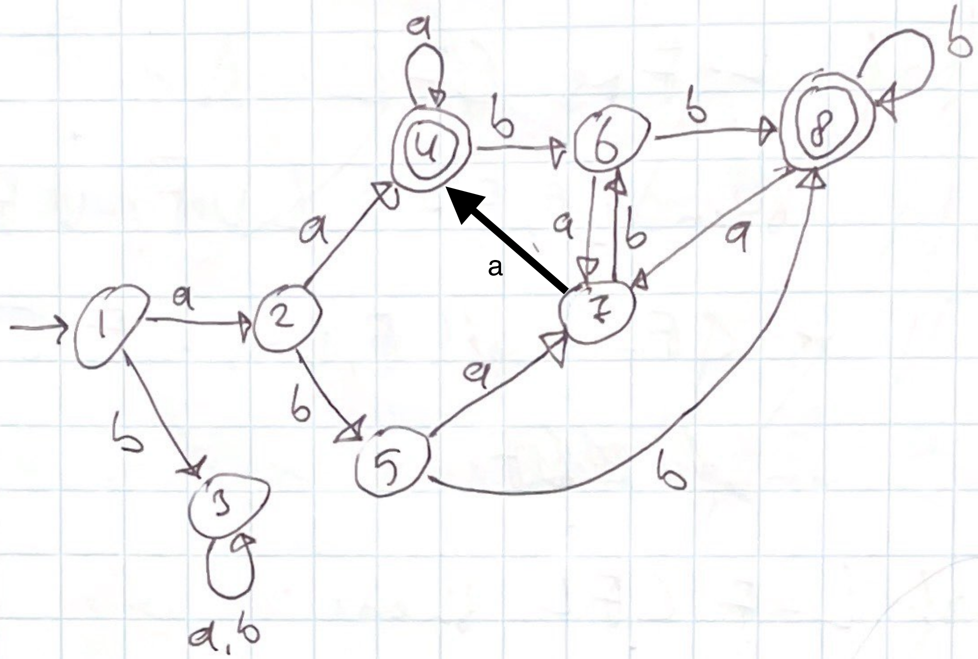
1.



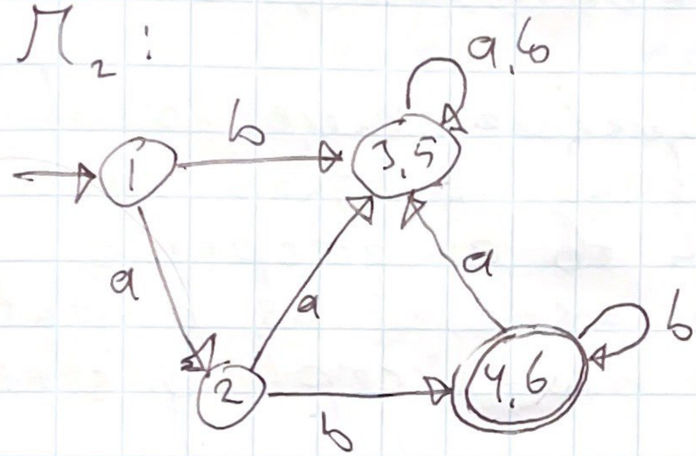
2 a

2	2b				
3	3a	2b			
4	1	1	1		
5	3a	2b	·	1	
6	1	1	1	·	1
	1	2	3	4	5

So 3,5 and 4,6 are equivalent.

b We merge states 3 and 5 and 4 and 6.

M_2:



3a We only have to change the accepting states:

$Q_2 = Q_1$
$q_2 = q_1$
$A_2 = Q_1 - A_1$
$\delta_2 = \delta_1$

(all accepting states in M_1 are not accepting on M_2 and vice versa.)

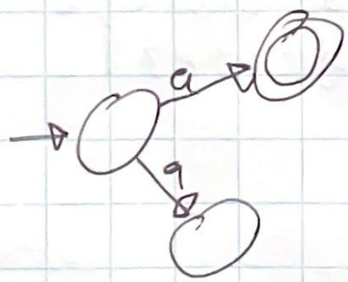
b Three reasons:

* The PDA M_1 could be non-deterministic.

And therefore there could be a string x that has a path to an accepting state, as well as a path to a non-accepting state.

For example:

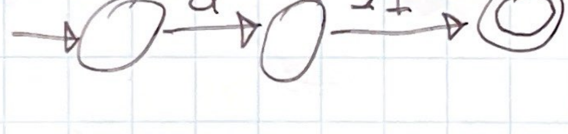
M_1 :



String x= a would be accepted by both M_1 and M_2.

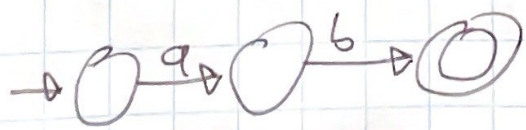
* M_1 could contain Λ-transitions which lead non-accepting states to accepting states.

For example:

M_1 : 

$x = a$ would be accepted by both M_1 and M_2.

* M_1 could 'miss' transitions, such that it always crashes for certain input, whatever the accepting states may be.

M_1: 

$x = b$ wouldn't be accepted by both M_1 and M_2.

c For example:

$$L_1 = XX' = \{ss \mid s \in \{a,b\}^*\}'$$

or $L_1 = A \cap B \cap C' = \{a^i b^i c^i \mid i \geq 0\}$

or $L_1 = \{x \in \{a,b,c\}^* \mid n_a(x) = n_b(x) = n_c(x)\}'$

4a A regular expression for L_1:

$$aa^*bb^*cc^*$$

4 b A regular expression for L_1' :

$$(a+b+c)^*(ba+ca+cb)(a+b+c)^* + (a+b)^* + (a+c)^* + (b+c)^*$$

If $x \in \{a,b,c\}^*$ is not in L_1, there are two possibilities.

1. x contains a's, b's and c's but not in the right order. Then x has to contain a substring ba, ca or cb. Which would put the characters in the wrong order. This is described by the first long term in the regular expression.

2. x contains not all of the three characters. Then x contains either only a's and b's, a's and c's or b's and c's. This is described by the three last terms of the regular expression.

5 a(i) No, $L \not\subseteq L(G_1)$. Because for example $x = abbbbbc$ is in L, but not in $L(G_1)$. The b's that 'belong to the a's', get put at the very end by G_1.

a(ii) No, $L(G_1) \not\subseteq L$. Because for example $x = abbbbcb$ is in $L(G_1)$, but not in L. $S \Rightarrow aSb \Rightarrow abbbCb \Rightarrow abbbbCcb \Rightarrow abbbbcb$.

b(i) No, $L \not\subseteq L(G_2)$ because for example $x = bbb$ is in L, but not in $L(G_2)$. In G_2 at least one a or c is generated through A or C.

b(ii) Yes, $L(G_2) \subseteq L$

c(i) Yes, $L \subseteq L(G_3)$

c(ii) Yes, $L(G_3) \subseteq L$

6a $N_0 = \emptyset$
$N_1 = N_0 \cup \{B\} = \{B\}$
$N_2 = N_1 \cup \emptyset = N_1$

Therefore only B is nullable

b B is nullable, so we add productions where B is left out.

And we remove $B \to \Lambda$

G_2:

$S \to Sa \mid bb \mid AB \mid A$
$A \to aAb \mid BBa \mid Ba \mid a$
$B \to SB \mid a \mid S$

c. S-derivable: $\{A\}$

A-derivable: $\{\} = \emptyset$

B-derivable: $\{S, A\}$

d Now we add the productions of X-derivable variables to X and remove unit productions.

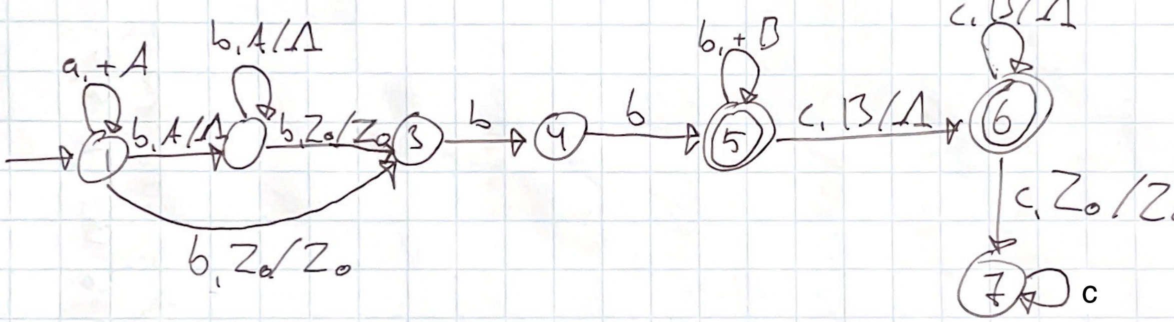
This gives G_3:

$S \to Sa \mid bb \mid AB \mid aAb \mid BBa \mid Ba \mid a$

$A \to aAb \mid BBa \mid Ba \mid a$

$B \to SB \mid a \mid Sa \mid bb \mid AB \mid aAb \mid BBa \mid Ba$

7



7 M_1 reads in state 1 a^i, and puts an A on the stack for every a it reads. In state 2 the same amount of b's are read as there were a's. To count this, for every b read an A get taken off the stack. When the stack is empty we read 3 more b's through states 3, 4 and 5, this is the minimum amount of $b's$ needed for a string in L.

Then we can accept in 5 but we can also read more b's. We count these $b's$ by putting B on the stack for every b read. Then we are allowed to read as many c's as there are B's on the stack. If we have c's left when the stack is empty there are too many c's and we go to state 7 which is non-accepting.

When the letters are in the wrong order the PDA crashes and the string is not accepted.

8 u_1 is not useful, it is not part of L (there is a b missing).

u_2 can be used

u_3 can't be used. !

the decomposition

$v = a^n,\ w = b,\ x = \Lambda,\ y = \Lambda,\ z = b^{3n-1} c^n$

fully obeys the pumping lemma.

u_4 can't be used.

the decomposition:

$v = \Lambda,\ w = abbc,\ x = \Lambda,\ y = \Lambda,$

$z = (abbc)^{n-1} abc$

fully obeys the pumping lemma.