Name: Aytaç SEKMEN

Student ID: 2575983

## QUESTION 1:

**1)** M = (K, Σ, Γ, Δ, S, F)  where K = {q0, q1, q2, q3, q4, q5, q6, q7}, Σ = {a, b}, Γ = {a, b, M0, M1, &}, S = q0, F = {q7}

M0 and M1 are markers to distinguish the $x^R$ part with the randomly given a's and b's as a prefix and suffix to $x^R$. It is not necessary to use this kind of markers but I preffered to using them to distinguish the states between each other.

&: is the bottom marker

Δ = {((q0, e, e),(q1, &)),

((q1, a, e),(q1, a)), ((q1, b, e),(q1, b)),

((q1, e, e),(q2, M0)),

((q2, b, e),(q2, b)), ((q2, a, e),(q2, a)),

((q2, e, e),(q3, M1)),

((q3, a, e),(q3, a)), ((q3, b, e),(q3, b)),

((q3, #, e),(q4, e)),

((q4, e, a),(q4, e)), ((q4, e, b),(q4, e)),

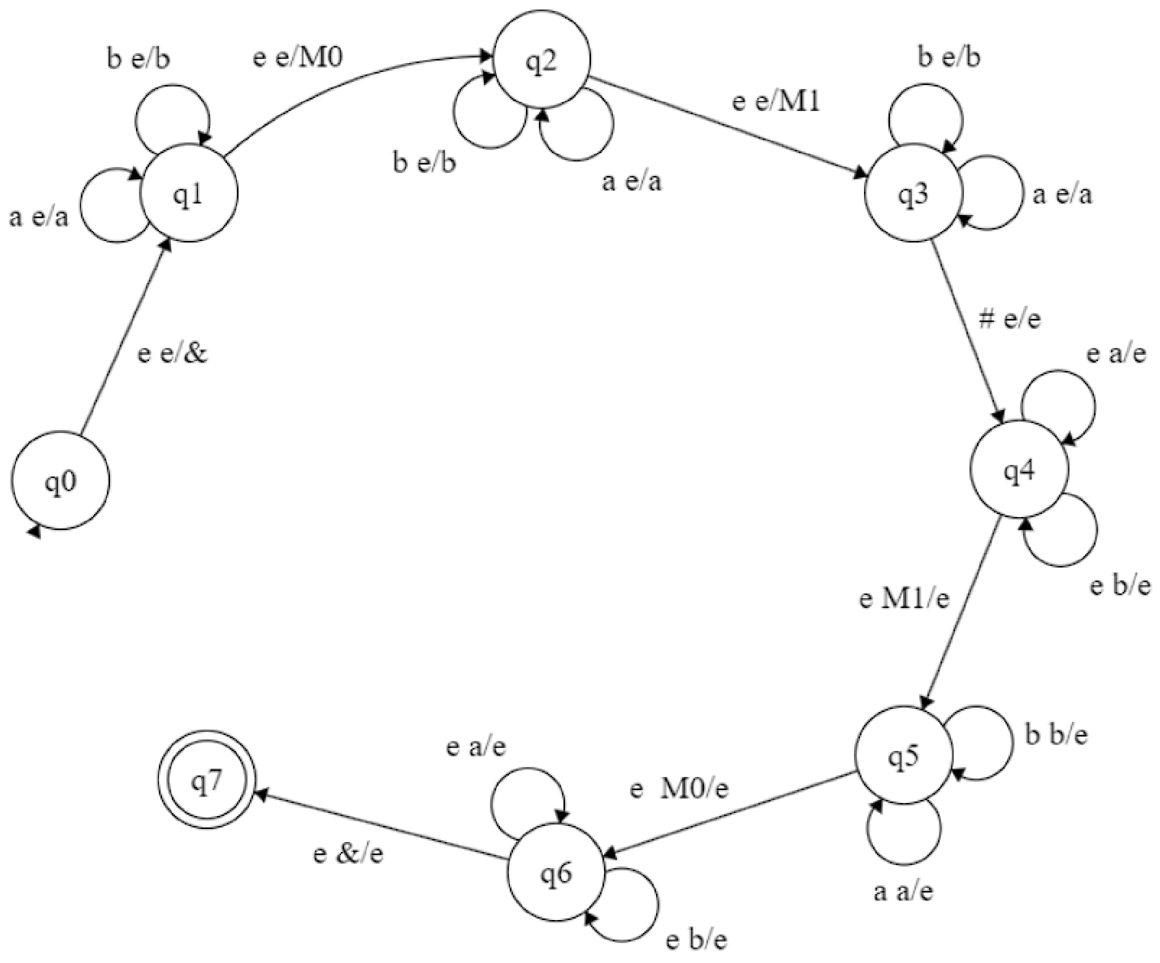
((q4, e, M1),(q5, e)),

((q5, b, b),(q5, e)), ((q5, a, a),(q5, e)),

((q5, e, M0),(q6, e)),

((q6, e, a),(q6, e)), ((q6, e, b),(q6, e)),

((q6, e, &),(q7, e))}



**2)**

M = (K, Σ, Γ, Δ, S, F) where K = {q0, q1, q2, q3, q4, q5}, Σ = {a, b, c}, Γ = {a, b, c, #}, S = q0, F = {q5} and

Note: In this question since there is no seperator input symbol between w and $w^R$, I thought it will be more efficient not to use the Markers as i used in question-1.

Δ ={((q0, e, e),(q1, #)),

((q1, c, e),(q1, c)),

((q1, e, e),(q2, e)),

((q2, a, e),(q2, a)),

((q2, b, e),(q2, b)),

((q2, e, e),(q3, e)),

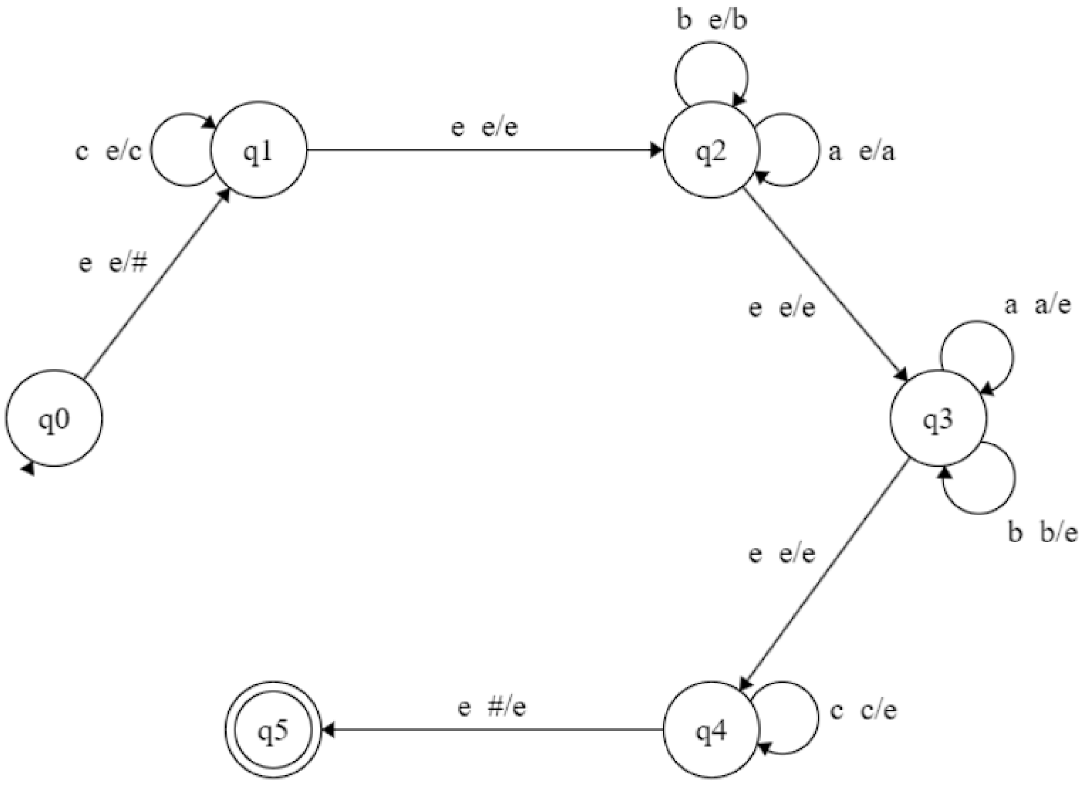
((q3, a, a),(q3, e)),

((q3, b, b),(q3, e)),

((q3, e, e),(q4, e)),

((q4, c, c),(q4, e)),

((q4, e, #),(q5, e))}



## QUESTION 2:

To give a counterxample to this statement. Let's think about a context-free language called L , and its corresponding grammar G = (V, Σ, R, S)   where V={S}, ∑={a,b}, R={S→ aSb, S→e} and S as an start state. Then if we add S→SS and let's say we get the new grammar G'. And for example the language generated by G' will have this string aababb:

Derivation for aababb:

S=>aSb=>aSSb=> aaSbSb=> aaSbaSbb=> aabaSbb=> aababb

But we should notice that our original grammar G actually forms the strings in the form $a^n b^n$ as n≥0. So that L* should actually include the strings that are combination of $a^n b^n$ in a sequence. But one can notice that the string aababb is not in this form. Hence, the new grammar G' actually does not generate L* in this case.

So we concluded that adding rule S → SS does not generate L*.

## QUESTION 3:

**1)**

- L1 is in the class of S-CFL. Because whenever I read "a" as input, I can push "#" to the stack ,which is the only stack symbol. And whenever I read "b" as input, I can pop "#" from the stack. And at the end of reading the input tape, I can look at the stack. If the stack is empty then I can say that the string is in language. Also, if the stack is not empty I can say that the string is not in the language.
- L2 is not in the class of S-CFL. Because I can push "#" ,which is the only stack symbol, whenever I read the input a, and pop that symbol from stack. But there is one thing we should keep in mind. What will happen if there is no symbol in stack and I read b? Since I can not distinguish between the situation where I read b and stack is empty and where I read b and stack has symbol "#", this language is not in the class of S-CFL.
- L3 is not S-CFL. Because in this language there is actually two number to be tracked, m and n. If we had another stack symbol, which shows us that we finished the part related with $a^n b^n$ and lead us to move another state, we could actually show this language. But since we can not distinguish between whether we are at $a^n b^n$ or at $a^m c^m$, this language is not in S-CFL.

**2)** To find another language that is in the class of S-CFL, I found a way of thinking. Since we can only hold the amount of 1 kind of thing, I thought what would happen If I use partition. For example, the language $L_1 = \{a^n b^n \mid n \geq 0$ and $n \in N \}$ could be thought like this $L_2 = \{a^n b^m c^{m+n} \mid n,m \geq 0$ and $m, n \in N\}$. Which actually means the order of a's, b's and c's are fixed and the the number of c's in string equals to the sum of number of a's and b's. And its corresponding PDA has this structure: Whenever I read a or b I can push "#",which is the only stack symbol, and whenever I read c, I can pop "#" from the stack. And at the end of input tape, I can look at the stack, and if stack is empty I can call that string is in the language. And the grammar for the correponding language $L_2$ should be like this:

G = (V, Σ, R, S) where V = {S,T} ∪ Σ, Σ = {a,b,c} S=S and R = {S→aSc | T, T→ bTc | e}

**3)** To decide which memory element, I actually looked at the languages that are recognized by S-CFL's. For example the language $a^n b^n$ can not be recognized by the finite automatons. But if we somehow add a memory element that will keep the count of the number of a's then we could express the language $a^n b^n$ using finite automatos. And actually this memory element is called counter.

**4)**

Counters basically consist of flip-flops that can keep a number, and can be incremented when we face a specific input, and can be decremented when we face a specific input. Let me give more information about the function of this "counter". Thanks to counter, that I will use in my finite automata, I can hold the number of a's that I countered while reading the input tape. For example, whenever I read a "a" from input tape, I can increase the counter by "+1" and whenever I start reading b, I can decrease the counter. That's way, when I came at the end of input tape, I can look at counter and decide whether the given string is in language or not. For example, If counter is "0", then I can say that the string is in language, otherwise I can call that the given string is not in the language.

If I have to mention about, why this counter is needed to give the finite automatos to be able to show the languages in the class of S-CFL, I can say these:

-Thanks to counter I used, I can hold the number of a, I encountered. In finite automatos this process is impossible, because the number n can go to infinity. If we limit like this: $n \leq x$ for x=1,2,3.... Then It would be possible to draw such finite automato to represent that language but in our case n is not limited, so we can't actually draw such finite automatos that can represent that language.

**5)** Hocam in this question, I was actually a lot confused, since it is first time I encounter pda's with such limitations. When I try to come up with a solution I first tried the solution way 1 and then I decided to follow path-2. Since I'm confused which is better way to come up with solution, I decided to write both way of my thinking.

1st way:

If the class of S-CFLs closed under complementation, then complement of any language in the class of S-CFL should also be in the class of S-CFL. For example, let's think about the language L = {w | w=$a^nb^n$ and n≥0}. This language is actually in the class of S-CFL. And any language that consists of the strings that is not in this language should also be in the class of S-CFL. For instance, the language L' = {w | w=$a^nb^ma^mb^n$ and n,m≥1} is at the complement of the language L. But L' actually is not in the class of S-CFL, because there should be at least 2 counter to hold the number n and m seperately, which is not possible with only 1 counter. Since L', which consists of the strings that are not in language L, (I mean L' is a language which stands at the region of complement of L) is not in the class of S-CFLs, I can deduce that the class of S-CFL is not closed under complementation by using proof by contradiction.

2nd way:

For finite automatas, we accept the strings that are at accepting states. And when we say that they are closed under complementation, we can change non-accepting states to accepting ones and accepting ones to non-accepting ones. On the other hand, for improved version of finite automatas, we accept strings when the counter is 0 and we are at one of the final states. And we want this improved version to be also closed under complement. We can again interchange the accepting and non-accepting states but what about the counter? Since our only mechanism is to check whether counter=0 or not, we can not apply this process of taking complement to the improved version of finite automatas which actually represents the language class of S-CFL's. So the class of S-CFL's are not actually closed under complementation.