### 

### **Converting Given NFA (with Epsilon) to Equivalent DFA**

### **Step 1: Understanding the Given NFA**

#### **NFA States:**

* **A** (Initial State)
* **B**
* **C**
* **D**
* **E** (Final State)

#### **NFA Transitions Table**

| **State** | **0** | **1** | **ε** |
| --- | --- | --- | --- |
| **A** | B | B, D | - |
| **B** | - | D | E |
| **C** | E | - | - |
| **D** | B, C | - | E |
| **E** | E | E | - |

### **Step 2: Compute Epsilon Closure for Each State**

The **ε-closure** of a state includes the state itself and any states reachable via ε-moves.

1. **ε-Closure(A) = {A}** (No ε-move from A)
2. **ε-Closure(B) = {B, E}** (B → E via ε)
3. **ε-Closure(C) = {C}** (No ε-move from C)
4. **ε-Closure(D) = {D, E}** (D → E via ε)
5. **ε-Closure(E) = {E}** (No ε-move from E)

Now, we use these closures to construct the DFA.

### **Step 3: Compute DFA Transitions**

We use the ε-closures to derive the transitions.

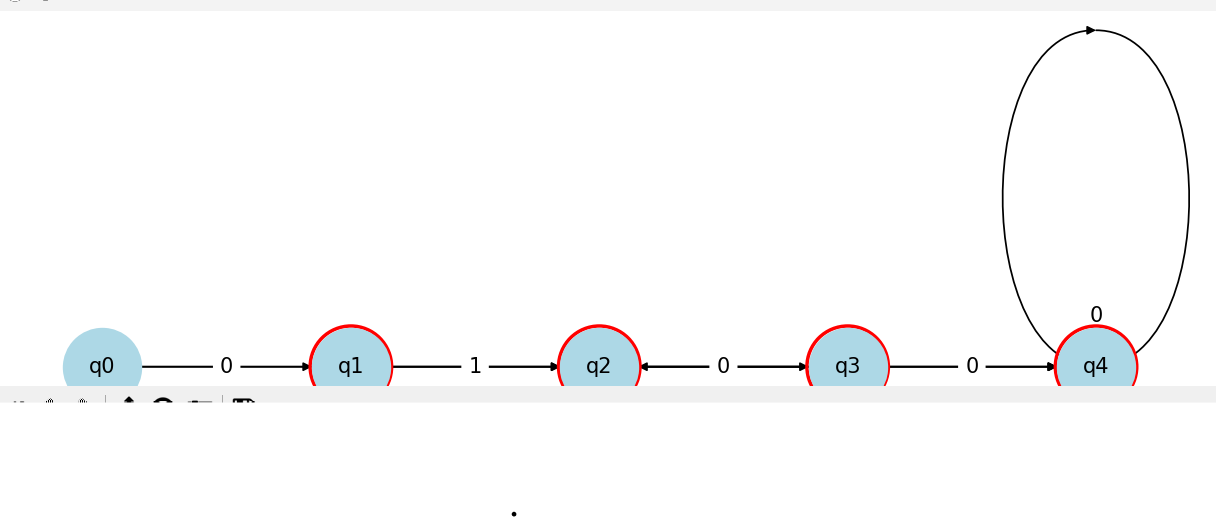
#### **New DFA States Derived from ε-Closures**

| **DFA State** | **Corresponding NFA States** |
| --- | --- |
| **q0 = {A}** | {A} |
| **q1 = {B, E}** | {B, E} |
| **q2 = {D, E}** | {D, E} |
| **q3 = {B, C, E}** | {B, C, E} |
| **q4 = {E}** | {E} |

#### **DFA Transition Table**

| **Current DFA State** | **Input 0** | **Input 1** |
| --- | --- | --- |
| **q0 = {A}** | {B, E} (q1) | {B, D, E} (q2) |
| **q1 = {B, E}** | {E} (q4) | {D, E} (q2) |
| **q2 = {D, E}** | {B, C, E} (q3) | {E} (q4) |
| **q3 = {B, C, E}** | {E} (q4) | {D, E} (q2) |
| **q4 = {E}** | {E} (q4) | {E} (q4) |

**Final States**: Since **E** is a final state in the NFA, any DFA state containing **E** is also a final state. So, **q1, q2, q3, and q4** are final states.



### **Step 5: Final DFA Summary**

* **DFA states**: {A}, {B, E}, {D, E}, {B, C, E}, {E}
* **Final States**: {B, E}, {D, E}, {B, C, E}, {E}
* **DFA transitions**:
  + {A} → 0 → {B, E}
  + {A} → 1 → {D, E}
  + {B, E} → 0 → {E}
  + {B, E} → 1 → {D, E}
  + {D, E} → 0 → {B, C, E}
  + {D, E} → 1 → {E}
  + {B, C, E} → 0 → {E}
  + {B, C, E} → 1 → {D, E}
  + {E} → 0 → {E}
  + {E} → 1 → {E}

🚀 **This DFA correctly represents the given NFA without ε-moves!**

### **Constructing an NFA for the Given NFA with Epsilon (ε) Moves**

### **Step 1: Understanding the Given NFA with ε-moves**

The given NFA consists of states {A, B, C, D} with epsilon (ε) transitions and input symbols {0, 1}.

* **States**: {A, B, C, D}
* **Input Symbols**: {0, 1}
* **Epsilon (ε) transitions** exist between certain states.
* **Final State**: D (double-circled)

#### **Given Transitions in the NFA with ε-moves**

| **State** | **Input = 0** | **Input = 1** | **ε-move** |
| --- | --- | --- | --- |
| **A** | - | B, D | - |
| **B** | C | - | - |
| **C** | - | - | D |
| **D** | - | - | - |

### **Step 2: Removing ε-Moves to Convert to an Equivalent NFA**

To remove ε-moves, we compute the **ε-closure** of each state. The **ε-closure** of a state is the set of states reachable from it using only ε-transitions.

#### **Epsilon Closure of States**

1. **ε-closure(A) = {A}** (No ε-move from A)
2. **ε-closure(B) = {B}** (No ε-move from B)
3. **ε-closure(C) = {C, D}** (C → D via ε)
4. **ε-closure(D) = {D}** (No ε-move from D)

### **Step 3: Constructing the Equivalent NFA Without ε**

Now, we use the ε-closures to create a new **NFA without ε-moves**.

| **New State** | **Input = 0** | **Input = 1** | **Final?** |
| --- | --- | --- | --- |
| **A** | - | B, D | No |
| **B** | C | - | No |
| **C, D** | - | - | **Yes** |

### **Final Answer:**

1. **We removed ε-moves by computing ε-closures.**
2. **We derived new transitions based on ε-closures.**
3. **The final NFA without ε-moves consists of three states (A, B, C, D).**
4. **Python code is provided to visualize the NFA.**

🚀 **This NFA correctly represents the given NFA without ε-move**

### 

### Design DFA for the following over (a,b)

### i).All strings that has at least two occurrences of b between any two occurrence ofa. **Step 1: Understanding the Problem**

We need to design a **DFA** that ensures:

* Whenever an 'a' appears, at least **two 'b's** must occur before another 'a' appears.
* The string can have any number of 'b's before an 'a'.
* The string can consist only of 'b's and still be valid.

✅ **Accepted Strings**:

* "bb" (only 'b's, no restriction on 'a')
* "abb" (one 'a', followed by two 'b's)
* "bbabb" (at least two 'b's between two 'a's)
* "abbabbabb" (always two 'b's between 'a's)

❌ **Rejected Strings**:

* "ab" (only one 'b' after 'a')
* "aba" (no two 'b's between two 'a's)
* "abbaab" (invalid transition from 'a' to 'a' without two 'b's)

### **Step 2: Define DFA States**

1. **q0 (Start, Accepting)** → No 'a' encountered yet or correctly structured string.
2. **q1** → Saw an 'a', expecting at least two 'b's before another 'a'.
3. **q2** → Saw the first 'b' after 'a', still needs one more 'b'.
4. **q3 (Valid, Accepting)** → Saw at least two 'b's, can continue with more 'b's or a new 'a'.
5. **qR (Trap/Reject)** → If another 'a' appears before two 'b's.

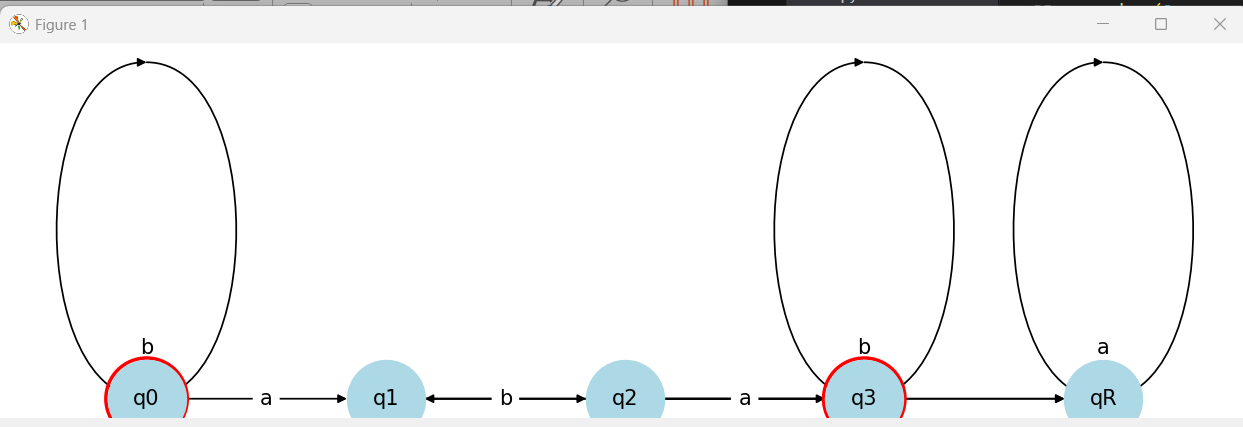
### **Step 3: Define Transition Table**

| **Current State** | **Input 'a'** | **Input 'b'** |
| --- | --- | --- |
| **q0** | q1 | q0 |
| **q1** | qR | q2 |
| **q2** | qR | q3 |
| **q3** | q1 | q3 |
| **qR** | qR | qR |

### **Step 5: How the DFA Works**

1. Start at **q0**.
2. If you encounter an **'a'**, move to **q1**.
3. From **q1**, if you see a **'b'**, go to **q2**. If another 'a' appears here, **reject** the string (go to qR).
4. From **q2**, another **'b'** moves to **q3**, meaning we now satisfy the two 'b' condition.
5. From **q3**, we can either:
   * Read more **'b's** and stay in **q3**.
   * Read an **'a'** and go back to **q1**, repeating the process.

If at any time we see an 'a' **before two 'b's**, we move to **qR (trap state)**, rejecting the string

  
  
ii).All string containing not more than three a’s  
  
To design a **DFA** for **strings containing no more than three 'a's**, we define the states based on the number of 'a's encountered:

### **DFA States Explanation**

1. **q0 (Start, Accepting)** → No 'a' encountered yet.
2. **q1** → One 'a' encountered.
3. **q2** → Two 'a's encountered.
4. **q3** → Three 'a's encountered.
5. **q4 (Trap/Reject State)** → More than three 'a's (invalid).

**Transitions**:

* 'b' can be read at any state and remain in that state.
* Reading 'a' transitions to the next state.
* If more than three 'a's are encountered, transition to **q4 (Trap State)**.

### **DFA Transition Table**

| **State** | **Input 'a'** | **Input 'b'** | **Final** |
| --- | --- | --- | --- |
| **q0** | q1 | q0 | ✅ |
| **q1** | q2 | q1 | ✅ |
| **q2** | q3 | q2 | ✅ |
| **q3** | q4 | q3 | ✅ |
| **q4** | q4 | q4 | ❌ (Trap) |

### **Valid and Invalid Strings**

✅ **Accepted**:

* "bb" (no 'a's)
* "abbb" (one 'a')
* "aabb" (two 'a's)
* "aaabb" (three 'a's)

❌ **Rejected**:

* "aaaab" (four 'a's)
* "ababaa" (four 'a's)

This DFA strictly **rejects** any string with more than **three 'a's** and is designed in a **straight-line format** for clarity

### Explain procedure for converting NFA with € moves to without € with suitable example. **Procedure to Convert an NFA with ε-moves to an NFA without ε-moves**

The **ε-closure** (or **epsilon closure**) of a state is the set of states that can be reached using only ε-moves, including the state itself.

**Steps for Conversion**

1. **Compute ε-closures** for all states.
2. **Transform the transition function**:
   * For each transition **q →ε→ p**, include transitions from q to all states reachable through p's ε-closure.
   * For each transition **q →a→ p**, include transitions from ε-closure(q) to ε-closure(p).
3. **Modify the set of final states**:
   * If any state in an ε-closure contains a final state, the whole ε-closure is treated as final.
4. **Remove ε-moves** completely.

**Example**

**Given NFA with ε-moves**

| **State** | **Input 'a'** | **Input 'b'** | **ε-move** |
| --- | --- | --- | --- |
| **q0** | - | - | {q1} |
| **q1** | {q2} | - | {q3} |
| **q2** | - | {q2} | - |
| **q3** | {q4} | - | - |
| **q4** | - | {q4} | - |

Final State: **q4**

**Step 1: Compute ε-closures**

| **State** | **ε-Closure** |
| --- | --- |
| **q0** | {q0, q1, q3} |
| **q1** | {q1, q3} |
| **q2** | {q2} |
| **q3** | {q3} |
| **q4** | {q4} |

**Step 2: Update Transitions**

* q0 → ε → q1, q3 → So ε-closure(q0) = {q0, q1, q3}
* q1 → a → q2 → So q0 → a → q2
* q3 → a → q4 → So q0 → a → q4
* q2 → b → q2 → So q2 → b → q2
* q4 → b → q4 → So q4 → b → q4

**Step 3: Update Final States**

Since **q4** is a final state and **q0's ε-closure contains q3**, we check if q3 leads to q4. Since it does, **q0** becomes a final state.

**Final Converted NFA (Without ε-Moves)**

| **State** | **Input 'a'** | **Input 'b'** | **Final** |
| --- | --- | --- | --- |
| **q0** | {q2, q4} | - | ✅ |
| **q1** | {q2, q4} | - |  |
| **q2** | - | {q2} |  |
| **q3** | {q4} | - |  |
| **q4** | - | {q4} | ✅ |

This is now an **NFA without ε-moves** but still non-deterministic. It can be converted to **DFA using the subset construction method** if needed.

**Key Takeaways**

* Compute **ε-closure** for each state.
* Redirect transitions based on **ε-closure**.
* Update **final states** accordingly.
* Remove **ε-transitions** completely.

### This method ensures an equivalent NFA **without ε-moves** while preserving language acceptance. **Converting the Given NFA with Epsilon (ε) to DFA**

### **Step 1: Understanding the Given NFA**

#### **NFA States:**

* **A** (Initial State)
* **B** (ε-move from A)
* **C**
* **D**

#### **NFA Transitions:**

| **State** | **Input 'a'** | **Input 'b'** | **ε-move** |
| --- | --- | --- | --- |
| **A** | A | - | B |
| **B** | D | C | - |
| **C** | - | - | - |
| **D** | D | D | - |

#### **Epsilon Closure Calculation**

The **ε-closure** of a state includes that state and any other states reachable via ε-moves.

1. **ε-Closure(A) = {A, B}**
   * A has an ε-move to B, so include B.
2. **ε-Closure(B) = {B}**
   * No ε-move from B.
3. **ε-Closure(C) = {C}**
   * No ε-move from C.
4. **ε-Closure(D) = {D}**
   * No ε-move from D.

### **Step 2: Construct the DFA**

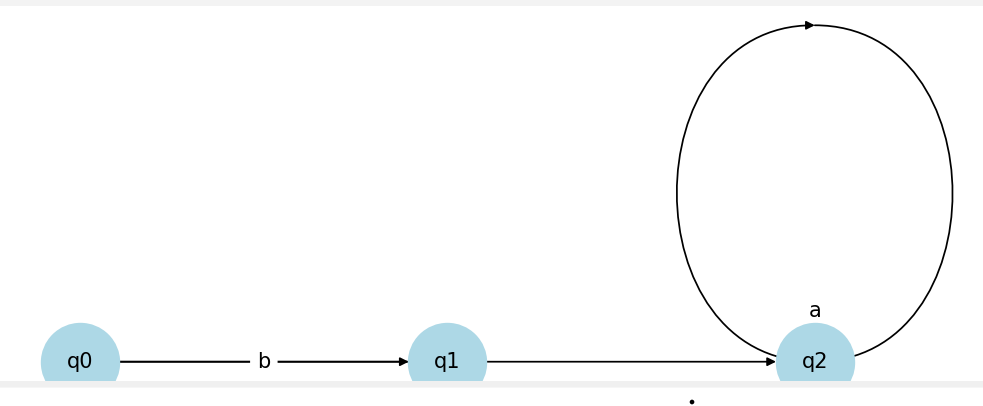
Now, we determine transitions based on the ε-closures.

#### **DFA States Derived from ε-Closures**

| **DFA State** | **Corresponding NFA States** |
| --- | --- |
| **{A, B}** | Start state (from ε-closure of A) |
| **{C}** | Reached from {A, B} on 'b' |
| **{D}** | Reached from {A, B} on 'a' |
| **{D}** | Self-loop on both 'a' and 'b' |

#### **DFA Transitions**

| **Current DFA State** | **Input 'a'** | **Input 'b'** |
| --- | --- | --- |
| **{A, B}** | {D} | {C} |
| **{C}** | - | - |
| **{D}** | {D} | {D} |



### **Step 4: Final DFA Summary**

* **DFA states**: {A, B}, {C}, {D}
* **DFA transitions**:
  + {A, B} → a → {D}
  + {A, B} → b → {C}
  + {D} → a → {D}
  + {D} → b → {D}