**Name:** Y. Tejaswi

**Roll No:** 23691A05L8

**Branch:** CSE-D

**AUTOMATA THEORY AND COMPILER DESIGN**

**ASSIGNMENT-1**

**1.Let M = ({q0, q1, q2, q3}, {0,1}, δ, q0, {q2, q3}) be ε-NFA. Where δ(q0, 0) ={q0, q1}, δ(q0, 1) = {q1}, δ(q1, 0) = {q2, q3}, δ(q1, ε) = {q1}, δ(q1, 1) ={q0, q1}, δ(q2, 0) = {q2}, δ(q2, ε) = {q3}, δ(q2, 1) = {q0, q3,}, δ(q3, 0) ={q3}, δ(q3, 1) = {q2, q3}, δ(q3, ε) ={q0}. Construct its equivalent DFA.**

**Answer:**

**1) ε-closures**  
Using q1 -ε-> q1, q2 -ε-> q3, and q3 -ε-> q0 ε-transitions:  
Q0 = {q0} for ε-closure(q0)  
= {q1} ε-closure(q1)  
Since q2→q3→q0, ε-closure(q2) = {q2, q3, q0}  
(q3) = {q3, q0} ε-closure  
DFA's initial state is ε-closure({q0}) = {q0}.

**2) Subset construction (states of attainable DFA)**Each DFA state should be written as a collection of NFA states.  
Let's  
A = {q0} (start)  
B = {q0, q1}  
C = {q1}

D = (all states) {q0, q1, q2, q3}  
E = {q0, q2, q3} (derived from {q2,q3} closure)  
**Transitions are now calculated as follows: δ\_DFA(state, symbol) = ε-closure( ⋃ δ(q, symbol)):**  
From A = {q0}:  
q0,q1 = δ(q0,0) → ε-closure = {q0,q1} = B on 0  
δ(q0,1) = {q1} → ε-closure = {q1} = C on 1

From B = {q0, q1}:  
δ(q0,0) ∪δ(q1,0) = {q0, q1} ∪ {q2, q3} = {q0, q1, q2, q3} → ε-closure = D on 0  
δ(q0,1) ∪δ(q1,1) = {q1} ∪ {q0, q1} = {q0, q1} → ε-closure = B on 1

From C = {q1}:  
δ-closure ({q2, q3}) = {q0, q2, q3} = E on 0: δ(q1,0) = {q2, q3}  
δ(q1,1) = {q0, q1} → ε-closure = B on 1

From {q0,q2,q3} = E:  
on 0: δ(q0,0)∪δ(q2,0)∪δ(q3,0) = {q0,q1}∪{q2}∪{q3} = {q0,q1,q2,q3} → ε-closure = D  
on 1: δ(q0,1)∪δ(q2,1)∪δ(q3,1) = {q1}∪{q0,q3}∪{q2,q3} = {q0,q1,q2,q3} → ε-closure = D  
**From the whole set D = {q0,q1,q2,q3}:**  
goes to D on 0:  
travels to D on 1  
Any DFA state that contains q2 or q3 is considered to be acceptable, therefore D and E.

**3) DFA transition table (before minimization)**

| **DFA state** | **NFA-set** | **on 0** | **on 1** | **accepting?** |
| --- | --- | --- | --- | --- |
| A | {q0} | B = {q0, q1} | C = {q1} | No |
| B | {q0, q1} | D = {all} | B | No |
| C | {q1} | E = {q0, q2, q3} | B | No |
| E | {q0, q2, q3} | D | D | **Yes** |
| D | {q0, q1, q2, q3} | D | D | **Yes** |

**4) Reduce DFA**Separation by acceptance or rejection:  
Accepting: {D, E}  
Refusing to accept: {A, B, C}  
Refine:  
D and E merge because they have the same transitions (both map inputs to accepting class).  
B and C merge because they act similarly with regard to partitions (0 → accept, 1 → non-accept).  
(0 -> non-accept, 1 → non-accept) A is unique.  
**Therefore, the minimised DFA is in three states:**

Give them names:  
S0 (start, non-accept) = A = {q0}  
S1 = merge (B, C) — representative {q0, q1} (non-accept)  
S2 = merge (D, E) — representative q0, q1, q2, q3 (accepting sink)

**Reduced size of the transition table:**

| **State** | **on 0** | **on 1** | **Accepting?** |
| --- | --- | --- | --- |
| S0 | S1 | S1 | No |
| S1 | S2 | S1 | No |
| S2 | S2 | S2 | **Yes** |

More explicitly:

* From S0: 0 → S1, 1 → S1
* From S1: 0 → S2, 1 → S1
* From S2: 0 → S2, 1 → S2

An equivalent **minimal DFA** has 3 states {S0, S1, S2} with start S0, accepting {S2}, and transitions:

* δ(S0,0) =S1, δ(S0,1) =S1
* δ(S1,0) =S2, δ(S1,1) =S1
* δ(S2,0) =S2, δ(S2,1) =S2

**2. Convert the following grammar to GNF:**

**S→CA|BB**

**B→b|SB**

**C→b**

**A→a**

**Answer:**

Convert the grammar into **Greibach Normal Form (GNF).**

S → C A | B B

B → b | S B

C → b

A → a

**Step 1: Recall definition of GNF**

If each production takes the following form, then a CFG is in GNF:

A→ a A1 ​A2​ … Ak ​(a∈T, Ai​∈N, k≥0)

In other words, a terminal may be followed by non-terminals at the beginning of each RHS.  
(The start symbol can only use ε if the grammar produces it.)

**Step 2: Put the nonterminal in the correct order.**  
Consider the following nonterminal ordering:

C, A, S, B

When a production's leftmost symbol is another nonterminal, this aids in determining the replacement sequence.

**Step 3: Rewrite simple terminal rules:**

C → b (already GNF)

A → a (already GNF)

**Step 4: Deal with S → C A | B B**  
Change C to b:

S→ b A ∣ BB

S → b A, the first rule, is already GNF.  
The nonterminal B → must be fixed later is where the second rule starts.

**Step 5: Handle B → b | S B**Production B → b has already reached GNF status.  
Nonterminal S starts the synthesis B → S B.  
Next, use its rules to substitute S (S → bA | B B):

B→ b∣(bA)B∣(BB)B

That is:

B→ b∣ bAB ∣BBB

**Step 6: Get rid of B's immediate left recursion**B → B B B is an example of left recursion.  
Use the conventional approach:  
If

B→α1​∣α2​∣Bβ

then rewrite as:

B→α1​B′∣α2​B′

B′→βB′∣ε

Here:

* Non-left-recursive parts = b, b A B
* Left-recursive part = B B B

So:

B → b B' | b A B B'

B' → B B B' | ε

**Step 7: Return to GNF form by expanding**Now, the first letter of every B output is b (good).  
Expand again by using B's rules (b B', b A B B'), since B' still starts with B.

Starting with terminal b, this produces RHS.  
Fix S → B B in step eight.  
Substitute its GNF rules (b B' and b A B B') for the leading B:

S → b A

S → b B

S → b A B B

**Final GNF Grammar**

Thus, the grammar in GNF is:

A → a

C → b

S → b A

| b B' B

| b A B B' B

B → b B'

| b A B B'

B' → b B' B B'

| b A B B' B B'

| ε