Problem Statement

Let **X = (Qx, ∑, δx, q0, Fx)** be an NDFA which accepts the language L(X). We have to design an equivalent DFA **Y = (Qy, ∑, δy, q0, Fy)** such that **L(Y) = L(X)**. The following procedure converts the NDFA to its equivalent DFA −

Algorithm

**Input** − An NDFA

**Output** − An equivalent DFA

**Step 1** − Create state table from the given NDFA.

**Step 2** − Create a blank state table under possible input alphabets for the equivalent DFA.

**Step 3** − Mark the start state of the DFA by q0 (Same as the NDFA).

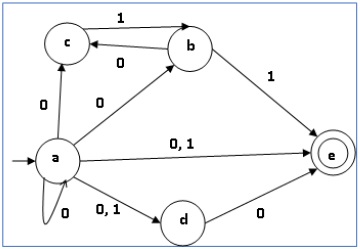
**Step 4** − Find out the combination of States {Q0, Q1,... , Qn} for each possible input alphabet.

**Step 5** − Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.

**Step 6** − The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

Example

Let us consider the NDFA shown in the figure below.

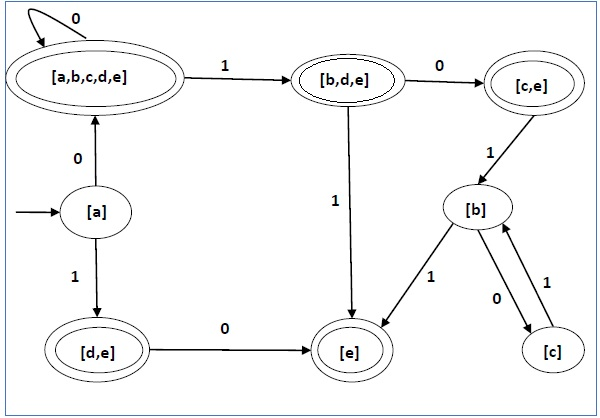


|  |  |  |
| --- | --- | --- |
| **Q** | **δ(q,0)** | **δ(q,1)** |
| a | {a,b,c,d,e} | {d,e} |
| b | {c} | {e} |
| c | ∅ | {b} |
| d | {e} | ∅ |
| e | ∅ | ∅ |

Using the above algorithm, we find its equivalent DFA. The state table of the DFA is shown in below.

|  |  |  |
| --- | --- | --- |
| **q** | **δ(q,0)** | **δ(q,1)** |
| [a] | [a,b,c,d,e] | [d,e] |
| [a,b,c,d,e] | [a,b,c,d,e] | [b,d,e] |
| [d,e] | [e] | ∅ |
| [b,d,e] | [c,e] | [e] |
| [e] | ∅ | ∅ |
| [c, e] | ∅ | [b] |
| [b] | [c] | [e] |
| [c] | ∅ | [b] |

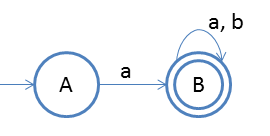
The state diagram of the DFA is as follows −



# **Set Substitution Method to convert NFA to DFA**

We convert NFA to DFA so that we can implement the state machine represented by DFA.  
The method is as follows:

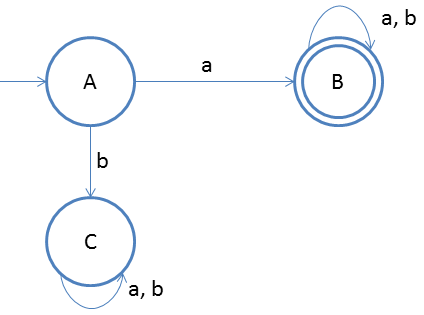
1. First find out the state transition table
2. Then take one state from the transtion table and then whenever you find out that output is not defined then put dead state there
3. Create new DFA

We will understand the whole method step by step:  
**Step 1:**  
Following is the NFA for strings starting with 'a'  
  
**Step 2:**  
Create state transition table

|  |  |  |
| --- | --- | --- |
| **State** | **a** | **b** |
| A | {B} | ϕ |
| B | {B} | {B} |

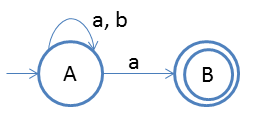
**Step 3:**  
Now create new transition table,  
Rule is whenever there is ϕ, we will include one more state as dead state.

|  |  |  |
| --- | --- | --- |
| **State** | **a** | **b** |
| 🡪 A | B | C |
| \*B | B | B |
| C | C | C |

***Note: Here C is a dead state***  
**Step 4:**  
Now create new DFA from the new transition table  


**Note:** Do not think that in the new table we will take the states which we have already, for example  
If on state A for input 'a' we go to A and B both in NFA then we will write as

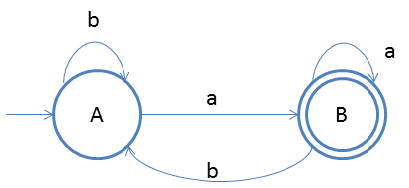
|  |  |  |
| --- | --- | --- |
| **State** | **a** | **b** |
| A | AB | D |
| AB | AB | B |
| D | D | D |

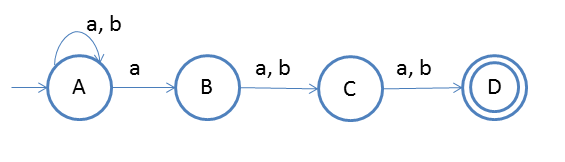
If you did not get the point I have just mentioned then follow the below example, you will understand  
**Step 1:**  
Following is the NFA for strings ending with 'a'  
  
**Step 2:**  
Create state transition table

|  |  |  |
| --- | --- | --- |
| **State** | **a** | **b** |
| 🡪A | {A,B} | {A} |
| \*B | ϕ | ϕ |

**Step 3:**  
Now create new transition table,  
Rule is whenever there is ϕ, we will include one more state as dead state.

|  |  |  |
| --- | --- | --- |
| **State** | **a** | **b** |
| 🡪A | [AB] | [A] |
| \*[AB] | [AB] | [A] |

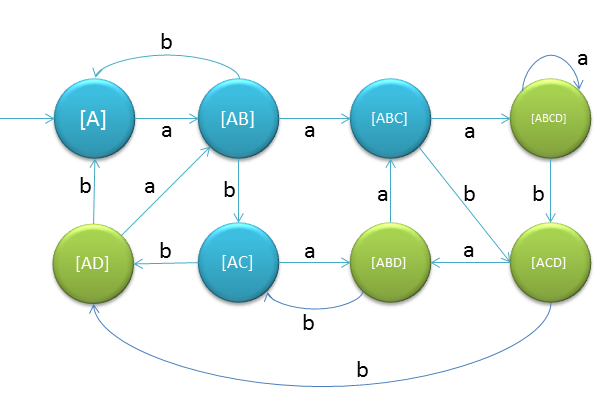
**Note: 1***Here State D goes to D on both inputs so we have not written it.*  
**Note: 2***Here State [AB] is denoted by state B and [A] is denoted by state A.*  
  
**Step 4:**  
Now create new DFA from the new transition table  
  
***Note: If there are n states in NFA then there could be 2n states in DFA after conversion.***  
***Note: Final state(s) of DFA will be the one which includes final state of NFA***

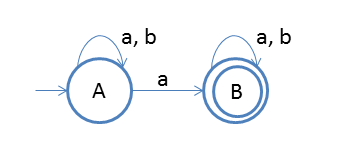
The proof of the above statement is explained by the below example:  
**Step 1:**  
Following is the NFA for strings having 3rd symbol 'a' from r.h.s.  
  
**Step 2:**  
Create state transiton table

|  |  |  |
| --- | --- | --- |
| **State** | **a** | **b** |
| A | {AB} | {A} |
| B | {C} | {C} |
| C | {D} | {D} |
| D\* | ϕ | ϕ |

**Step 3:**  
Now create new transition table,  
Rule is whenever there is ϕ, we will include one more state as dead state.

|  |  |  |
| --- | --- | --- |
| **State** | **a** | **b** |
| [A] | [AB] | [A] |
| [AB] | [ABC] | [AC] |
| [AC] | [ABD] | [AD] |
| [AD] | [AB] | [A] |
| [ABC] | [ABCD] | [ACD] |
| [ABD] | [ABC] | [AC] |
| [ACD] | [ABD] | [AD] |
| [ABCD] | [ABCD] | [ACD] |

**List of final states**  
[AD], [ABD], [ACD] and [ABCD] because they include D, final state of NFA.  
**Step 4:**  
Now create new DFA from the new transition table  




Transition table for NFA:

|  |  |  |
| --- | --- | --- |
|  | a | b |
| 🡪A | {A, B} | {A} |
| \*B | {B} | {B} |

Transition table for DFA:

|  |  |  |
| --- | --- | --- |
|  | a | b |
| * A | [AB] | [A] |
| \*[AB] | [AB] | [AB] |

Practice Problem:

String starts with ab

String ends with ab

String does contain with ab