Student Information

Full Name : Doruk Gerçel
Id Number : 2310027

Answer 1

a.

(i) $G = (V, \sum, R, S)$
$V = (S, A, a, b)$
$\sum = \{a, b\}$
R:
$S \to aSa \mid bA$
$A \to aAa \mid b$

(ii) $G = (V, \sum, R, S)$
$V = (S, A, B, a, b)$
$\sum = \{a, b\}$
R:
$S \to aaAaa \mid abAaa \mid aaAba \mid abAba \mid bbBbb \mid baBbb \mid bbBab \mid baBab$
$A \to a \mid aAa \mid bAa \mid aAb \mid bAb$
$B \to b \mid aBa \mid bBa \mid aBb \mid bBb$

(iii) $G = (V, \sum, R, S)$
$V = (S, A, B, C, T, X, Y, a, b, c)$
$\sum = \{a, b, c\}$
R:
$S \to aAC \mid BbC \mid TbX \mid TYc$
$A \to aAb \mid aA \mid e$
$C \to cC \mid e$
$B \to aBb \mid Bb \mid e$
$T \to aT \mid e$
$X \to bXc \mid bX \mid e$
$Y \to bYc \mid Yc \mid e$

(iv)

b.

When we look at the rules, we can observe that when we choose A, we have one more a's than b's, and when we choose B, we have one more b's and a's. When we choose S it directs us to two rules which are aB and bA. Therefore in first rule we have one more a that comes from a, but as B has one more b, the number of a's and b's are equaled. The same thing applys for the second

rule aswell. Therefore we have seen that these rules produce equal number of a's and b's. Also we need to show that when a string has equal number of a's and b's we form this grammer to prove it. We will show it by induction.
Base Case : (n = 1)
When our string's length is equal to 1 we can either form string a or b, but we know that when a string's length is 1, it can't have equal number of occurences, therefore our base case holds.
Inductive Case: When we have a string in the length of n+1, as in the form of aw, or bw. We can see that our string has one more a's than b's or vice versa. We know that our w has equal number of occurences. So in our example we have one more a in this non balanced string (as it's length is odd). To balance this string we need to form it as 'bs' and to achieve it we add a B character. Also in our second example we add a A character.
Therefore the rules for grammer that we form to obtain this string is also the grammer that was given to us. So as we can both obtain equal number of occurences with this grammer and the grammer that we need to form equal number occurences are the same we can conclude that this statement holds.

Answer 2

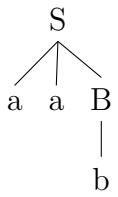
a.

In our first tree, we chose 'Ab' as our starting character, then we chose 'A' as 'Aa' and chose this 'A' as 'a' and as we couldn't move any forward we formed our string which is 'aab'. In our second tree, we chose 'aaB' as our starting character and chose 'B' as 'b' and formed the string 'aab'. Both of them are left parse tree as we always use the leftmost derivable character. Our string s is 'aab'.

b.

$G = (V, \sum, R, S)$
$V = (S, A, T, B, a, b), \sum = (a, b)$
R:
$S \rightarrow Ab \mid aaB$
$A \rightarrow Ta \mid T$
$T \rightarrow e$
$B \rightarrow b$

c.



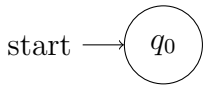
We chose, 'aaB' as our starting character and chose 'B' as 'b' and obtained 'aab'.

Answer 3

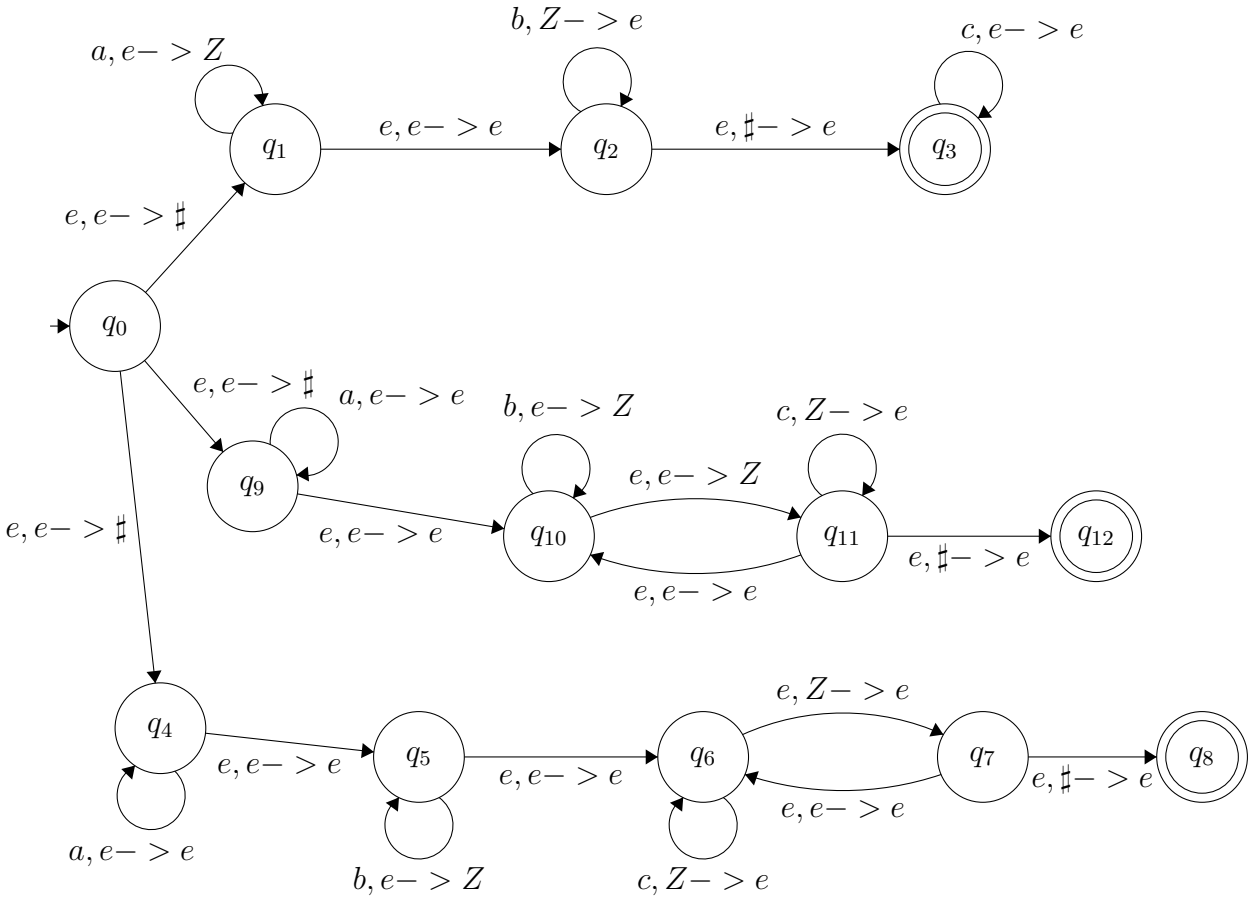
a.

$L = \{a^{2i}b^{3i} \mid i \geq 0\}$
When we go throught the automata we observe that, we can exit without taking any characters, therefore it is possible to produce empty string. Also when we go through the automata we observe that, every time we take i number of x character to our stack, we form 2i number of 'a' characters, and to push out these x characters to reach the control character in the stack, we need to form a following 3i number of 'b' characters. This is how we obtained that language.

b.



c.


$a, e->Z$
$b, Z->e$
$c, e->e$
q_1
$e, e->e$
q_2
$e, \sharp->e$
q_3
$e, e->\sharp$
q_0
$e, e->\sharp$
$a, e->e$
$b, e->Z$
$c, Z->e$
q_9
$e, e->Z$
$e, \sharp->e$
q_{10}
q_{11}
q_{12}
$e, e->e$
$e, e->e$
q_4
$e, Z->e$
q_5
$e, e->e$
q_6
q_7
q_8
$e, \sharp->e$
$e, e->e$
$a, e->e$
$b, e->Z$
$c, Z->e$

(i)

(ii)

State	Input	Stack	Transition
q_0	aabcc	e	-
q_9	aabcc	$\sharp$	$(q_0, e, e), (q_9, \sharp)$
q_9	abcc	$\sharp$	$(q_9, a, e), (q_9, e)$
q_9	bcc	$\sharp$	$(q_9, a, e), (q_9, e)$
q_{10}	bcc	$\sharp$	$(q_9, e, e), (q_{10}, e)$
q_{10}	cc	$Z\sharp$	$(q_{10}, b, e), (q_{10}, Z)$
q_{11}	cc	$ZZ\sharp$	$(q_{10}, e, e), (q_{11}, Z)$
q_{11}	c	$Z\sharp$	$(q_{11}, c, Z), (q_{11}, e)$
q_{11}	e	$\sharp$	$(q_{11}, c, Z), (q_{11}, e)$
q_{12} (Accepts)	e	e	$(q_{11}, e, \sharp), (q_{12}, e)$

State	Input	Stack	Transition
q_0 (Alternative 1)	bac	e	-
q_1	bac	$\sharp$	$(q_0, e, e), (q_1, \sharp)$
q_1	bac	$\sharp$	$(q_1, e, e), (q_2, e)$
q_2	ac	e	$(q_2, b, Z), (q_2, e)$
q_9 (Alternative 2)	bac	$\sharp$	$(q_0, e, e), (q_9, \sharp)$
q_{10}	bac	$\sharp$	$(q_9, e, e), (q_{10}, e)$
q_{10}	ac	e	$(q_{10}, b, Z), (q_{10}, e)$
q_4 (Alternative 3)	bac	$\sharp$	$(q_0, e, e), (q_4, \sharp)$
q_5	bac	$\sharp$	$(q_4, e, e), (q_5, e)$
q_5	ac	Z	$(q_5, b, e), (q_5, Z)$
q_6	ac	Z	$(q_5, b, e), (q_6, e)$
q_7	ac	$\sharp$	$(q_6, b, Z), (q_7, e)$
q_7	ac	$\sharp$	$(q_6, b, Z), (q_8, e)$

First can't happen as there is no Z in stack, rejects.

Second can't happen as there is no Z in stack, rejects.

Third can't happen as we can't pop control character as the string is not finished, rejects.

Answer 4

a.

$M = (\{p, q\}, \sum, V, \triangle, p, \{q\})$
$\triangle =$
{((p,e,e),(q,E))
((q,e,E),(q,E+T))
((q,e,E),(q,T))
((q,e,T),(q,T × F))
((q,e,T),(q,F))
((q,e,F),(q,(E)))
((q,e,F),(q,a))
((q,a,a),(q,e))
((q,+,+),(q,e))
((q,×, ×),(q,e))
(q,(,( ),(q,e))
(q,),)),(q,e))}

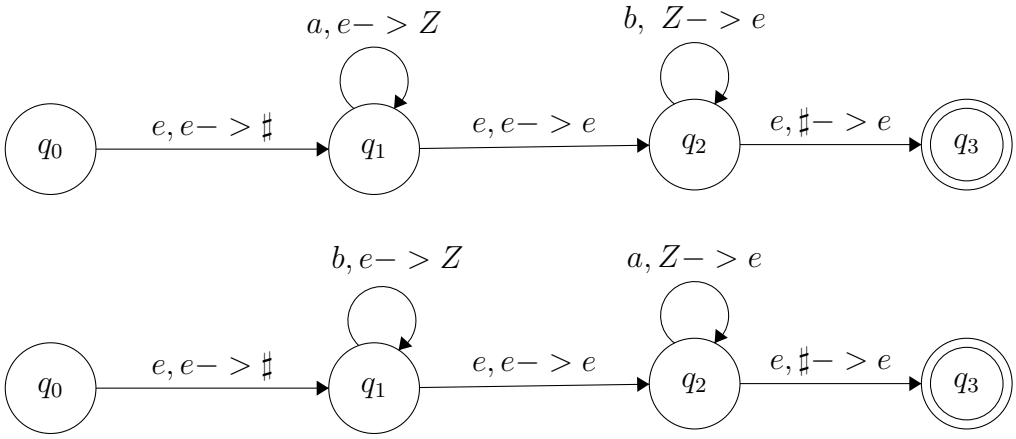
b.

We can see that as they both accept a CFL they are a well constructed PDA by the use of CFG. In a PDA there are only three type of operations on the stack that we can either have empty character and continue to have empty character and both of their lengths are equal to zero and therefore their sum is less than 1 ($0+0 \leq 1$). We can also have a push stack operation in which we express as having an empty character and then adding a new character, therefore empty string has length 0 and our new stack character has a length 1, and their sum is equal to 1 ($0+1 \leq 1$). Also we may have a pop stack operation that we have a stack character and we remove this character in the following step, so first we have character that has length 1 and then empty character with length 0, so their sum is equal to 1 aswell ($1+0 \leq 1$). So we proved that every possible operation that we can do with our stack obeys to the given rule.

Answer 5

a.

(i) We reshape our string as $a^m b^m b^n a^n$ and divide it into two as $a^m b^m$ and $b^n a^n$. As we know the context free languages are closed under concatenation, therefore if both substrings are CFL therefore this expression will be a CFL as well. So if we can show both of them as an automata we can prove that they are CFL.



As we showed both of them are CFL their concetanetion is CFL aswell.

(ii) We know that CFL is not closed under complemantation. Therefore $\{a, b\}^* - L$ is CFL as long as L is not CFL. To prove that L is not CFL we will use pumping theorem for CFLs. We choose our pumping length P.
Case:
When we choose vyx as any part of $ba^x(x < n)$ we obtain a string that doesn't obey the pattern. For example we will obtain a string like $....ba^iba^{i+1}ba^{i-x}bba^{i-y}.....$ Therefore L is not a CFL according to the pumping theorem.
Therefore, as our expression is its complement we can state that our expression is a CFL.

b.

(i) We choose our pumping length p, our string is a^xb^p in which $x \leq p^2$, $vy \neq e$ and $\mid vxy \mid \leq p$.
Case 1:
We choose vxy from 'a' in length p. vxy = a^k and $k < x$. When we pump x it will only increase the number of a's not b's, so x will be bigger than p^2.
Case 2:
We choose vxy in the boundary of ab's intersection, choose y empty and we in the part of intersection of ab. When we pump v, we obtain $a^ib^ja^xb^y$ so the order of our string doesn't obey the rules.
So it is not CFL.

(ii) We choose our pumping length p, our string is $a^pb^pa^pb^pa^pb^p$, $vy \neq e$ and $\mid vxy \mid \leq p$.
Case 1:
We first choose our vxy from 'a' in length of p. (Choose first group of a's in our example.) When we pump v or y there will be more number of a's in one w. For example our string will be like $a^xb^pa^pb^pa^pb^p$, in whix x is a bigger number than p, therefore our string is now in the form of 'xww' in which three of our substrings are equal to each other. This case holds for every 'vxy' that was chosen in the single group of 'a's and 'b's.
Case 2:
In this case we choose the parts that 'a's and 'b's intersect in a w. When we choose 'vxy'

in the intersection of 'ab'(In our example we choose in first w), when we punp 'v' or 'y', our string becomes like $a^i b^j a^x b^y a^p b^p a^p b^p$. So as we see, two of the w are still the same but there is one non-similar string. This is a possible case to occur in both second and third w aswell.
Case 3:
In this case we choose the parts of intersection between 'b's and 'a's that are in different w's. When we choose one part of our 'vxy' in both 'b' part of first w and 'a' part of the following w, adn when we pump v or y, our string becomes like $a^p b^i a^j b^x a^y b^p a^p b^p$. So three w's are no longer identical. This case may occur between the first and second w, and between second and third w.
We can't choose any bigger 'vxy' string as they must not be bigger than the length p.
As we see from these 3 cases it is not CFL.

Answer 6

(i) FALSE

(ii) TRUE

(iii) TRUE

(iv) FALSE