

Defining x and y :

$x \rightarrow$ A set of all strings over all the alphabets in Σ such that total no. of "a" & "b" in the string is even as it equals to $2k$ where " k " is a natural no.

$y \rightarrow$ A set of all strings over all the alphabets in Σ such that the string can be split into equal halves "u" and "v"

Proof using 2 conclusions:

i) $x \subseteq y$: Assume w is an arbitrary string in x . By the definition above we know that the total no. of "a" & "b" in w is even.

- As the total is even $\rightarrow 2k$ thus we can divide w in two halves of equal length strings 'u' & 'v', such that
 $\text{length}(u) = \text{length}(v) = k$

- Thus w can be expressed as 'uv' where $u \in \Sigma$ and $v \in \Sigma$ and $\text{length}(u) = \text{length}(v)$
 $\Rightarrow w$ is in y

- As w is an arbitrary string in Σ ,
Thus every string in x present in y .

Thus proving $\underline{x \subseteq y}$

2. $y \subseteq x$: Assume ' w ' is an arbitrary string in y . By definition above ' w ' can be split into two strings 'u' & 'v' $\rightarrow \text{length}(u) = \text{length}(v)$
- Thus, we can conclude the total length of ~~w~~ ' w ' is $\text{length}(u) + \text{length}(v) = 2 \times \text{length}(u) \rightarrow$ even which is even no. (anything $\times 2$ is even)
- Thus, the total no. of 'a' and 'b' in the string ' w ' is also even $\rightarrow 2k$
Hence, proving ' w ' is in x .
- As ' w ' is an arbitrary string in y , this shows that every string in y is present in x .

Hence, $y \subseteq x$.

As we proved both $x \subseteq y$ and $y \subseteq x$ by the double inclusion method, we can conclude $x = y$.

Therefore language x & y are equal.

3] $\Sigma = \{a, b, c\}$ contain abc but not substring bbb

As we require abc but not substring bbb
but we can have 'bb' & also single b

a - 'a' is allowed single or multiple times

c - 'c' is allowed

ba } single time 'b' allowed
bc }

bab } double times 'b' allowed
bbc }

abc . requirement / allowed.

hence,

$(a|c|ba|bc|bb|abbc)^*$... to start with.

+ $a|c|ba|bc|bb|abbc$... allowed substring

again $(a|c|ba|bc|bb|abbc)^*$... to cover every possible
unit to make substring in string.

No matter how many times we use of $ba|bc|bb|abbc^n$
to form string having 'b', consecutively in
any sequence it will not form 'bbb' substring.

Final expression

$(a|c|ba|bc|bb|abbc)^* abc (a|c|ba|bc|bb|abbc)^*$

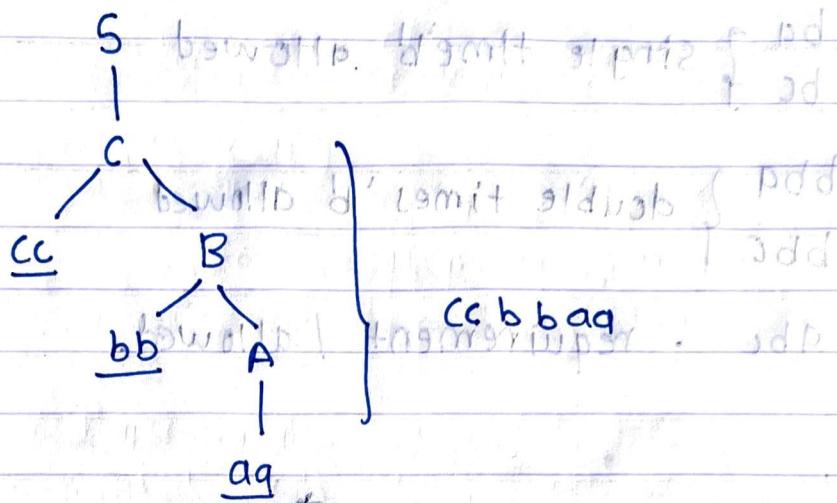
4] $\Sigma = \{a, b\}$, $G: S \rightarrow A \mid B \mid C$

$$A \rightarrow aa \mid B \mid C$$

$$B \rightarrow bb \mid A \mid C$$

$$C \rightarrow cc \mid A \mid B$$

a) Tree For $ccbbbaa \in L(G)$



b) In the set theoretic terms, the language of this grammar can be described as a set of string over alphabets $\{a, b, c\}$ such that each string is a concatenation of either two or more of 'aa', 'bb', 'cc' or the string is 'aa', 'bb' or 'cc' alone.

c) Regular expression

'aa', 'bb', 'cc' individually considered in pair with either of them in every possible combination.

$$((aa(bb|cc))|aa) \mid (bb(aa|cc)|bb) \mid (cc(aa|bb))|cc)) +$$

7]

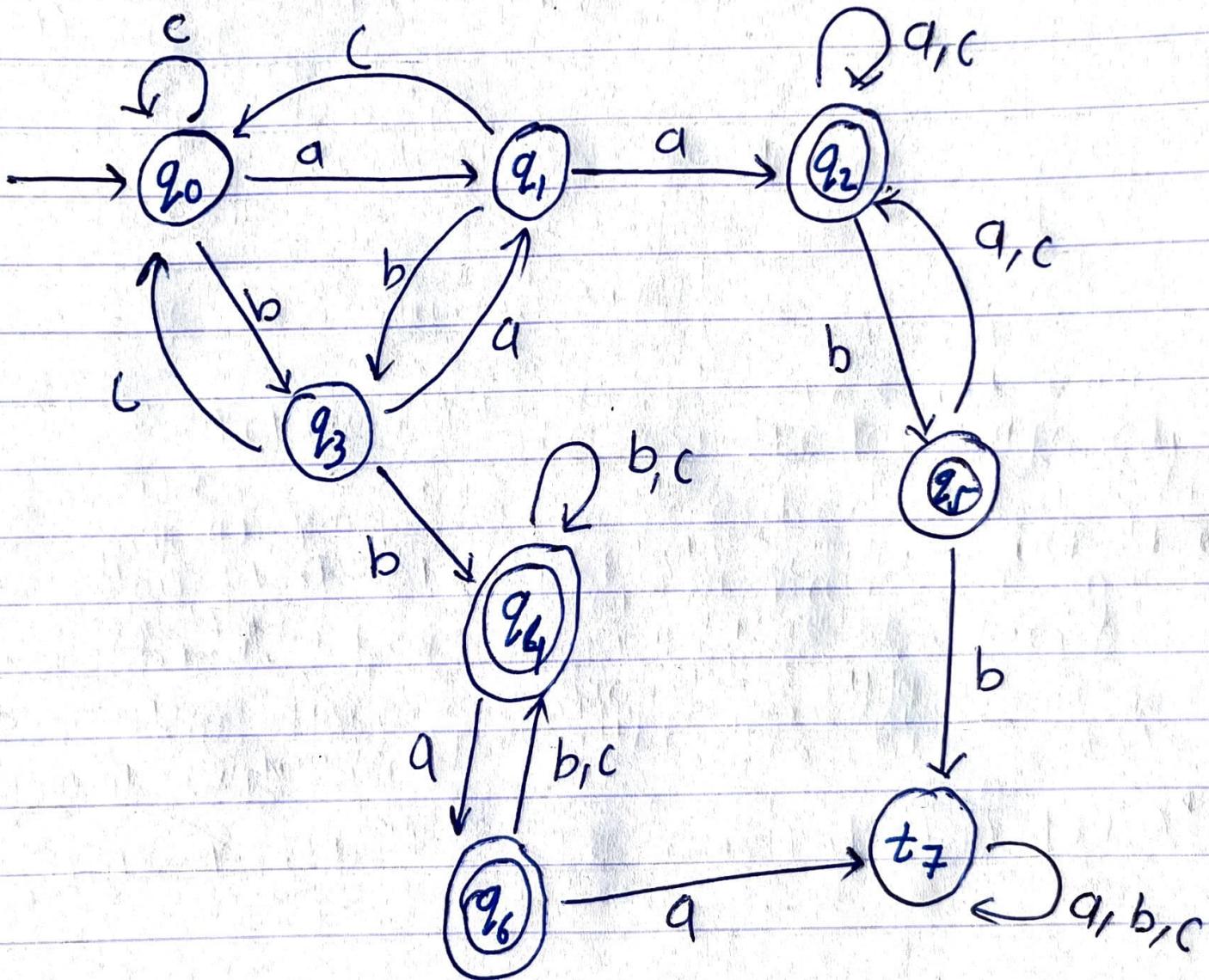
$$\mathcal{Q} \rightarrow \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, t_7\}$$

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

initial state $\rightarrow q_0$

final state $\rightarrow \{q_2, q_4, q_5, q_6\}$

trap state $\rightarrow t_7$



6] b $G: S \rightarrow aa | P$
 $P \rightarrow aa | aQaQ$
 $Q \rightarrow bb | bPbP$
 $R \rightarrow cc | bPQPRPQb | aPRQRaPaQaR | abc$

Stage 1:	$U_a \rightarrow a$	$A \rightarrow QP$
	$U_b \rightarrow b$	$B \rightarrow RP$
	$U_c \rightarrow c$	$C \rightarrow Q U_b$
	$X \rightarrow U_a Q$	$D \rightarrow U_a P$
	$Y \rightarrow U_b P$	$E \rightarrow RQ$
	$U_d \rightarrow U_b U_a$	$F \rightarrow RUa$
	$U_e \rightarrow U_a$	$G \rightarrow PX$
		$H \rightarrow U_a R$
		$I \rightarrow YA$
		$J \rightarrow BC$
		$K \rightarrow DE$
		$L \rightarrow FG$
		$M \rightarrow LH$

... replacing values

Final CNF is:

$G: S \rightarrow U_a U_a P$	$D \rightarrow U_a P$
$P \rightarrow U_a U_a XX$	$E \rightarrow RQ$
$Q \rightarrow U_b U_b YY$	$F \rightarrow RUa$
$R \rightarrow U_c U_c IS KM U_a U_d$	$G \rightarrow PX$
$U_a \rightarrow a$	$H \rightarrow U_a R$
$U_b \rightarrow b$	$I \rightarrow YA$
$U_c \rightarrow c$	$J \rightarrow BC$
$X \rightarrow U_a Q$	$K \rightarrow DE$
$Y \rightarrow U_b P$	$L \rightarrow FG$
$U_d \rightarrow U_b U_c$	$M \rightarrow LH$
$A \rightarrow QP$	
$B \rightarrow RP$	
$C \rightarrow Q U_b$	

$$6] a \quad A \rightarrow a | aA | Aa | BaA | AaB$$

$$B \rightarrow bbb | BBB$$

$$C \rightarrow bbb | BBB | ccc | CCC | cba | CBA$$

for 'A' we have escape character 'a'
we can write

$$A \rightarrow a | aA | BaA | aA' | aBA'$$

$$A' \rightarrow aA' | aBA' | \lambda$$

for 'B' we have escape character 'b'
we can write

$$B \rightarrow bbb | \lambda B'$$

$$B' \rightarrow \lambda B' | BBBB' | \lambda$$

for 'C' we have escape character 'c'
we can write.

$$C \rightarrow bbb | BBB | ccc | cba | \lambda C' | BAC'$$

$$C' \rightarrow \lambda C' | CCC' | BAC' | \lambda$$

Thus for $A \rightarrow BaA$ is allowed

for $B \Rightarrow C \rightarrow BBB$ is allowed hence

now our final rule with no left recursion
is

$A \rightarrow a aA BaA aA' aBA'$
$B \rightarrow bbb \lambda B'$
$C \rightarrow bbb BBB ccc cba \lambda C' BAC'$
$A' \rightarrow aA' aBA' \lambda$
$B' \rightarrow \lambda B' BBBB' \lambda$
$C' \rightarrow \lambda C' CCC' \lambda$

$$2] a) S_0 = \{a^3, b^3\}$$

(a) $a^3, b^3 \in GL$, basis step)

Thus,

S_1 , given $u, v \in \Sigma^*$ where $\Sigma = \{a, b\}$, we can assume the recursive step can occur on any substring of 'aa' or 'bb'

Thus:

- For 'aaa'
 - abaaaab and aaabaaa [Assuming $u=a$ and $v=\lambda$]
 - baaaba and aabbaaa [Assuming $u=\lambda$ & $v=a$]

→ For 'bbb'

- babbba and bbbabb [Assuming $u=b$ & $v=\lambda$]
- abbabb and bbabbabb [Assuming $u=\lambda$ & $v=b$]

$$\Rightarrow S_1 = \{abaaaab, aaabaaa, baaaba, aabbaaa, babbba, bbbabb, abbabb, bbabbabb\}$$

2] b) Base case:

Base case is S_0 , which is a^3 & b^3 both these strings can be written as $paaaq$ or $pbbbq$ respectively. where p and q are both empty strings. Thus base case is true.

Inductive step:

Assume for some $k \geq 0$, every string in S_k can be written as $paaq$ or $pbbbq$ for some $p, q \in \Sigma^*$.

Next we need to prove this holds for S_{k+1} .

Consider a string $w \in S_{k+1}$. By definition 'w' must be generated by applying the recursive step to string $w' \in S_k$, then we have 2 scenarios.

Case 1: $w = uav$

1. $w' = uaaav \in S_k$: $u = baa$, $v = dd$

Then

If $w' = uaaav$ where $u, v \in \Sigma^*$ we can see from part a that at any point where 'ba' is present, it is always joint with a substring.

Case 2: $w' = ubbv \in S_k$: $u = bbb$, $v = dd$

Thus, proving that it can be written as $paaq$.

2. $w' = ubbv \in S_k$: $u = bbb$, $v = dd$

If $w' = ubbv$ where $u, v \in \Sigma^*$ we can see from part a that at any point where 'bb' is present, it is always joint with a substring 'bbb'. It turns into 'babbb' and contains another substring 'bbb' itself.

Thus, proving that it can be written as $pbbbq$.

Therefore, if every string in S_k can be written as $Paaaq$ or $Pbbbq$ for some $P, q \in \Sigma^*$ then every string in S_{k+1} can also be written in the same form.

Thus, by the principles of induction, every string in language b can be written in the form $Paaaq$ or $Pbbbq$.