1] Defining x and y:

use standard notation.

x → 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 → A set of all strings over all the alphabets in Σ such that the string can be split into equal halves "u" and "u"

Proof using 2 conclusions:

i. x ⊆ 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 → 2k thus we can divide 'w' in two halves of equal length strings 'u' & 'v', such that

  length(u) = length(v) = k

- Thus 'w' can be expressed as 'uv' where u ∈ Σ and v ∈ Σ and length(u) = length(v)
  ⇒ 'w' is in y

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

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

2. $y \subseteq x$: Assume 'w' is an arbitrary string in y. By definition above 'w' can be split into two strings of equal length 'u' & 'v' → length(u) = length(v)

- Thus, we can conclude the total length of 'w' is length(u) + length(v) = 2 × length(u) which is even no. (anything × 2 is even)

- Thus we have total no. of 'a' and 'b' in the string 'w' is also even → 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, $\boxed{y \subseteq x}$

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.

Actually, pretty well done. It should be shorter and tighter, and you should NOT disguise your paragraphs to look like a list.

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
bc } single time 'b' allowed

bba
bbc } double times 'b' allowed

abc . requirement / allowed.

hence,

(a|c|ba|bc|bba|bbc)* ...to start with.
abc ... allowed substring

again
(a|c|ba|bc|bba|bbc) ...to cover every possible string.

No matter how many times we use ba|bc|bba|bbc
to form string having 'b', consecutively in
any sequence it will not form 'bbb' substring.

Final expression

(a|c|ba|bc|bba|bbc)* abc (a|c|ba|bc|bba|bbc)*

cannot match bbabccbb

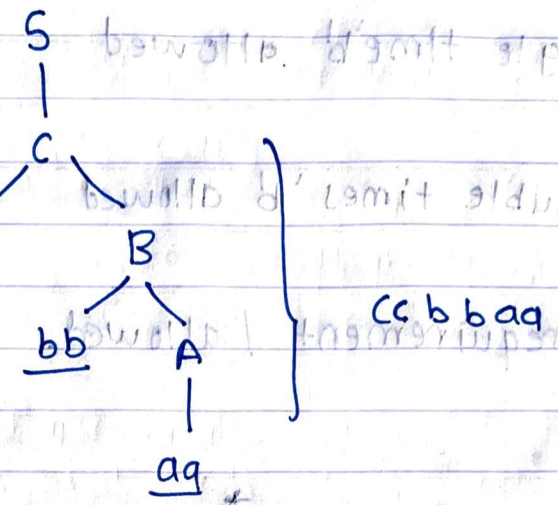
4] $\Sigma = \{a, b\}$. G: S → A | B | C

A → aaB | aaC | aa

B → bbA | bbC | bb

C → ccA | ccB | cc

a) Tree For ccbbaa ∈ L(G)



All nodes must be letters or variables, not strings.

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 an alternate sequence combination of either two or more of 'aa', 'bb', 'cc' or the string 'aabbcc' but not even two one.

The "syllables" never repeat! No

c) Regular expression

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

$((aa(bb|cc)|aa)|(bb(aa|cc)|bb)|(cc(aa|bb)|cc))^+$

If that is what you want, this is way too complicated.

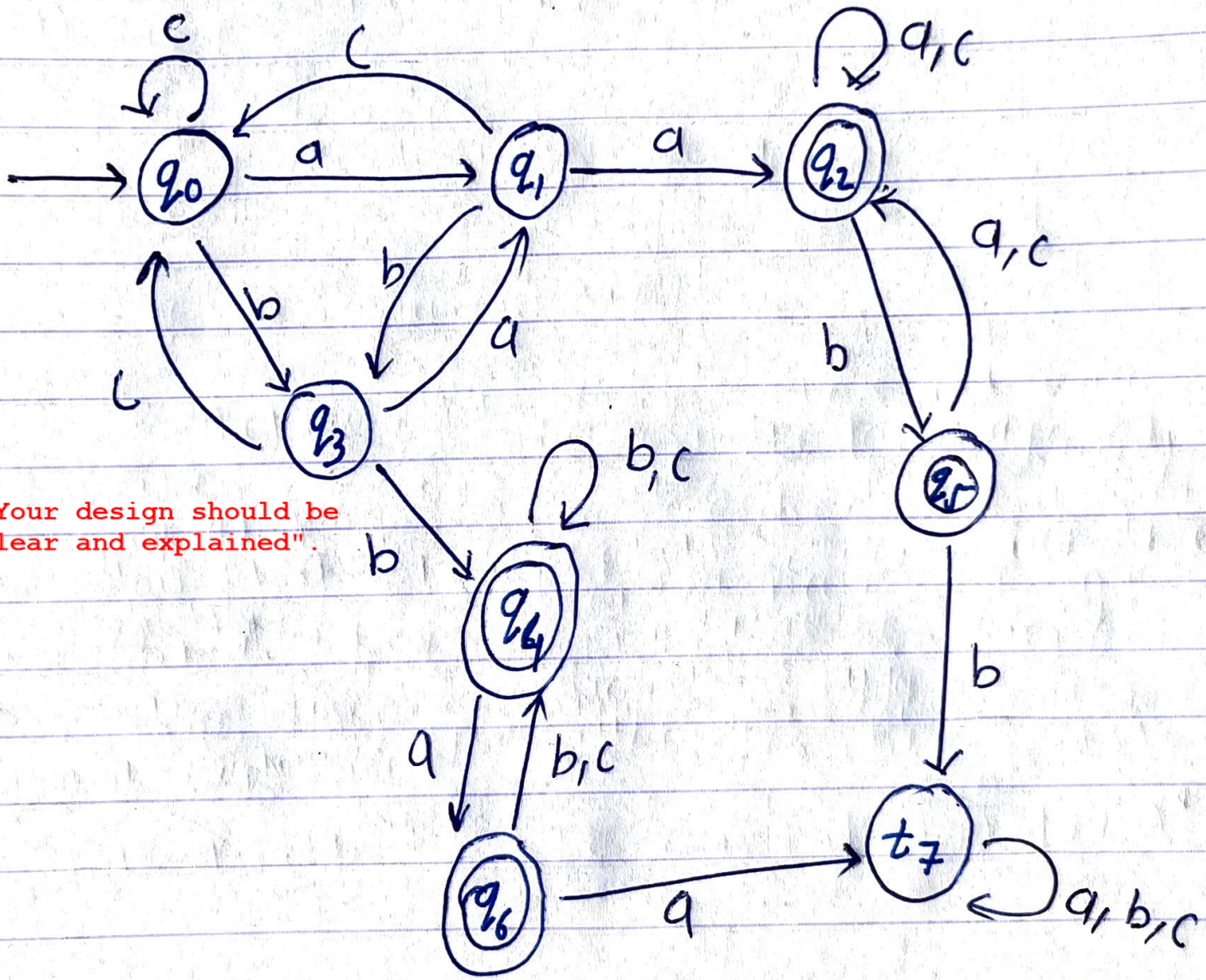
7] $Q \rightarrow \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, t_7\}$

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

initial state $\rightarrow q_0$

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

trap state $\rightarrow t_7$



"Your design should be clear and explained".

6] b G: S → aq | P
P → qq | aQaQ
Q → bb | bP bP
R → cc | bPQPRPQb | aPRQRaPaQaR | abc

Stage 1: $U_q \to q$
$U_b \to b$
$U_c \to c$
$X \to U_aQ$
$Y \to U_bP$
$U_d \to U_bU_c$

$A \to QP$
$B \to RP$
$C \to QU_b$
$D \to U_aP$
$E \to RQ$
$F \to RU_a$
$G \to PX$
$H \to U_aR$
$I \to YA$
$J \to BC$
$K \to DE$
$L \to FG$
$M \to LH$

∴ (replacing values)

Final CNF is:

G: S → U_aU_a | P
P → U_aU_a | XX
Q → U_bU_b | YY
R → U_cU_c | IJ | KM | U_aU_d
$U_a \to a$
$U_b \to b$
$U_c \to c$
$X \to U_aQ$
$Y \to U_bP$
$U_d \to U_bU_c$
A → QP
B → RP
C → QU_b
D → U_aP
E → RQ
F → RU_a
G → PX
H → U_aR
I → YA
J → BC
K → DE
L → FG
M → LH

6] a $A \to a \mid aA \mid Aa \mid BaA \mid AaB$

$B \to bbb \mid BBB$

$C \to bbb \mid BBB \mid ccc \mid CCC \mid cba \mid CBA$

This should give you 3 escapes and three escapes followed by the new recursive variable. You are missing one.

For 'A' we have escape character 'a'

Missing even more in the Rules

we can write

$A \to a \mid aA \mid BaA \mid aA' \mid aBA'$

$A' \to aA' \mid aBA' \mid \lambda$

For 'B' we have escape characters 'b'

we can write

$B \to bbb \mid \lambda B'$

$B' \to \lambda B' \mid BBBB' \mid \lambda$

For 'c' we have escape character 'c'

we can write.

$C \to bbb \mid BBB \mid ccc \mid cba \mid \lambda C' \mid BAC'$

$C' \to \lambda C' \mid CCCC' \mid BAC' \mid \lambda$

Thus for $A \to BaA$ is allowed

for $C \to BBB$ is allowed hence

now our final rule with no left recursion is

$A \to a \mid aA \mid BaA \mid aA' \mid aBA'$

$B \to bbb \mid \lambda B'$

$C \to bbb \mid BBB \mid ccc \mid cba \mid \lambda C' \mid BAC'$

$A' \to aA' \mid aBA' \mid \lambda$

$B' \to \lambda B' \mid BBBB' \mid \lambda$

$C' \to \lambda C' \mid CCCC' \mid \lambda$

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

.... $\{a^3, b^3 \in S \subseteq L$, 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'

- abaaab and aaabaa [assuming $u = a$ and $v = \lambda$]
- baaaba and aabaaa [assuming $u = \lambda$ & $v = a$]

→ For 'bbb'

- babbba and bbbabb [assuming $u = b$ & $v = \lambda$]
- abbbab and bbabbb [assuming $u = \lambda$ & $v = b$]

⇒ $S_1 = \{$abaaab, aaabaa, baaaba, aabaaa, babbba, bbbabb, abbbab, bbabbb$\}$

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 paaaq 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:

1. $w' = uaaav \in S_k$:

If $w' = uaav$ where $u, v \in \Sigma^*$ we can see from part a that at any point where 'aa' is present, it is always joint with a substring. 'aaa' or it turns into 'ubaaab' and contains the substring itself.

Thus, proving that it can be written as paaaq.

2. $w' = vbbv \in S_k$:

If $w' = vbbv$ where $u, v \in \Sigma^*$ we can see from part a that at any point where 'bb' is present it is always joint with substring 'bbb' or turns into 'vabbba' and contains the substring 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 L can be written in the form $paaaq$ or $pbbbq$.