Ozan Cinci

2448223

HW4

Q1.

To be clear, I first defined two CFLs then transformed them into PDAs.

Q1.1

K = { s,f }

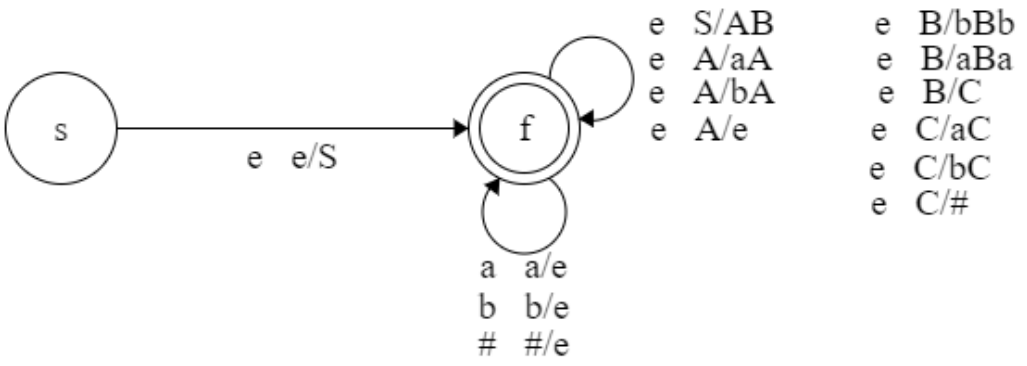
Σ = {a,b,# }

Γ = {a,b,#, S, A,B,C }

Δ = { ((s,e,e),(f,S)), ((f,e,S),(f,AB)), ((f,e,A),(f,aA)), ((f,e,A),(f,bA)), ((f,e,A),(f,e)), ((f,e,B),(f,aBa)), ((f,e,B),(f,bBb)), ((f,e,B),(f,C)), ((f,e,C),(f,aC)), ((f,e,C),(f,bC)), ((f,e,C),(f,#)), ((f,a,a),(f,e)), ((f,b,b),(f,e)), ((f,#,#),(f,e))}

S = s

F = f



Q1.2

K = { s,f }

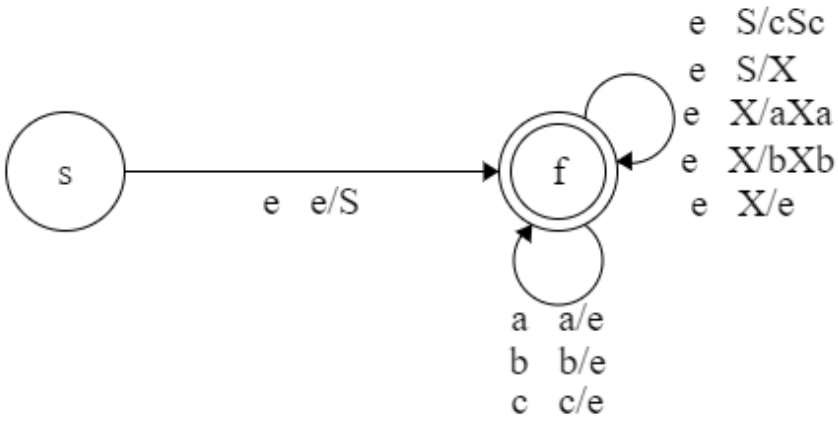
Σ = {a, b ,c}

Γ = {S, X, a, b ,c}

Δ = { ((s,e,e),(f,S)), ((f,e,S),(f,cSc)), ((f,e,S),(f,X)), ((f,e,X),(f,aXa)), ((f,e,X),(f,bXb)), ((f,e,X),(f,e)), ((f,c,c),(f,e)), ((f,a,a),(f,e)),((f,b,b),(f,e)) }

S = s

F = f



Q2.

G = {V, Σ ,R,S}

Σ = {a,b}

S = S

V = {S,a,b}

R = { S -> aSb | bSa | a }

Let's add S->SS as a rule. L* is constructable except "e". There is no way to create "e" using these rules since they cannot create L*. The shortest string it can create is "aa" now.

Q3.

For easiness, the stack symbol is called “$” for the next questions.

Q3.1:

L1 is S-CFL. We can push $ into stack every time we encountered an “a”. Then there will be an empty state transition from the start state to the final state. After transition, we pop $ from stack every time we encounter “b”. If we encounter “a” in this state, there will be no path to follow so fail. After consuming string, we check whether stack is empty or not to decide given string follows the rules.

L2 is not S-CFL. We need at least two symbol to show that one of their symbols’ counters is bigger than others. Let’s say we push $ into stack every time we encounter “a” and pop $ from stack in case of “b”. What happens if we encounter more “b” than “a”. Then it fails to represent minus values since we can only represent the numbers bigger or equal to zero. It also holds if we switch the roles.

L3 is S-CFL. We push $ into stack every time we encounter “a” and pop $ from stack every time we encounter “b”. Then there will be a transition only happening if stack is empty and we are reading “b”. After this transition we push $ into stack every time we encounter “b”, and pop $ from stack every time encounter “c”. After consuming all string there will be an empty transition to final state in case stack is empty.

Q3.2:

$L = \{ wcw^R \mid w \in \{a, b\}^* \}$

$G = \{V, \Sigma, R, S\}$

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

$S = S$

$V = \{S,a,b,c\}$

$R = \{ S \rightarrow aSa \mid bSb \mid c \}$

Q3.3: The memory element is an integer counter.

Q3.4: The main problem in FA is that we cannot detect the count of an element in FA because we do not have any memory. We only know the previous state. So, we cannot keep track of number of occurrences that observed. Let's consider L ={w|w ∈ {a,b}* and #a is not equal to #b }. In this context we can keep track of the differences between #a and #b thanks to the memory which is an integer counter.

If "a" then "counter+1"

If "b" then "counter-1"

After consuming the given string: if the result is not zero, then it is accepted.

Q3.5: No. To be closed under complementation, it must be deterministic so the exact opposite may exist. S-CFL is created by using S-PDA and it is non-deterministic. We could do it deterministic but then deterministic version cannot accept all the strings that can be accepted by non-deterministic PDA since non-deterministic PDA is more powerful than deterministic PDA when it comes to accepting strings. So, it loses its power. As a result, it is NO.