Student Information

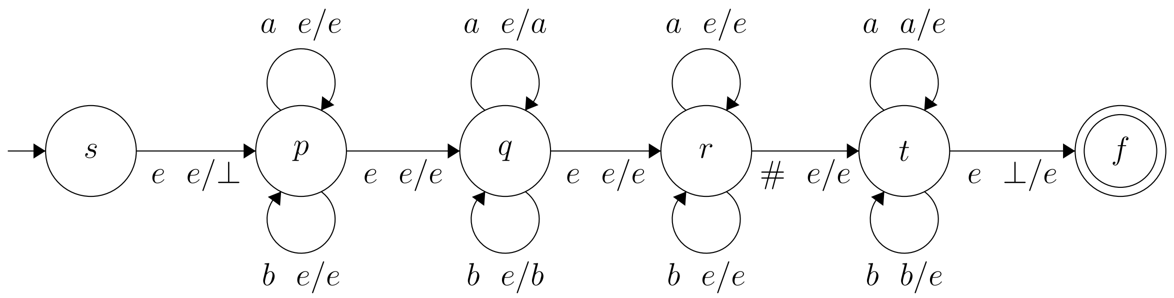
Full Name : Berk Ulutaş
Id Number : 2522084

Answer 1

1)

$M = (K, \Sigma, \Gamma, \Delta, s, F)$
$K = \{s, p, q, r, t, f\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, \perp\}$ ($\perp$ to mark bottom of the stack)
$F = \{f\}$

$$\begin{aligned}\Delta = \{&((s,e,e),(p,\perp)),\\ &((p,a,e),(p,e)),\\ &((p,b,e),(p,e)),\\ &((p,e,e),(q,e)),\\ &((q,a,e),(q,a)),\\ &((q,b,e),(q,b)),\\ &((q,e,e),(r,e)),\\ &((r,a,e),(r,e)),\\ &((r,b,e),(r,e)),\\ &((r,\#,e),(t,e)),\\ &((t,a,a),(t,e)),\\ &((t,b,b),(t,e)),\\ &((t,e,\perp),(f,e))\}\end{aligned}$$

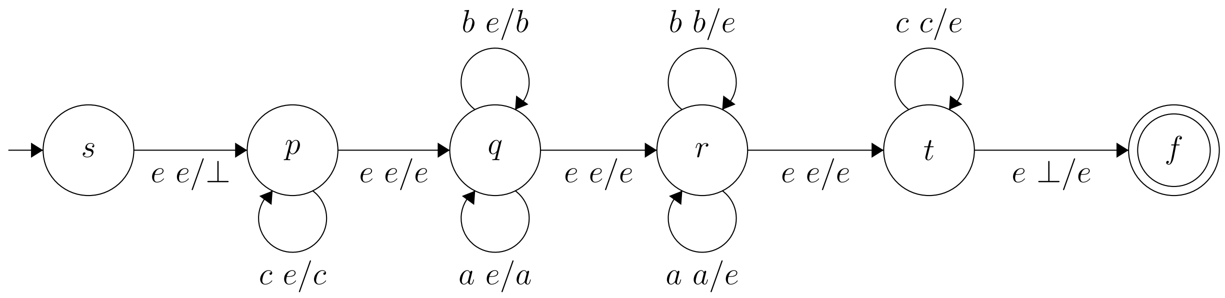


2)

$M = (K, \Sigma, \Gamma, \Delta, s, F)$
$K = \{s, p, q, r, t, f\}$

$\Sigma = \{a, b\}$
$\Gamma = \{a, b, \perp\}$ ($\perp$ to mark bottom of the stack)
$F = \{f\}$

$$\begin{aligned}\Delta = \{&((s, e, e), (p, \perp)),\\ &((p, c, e), (p, c)),\\ &((p, e, e), (q, e)),\\ &((q, a, e), (q, a)),\\ &((q, b, e), (q, b)),\\ &((q, e, e), (r, e)),\\ &((r, a, a), (r, e)),\\ &((r, b, b), (r, e)),\\ &((r, e, e), (t, e)),\\ &((t, c, c), (t, e)),\\ &((t, e, \perp), (f, e))\}\end{aligned}$$



Answer 2

Let $G = (V, \Sigma, R, S)$ where
$V = \{S, 0, 1\}$
$\Sigma = \{0, 1\}$
$R = S \rightarrow 0S0 \mid 1$

This context-free grammar G, generates Context-free language $L = L(G) = \{0^n10^n \mid n \geq 0\}$. It generates strings:

$$1, 010, 00100 \ldots$$

.

Add rule $S \rightarrow SS$ to grammar and R becomes:

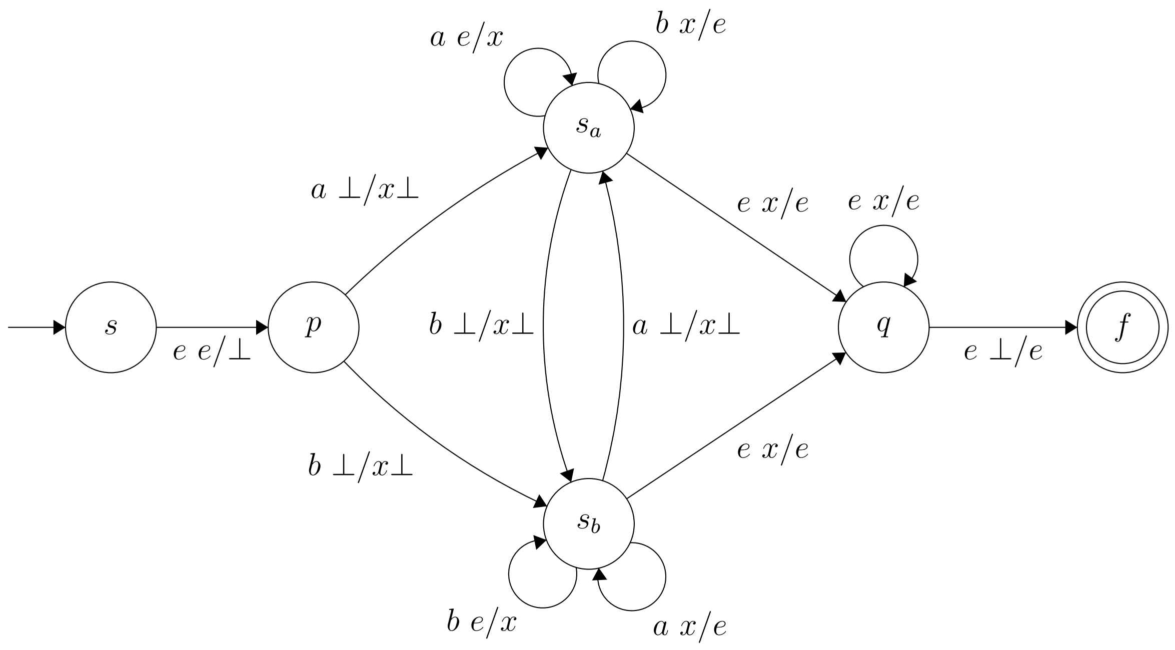
$$R = S \rightarrow SS \mid 0S0 \mid 1$$

It is clear that with the newly added rule, this grammar cannot produce empty string(e). However, L^* includes empty string(e). Thus, adding rule $S \rightarrow SS$ does not produce L^*

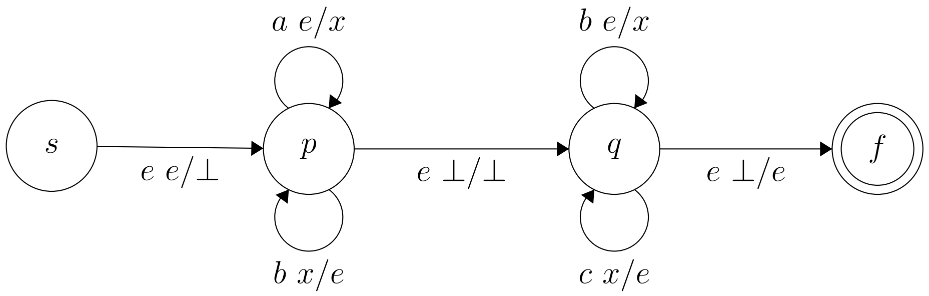
Answer 3

1)

- Let x will be our stack symbol
- $L_1 = \{a^n b^n | n \geq 0\}$. L_1 is an S-CFL. We can construct an S-PDA which recognizes L_1. For each "a" read from the input, push "x" onto the stack. When machine starts reading b's, pop "x" for every "b" read. If the input is completely read and the stack is empty, the S-PDA accepts input string. This S-PDA works since, b's come after a's. First we count number of a's then during popping we check number of b's is equal to number of a's.
- $L_2 = \{w | w \in \{a, b\}^*$ and the number of a's in w is not equal to number of b's in w$\}$. L_2 is an S-CFL. We can construct an S-PDA which recognizes L_2.
  - Let's say there is two states s_a and s_b. s_a indicates that number of a's more than number of b's. s_b indicates that number of b's more than number of a's.
  - If the machine reads an 'a' first, go to state s_a and push "x" onto the stack. While in s_a, if the machine reads 'a', push 'x', if it reads 'b', pop 'x'. If the machine reads 'b' and the stack is empty, go to state s_b and push 'x' onto the stack.
  - If the machine reads an 'b' first, go to state s_b and push "x" onto the stack. While in s_b, if the machine reads 'b', push 'x', if it reads 'a', pop 'x'. If the machine reads 'a' and the stack is empty, go to state s_a and push x onto the stack.
  - After finishing processing the input, if the stack is not empty, go to the final state, pop everything from the stack.
  - If the input is completely read and the stack is empty, the S-PDA accepts input string.



- $L_3 = \{a^n b^{n+m} c^m | m, n \in \mathbb{N}\}$ L_3 is an S-CFL. We can construct S-PDA which recognizes L_3. Think L_3 like this $\{a^n b^n b^m c^m | m, n \in \mathbb{N}\}$. For each "a" read from the input push "x" onto the stack. When machine starts reading b's, pop "x" from stack for each b read(these are the 'n' number of b's). When stack becomes empty push "x" onto stack for each b read (these are the 'm' number of b's). When it starts reading c's. pop "x" from stack for each c read. If the input is completely read and the stack is empty, the S-PDA accepts input string. This S-PDA works since, c's comes after b's and b's comes after a's. First we count number of a's(n) with pushing x onto the stack, then we process n b's with popping x from stack. After that we need to determine m. So, we count b's(m) with stack. Then we check is there m c's.



2)

$L_4 = \{a^n b^{2n} | n \geq 0\}$ L_4 is an S-CFL. We can construct an S-PDA which recognizes L_4. For each "a" read from the input, push two "x" onto the stack. When machine starts reading b's, pop "x" for every "b" read. If the input is completely read and the stack is empty, the S-PDA accepts input string.
$G = \{V, \Sigma, R, S\}$ where

- $V = S, a, b$
- $\Sigma = a, b$
- $R = S \rightarrow aSbb \mid e$

3)

The minimal memory element that must be added to finite automata to recognize S-CFLs is a **Counter**.

4)

- Function of counter to keep track of a single integer value. It can be incremented or decremented by automaton during processing input.
- The counter is less powerful than a stack since, it cannot keep multiple elements.

- By using counter, a finite automaton can recognize S-CFLs. Since, Stack with just one stack symbol basically a counter. It cannot keep multiple elements, just keeps a single integer value like a counter. We can mimic what stack(with single stack symbol) do with a counter, by increasing the counter instead of pushing the stack.

5)

No it is not closed under complementation. Since normal CFL are not closed under complementation S-CFL are also not closed under complementation. Consider $L_1 = \{a^n b^n | n \geq 0\}$ it is a context free language. It can accepted by normal PDA and it can also be accepted by our S-PDA. Theorem 3.5.4: The context-free languages are not closed under intersection or complementation. (from textbook). So, complementation of L_1 is not context free language. Since it is not context free we cannot draw a S-PDA to recognize it. Therefore S-CFL is not closed under complementation.