CS 321 HW 5 – 25 points

1. (5 pts) Convert the grammar below to CNF.

G = (V,T,S,P) where
V = { S, A, B, C, D}
T = { 0, 1, 2 } and P is given below.

S -> A | ABD | 0BB
A -> 0 | BAA
B -> BB | 1 | 2 | λ
C -> CD | 0
D -> D1 | DD

Step 1 : Removing Useless productions – C & D

S -> A | 0BB
A -> 0 | BAA
B -> BB | 1 | 2 | λ

Step 2 : Removing λ productions
S -> A| 0BB | 0B |0
A -> 0 | BAA | AA
B -> BB | 1 | 2 | B

Step 3: Remove Unit productions :
S -> 0BB | 0B |0| BAA | AA
A -> 0 | BAA | AA
B -> BB | 1 | 2

Step 4 : Adding Terminal Symbols :
S -> T_0BB | T_0B |0| BAA | AA
A -> 0 | BAA | AA
B -> BB | 1 | 2
T_0 -> 0

Step 5 : Adding intermediate variables :
S -> T_0V_1 | T_0B |0| BV_2 | AA
V_1 -> BB
V_2 -> AA
A -> 0 | BV_2 | AA
B -> BB | 1 | 2
T_0 -> 0

2. (5 pts) Consider the CNF grammar
G = (V,T,S,P) where V = {S, A, B, C, D }, T = { a, b, c }, S = S and P is given below.

S -> AB | AD | AC
A -> AA | a
B -> BB | AB | b
C -> AC | DC | c
D -> DD | b | c

Use the CYK algorithm to determine if the strings w1 = babbc and w2 = aaaabb are in the language L(G). Show the DP table. If the string is in L(G) construct the parse tree.

a) String w1 = babbc

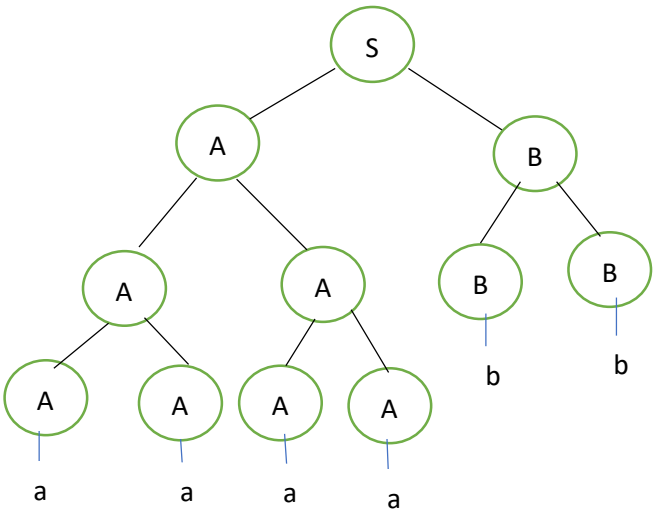
B,D	Ø	B	B	C
	A	S, B	S, B	S, C
		B,D	B,D	C, D
			B,D	C,D
				C,D

The string babbc cannot be generated.

b) String w2 = aaaabb

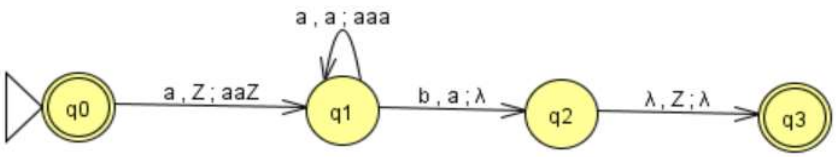
A	A	A	A	S, B	S, B
	A	A	A	S, B	S, B
		A	A	S, B	S, B
			A	S, B	S, B
				B, D	B, D
					B, D

Therefore the string w2 = aaaabb can be generated.



3. (15 pts) Construct npda's that accept the following languages on ∑ = {a, b, c}. Give both a verbal explanation on how your npda works and the formal definition including the transition function and/or transition graph. You must use JFLAP. Submit the transition graph in the HW pdf and the JFLAP code file for each problem.

a) L = { $a^n b^{2n}$: n ≥ 0 }



Let M be the npda for the given language

M = (Q, ∑, ⌈, δ, q0,z, F) where

Q = {q0, q1, q2, q3}

∑ = {a, b}

F = {q0, q3}

Transisiton function δ :

δ (q0,a,z) = {(q1,aaz)}

δ (q1,a,a) = {(q1,aaa)}

δ (q1,b,a) = {(q2,λ)}

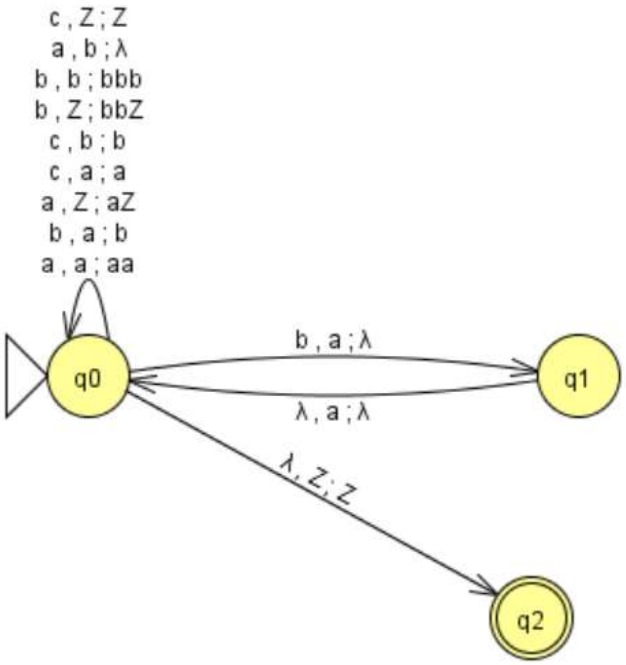
δ (q2,b,a) = {(q2,λ)}

δ (q2,λ,z) = {(q3,λ)}

Since the language accepts empty string, we have made the initial state as the final state.

On seeing an ‘a’, it pushes two a’s on to the stack so that makes a total 3 a’s. On seeing a ‘b’, it pops out one a each time and this way we can confirm that there are twice b’s as a’s. Finally it accepts when the input is over and there are no a’s on stack.

b) L = { w : $n_a(w) = 2\ n_b\ (w)$ }



Let M be the npda for the given language

M = (Q, ∑, ⌈, δ, q0,z,F) where

Q = {q0, q1, q2}

∑ = {a, b, z}

F = {q2}

Transisiton function δ :

δ (q0,λ,z) = {(q2,z)}

δ (q0,c,z) = {(q0,z)}

δ (q0,c,a) = {(q0,a)}

δ (q0,c,b) = {(q0,b)}

δ (q0,a,z) = {(q0,az)}

δ (q0,b,z) = {(q0,bbz)}

δ (q0,a,a) = {(q0,aa)}

δ (q0,b,b) = {(q0,bbb)}

δ (q0,a,b) = {(q0,λ)}

δ (q0,b,a) = {(q0,b)}

δ (q0,b,a) = {(q1,λ)}

δ (q1,λ,a) = {(q0,λ)}

Since the language accepts empty string, on lambda transition it is made to go to a final state.

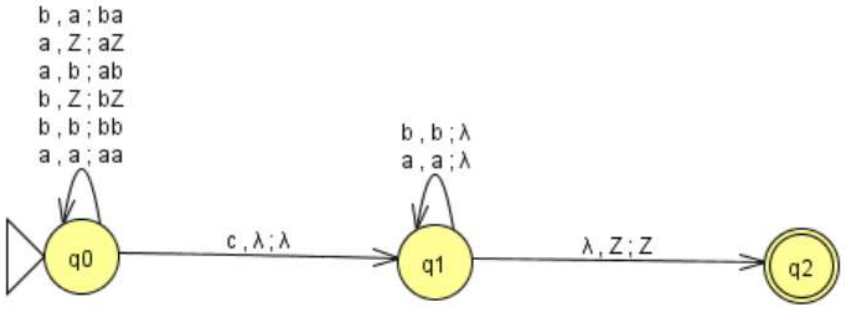
On seeing an a, if the stack is empty or the top of the stack is ‘a’ then it pushes an ‘a’ on to the stack.

On seeing an b, if the stack is empty or the top of the stack is ‘b ‘then it pushes an 2b’s on to the stack.

On seeing an ‘a’ & top of stack is ‘b’ -> it pops b

On seeing a ‘c’ -> just reads it

c) L = { wcwR : w ∈ {a,b}* }



Let M be the npda for the given language

M = (Q, ∑, ⌈, δ, q0,z,F) where

Q = {q0, q1, q2}

∑ = {a, b, z}

F = {q2}

Transisiton function δ :

δ (q0,a,a) = {(q0,aa)}

δ (q0,b,a) = {(q0,ba)}

δ (q0,a,b) = {(q0,ab)}

δ (q0,b,b) = {(q0,bb)}

δ (q0,a,z) = {(q0,az)}

δ (q0,b,z) = {(q0,bz)}

δ (q0,c,b) = {(q1,z)}

δ (q0,c,a) = {(q1,a)}

δ (q0,c,b) = {(q1,b)}

δ (q1,a,a) = {(q1,λ)}

δ (q1,b,b) = {(q1,λ)}

δ (q1,λ,z) = {(q2,z)}

The language accepts a c. On seeing an ‘a’ or a ‘b’, it pushes a or b respectively on to the stack. On seeing a ‘c’, without pushing anything it moves to q1.

When it is in q1, when the top of the stack is ‘a’, then on seeing an ‘a’ it pops an ‘a’.

Similarly, while in q1, when the top of the stack is ‘b’, then on seeing an ‘b’ it pops an ‘b’.

By this way it checks the start after the c and confirms the w and w^R. Finally it accepts if there are no more a’s and b’s on the stack and the input is done.