, intersection of two language need not to be CFL,

① it may be CSL.

② $L_1 \cap L_2 = a^n b^n c^n$, Intersection not closed under DCFL

① **Complementation** ⇒ DCFL are closed under complement, interchanging i final to non-final and non final to final.

Like regular L^c ; L

Famous problem →

$L_1 : a^m b^{m+n} c^n \mid m,n \geq 1$

$L_2 : a^n b^{2n} c^n \mid n \geq 1$

Find who are CFL → DCFL → Write CFG or Draw DPDA.

→ Two comparison required. Hence, CSL

$S \to AB \Rightarrow a^m b^m b^n c^n \Rightarrow a^m b^{m+n} c^n$

$A \to aAb \mid ab$ $\quad a^m b^m$

Misc → 2

1:01:00 $B \to bBc \mid bc$

$L \to 23$

$L_1 \to a^m b^n c^n d^m \mid m,n \geq 1$ → ① push a's, push b, pop(b), with comparing c, then pop(a), d comparison

Hence, DPDA

CFL ✓

$L_2 = a^m b^n c^m d^n \quad m,n \geq 1$

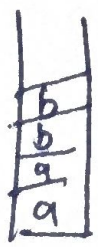
aabbccdd → ②

a will cancel c

① a is cancelled by c

② b is cancelled by d.

poped



a must be cancelled by a.

But it is bottom.

$L_3 : a^n b^n c^m d^m \longrightarrow$ DCFL.

Also, if CFL $\longrightarrow$ CSL

Consider the following PDA :→ (It is big paragraph)

$\delta(q_0, 1, Z_0) = (q_0, XZ_0)$

$\delta(q_0, \epsilon, Z_0) = (q_0, \epsilon)$

$\delta(q_0, 1, X) = (q_0, XX)$

$\delta(q_0, 0, X) = (q_1, X)$

$\delta(q_1, 1, X) = (q_1, \epsilon)$

$\delta(q_1, 0, Z_0) = (q_0, Z_0)$

q_0 is starting state.

* PDA accepted by empty stack.

Rewriting :→

$\delta(q_0, 1, Z_0) = (q_0, XZ_0)$

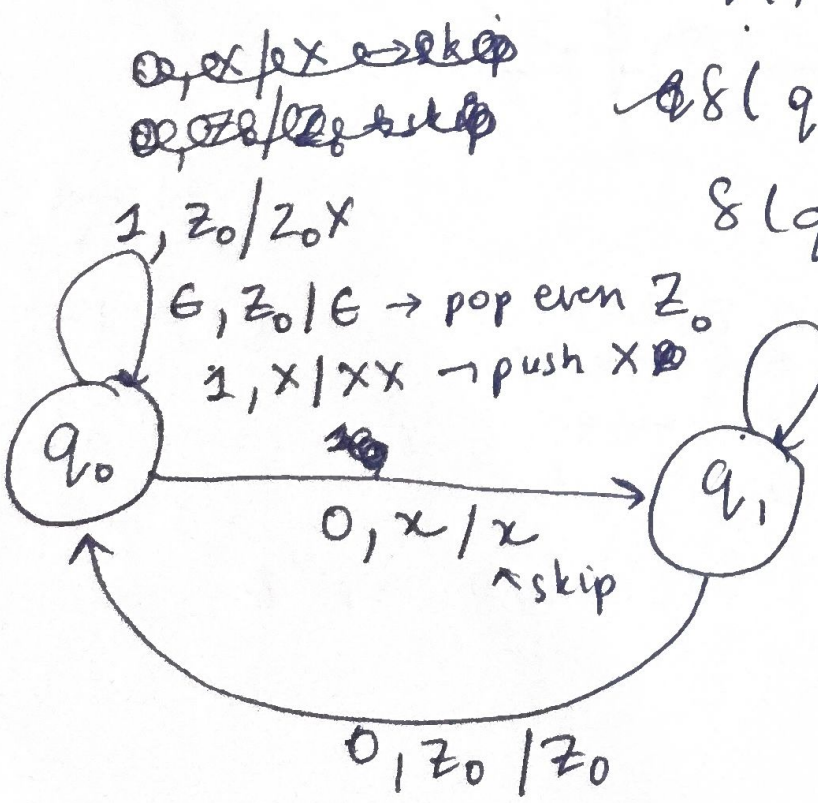
$\delta(q_0, \epsilon, Z_0) = (q_0, \epsilon)$

$\delta(q_0, 1, X) = (q_0, XX)$

$\delta(q_1, 1, X) = (q_1, \epsilon)$

$\delta(q_0, 0, X) = (q_1, X)$

$\delta(q_1, 0, Z_0) = (q_0, Z_0)$

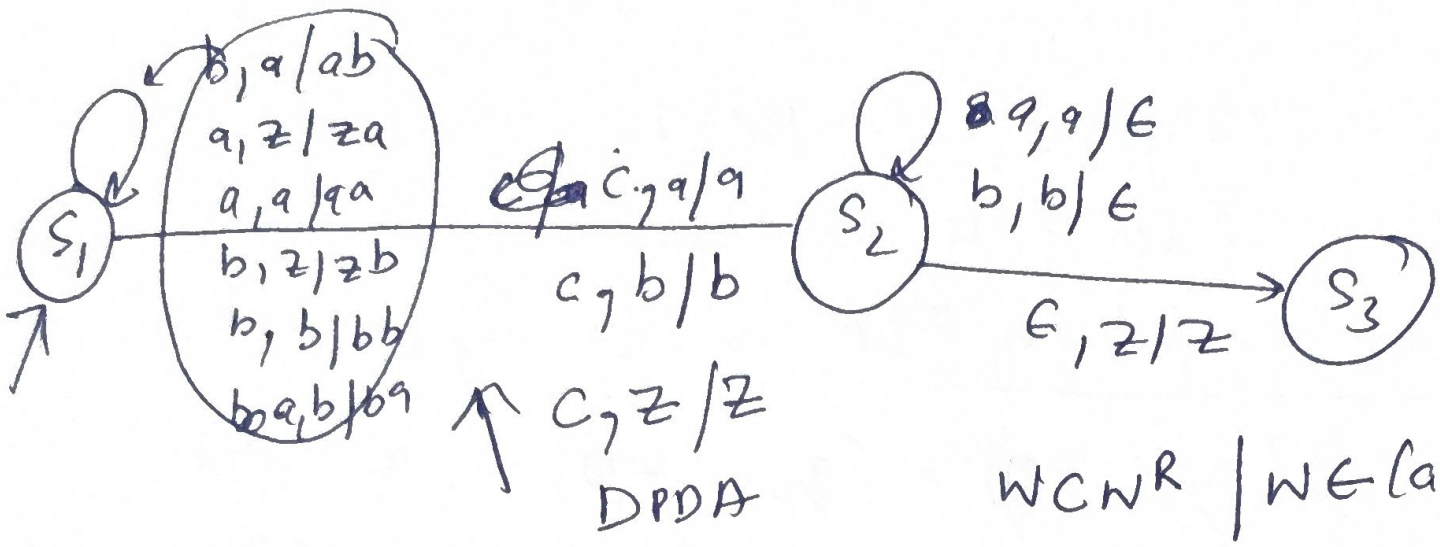


① $0^n 1 0^n 1 \mid n \geq 1$

✓ ② $1^n 0 1^n 0 \mid n \geq 0$

③ $1^n 0 1^n \mid n \geq 1$

④ $1^n 0 1^n 0 1^n \mid n \geq 1$



$WCW^R \mid W \in (a+b)^*$

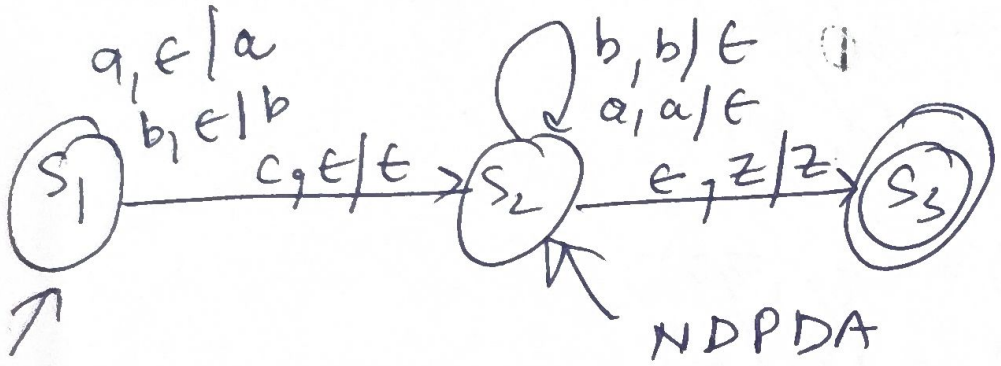
① DPDA

② odd length Pallindrom

→ DCFL → CFL

↓

CSL



$a, \epsilon / a \rightarrow \epsilon$, we donot care top of the stack (blind push to a)

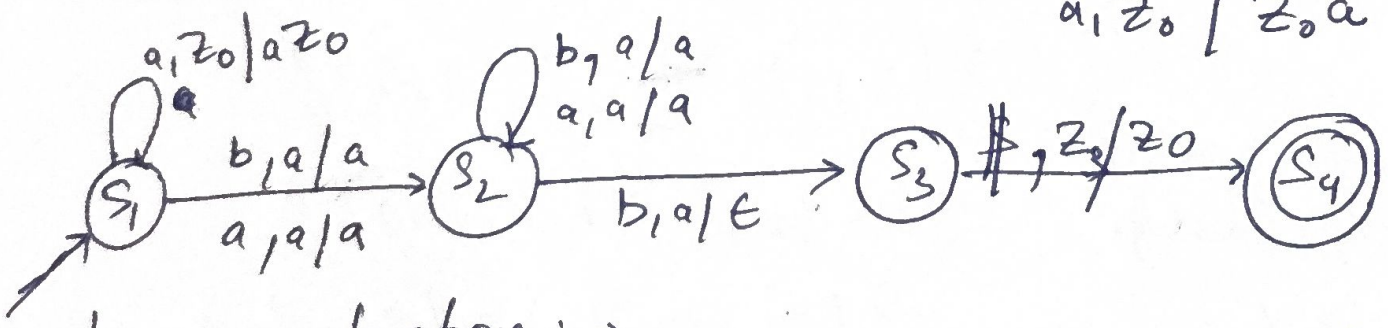
$a, \epsilon / a$ → single insted of ~~last~~ top of stack.

N-DPA :→

You don't care what is the top of the stack, push, pop or skip.

Consider the following PDA?

PTR: $a, Z_0 / aZ_0$] → both are same.
$a, Z_0 / Z_0 a$



Language of above :→

① Regular & finite

② " # infinite

③ CFL but not DCFL

④ Finite language.

① a a → a
② a a → b
③ a b → a
a b → b

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

Observe :→

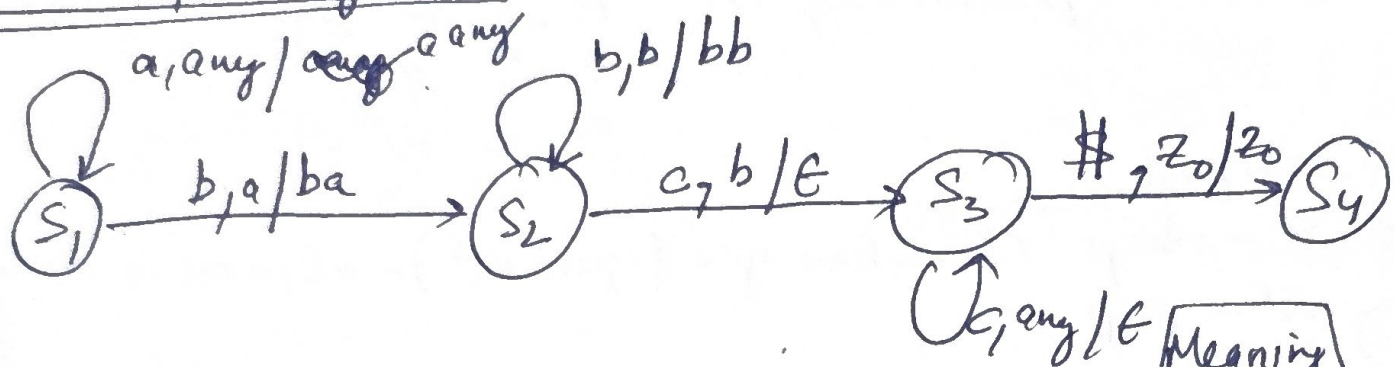
① No push after seeing first a.

② In the end, after seeing last b, pop.

Observation :→

① The PDA is N-DPDA.
② A Few PDA accepts regular.
③ Take DCFL.
④ Language is regular.
⑤ Without checking, PDA.

Consider the following PDA :→



Meaning

- a, any / any → skip.
- a, any / ∈ → pop
- a, any / a any or a, any / any a → push a on the top of the stack.

Observe :→

① a, any / any → ~~NDPDA~~

$a^+b\ b^*\ cccc$

$a^m b^n c^{m+n} / m, n \geq 1$

Question :→ Language of above PDA

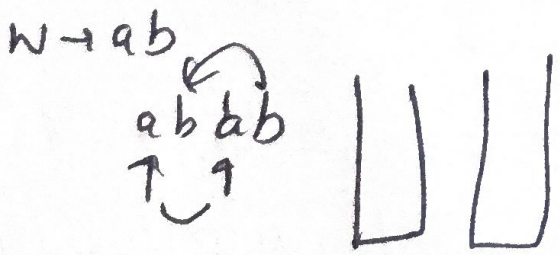
~~Regular~~ DCFL → CFL

Problem :→ $L_1 = \{ wcw^R / w \in (a+b)^*, c \notin \{a, b\} \}$ Famous problems :→

→ DCFL (pallindrome)
↳ CFL → CSL.

$L_2 = \{ ww^R / w \in (a+b)^* \}$ ⇒ Just CFL (NPDA) but not ~~CFL~~ DCFL.
↳ CSL

$L_3 = \{ ww / w \in (a+b)^* \}$ = First of W, LBA
CSL
RoE



Careful

$L_4 = \{ w x w^R \mid w, x \in (a+b)^* \}$

x is not a special symbol.
x may be a string $(a+b)^*$

Some string ab aayba or ab ∈ ba

① ab aayba — aay = X
ab bbb ba — ab = W, bbb = X, ba = W^R

② ∈ → possible? , Yes ∈.∈.∈. = ∈
(↑ W, ↑ x, ↑ W)

① w → push
② x → skip (Somehow you figure out) → Guess is required.
③ w^R → pop

Hence, confusing, but guess is possible,
So, DPDA ✗, NPDA ⇒ CFL

But behaviour → {∈, a (∈.a.∈ = W X WR), b, aa, ab, ba, ... ⇒ $(a+b)^*$ Regular
DCFL
CFL

$L_5 = \{ x w w^R \mid w, x \in (a+b)^* \}$

⇒ $(a+b)^*$ (Regular)

{ ∈ , a , b , ab∈∈ }
∈.∈.∈ (↑), a∈∈ (↑), b,∈,∈ (↑ $x w w^R$), $x w w^R$ → Regular.

$L_6 = \{ w w^R x \mid w, x \in (a+b)^* \} \Rightarrow (a+b)^*$

$L_7 = \{ w x w \mid w, x \in (a+b)^* \} \Rightarrow (a+b)^*$

(Miscellaneous-3 1:13:00)

[spoiled by x] [$(a+b)^*$ generate]

Careful

$L_8 = \{ wxw^R \mid w, x \in (a+b)^+ \}$

Actual format :- wxw^R as this time it cannot.

① $a \rightarrow x$, as ϵ is not allowed,

② Atleast three length is required.

aaa, bbb, aba, bab, abaaba, babbab

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

For four length :-

① $a(a+b)^2a$

② $b(a+b)^2a$ Starting ending should be same.

Critical

① a bba ba

w x w^R

Difficult :→

$L_9 = \{ wxw^Rx \mid w, x \in (a+b)^+ \}$

aaaa, bbba, aabb aba → invalid → CFL format is not destroyed

$L_{10} = \{ xwxw^R \mid w, x \in (a+b)^+ \}$ →

$L_{11} = \{ wxw \mid w, x \in (a+b)^+ \}$ → + operator

abb x abb → $\{\epsilon, a, b, ab, ba, \ldots\}$

can generate anything.

a b b	a

w

a b a, b a b, ab ab

w w

first with first

Comparison (Queue)

but in stack, it is not possible.

$L_{14} \Rightarrow \{xnww \mid wwnx\}$ → CSL

$w, x \in \{a, b\}^+$

pattern are not spoiled.

L_{15}

$L_{16} \Rightarrow \{wxnwy \mid w, x, y \in \{a+b\}^+\}$

a b b x a b b y

a b b x a b b y

w x w y

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

$L_{17} = \{xnwynw \mid w, x, y \in \{a+b\}^+\}$

same as above

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

$L_{18} = \{wxywn \cdot xwwy$

Simplification of Context Free Grammar

① Eliminate Null productions
② Eliminate Unit productions
③ " Useless productions

Order is imp.

①, ② then ③

[Donot change the order.]

Minimization Guranttee
① CFG
② Finite automata.
→ DFA → minDFA

Ex :→ $S \rightarrow AB$
$A \rightarrow a \mid b \mid \epsilon$
$B \rightarrow D \mid e \mid g$
$E \rightarrow f \mid h$
$D \rightarrow eD \mid eD$

Q How many productions?
Ans 1 1, $A \rightarrow a \mid b$, count as 2 productions

① Eliminate null productions

① Eliminating the null production

① S → AB
A → a | b | ∈
B → D | e | ~~f~~
E → f | h
D → d b | eD

~~Is Ref B~~

① Check production which produce.

~~S → AB | B~~

① S → AB
② A → a
③ A → b (We need to remove)
④ A → ∈ → ∈ → production
⑤ B → D | e | ~~f~~
(6) (7) (8)
E → f | h
(9) (10)
D → d | eD
(11) (12)

As, A → a, A → b. is there, you still have to consider A, write B seperately.

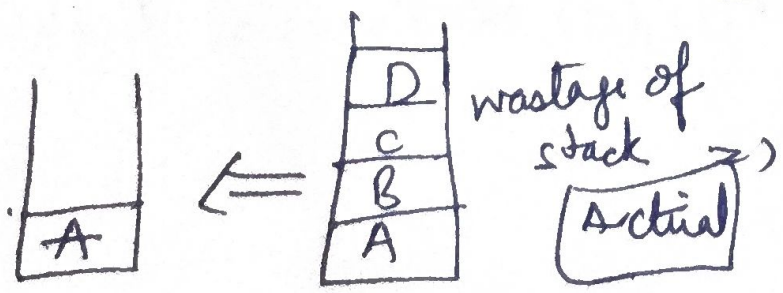
② S → AB | B → removed
A → a | b → removed ~~null~~ ∈ production

Unit production elimination :→

② S → AB | e | f | dD | eD → replace B with B content
A → a | b
B → e | f | dD | eD
E → f | b
D → dD | eD

```
A() {
    B()
    }
B() {
    C()
    }
C() {
    D()
    }
D() {
    s statement
    }
```

write here



(iii) Useless

(i) Eliminate those symbols which are not reachable from start symbol.

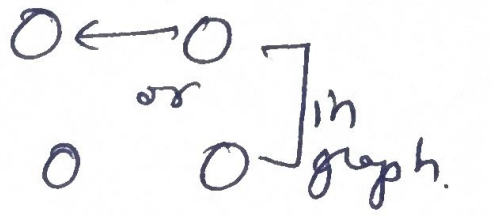
$S \rightarrow AB \mid e \mid g \mid ab \mid eD$

$A \rightarrow a \mid b$

$B \rightarrow e \mid g \mid aD \mid eD$

$E \rightarrow f \mid h$ ⟶ you cannot reach here (unreach)

$D \rightarrow aD \mid eD$



⇓

$S \rightarrow AB \mid e \mid g \mid ab \mid eD$

$A \rightarrow a \mid b$

$B \rightarrow e \mid g \mid aD \mid eD$

$D \rightarrow dD \mid eD$

(ii) Reachable but not useful. → Eliminate those productions which does not ~~stor~~ at least one string.

~~terminal~~

$S \rightarrow AB \mid e \mid g \mid \cancel{aD \mid eD}$ → cannot generate a string

$A \rightarrow a \mid b$

$B \rightarrow e \mid g \mid dD \mid eD$ → cannot generate a string

$\cancel{D \rightarrow dD \mid eD}$ ↔ useless (ii), as it contains a strings but recursions only. doesn't

$S \rightarrow AB \mid e \mid g$

$A \rightarrow a \mid b$

$B \rightarrow e \mid g$

The above steps (i), (ii) are known as Reduced Grammar.

Problem-2:

$S \to aAa$

$A \to Sb \mid bCc \mid DaA$ Convert to reduced grammar only.

$C \to abb \mid DD$

$E \to ac$

$D \to aDA$

$\to$ $S \to aAa$
$A \to Sb \mid bCc$
$C \to abb$
} Reduced Grammar.

Problem-3

$S \to Aa \mid B$

$B \to \not{A} \mid bb \mid a \mid bc$

$A \to a \mid bc \mid \not{B} \mid bb$

$S \to A \to B$ (A)

Replace Variable by its content..

$S \to Aa \mid bb \mid a \mid bc$

$B \to bb \mid a \mid bc$

$A \to a \mid bc \mid bb$

Single variable generate single variable, then directly replace the right side.

$A \to B$
$B \to ac$
$A \to ac$

Problem

$S \to AB$

$A \to a$

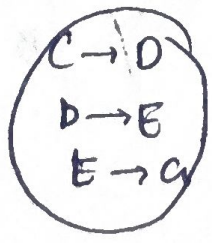
$B \to C \mid b$

$C \to D$

$D \to E$

$E \to a$

$\Rightarrow$ $S \to AB$
$A \to a$
$B \to b \mid a$
$C \to a$
$D \to a$
$E \to a$



$\Rightarrow$ $C \to D$
$D \to E \to a$
$C \to a$

Replace variable with its right side.

(P4)

$S \to aS \mid AB$

$A \to \epsilon$

$B \to \epsilon$

$D \to b$

Null production elimination only

⟶ ① $S \to aS \mid AB$ '

~~Se~~

② $A \to \epsilon$

③ $B \to \epsilon$

④ $D \to b$

Null productions (2), (3)
wherever A is visible
replace with ϵ

$A \Leftrightarrow AB \to \epsilon.B$

① $S \to aS \mid AB \mid B \mid A \mid \epsilon$ $AB \to A.\epsilon$

$A \to \epsilon$ $AB \to \epsilon.\epsilon$

$B \to \epsilon$ Then, ϵ, must also

$D \to b$ be removed

i/p : string → your program that you feed.

Grammar → Compiler

Parser → create a tree using Grammar.

① $S \to aS \mid AB \mid B \mid A$

$D \to b$ → unit came

(P5) $S \to a \mid Xb \mid aya \mid b \mid aa$

$X \to Y \mid \epsilon$

$Y \to b \mid \epsilon$

A or B, may generate other terminal.

ϵ → production is there,

but ϵ is not their string is not there

Y replace with ϵ, and add to the production

wherever

rewrite

$X \to Y \mid \epsilon \mid \epsilon$

$S \to Xb$

$X \to y \mid \epsilon$

$y \to b \mid \epsilon$

$yb \Rightarrow b$

$\epsilon.b = b$ → no null string

$S \to a \mid xb \mid aya \mid b \mid aa$

$x \to y$

$xy \to b$

Normal Forms :→

① Chomsky Normal Form (CNP)
② Greybach Normal Form (GNF)

① CNF :→ Every context free Grammar, every production, right side has single terminal or exactly two variables..

$V \rightarrow V_1 V_2 \mid T$ → single terminal

Eg →

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

② GNF :→ Every CFG, every production in the right side has has the form $V \rightarrow TV^*$

P1 $S \rightarrow \overset{①}{bA} \mid \overset{②}{aB}$
$A \rightarrow \overset{③}{bAA} \mid \overset{④}{aS} \mid \overset{⑤}{a}$
$B \rightarrow \overset{⑥}{aBB} \mid \overset{⑦}{bS} \mid \overset{⑧}{b}$

Convert into CNF

$V \rightarrow V_1 V_2 \mid T \rightarrow$ single terminal

① Productions → 8,
⑤, ⑧ only in CNF.

$S \rightarrow DA \mid EB$
$D \rightarrow b \quad E \rightarrow a$
$A \rightarrow a \mid ES \mid DF$
$F \rightarrow AA$
$B \rightarrow b \mid DS \mid EG$
$G \rightarrow BB$

P3

$S \to AB$

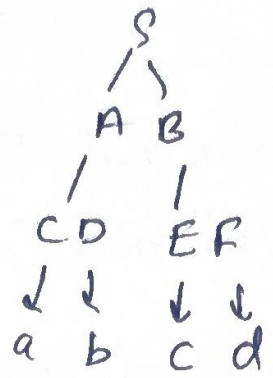
$A \to CD$

$B \to EF$

$C \to a \quad D \to b$

$E \to c \quad F \to d$

abcd ?



abcd

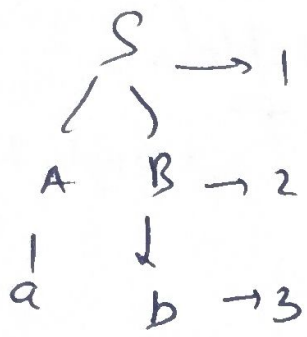
Seven productions are used for four length.

$[2^n - 1]$ productions for n length strij.

$S \to AB$

$A \to a$

$B \to b$



two generate ~~four~~ do 2 length strij, $2^2 - 1 \Rightarrow$ 3 productis

In CNF :→

Hence for n-strij length :→ $2^n - 1$ productions needed

50 strij $\Rightarrow 2 \times 50 - 1 \Rightarrow$ 99 productions will check.

for n-strij $\Rightarrow 2 * n - 1$ production will be Checked.

Ex

$S \to bA$ ① | aB ②

$A \to bAA$ ③ | aS ④ | a ⑤

$B \to aBB$ ⑥ | bS ⑦ | b ⑧

CAA

↓

E

Identify

① only 5, 8 are in CNF.

$S \to CA \mid DB$

$C \to b \quad , D \to a$

$A \to DS \mid a \mid CE$

$E \to AA$

$B \to DF \mid b \mid CS$

$F \to BB$

$S \to aAD$ ①

$A \to aB$ ② $| \; bBB$ ③

$B \to b$ ④

$D \to b$ ⑤

$\longrightarrow$

Productions → ④, ⑤ in CNF

$S \to CE$

$C \to a \quad E \to AD$

$A \to CB \mid BF$

$B \to$ ~~b~~ $F \to BB$, $B \to b$, $D \to b$

two ways → ⓐ Ⓐ Ⓓ or ⓐ Ⓐ Ⓓ

Parser have upper limit which makes.

Greibach Normal Form :→

In CNF

① First Simplification must be done.

② CNF formulae must be applied.

P ① $S \to aSb \mid ab$

① Simplification CFG

② Convert to GNF (terminal followed by Variable) only

Aa not allowed.

$S \to aB \mid aSB$

$B \to b$

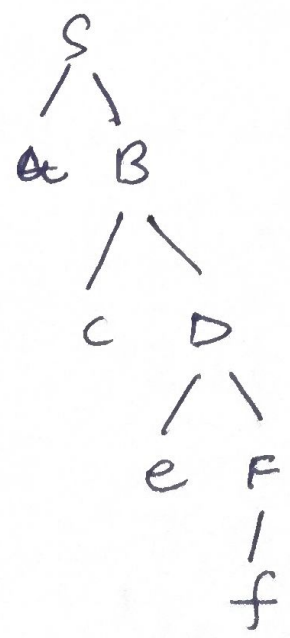
P ④ $S \to aB$

$B \to cD$

$D \to eF$

$F \to f$

acef.



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

① Simplification

①

$S \to a \mid b \mid aSA \mid bSB$

$A \to a$, $B \to b$

$S \to abcd$

or

$S \to aT$ $\qquad$ $S \to aaBC'D$

$T \to bU$ $\qquad$ $B \to b$

$U \to cV$ $\qquad$ $C \to c$

$V \to d$ $\qquad$ $D \to d$

$S \to \underset{\text{①}}{S}a \mid \underset{\text{②}}{b}$

① Simplification ✓

② GNF → first place variable there ?

If recursion is there, GNF will become complex.

most left recursion

The above problem is known as Left Recursion Problem.

$S \to Sa \to ba, baa, baaa, b\ldots a$

$\Rightarrow ba^*$ $\approx$ $S \to b \mid bS'$

$S' \to a \mid aS'$

$S \to b \mid bS'$

② don't

$S' \to a \mid aS'$

```
A ()
{ ① A() → stack
  ② ~ only
  ③ ~ call
  ①~
}  this will
   never end
```

Problem $S \to Sa \mid b \mid c \mid d \mid e$

GNF

⇓ Left recursion →

ba^*, ca^*, da^*, ea^*

$S \to bX \mid bcX \mid dX \mid eX \mid b \mid c \mid d \mid e$

$X \to aX \mid a$

then write with right side

$\to aX \mid a$ → with a terminal

Problem $S \to Sa \mid Sb \mid Sc \mid d \mid e$

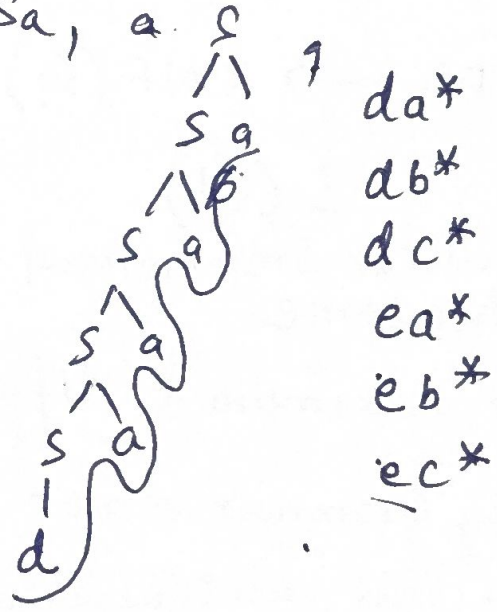
⇓ GNF

① Simplification ✓

② GNF

How many ways to end a string

① Sa, a



da^*
db^*
dc^*
ea^*
eb^*
ec^*

$S \to dS' \mid eS' \mid d \mid e$

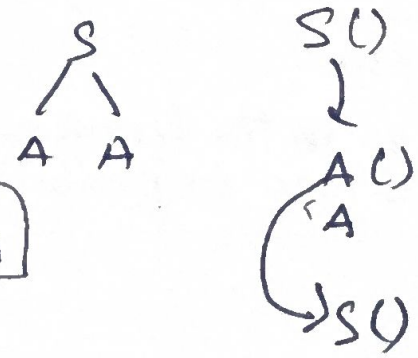
$S' \to a \mid aS' \mid b \mid bS' \mid c \mid cS'$

(Misc-V) L25

Problem

Hardest problem

$S \to AA \mid a$
$A \to SS \mid b$ } Indirect left Recursion.



$S \to SSA \mid bA \mid a$
$A \to SS \mid b$

⇒ Left Recursion

$S \to bA \mid a \mid bAS' \mid aS'$

$S \to SA \mid SAS'$

$A \to SS \mid b$

$S \to bA \mid a \mid bAS' \mid aS'$

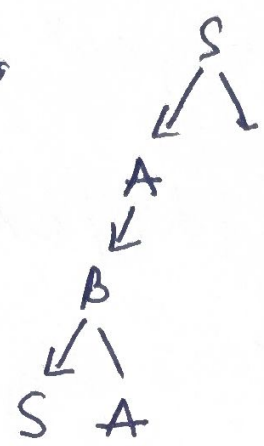
$\Rightarrow$ $S' \to bAA \mid aA \mid \bigcirc^{4} AS'$

17 productions $A \to \bigcirc^{4'} S \mid b$

Problem:→

$S \to AB$
$A \to BS \mid b$
$B \to SA \mid a$

Left Recursion?
Yes, indirectly



Left recursion.

① Simplification → ~~Redundant~~

② GNF:→

$S \to BSB \mid bB$
$A \to BS \mid b$
$B \to SA \mid a$

$\Rightarrow$

$S \to SASB \mid aSB \mid bB$
$A \to BS \mid b$
$B \to SA \mid a$

Please write steps ?→

① ~~CFG~~ CFG $\longrightarrow$ CNF(G) = G'

L(G) L(G')

Relation between Grammar and L(G')
① either same
② Or maximum ∈ difference.

For every Grammar donot have ∈-production, an equivalent CNF, GNF grammar.

If ∈ is in the Grammar, CNF, GNF will have ∈-production is missing.

$S \to aSBS' \mid bBS' \mid aSB \mid bB$ → replace.
$S' \to ASB \mid ASBS'$ → left recursion
$A \to BS \mid b$ → 3 productions, replace
$B \to SA \mid a$ → 2 productions

$\Rightarrow$ ~~∈~~ ∈-production

Ⓛ Miscellaneous - V

GNF production

Total productions → 27

Decidable Problems of PDA

① Emptiness Problem
② Finiteness Problem
③ Equality of two PDA :→ Checking two PDA is undecidable (No algorithm exist)
④ Membership Problem → Does the given Grammar generates a given string. (CYK Algorithm)
⑤ Two DPDA → decidable
⑥ But two NPDA are equal are is undecidable

① Emptiness problem :→ A Grammar is given,
① First use simplification on the Grammar production
② No production left

$S \rightarrow aS \mid bS$

Q

② Finitness Problem :→ Production generate infinite language.

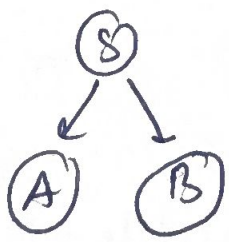
Steps

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

G

⟹

① Simplification of the Grammar.
② Convert into CNF
③ Draw Variable Dependency Graph :→



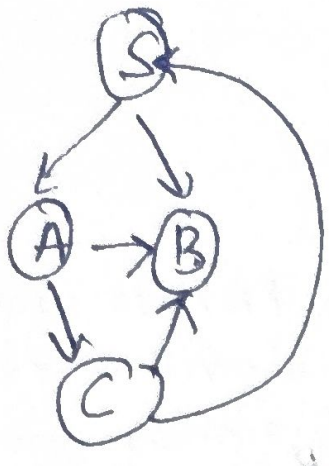
① No cycle is there.

So, after this (3), you get a cycle is then it is not finite.

$S \rightarrow AB \mid b$
$A \rightarrow BC \mid a$
$C \rightarrow BS \mid d$

① Simplification
② CNF
③ Variable Dependency Graph :→

(3)



~~Loop~~ Loop is there

Infinite ↝

Membership-Problem in ~~CFG~~ PDA :→ CYQ Algorithm

① Using CYQ algorithm we can ~~store~~ solve this

② CYQ Algorithm uses Dynamic Programming.

③ Time Complexity $O(n^3)$, Space Complexity $O(n^2)$

$S \rightarrow AB$

$A \rightarrow a/c$

$B \rightarrow b/d$

Single "a" → cannot be generated ✗

ab ↝ member.

* Check manually., No need to use CYQ algorithm.

$\theta \rightarrow$ zero

$S \rightarrow \theta S / S\theta / \theta\theta$

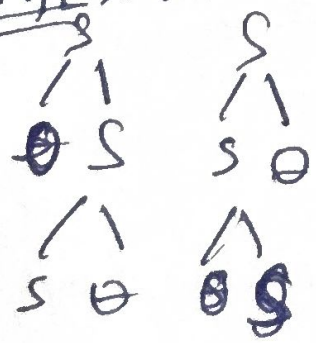
two zeros → ✓

two trees possible. →

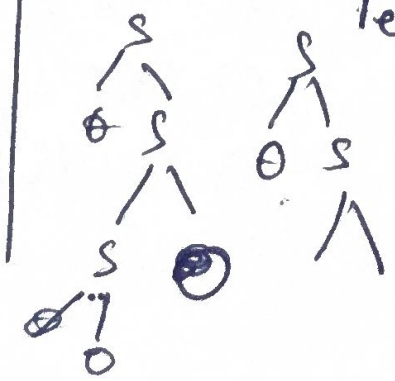
For first :→

2 2 2

0 0 0 =>



Only one variable → left and right must both are same.



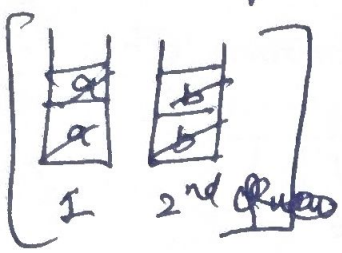
Turing Machine :→

L⇒ MISV → 5 (1:30:11)

① Finial Automata + two stacks. → Turing Machine

② PDA + one stack. → Turing Machine.

$a^n b^n c^n$ → two comparison. aabb cc



Turing Machine $M = \{Q, \Sigma, \delta, S, F, T, B\}$

Q, Σ, δ, S, F → FoA. ; F → set of final states ; T → "tau" ; B → "tao"

T → Tape Alphabet.

Two stacks are combined into one →

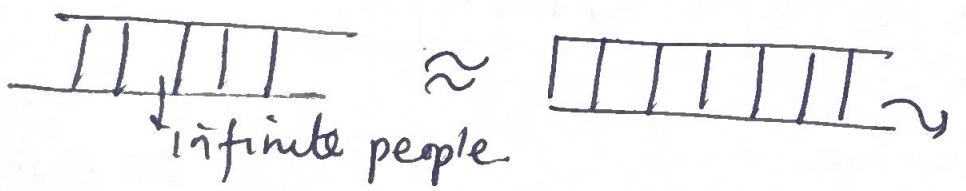
B → tape initially contain Blank symbol. B.

$B \in T$

Linear Bounded Automata, is a turing machine with some boundaries.

① There are ~~finite~~ infinite symbols but tape length is finite

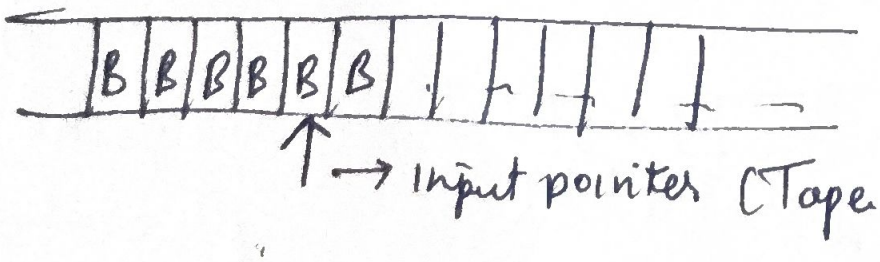
Turing Machine → Tape is open ended

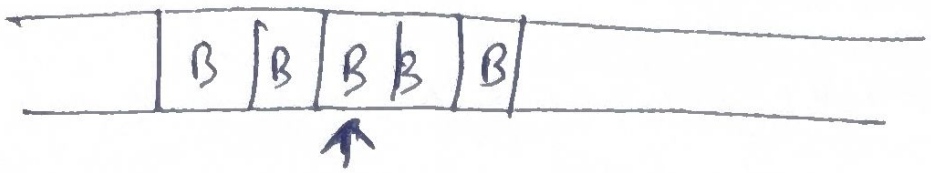


① Input Alphabet $\in T$ or $\Sigma \in T$

② Output Alphabet $\in T$

③ Inside tape → Input, output and blank symbol are the elements of T.





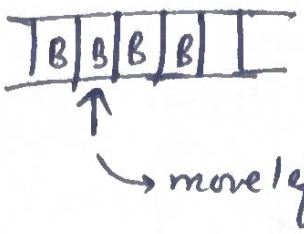
B/x → blank is replaced by x and then you have three choices

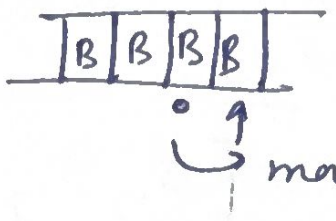
① move to left
② move to right
③ stay as it is.

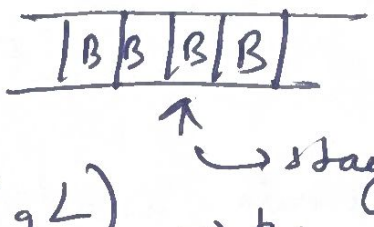
① S_1 —B/x/move left→ S_2

② S_1 —B/x/move right→ S_2

③ S_1 —B/x/stay→ S_2





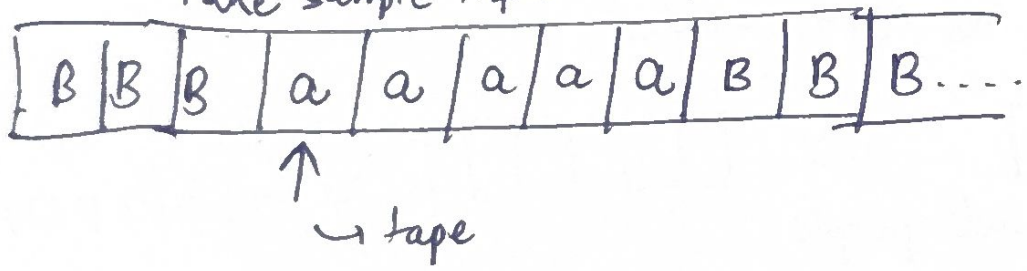


$\delta(S_1, B) = (S_2, X, L)$ → b.

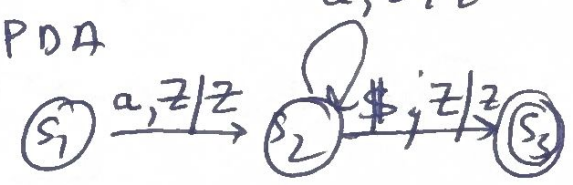
X: write; L: move left

Construct TM $L = \{a^n \mid n \geq 1\}$

① Take sample input :- a5



① Construct PDA



① Take sample input BB B input symbols B B B

② Tape pointer must point to first character

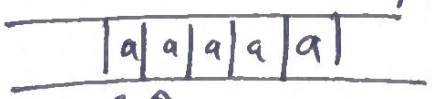
B a
↑
① first, as left side is Blank

② end of the string, after reading last a, we have B.

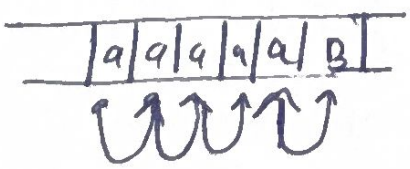
B a ~~~ a B
↑ ↑
start end

① S_1 —a/a, R (overwrite, move right)→ S_2

[Remember, you have to complete the input.]



② S_1 —a/a, R→ S_2; S_2 —a/a, R→ S_2



③ S_1 —a/a, R→ S_2; S_2 —a/a, R→ S_2; S_2 —B/B, R→ S_3

S_3 → Also known as halt final meaning string is accepted.

S_1 → Halt state X

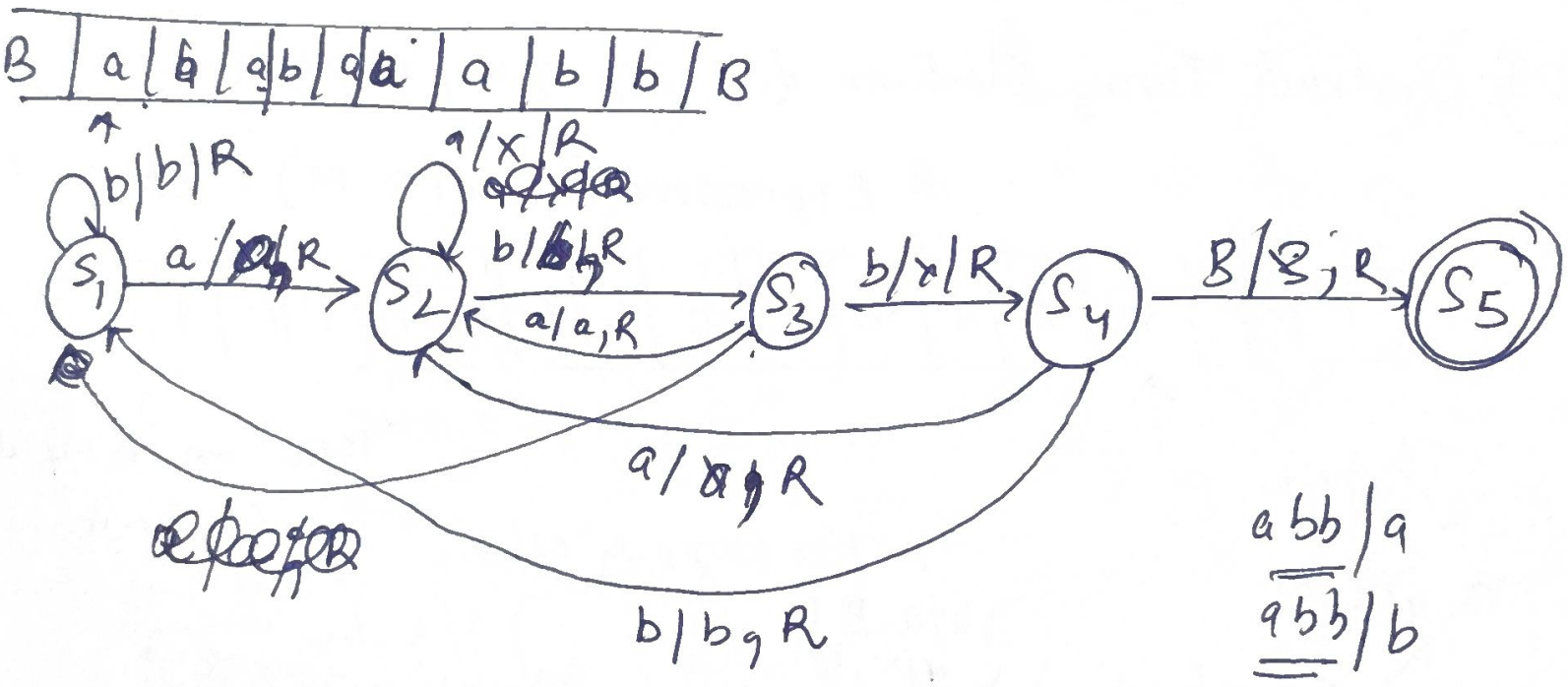
S_2 → Halt state X

S_3 → stop and string is accepted → known as HALT FINAL

Non-Determinstic Turing Machine ≈ Deteriministic TM
same power

(P1) Construct DM $L = \{(a+b)^* abb\}$

Input → ababaabb ↳ ending is abb.



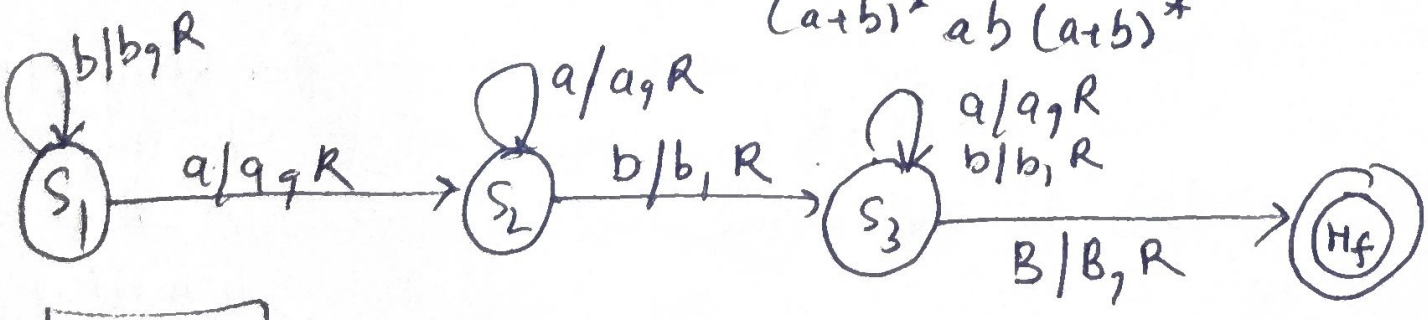
abb|a
abb|b

(P2) Construct DTM: L = {all string of a's and b's where each string contains ab as a sub string}

aba

B a b a b a B

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



Read till the end.

(3) ϵ is input : ϵ is replaced with B.

B	B	B	B

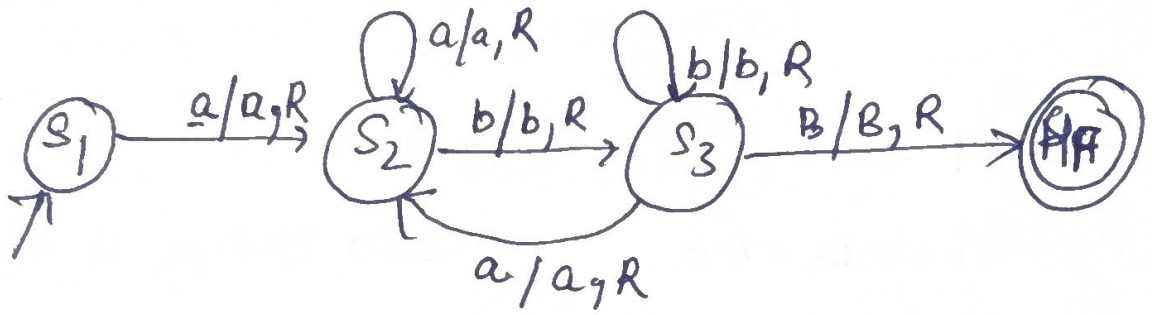
Any empty.

Blank . ϵ = Blank,

ϵ . Blank = Blank

(P3) Construct Turing Machine $L = \{ a(a+b)^* b \}$.

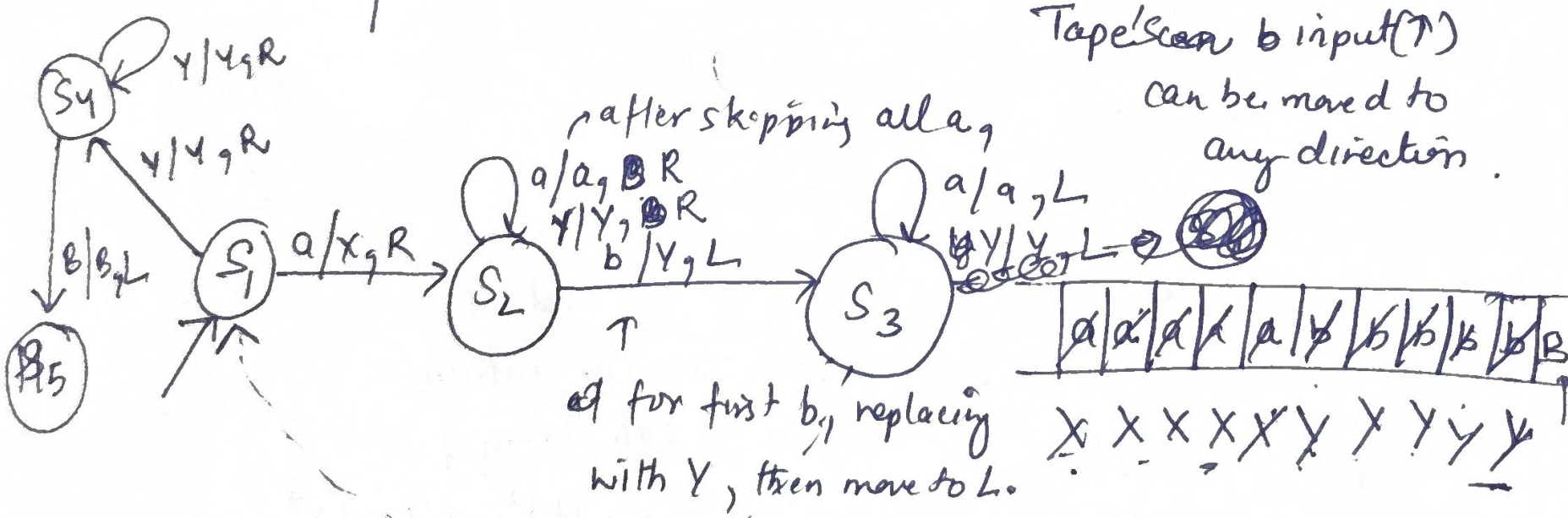
B	B	B	a	—	b	B



(P4) Construct Turing Machine $L = \{ a^n b^n \mid n \geq 1 \}$

* Expressive Power (DTM) = EP(NTM)

B	B	B	B	a	a	a	a	a	b	b	b	b	b	B	B	B	B



after skipping all a,

Tape scan b input(↑) can be moved to any direction.

↑ for first b, replacing with Y, then move to L.

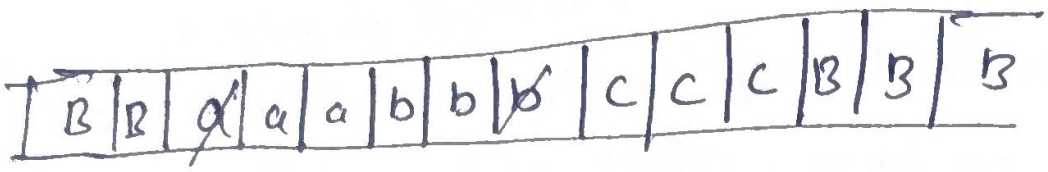
a	a	a	a	a	b	b	b	b	b	B
X	X	X	X	X	Y	X	Y	Y	Y	

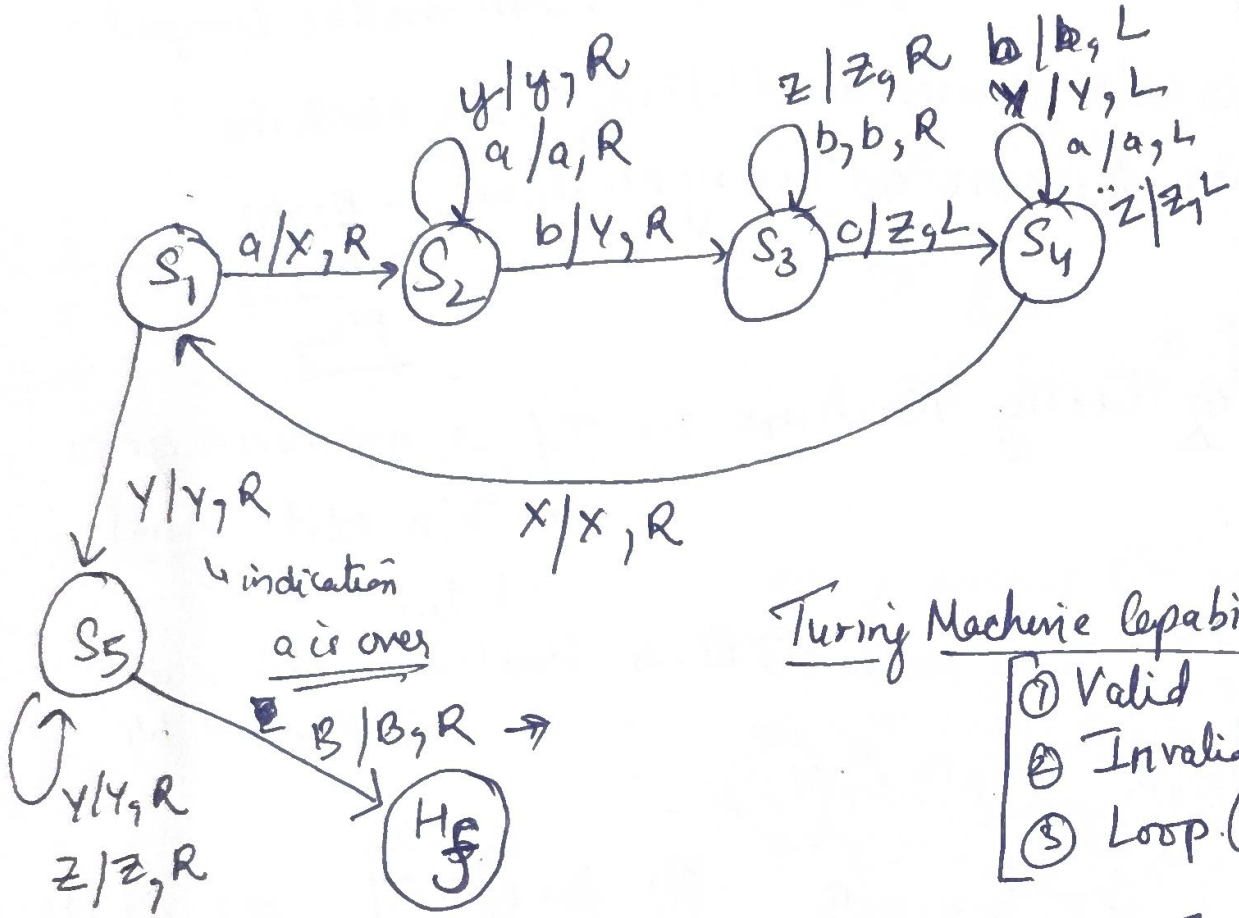
for every a, for every b, replace a with X. As it will act as a marker for number

For 2nd Bb, be careful.

①

Q Construct TM = { $a^n b^n c^n$ / $n \geq 1$ }



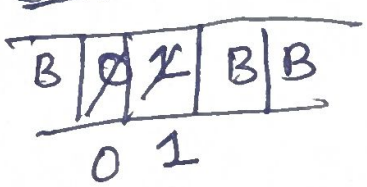


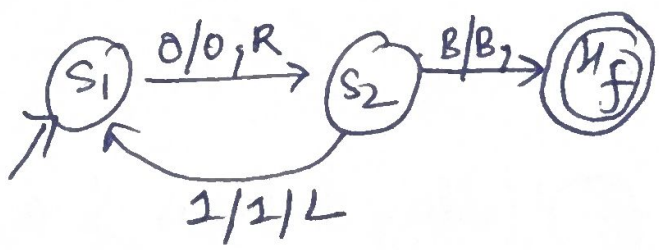
a is replaced by X
b is replaced by Y
C is replaced by Z
now → from C you will se Z; b, a ~~a~~ ~~first~~ second to get to the first X. Then only get a

Turing Machine Capability :→ Undecidable
① Valid
② Invalid
③ Loop (Infinite loop)

Turing Machine capability :→
① Turnaround Capability
②

Imp. :→





Drawback :→ Turnaround capability
① Turing machine will go to infinite loop. → Program never end.

Undecidable for T.M :→
① Emptiness problem
② Membership problem → Invalid, Valid, Infinite loop.

The turnaround property → can move to left or right any point of time.
The feature is not available in PDA, FA.

Halting Turing Machine :→ A TM which has only two states Yes and No. Hence, * decidable for membership.

Algorithm is given → [input] → Theortical on paper.

① Algorithm should stop running after finite time.

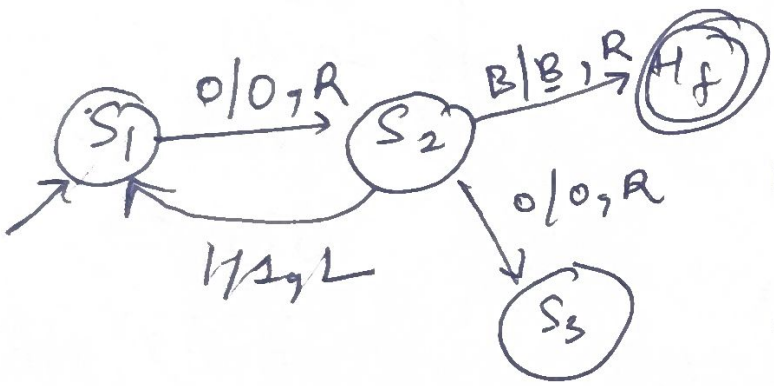
② Program can hang, which is run on the computer.

Algorithm is also known as Halting Turing Machine.

Program is known as Turing Machine ← Right, Wrong, Hang

(*) Halting Problem of a Turing Machine :→ If a machine goes to Halt final state.

In TM → Halting Problem is undecidable.



① for B|0|1|B => infinite loop

② for B|0|B => Yes

③ for B|0|0|B => Rejected

(#) Halting Problem of a Halting Turing Machine → As machine (HTM) has only two choice.

(HTM) Yes / No only.

Hence Decidable.

Notes :→

① HTM language is Recursive language.

② Turing Machine language is Recursive Enumerable Language.

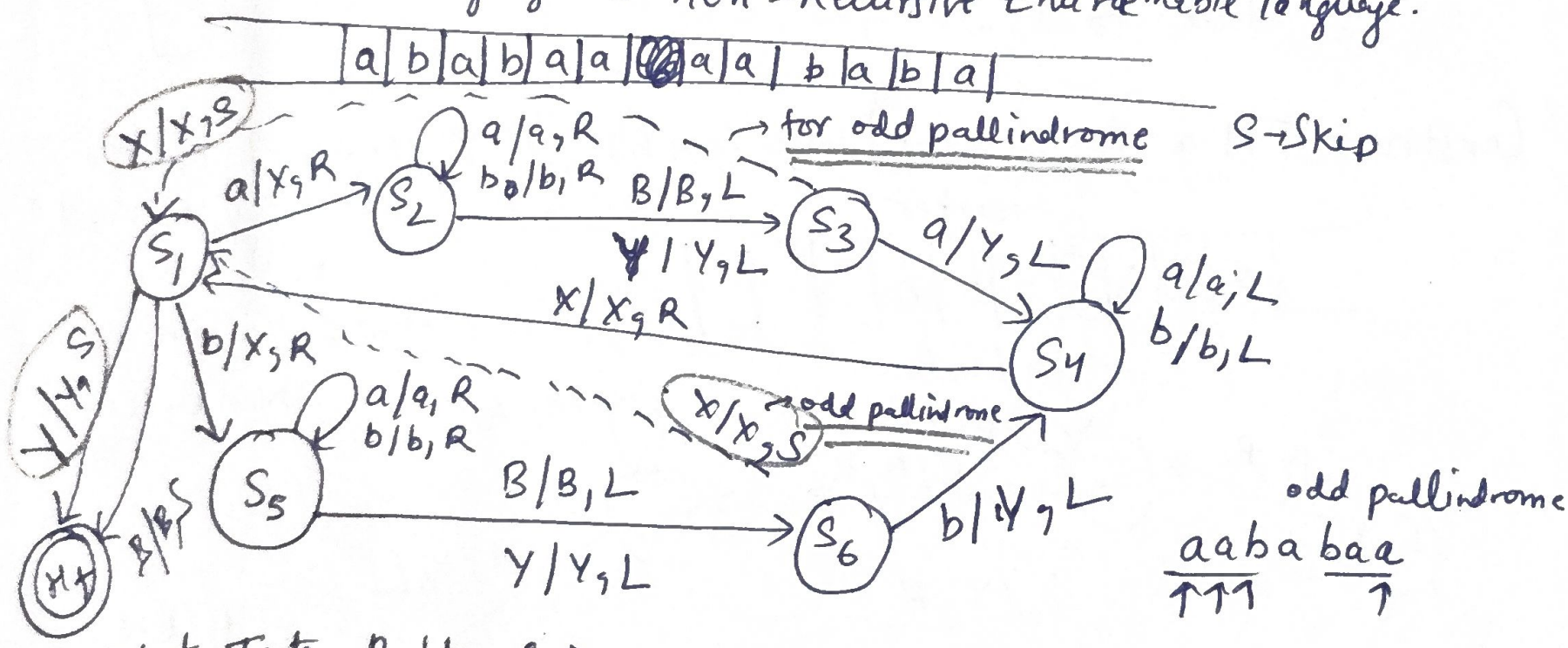
~~HTM is subset of TM~~

Recursive language is a subset of Recursive Enumerable Language.

Construct TM → $L = \{wcw^R \mid w \in (a+b)^*\}$

For any language if turing is possible then that language is Recursive Enumerable language.

Otherwise the language is non-Recursive Enumerable language.



State Entry Problem :→

Given a Turing Machine and a string, will it enter into S_3.

Input : Turing Machine, string

Ques. Will it enter into S_3 from S_1 (initial state). This is known as State Entry Problem.

Ans It is undecidable → Yes
→ No
→ Loop. (Stuck) → State entry problem.

Careful ⊗ → If a TM and a string is given, then from S_1 to given a string enter into state S_3.

[Moves are given → 100 moves, 1000 moves]

These one decidable.

one more way

[S_1 to S_3 state, in atmost 5 moves.
S_1 to S_3 state in 5 moves.]

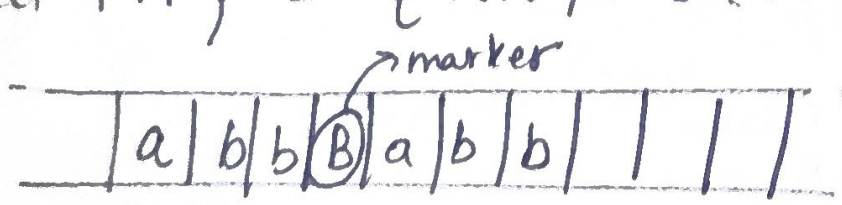
This is State Entry Problem, the above two is decidable problem.

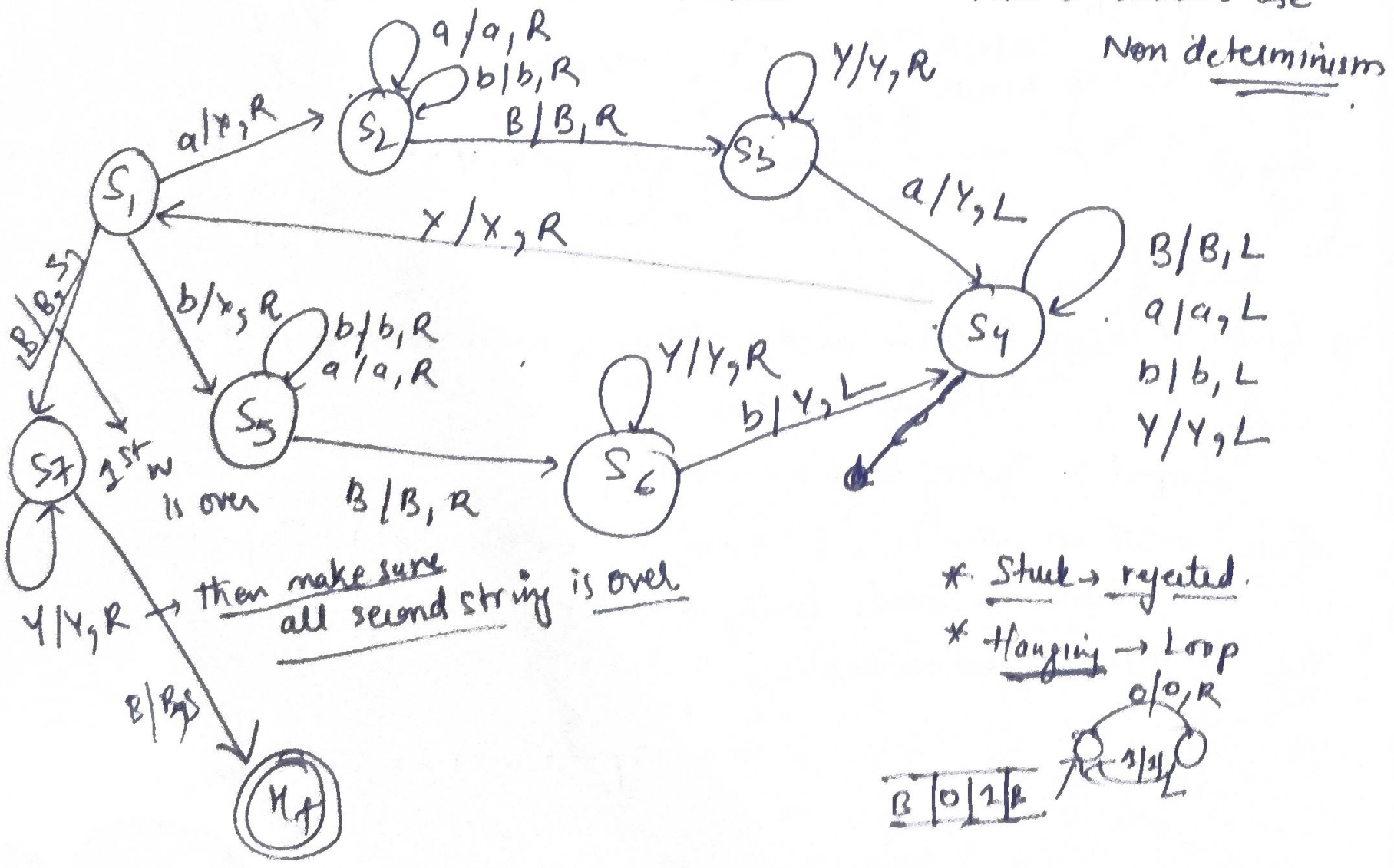
More variant :→
[Does a TM with a string, within 100 moves, from initial state to S_3.]

Construct TM; $L = \{ ww \mid w \in (a+b)^* \}$

Q How to know the start of second "w"?

Hint : You can use Non determinism





then make sure all second string is over

* Stuck → rejected.
* Hanging → Loop

"Transducers" :→ Input → [TM] → Output

2+3 → [TM] → 5 ⇒ Turing Machine

Computation :→

(B1) i/p : 5, 9 $\xrightarrow{5+9}$ o/p : 14

Construct TM to perform addition b/w 2 positive integers m and n. $(m, n \geq 1)$

Decimal to unary repre

5 → $(11111)_{unary}$

9 → $(11111\ 1111)_{unary}$.

"B" must be replaced by continious "ones".

* ① You can either move all ones in the right side of blank, o
* ② You can shift, B to the right most side, replacing current B with one. (This may not work)

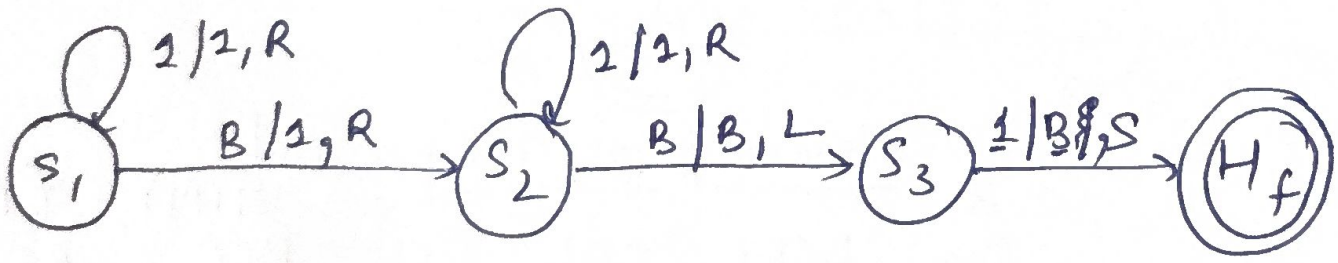
B	1	1	1	1	1	B	1	1	1	1	1	1	1	1	1	B	B

with 1

* ③ Replace the middle B, move it B to Last 1, replace to B.

1 1 1 B 1 1

⇒ 1 1 1 1 1 B ⇒ 5



(P2) Construct TM

i/p 5, 9 ⟹ o/p : 4 (9−5=4)

Hint: Do like a kid ① 1 1 X / X 1 ② 1 X 1 / X X

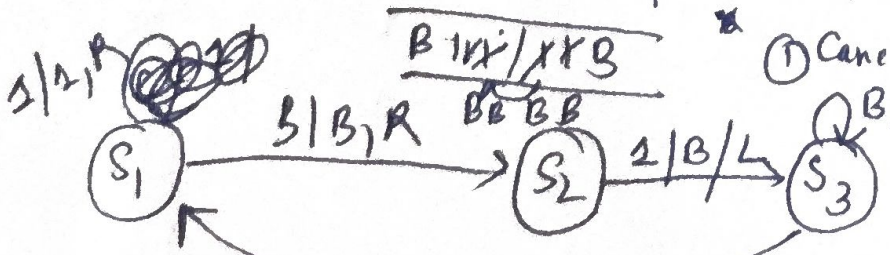
B	1	1	1	1	1	1	1	1	1	B	1	1	1	1	1	B

B 1 X / X 1 B

B B B B

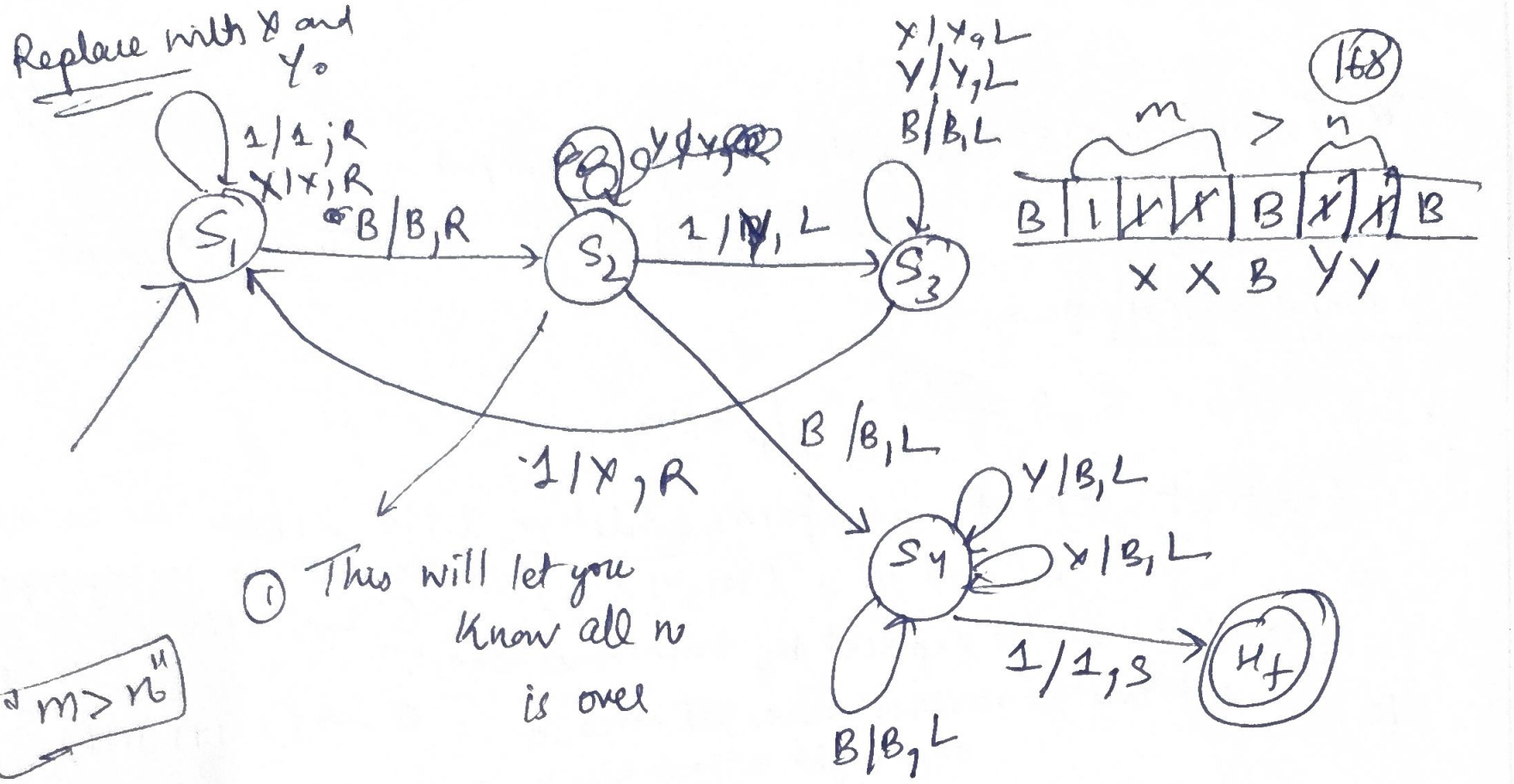
① Cancel the smaller one

Hint: 1 1 1 / 1 1 1 /



This will cause loop infinite loop.

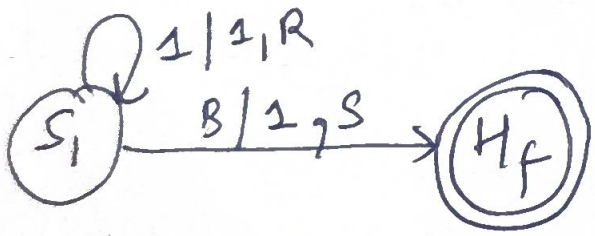
Hint



Construct Turing Machine for unary incrementation [i++]

Input: 5 $\xrightarrow{+1}$ o/p 6

B	B	1	1	1	1	1	B	



Closure properties [Closed]

		Reg	DCFL	CFL	CSL *	REC (HTM)	REL (TM)
1	Union	✓	X	✓	✓	✓	✓
2	Concatenation	✓	X	✓	✓	✓	✓
3	Kleene Closure	✓	X	✓	✓	✓	✓
4	Reversal	✓	X	✓	✓	✓	✓
5	Intersection	✓	X	X *	✓	✓	✓
6	Compliment	✓	✓	X *	✓	✓	X → important
*7	Inverse-Homo	✓	✓	✓	✓	✓	✓
8	Homomorphism	✓	X	✓	✓	X	✓
9	Substitution	✓	X	✓	✓	X	✓

REC is closed over complementation

REL is not closed over complementation

$(REL)^c$ ~complementation ⟶ may be REL or may be not REL (emotions)

$(REC)^c \Longrightarrow$ REC, REL as ~~REC~~ REC $\subseteq$ REL

↑ always!

$(REL)^c \approx$ always REL this language is "REC"

~~this~~ $(REL)^c$ is fake REL.

Similarly :→

Complement :→ $(DCFL)^c$ is always DCFL $\supsetneq$ CFL $\supsetneq$ CSL

~~Either~~ $(CFL)^c$ ~~is always~~ is DCFeL (A fake kind of CFL)

	Reg	DCFL	CFL	CSL	REC	REL
Subset						
Infinite-Union						
Infinite-Intersection						
Infinite-D						
Prefix	✓	✓	✓	✓	✓	✓
Quitient	✓	✗	✓	✓	✓	✓
Cycle	✓	✗	✓	✓	✓	✓
Min	✓	✗	✗			
Max	✓	✗	✗			
———	✓	✗	✗			

This are empty

Closure properties

	Reg	DCFL	CFL	CSL	REC	REL
L ∪ Regular	✓	✓	✓	✓	✓	✓
L ∩ Regular	✓	✓	✓	✓	✓	✓
L − Regular	✓	✓	✓	✓	✓	✓
R − L	✓	✓	✗	✓	✓	✗
L | R	✓	✓	✓	✓	✓	✓
Suffix	✓	✗	✓	✓	✓	✓
Difference	✓ (Reg-Reg)	✗ (DCFL-DCFL)	✗ (CFL-CFL)	✓ (CSL-CSL)	✓ (REC-REC)	✗ (REL-REL)
Substring	✓	✗	✓	✓	✓	✓

Most Imp

reverse

Both are same

Imp

① Regular Union Regular ⇒ Regular

② DCFL ∪ Regular ⇒ DCFL

③ CFL ∪ Regular ⇒ CFL

Which is bigger Union most of the time will give bigger language.

Example

$a^n b^n \cup \phi \Rightarrow a^n b^n$ [CFL] ✓ [DCFL]

$a^n b^n \cup (a+b)^* \Rightarrow (a+b)^*$

CFL Regular Regular ⇝ "a fake CFL"

CFL in intersection

$L - R \Rightarrow L \cap R^C$

Decidable / Undecidable

→ carefully

	Reg	DCFL	CFL	CSL	REC	REL
Membership	✓	✓	✓	✓	✓	✗
Emptiness	✓	✓	✓	✗	✗	✗
Finiteness	✓	✓	✓	✗	✗	✗
Equivalence	✓	✓	✗	✗	✗	✗
(✗) Regularity	✓	✓	✗	✗	✗	✗
Ambiguity	✓	✓	✗	✗	✗	✗
Disjoint	✓	✗	✗	✗	✗	✗
• Co-infiniteness	✓	✓	✗	✗	✗	✗
Completeness	✓	✓	✗	✗	✗	✗
Complementation	✓	✗	✗	✓	✓	✗
Subset Regular	✓	✗	✗	✗	✗	✗

Thm.

Derived * from above

$F_1 \subset F_2$
$F_1 - F_2 = \phi$
*

If given language is DCFL, can you say Regular.
Yes, because there is algorithm.

* The rest do not have any algorithm.

(✗) **Ambiguity** :→ I have a Regular Language, unambiguous or not.

Decidable

a^*

(✗) $S \to aS \mid Sa \mid a$ $\quad S \to Sa \mid a$

$a^* \to$ Regular
$S \to aS \mid a \to$ Regular Grammar (Right Linear)

For a language → Many Grammar possible.
from these, Atleast one is Regular grammar which is unambiguous.

Cofinitness :→ ① Complementation + ② Finitiness

For a lang → Do first Complementation then finitness.

DFA

DFA1 $\subseteq$ DFA2

DFA1 $\cap$ DFA2 =>

Miscellaneous-8 :→

Problems :→ ① Halting Problems

② Blank Tape Halting Problem → ∅ string is $\in$. TM halts? Undecidable.

③ State Entry Problem → Undecidable

④ Post Correspondence Problem → Undecidable

⑤ Modified Post Correspondence problem. → Undecidable

$N = \{1, 2, 3, 4 \text{ ---- } \}$

Countabily Finite → Countable and takes less time.

Countabilty Infinite :→ Countable but take infinite time : (Listing possible)

① $\underline{\underline{Z}} = \{ \underset{\text{CoI}}{Z^+}, \underset{0}{0}, \underset{\text{CoI}}{Z^-} \}$ => Z made up of + natural number, natural number with -ve sign, +zero

Still countabilty Infinite

② $\underline{\underline{Q}} = \{ \frac{p}{q}, \text{ rational numbers } q \neq 0 \}$ $\frac{p(\text{CoI})}{q(\text{CoI})}$ => Countabily Infinite

③ R = { Real numbers } => (1, 2, 3, 4 ; 1 to 2, 2 to 3, ...

(a+ib) → complex number from 1 to 2 → infinite number

[You may not be able listing.

Real numbers are uncountable

Uncountables :→

C = set of complex => {a+ib .... } → Uncountable

① Set of all regular languages → countably infinite.

② " " " CFL → " "

" " " DCFL " → " "

" " " CSL " → " "

" " " REC " → " "

" " " Recursive " → " "

" " " Enumerable → " "

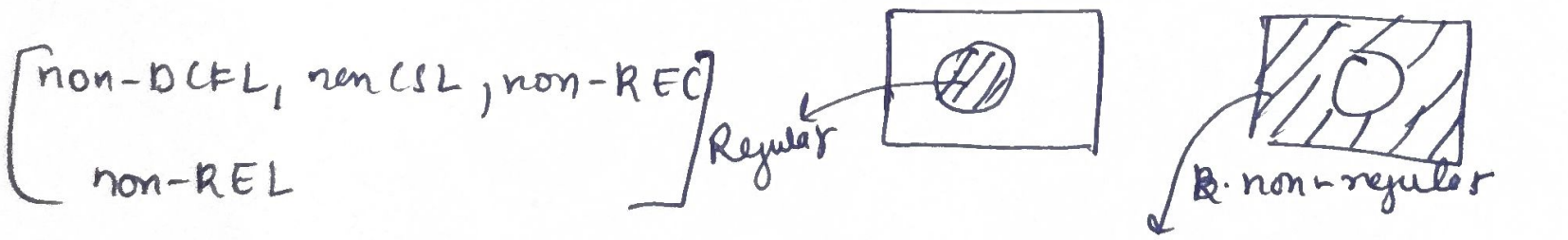
Important Theorems :→

① Σ^* => one language. → countable. i.e (C.I)

② 2^{Σ^*} => set of all language → uncountable

Set of all non-REL => uncountable (Human emotions)

Set of all non-Regular => uncountable



All are uncountable, reason.
Non-REL exist.

* Non-REL exist here

Theorems :→

If L is countably infinite then 2^L is uncountable.

If L is regular then then

Σ^* is regular & is countably infinite.

2^{Σ^*} is regular but uncountable.