Theory of Computing:

3. Finite Automata : NFA

Professor Imed Bouchrika
National Higher School of Artificial Intelligence
imed.bouchrika@ensia.edu.dz

Outline :

- Revision : DFA
- Non-Determinism
- Constructing NFA
- Converting NFA to DFA
- Minimization Algorithms
- Software and Tools

Finite Automata

- **Automaton :**
  - A machine that can do things or actions on its own.
  - Machine takes input, moves from one to another state and produces an output (Decision or computed)
  - Machines can be abstract, mechanical or electrical or even quantum..
  - Machines have different capabilities in terms of memory, processing speed....

Finite Automata

- **Automaton :**
  - Mechanical Vending Machine : (No electronics, no raspberry PI or arduino or high-level programming language).
    - Assume a single product
    - Price of the product is **30DA**
    - Accepted Coins are : **5DA**, **10DA** and **20DA**

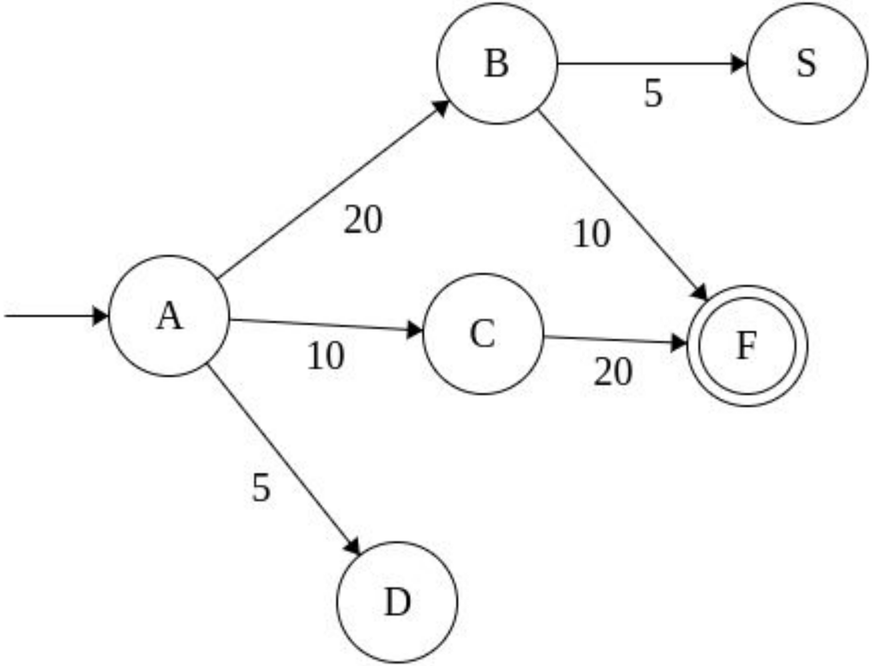
Finite Automata

- **Automaton :**
  - Mechanical Vending Machine : (No electronics, no raspberry PI or arduino or programming languag
    - Assume a single product
    - Price of the product is **30DA**
    - Accepted Coins are : **5DA**, **10**

What's the alphabet ?

What's the language accepted ?

Finite Automata



e : (No electronics, no raspberry PI or

guage

t

DA

A, 10

What's the alphabet ?

What's the language accepted ?

Finite Automata

- **Automaton :**
  - Mechanical Vending Machine : (No electronics, no raspberry PI or arduino or programming languag
    - Assume a single product
    - Price of the product is **30DA**
    - Accepted Coins are : **5DA**, **10**

Can we automate the process to return the change ?

We insert 40DA for a price of 30DA, it would give us back 10DA ?

Finite Automata

- **Automat**
  - Me ctronics, no raspberry PI or ar
    -
    -
    -

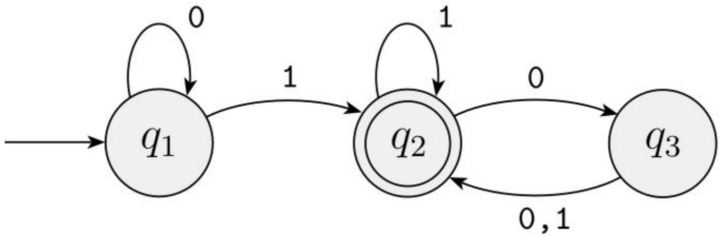
Not yet, the machines being built at this stage are decider (Yes/No Problem Solver) not Transducers

we automate the process to return the change ?

e insert 30DA for a price of 25DA, it would give us back 5DA ?

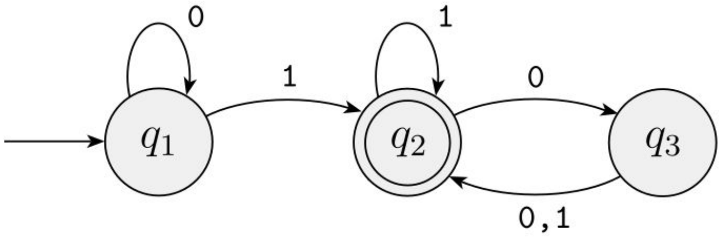
Finite Automata

- **State Diagram** :
  - is the visual representation of finite automata as shown below :



  - **States :** Circles
  - **Transitions :** Arcs/Arrows/Directed Edges

Finite Automata



- **States** :
  - **Start State (q1)**:
    - There must **be one** start state with the inbound arrow
  - **Accepting State (Final or Terminating state) (q2)**
    - Drawn as double-line circle
    - There can be multiple or even none,
    - If the string/word ends at this state, the machine would decide an ***accept*** for the word.
  - **Non-Accepting State (q3)**
    - Drawn as single-line circle
    - There can be many or none
    - If the machine reaches the end of the word at this state, the word is **rejected.**

Finite Automata

- **Deterministic Finite Automaton** :
  - Deterministic = Events can be determined precisely
  - Finite = Finite and small amount of space used
  - Automaton = Computing machine
- **Deterministic ?**
  - From a given state there is only one transition for each symbol + Each transition must have non-empty string symbol.
  - The q_1 state : There are two transitions for the same symbol 1. Therefore, this is not a deterministic finite automaton

Finite Automata

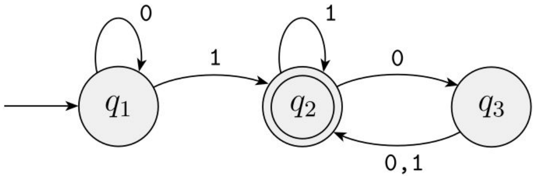
- **Formal Definition of Deterministic Finite Automaton** :
  - Usually Automated are denoted by the letter ***M***
  - It is a 5-tuple defined as $(Q, \Sigma, \delta, q_0, F)$

    Such that :

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Finite Automata

- **Transition Table** :
  - Finite Automata can be additionally represented by a Transition Table showing all possible transitions at different configuration:
    - For automaton shown M = (Q, Σ, δ, q1 , F) such that
      - Q = { q1,q2,q3}
      - F= { q2 }
      - Σ = { 0 , 1 }



Current Configuration

	0	1
$\to q_1$	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

Finite Automata

- **Representation of Automata Machines** :
    - State Diagram (Graphical)
    - Transition Table
    - ?

Finite Automata

- **Regular Language**
  - Let **M** a finite Automata:
  - We say that M recognizes language L if L = { w | M accepts/recognizes w}

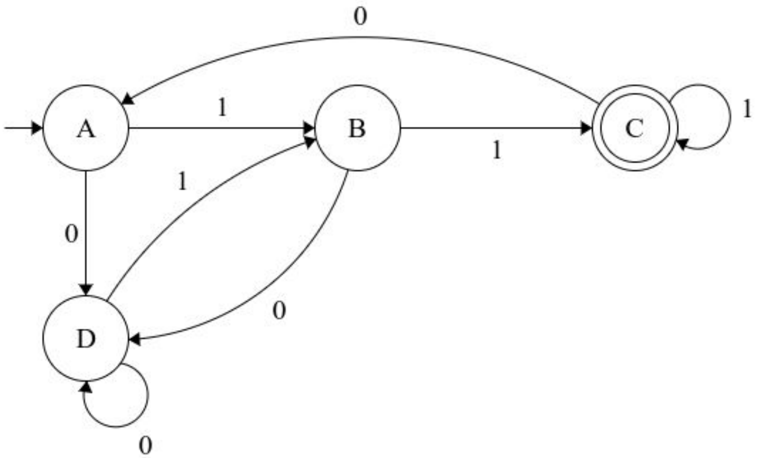
*language is called a **regular language** if some finite automaton recognizes it*

Finite Automata

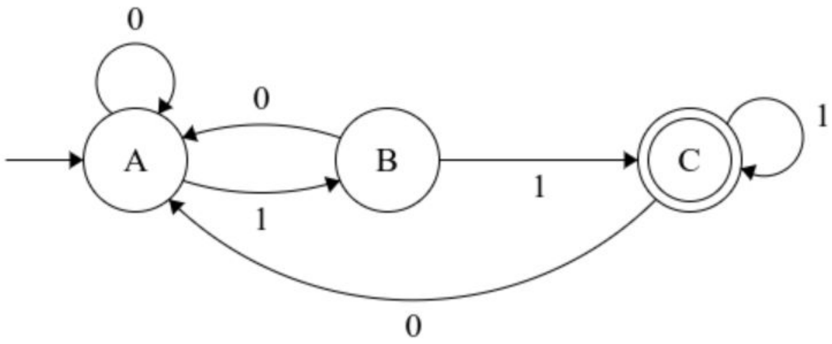
- **Designing the Deterministic Finite Automata**
  - Start always with **obvious case** and make sure it is accepted.
  - Move on to bigger words and each time:
    - Test if the language words are accepted
    - Test also that non-language words are rejected.
  - At each state, always say “what if” i have a symbol:
    - You can expect all alphabet symbols at each state
      - Create the Trap/Dead states **at the end**.

Finite Automata

- **Designing Deterministic Finite Automata** :
  - Alphabet Σ = { 0 , 1 }
  - Design M which recognizes L = {w ∈ {0, 1}* | w ends with 11 }



VS



Finite Automata

- **Example for Minimizing DFA :**
    -

Finite Automata

- **Example for Minimizing DFA** :
    -

Finite Automata

- **Question from last lecture :**
  - How to decide if two DFAs represent the same language ?

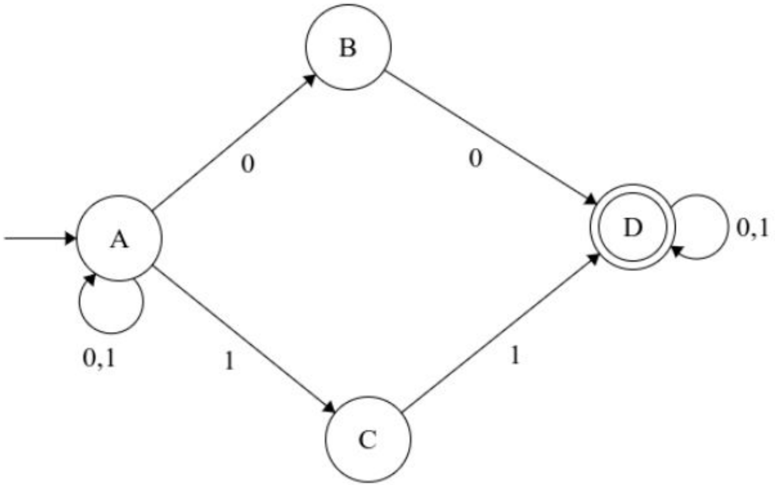
Nondeterministic Finite Automata

- **Nondeterministic Finite Automata (NFA)**
  - For each state there can be zero, one, two, or more transitions corresponding to a particular symbol.
  - Why :
    - *Because it is easier , compared to the conditions imposed by DFA*
      - *To construct*
      - *To understand*
    - *To simulate scenarios in real life (but computers are deterministic machines)*

Nondeterministic Finite Automata

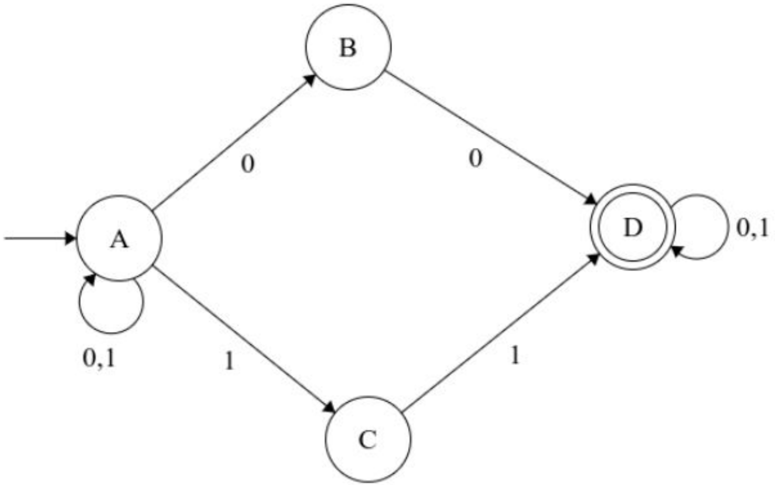
- **DFA vs NFA :**
  - Transition with the same label from a state
  - *State A :*
    - *Two outgoing arrows with 1*
    - *Two outgoing arrows with 0*

What language does the NFA recognize ?



Nondeterministic Finite Automata

- **DFA vs NFA :**
  - Transition with the same label from a state
  - *State A :*
    - *Two outgoing arrows with 1*
    - *Two outgoing arrows with 0*

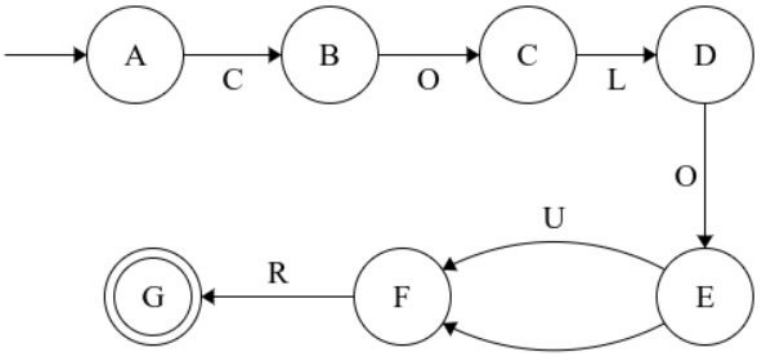


What language does the NFA recognize ?

{ w | w in $\{0,1\}^*$ and w contains **00** or **11** }

Nondeterministic Finite Automata

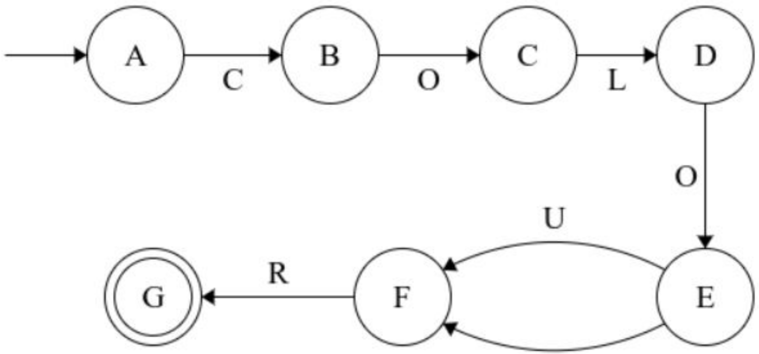
- **DFA vs NFA :**
  - Transition with the empty string
    - *State E can* ***Either****:*
      - *Move to* ***F*** *on reading symbol* ***U***
      - *Move to F without reading anything (Empty String ε)*



What language does the NFA recognize ?

Nondeterministic Finite Automata

- **DFA vs NFA :**
  - Transition with the empty string
    - *State E can* ***Either****:*
      - *Move to* ***F*** *on reading symbol* ***U***
      - *Move to F without reading anything (Empty String ε)*



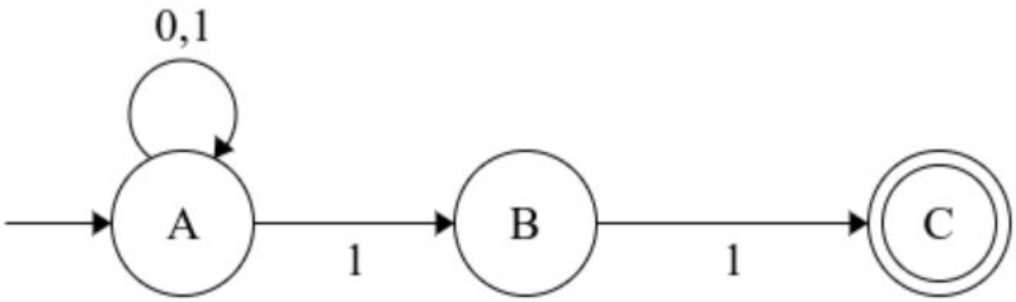
What language does the NFA recognize ?

L={ colour, color}

Nondeterministic Finite Automata

- **DFA vs NFA :**
  - Trap State for *Missing Transitions* : is not obligatory
    - *State **B :** is not providing the transition **for symbol 0***

What language does the NFA recognize ?

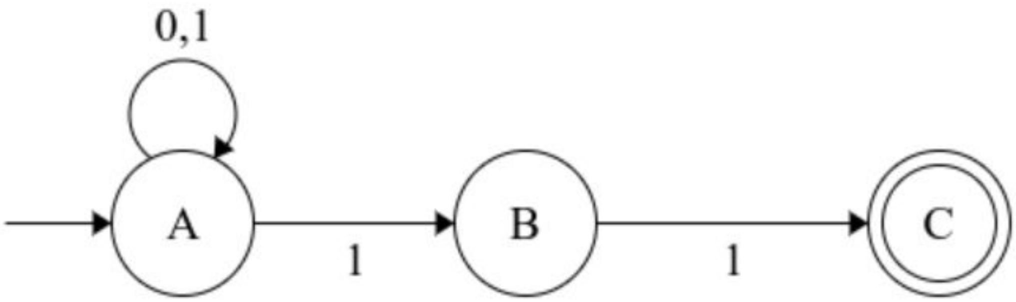


Nondeterministic Finite Automata

- **DFA vs NFA :**
    - Trap State for *Missing Transitions* : is not obligatory
        - *State **B :** is not providing the transition **for symbol 0***

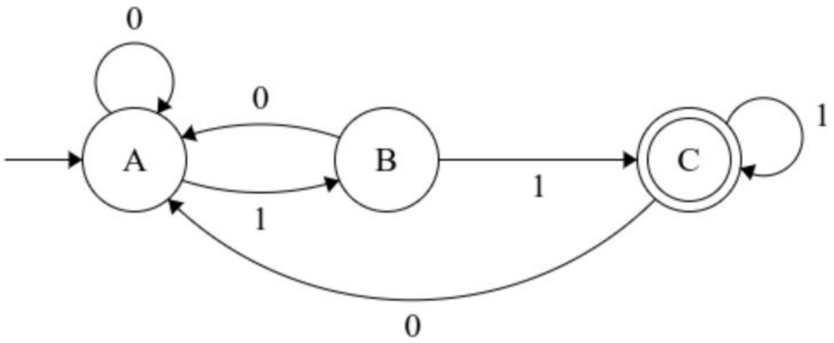
What language does the NFA recognize ?

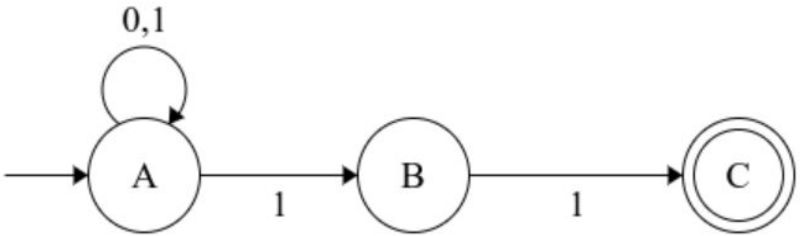
L={w | w ends with **11** }



Nondeterministic Finite Automata

- **Difficulty to construct and interpret : DFA vs NFA :**
  - Language : L={w | w ends with **11** }





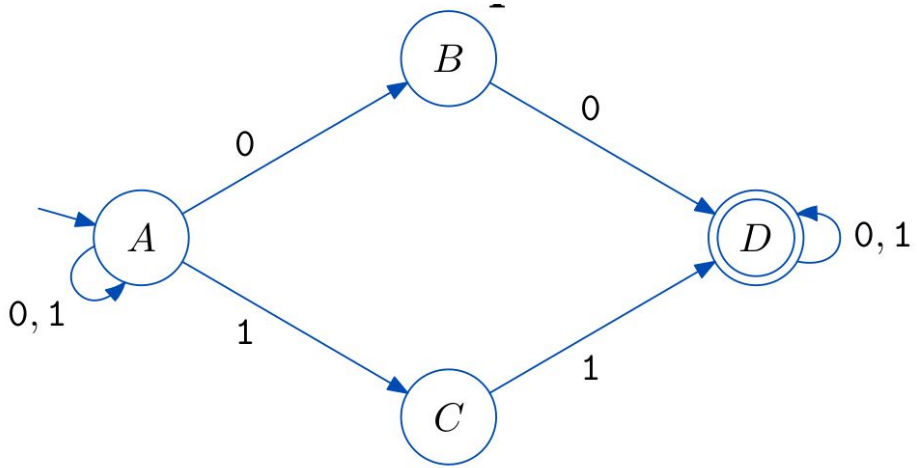
Nondeterministic Finite Automata

- DFA vs NFA :

	DFA	NFA
Transition with the same label from a state	Strictly one	Multiple
Transition with the empty string	Does not exist	It exists
Trap State for missing transitions	Obligatory	Optional

Nondeterministic Finite Automata

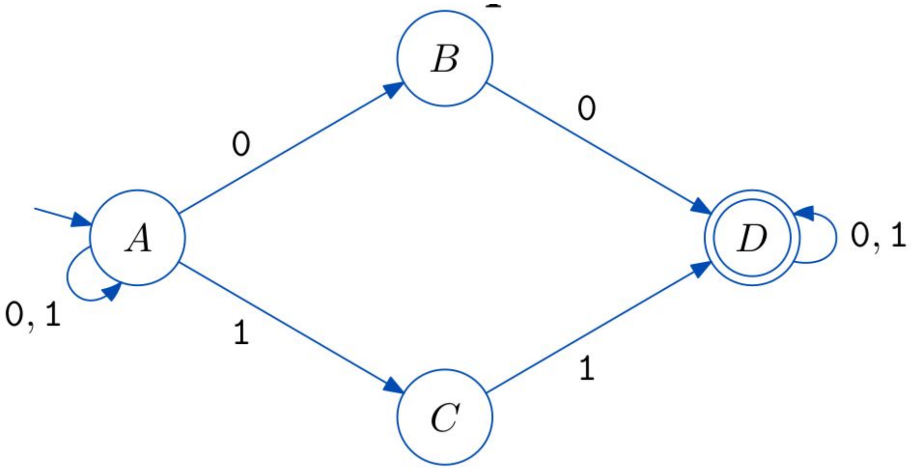
- **How to accept a word using NFA :**
  - Are the following words accepted ?
    - **10**
    - **00011**
    - **10101**
    - **10011**



Nondeterministic Finite Automata

- **How to accept a word using NFA :**
  - Are the following words accepted ?
    - **10 *not accepted***
    - **00011 *Accepted***
    - **10101 *not accepted***
    - **10011 *Accepted***
  - The language containing

    A substring of 00 or 11



Nondeterministic Finite Automata

- **How to accept a word using NFA :**
  - An NFA accepts the input string if there exists some choice of transitions that leads to ending in an accept state.
  - Thus,
    - **One** accepting branch is enough for the overall NFA to accept,
    - For rejection, **every** branch must reject word

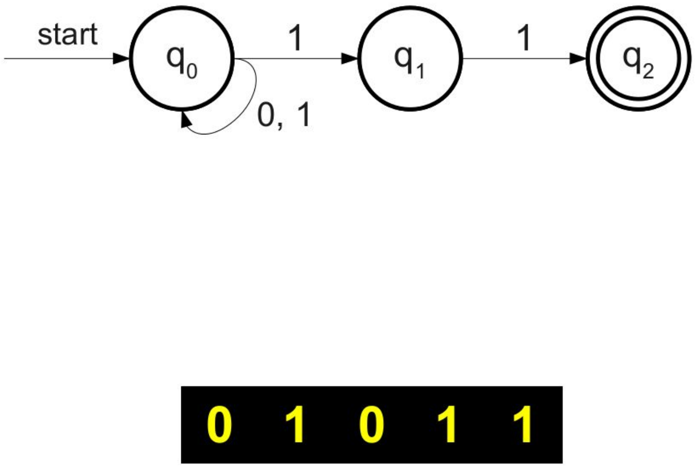
Nondeterministic Finite Automata

- **How to accept a word using NFA :**
  - An NFA accepts the input string if there exists some choice of transitions that leads to ending in an accept state.
  - Thus,
    - **One** accepting branch is enough for the overall NFA to accept,
    - For rejection, **every** branch must reject word

EXTREMELY IMPORTANT

Nondeterministic Finite Automata

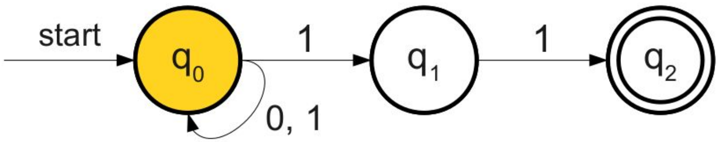
- **Simulating reading a word using NFA :**
  - L = { w | w ends with 11 }
  - Let's see the word : 11011



https://web.stanford.edu/class/archive/cs/cs103/cs103.1134/

Nondeterministic Finite Automata

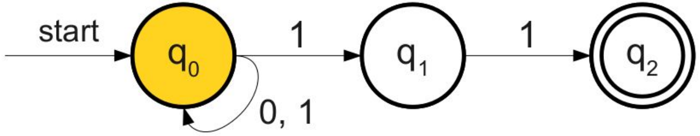
- **Simulating reading a word using NFA :**
  - L = { w | w ends with 11 }
  - Let's see the word : 11011
    - **Set the initial state as the start State**



https://web.stanford.edu/class/archive/cs/cs103/cs103.1134/

Nondeterministic Finite Automata

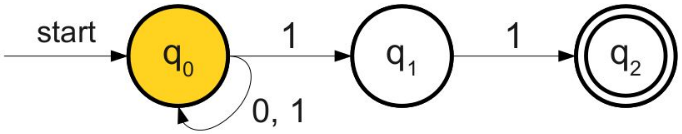
- **Simulating reading a word using NFA :**
  - L = { w | w ends with 11 }
  - Let's see the word : 11011
    - **Reading the first symbol From the word**
    - **Checking the relevant Transition**
    - **→ would stay at q0**

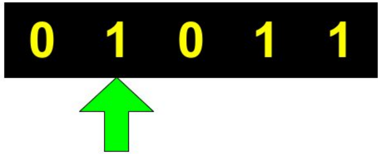


https://web.stanford.edu/class/archive/cs/cs103/cs103.1134/

Nondeterministic Finite Automata

- **Simulating reading a word using NFA :**
  - L = { w | w ends with 11 }
  - Let's see the word : 11011
    - **Read Symbol 1**
    - **Check the relevant transition**

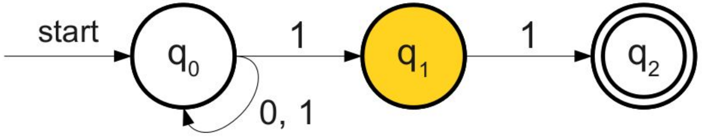


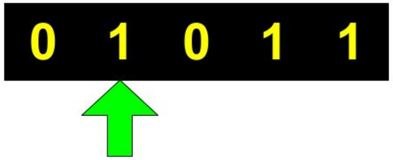


https://web.stanford.edu/class/archive/cs/cs103/cs103.1134/

Nondeterministic Finite Automata

- **Simulating reading a word using NFA :**
    - L = { w | w ends with 11 }
    - Let's see the word : 11011
        - → **Move to state q1**

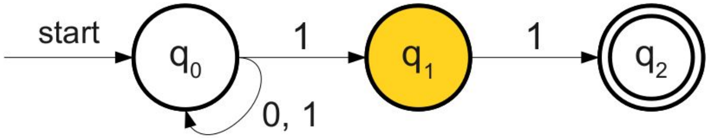


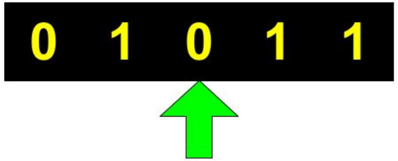


https://web.stanford.edu/class/archive/cs/cs103/cs103.1134/

Nondeterministic Finite Automata

- **Simulating reading a word using NFA :**
    - L = { w | w ends with 11 }
    - Let's see the word : 11011
        - **At state q1**
        - **Read Symbol 0**
        - **Check Relevant Transition**

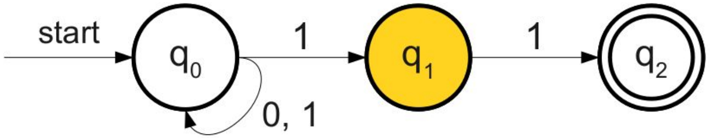


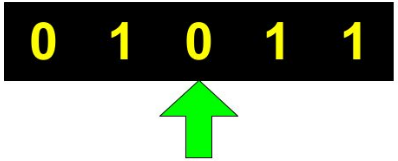


https://web.stanford.edu/class/archive/cs/cs103/cs103.1134/

Nondeterministic Finite Automata

- **Simulating reading a word using NFA :**
  - L = { w | w ends with 11 }
  - Let's see the word : 11011
    - **There is no transition,**
      - **What does it mean ? Rejected ?**
      - **What we do ?**

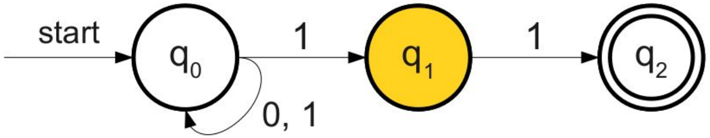


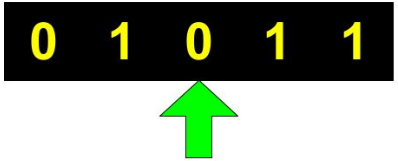


https://web.stanford.edu/class/archive/cs/cs103/cs103.1134/

Nondeterministic Finite Automata

- **Simulating reading a word using NFA :**
  - L = { w | w ends with 11 }
  - Let's see the word : 11011
    - **There is no transition,**
      - **The taken path or branch dies**
      - **Not necessary the word is rejected**
    - **What we do**
      - **We seek a different branch/path**

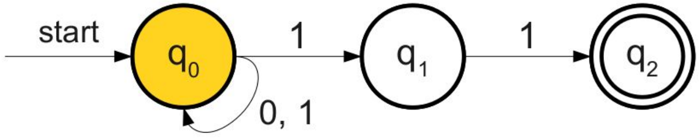




https://web.stanford.edu/class/archive/cs/cs103/cs103.1134/

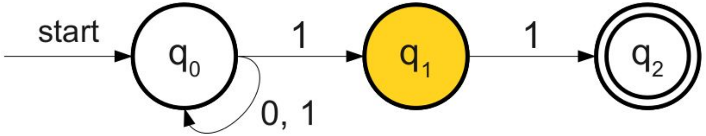
Nondeterministic Finite Automata

- **Simulating reading a word using NFA :**
  - L = { w | w ends with 11 }
  - Let's see the word : 11011
    - **We start again (At which level?)**
    - ***Read first symbol 0 : move to q0***
    - ***Read second symbol 1 : move to q0***
    - ***Read third symbol 0 : move to q0***

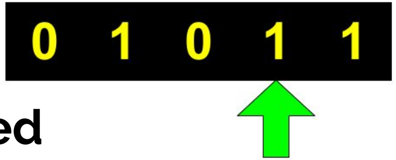


https://web.stanford.edu/class/archive/cs/cs103/cs103.1134/

Nondeterministic Finite Automata



- **Simulating reading a word using NFA :**
  - L = { w | w ends with 11 }
  - Let's see the word : 11011
    - **Read fourth symbol 1 : move to q1**
    - **Read fourth symbol 1 : move to q2**
    - **End of the word at q2:**
      - **q2 is accept state → Word is accepted**
    - **Shall we consider other branches ?**



Nondeterministic Finite Automata

- **Simulating reading a word using NFA :**
  - An NFA accepts the input string if there exists some choice of transitions that leads to ending in an accept state.
  - Thus,
    - **One** accepting branch is enough for the overall NFA to accept,
    - For rejection, **every** branch must reject word

EXTREMELY IMPORTANT

Nondeterministic Finite Automata

- **Simulating reading a word using NFA :**
  - NFA does the search for the relevant states and backtracks in case the selected path dies.
  - There are three strategies for the search for a given input :
    - Tree Computation
    - Parallel Computation
    - Guessing

Nondeterministic Finite Automata

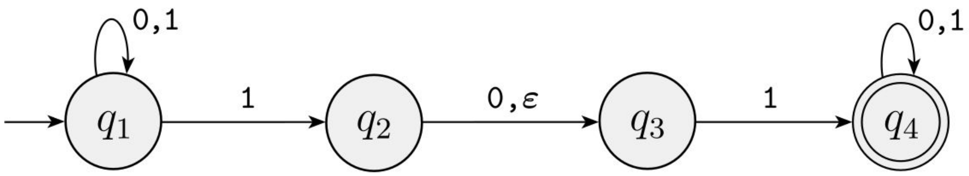
- **Simulating reading a word using NFA :**
  - NFA does the search for the relevant states and backtracks in case the selected path dies.
  - The transitions that leads to ending in an accept state.
  - Thus,
    - One accepting branch is enough for the overall NFA o accept,
    - For rejection, every branch must reject word

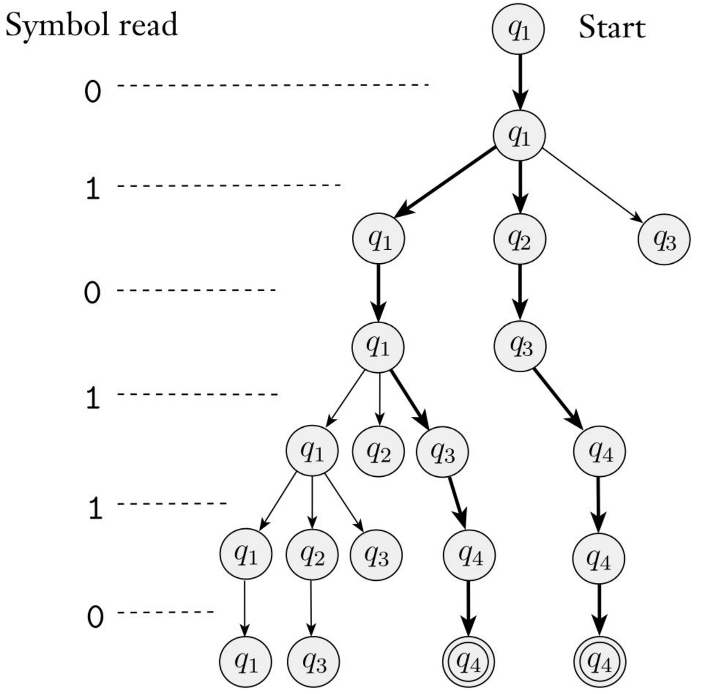
Nondeterministic Finite Automata

- **Branching : Computation of Trees**
  - The tree is constructed with the start state as the root.
  - A child is created
    - **for each transition** from the parent state
    - This includes self-transitions
  - Tree does not allow linking back children to (super-)parents in case states go back to each other, instead, new tree nodes are created.

Nondeterministic Finite Automata

- **Branching : Computation of Trees**
  - For the word: **010110**
  - From the tree, there are two possible paths



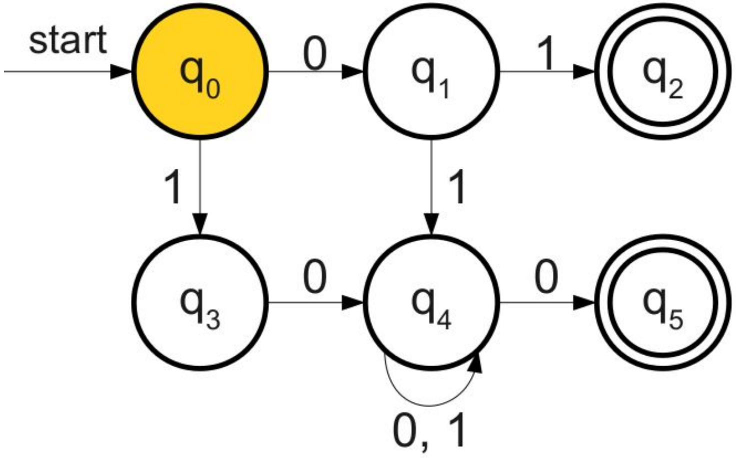


Nondeterministic Finite Automata

- **Branching : Execution in Parallel**
  - At each state with multiple choices for the same input:
    - We **fork** a new process or thread for each **choice**
    - Processes or threads run in parallel at the same time to search.
    - No backtracking is necessary

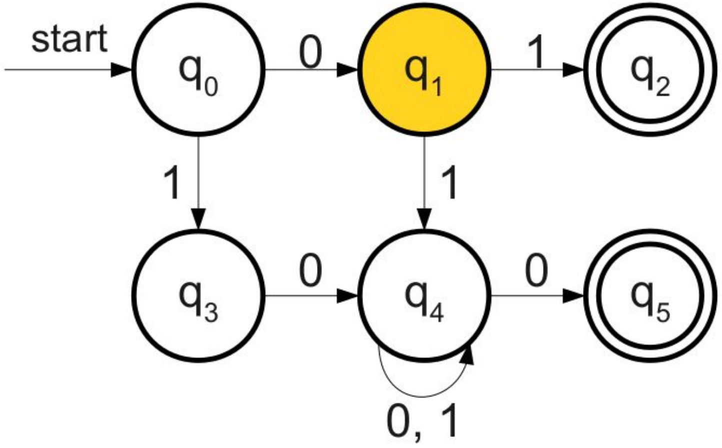
Nondeterministic Finite Automata

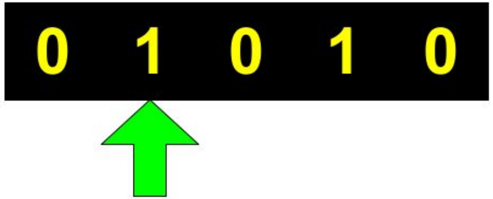
- **Branching : Execution in Parallel**
  - Setting initial states q_0
  - Reading Symbol 0



Nondeterministic Finite Automata

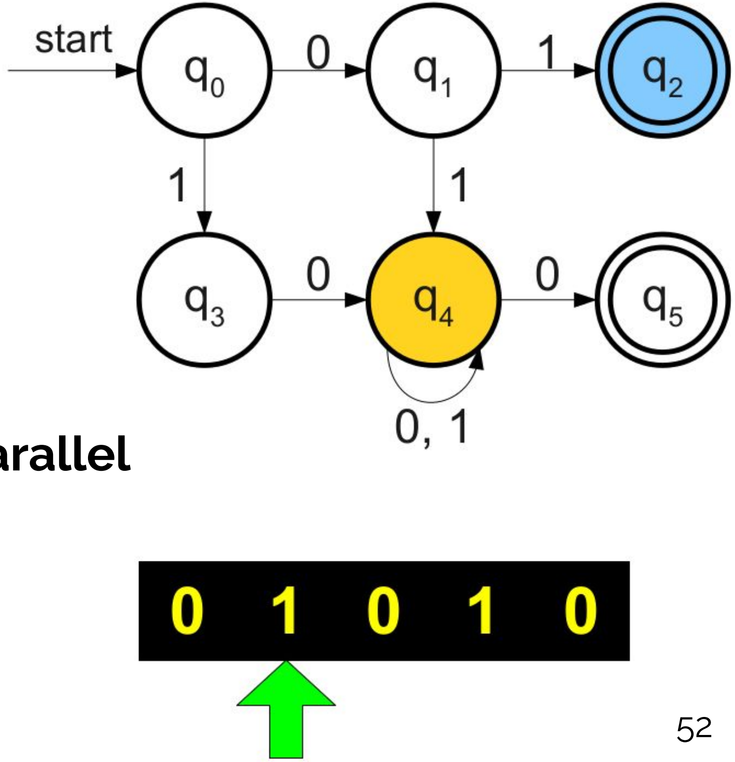
- **Branching : Execution in Parallel**
  - Moving to q1 when reading 0
  - Reading Symbol 1
  - We have **two choices**
    - **Move to q2**
    - **Move to q4**





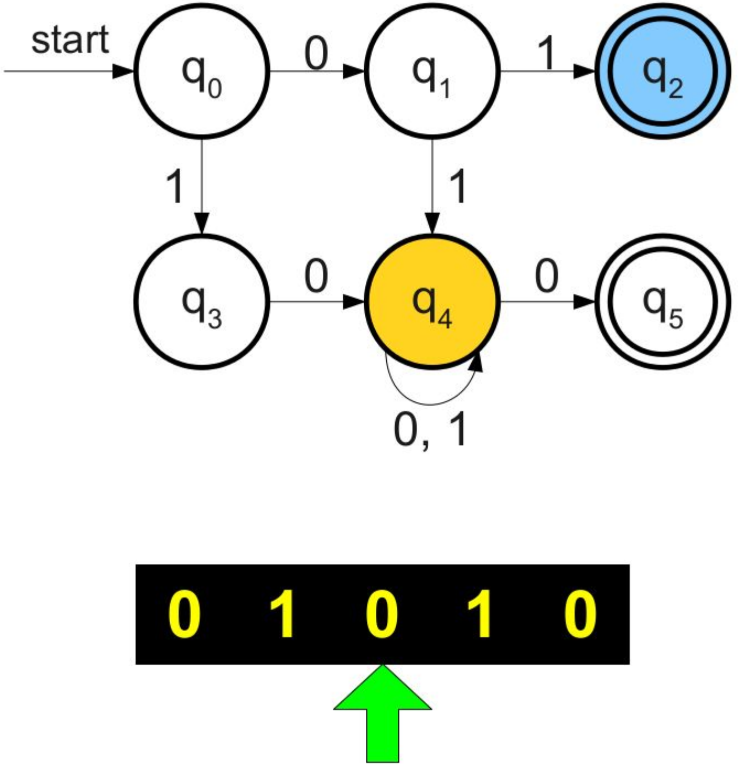
Nondeterministic Finite Automata

- **Branching : Execution in Parallel**
  - For the **two** choices
    - **Move to q2**
    - **Move to q4**
  - Two processes are created to run in **parallel**



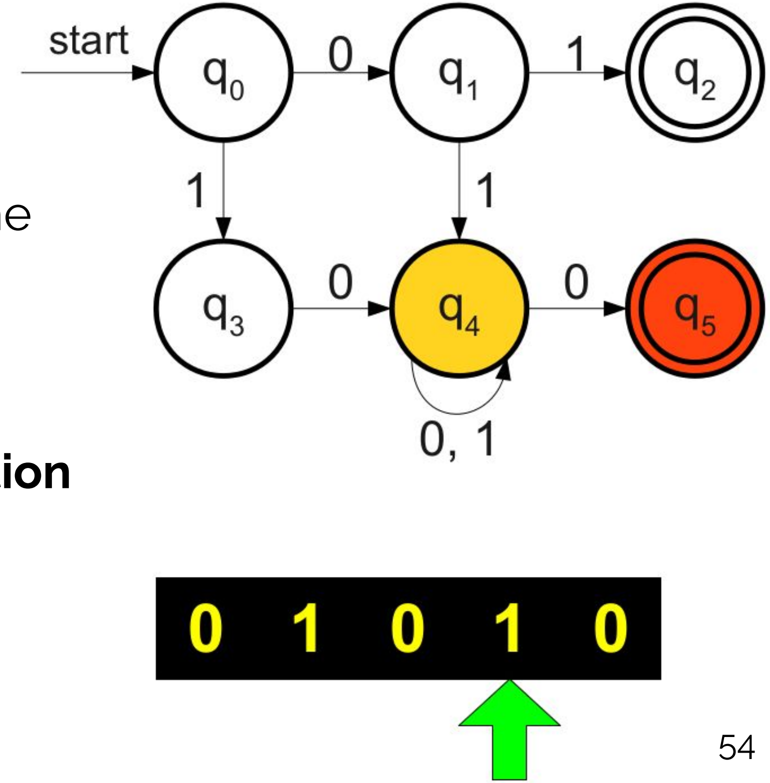
Nondeterministic Finite Automata

- **Branching : Execution in Parallel**
  - Reading Symbol 0
    - **Blue thread** dies
    - **Yellow thread : Two choices**
      - **Fork x 2 threads**
        - **Move to q5**
        - **Move to q4**



Nondeterministic Finite Automata

- **Branching : Execution in Parallel**
  - Two threads running at the same time
    - **Red and Yellow**
    - **When reading input 1 next**
      - **Red thread dies : no transition**
      - **Yellow thread: ?**



Nondeterministic Finite Automata

- **Branching : Guessing**
  - This is not to mean Random guessing
  - The machine or system has some “heuristics” or business logic for the machine to guess the correct choice/path to take.
  - This depends on the context and rules being treated.
  - Some literature may refer to the term supermagic guessing?

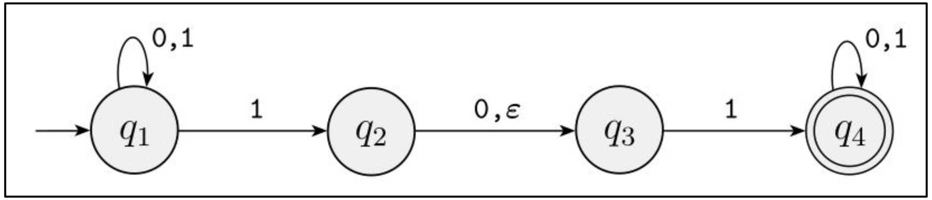
Nondeterministic Finite Automata

- **Formal Definition for NFA**
  - A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:
    - Q is a finite set of states,
    - Σ is a finite alphabet,
    - $\delta : Q \times \Sigma \cup \varepsilon \rightarrow 2^Q$ is the transition function
    - $q_0 \in Q$ is the start state
    - $F \subseteq Q$ is the set of accept states.

Nondeterministic Finite Automata

- **Transition Table for a formal definition**

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as



	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

4. q_1 is the start state, and
5. $F = \{q_4\}$.

Nondeterministic Finite Automata

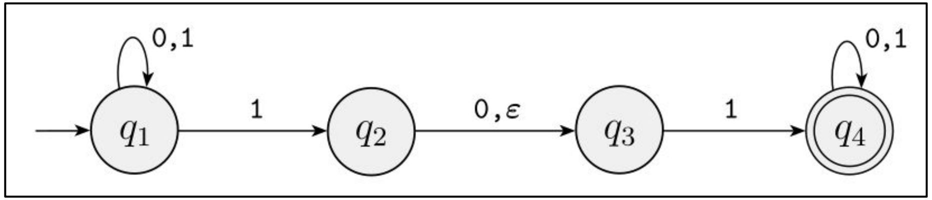
- **Transition Table for a formal definition**

Empty Symbol is considered as input

1. $Q = \{q_1, q_2$
2. $\Sigma = \{0,1\}$,
3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

4. q_1 is the start state, and
5. $F = \{q_4\}$.



Nondeterministic Finite Automata

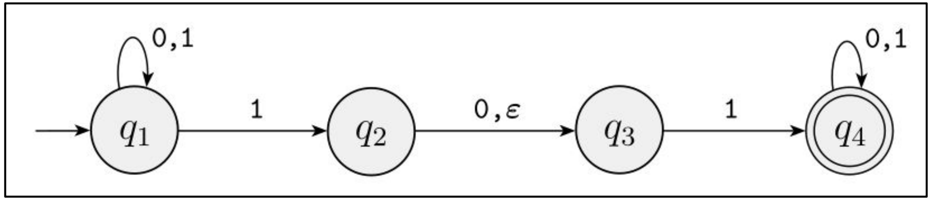
- **Transition Table for a formal definition**

For the missing transitions : no transition for 1

1. $Q = \{q_1, q_2$
2. $\Sigma = \{0,1\}$,
3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

4. q_1 is the start state, and
5. $F = \{q_4\}$.



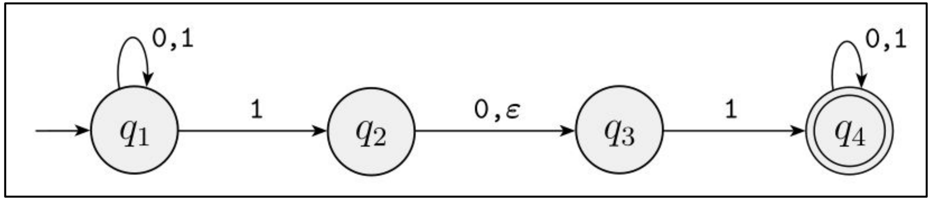
Nondeterministic Finite Automata

- **Transition Table for a formal definition**

For multiple choices on the same input from the same state

1. $Q = \{q_1,$
2. $\Sigma = \{0,1\}$,
3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$,



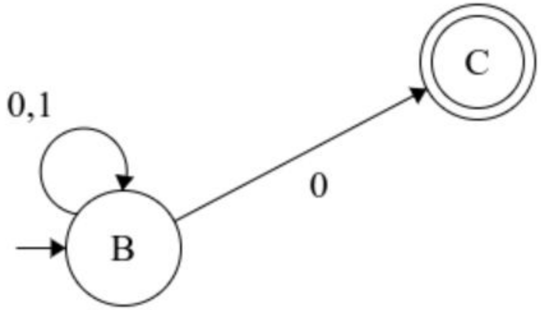
4. q_1 is the start state, and
5. $F = \{q_4\}$.

Constructing NFA

- **Example 1 :**
  - Design the NFA for the following Language L over Σ={1,0}
    - L= { w | w ends with 0 **or** contains only 1s }

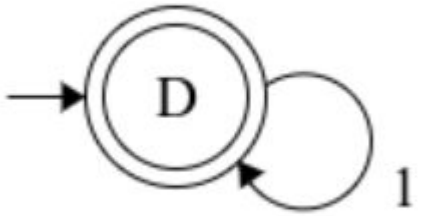
Constructing NFA

- **Example 1 :**
  - Design the NFA for the following Language L over **Σ**={1,0}
    - L= { w | w ends with 0 **or** contains only 1s }
      - Let's design the first part : ends with 0



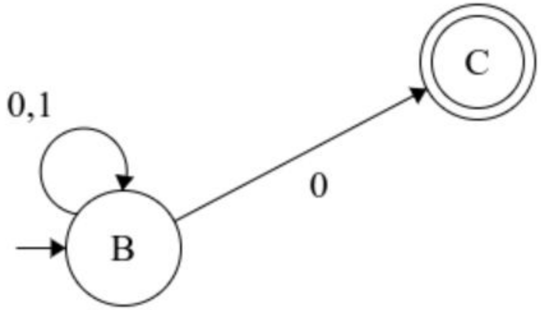
Constructing NFA

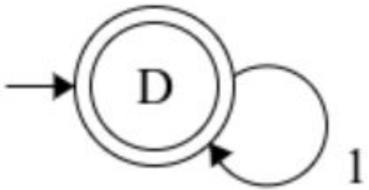
- **Example 1 :**
  - Design the NFA for the following Language L over Σ={1,0}
    - L= { w | w ends with 0 **or** contains only 1s }
      - Second part : contains only 1s



Constructing NFA

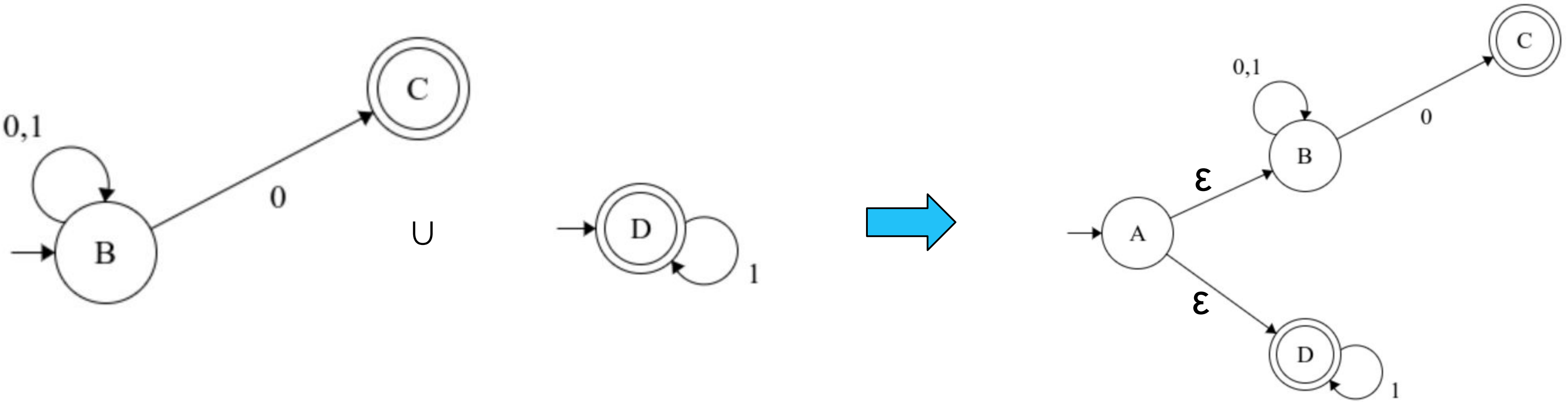
- **Example 1 :**
  - Design the NFA for the following Language L over **Σ**={1,0}
    - L= { w | w ends with 0 **or** contains only 1s }
      - We need to combine them : as a union → **use the epsilon**





Constructing NFA

- **Example 1 :**
  - Design the NFA for the following Language L over **Σ**={1,0}
    - L= { w | w ends with 0 **or** contains only 1s }
      - We need to combine them : as a union → **use the epsilon**



Converting NFA to DFA

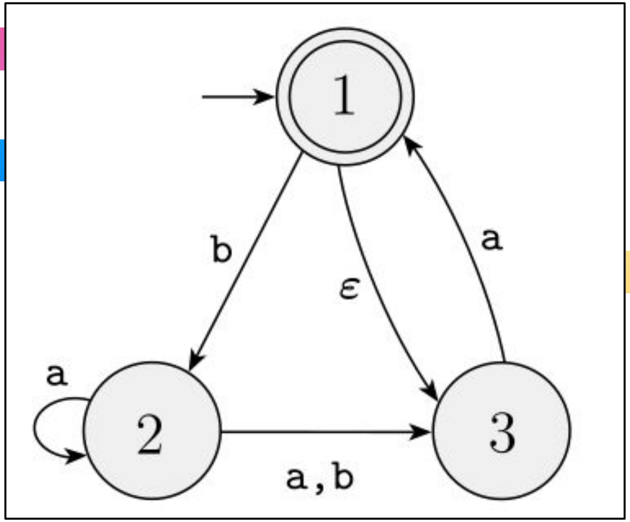
- **Equivalence Theories**
  - Every DFA is NFA
  - Inversely :
    - Every NFA is not a DFA
    - But, there is an **equivalent** DFA for each NFA

      (See Sipser Book for the proof : Theorem : 1.39

Converting NFA to DFA

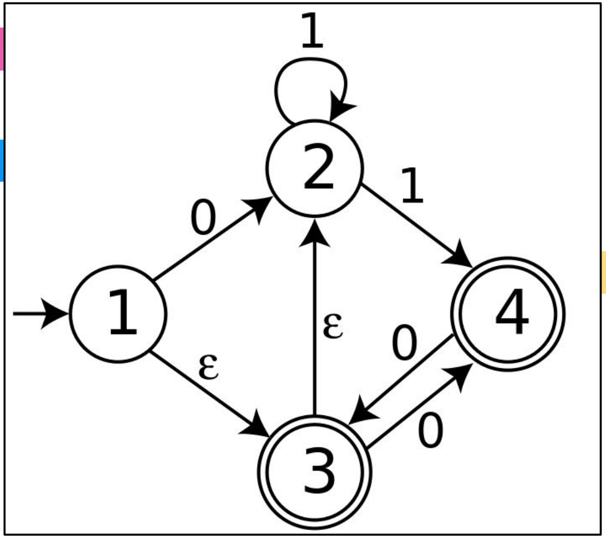
- **Algorithm to convert NFA to DFA :**
  - It is called conversion by subset construction :
  - States are represented as sets from the power set : 2^Q
    1. Determine the initial Start State
       - *It is the set containing the original start state union all other states reached from the original state by ε (directly or indirectly)*
    2. Determine the Accept States
       - *Any State set containing at least an original accept state*
    3. For each possible state created from 2^Q, find the possible transitions
       - *If there is missing transition, create a dead state*
    4. Draw the state diagram
    5. Remove any state without **incoming** transitions
  - Another Strategy : **Use the Transition Tables to facilitate the conversion**

Converting NFA to DFA



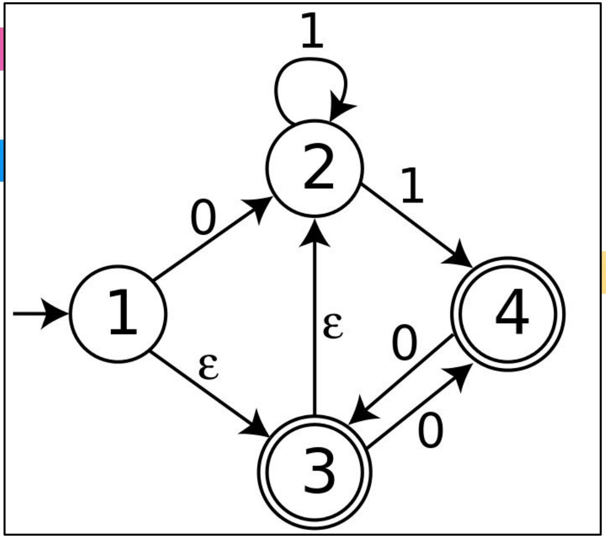
- **Algorithm to convert NFA to DFA :**
  - What's the start state for the following :
    - *It is the set containing the original start state union all other states reached from the original state by* **ε** *(directly or indirectly)*
      - 1
      - 3
        - {1,3}

Converting NFA to DFA



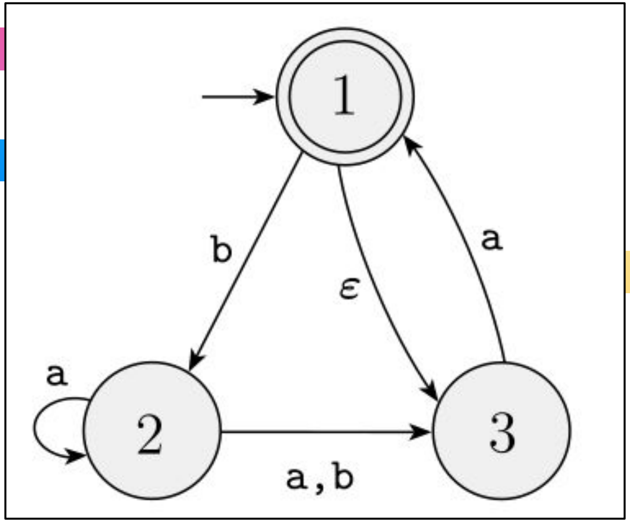
- **Algorithm to convert NFA to DFA :**
  - What's the start state for the following :
    - *It is the set containing the original start state union all other states reached from the original state by* **ε** *(directly or indirectly)*

Converting NFA to DFA



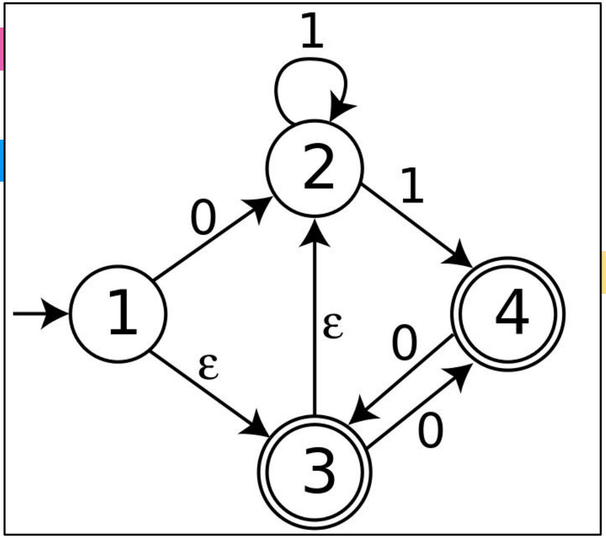
- **Algorithm to convert NFA to DFA :**
  - What's the start state for the following :
    - *It is the set containing the original start state union all other states reached from the original state by ε (directly or indirectly)*
      - 1
      - 3
      - 2
        - {1,2,3}

Converting NFA to DFA



- **Algorithm to convert NFA to DFA :**
  - What's the possible accept states
    - *Any State set containing at least an original accept state*
      - 1 (original state)
        - {1}
        - {1,2}
        - {1,3}
        - {1,2,3}

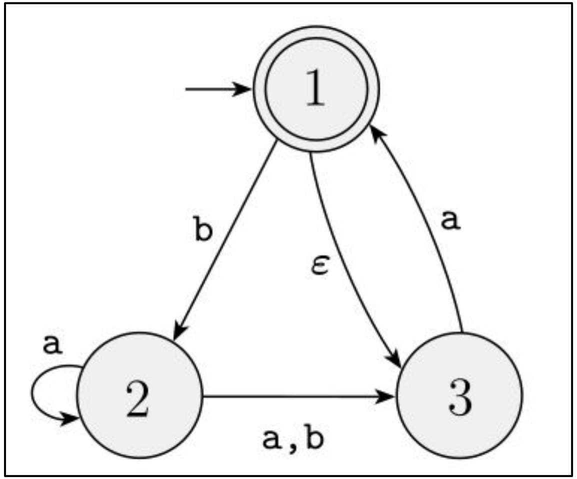
Converting NFA to DFA



- **Algorithm to convert NFA to DFA :**
    - What's the possible accept states
        - *Any State set containing at least an original accept state*
            - 3 (original state)
            - 4 (original state)
                - {3}
                - {4}
                - {3,1}
                - {3,2}
                - {3,1,2}
                - {4,1}
                - .....

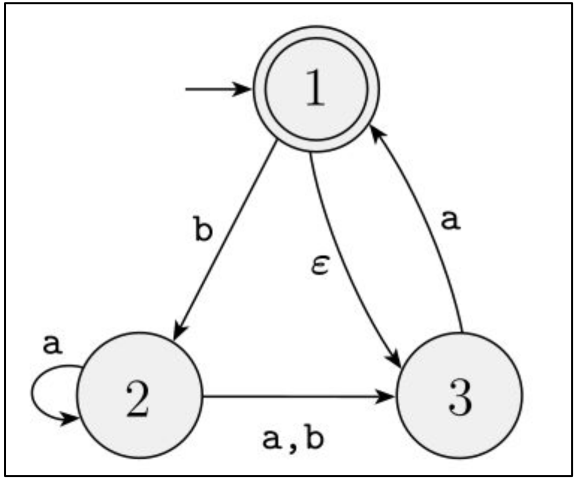
Converting NFA to DFA

- **Algorithm to convert NFA to DFA :**
    - Compute the transitions for all possible powerset elements over the alphabet
        - *{1} on reading*
            - *a → {3,1}*
            - *b → {2}*
        - *{2} on reading:*
            - *a → {2,3}*
            - *b → {3}*
        - *{1,3} on reading:*
            - *a → {1,**3**}*
            - *b → {2}*
        - *{1,2} on reading:*
            - *a → {2,3}*
            - *b → {2,3}*
        - *{2,3} on reading:*
            - *a → {1,2,3}*
            - *b → {3}*



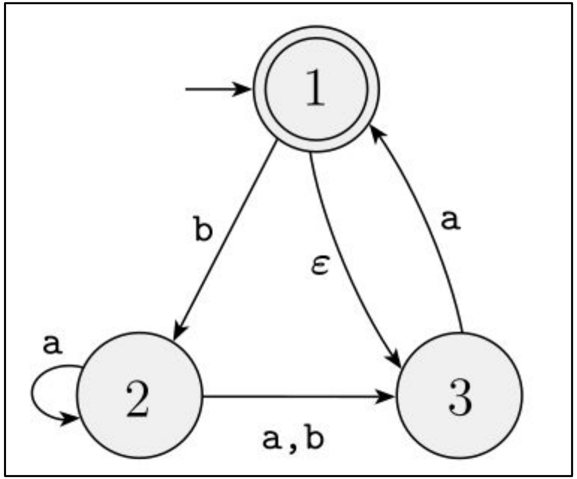
Converting NFA to DFA

- **Algorithm to convert NFA to DFA :**
  - *At state 1, reading a, what possible states we can reach :*
    - *?*

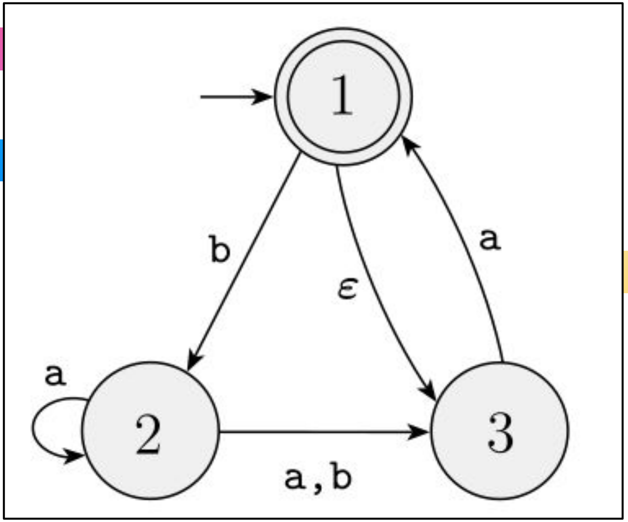


Converting NFA to DFA

- **Algorithm to convert NFA to DFA :**
  - *At state 1, reading a, what possible states we can reach :*
    - $\mathbf{1} - \varepsilon \rightarrow \mathbf{3} - a \rightarrow \mathbf{1}$
    - $\mathbf{1} - \varepsilon \rightarrow \mathbf{3} - a \rightarrow \mathbf{1} - \varepsilon \rightarrow \mathbf{3}$
  - *The possible states that can be reached from 1 are { 1 , 3 }*
    - *Remember to consider always :*
      - $\boldsymbol{\varepsilon^{*}}$ ***STATE*** $\boldsymbol{\varepsilon^{*}}$



Converting NFA to DFA

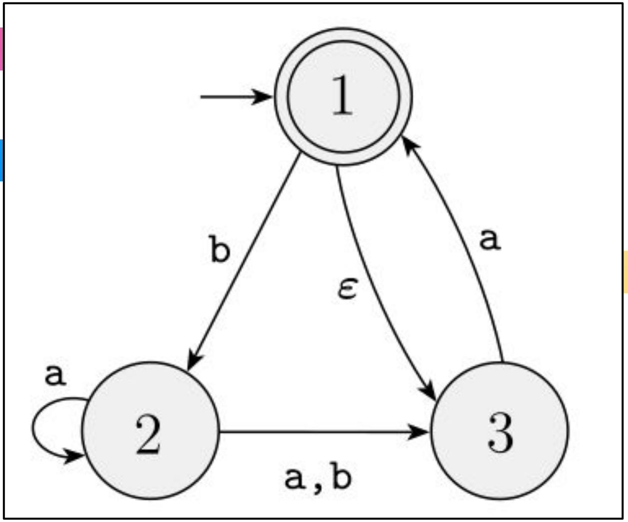


- **Algorithm to convert NFA to DFA : Example 1**
  - Using Transition Table

	a	b	ε
1	{1,3}	{2}	{3,1}
2	{2,3}	{3}	{2}
3	{1,3}	Φ	{3}

Every state, it has implicit Ɛ transition to itself !

Converting NFA to DFA



- **Algorithm to convert NFA to DFA : Example 1**
  - Using Transition Table

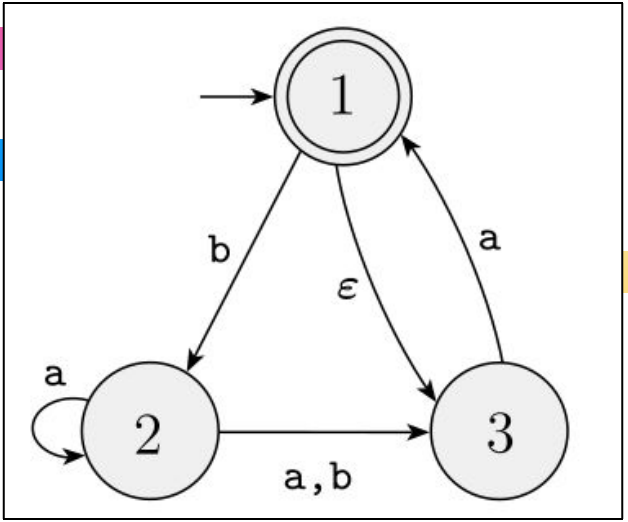
	a	b	ε
1	{1,3}	{2}	{3,1}
2	{2,3}	{3}	{2}
3	{1,3}	Φ	{3}

	a	b
{1,3}		
{1}	{1,3}	{2}
{2}	{2,3}	{3}
{3}	{1}	Φ

This is our new **start state**

Converting NFA to DFA

- **Algorithm to convert NFA to DFA : Example 1**
  - Using Transition Table



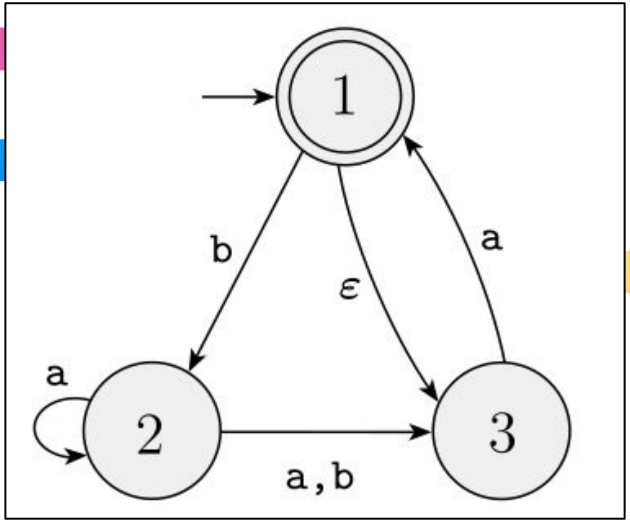
	a	b	ε
1	{1,3}	{2}	{3,1}
2	{2,3}	{3}	{2}
3	{1,3}	Φ	{3}

	a	b
{1,3}	{1,3}	{2}
{1}	{1,3}	{2}
{2}	{2,3}	{3}
{3}	{1}	Φ

On reading input a →
What are **the set of** the original states reached from either : 1 **or** 3

Don't overlook the ε !

Converting NFA to DFA



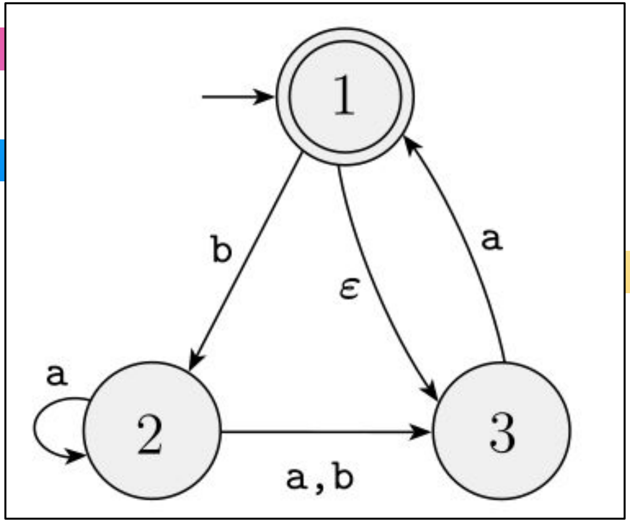
- **Algorithm to convert NFA to DFA : Example 1**
  - Using Transition Table

	a	b	ε
1	{1,3}	{2}	{3,1}
2	{2,3}	{3}	{2}
3	{1,3}	Φ	{3}

	a	b
{1,3}	{1,3}	{2}
{1}	{1,3}	{2}
{2}	{2,3}	{3}
{3}	{1}	Φ

A dead state is created for the empty set

Converting NFA to DFA



- **Algorithm to convert NFA to DFA : Example 1**
  - Using Transition Table

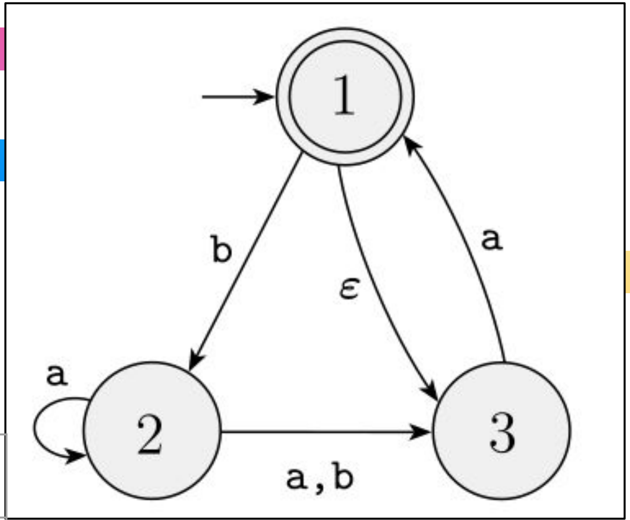
	a	b	ε
1	{1,3}	{2}	{3,1}
2	{2,3}	{3}	{2}
3	{1,3}	Φ	{3}

	a	b
{1,3}	{1,3}	{2}
{1}	{1,3}	{2}
{2}	**{2,3}**	{3}
{3}	{1}	D_1
{2,3}	**{1,2,3}**	**{3}**

We create a new row to represent this state

Converting NFA to DFA

- **Algorithm to convert NFA to DFA : Example 1**
  - Using Transition Table



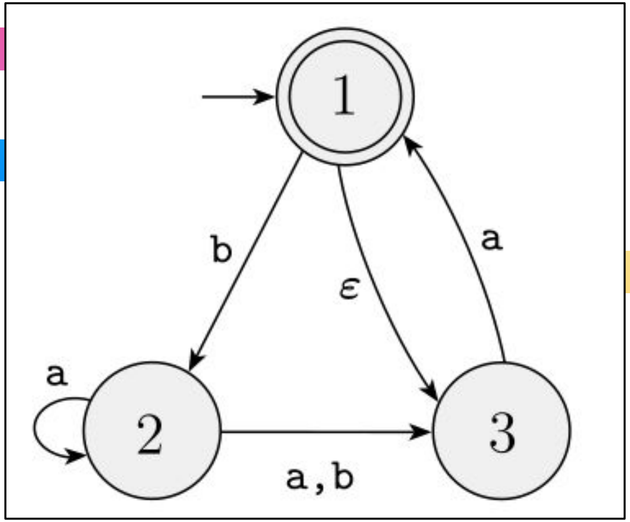
	a	b	ε
1	Φ	{2}	{3,1}
2	{2,3}	{3}	{2}
3	{1,3}	Φ	{3}

	a	b
{1,3}	{1,3}	{2}
{1}	D_1	{2}
{2}	{2,3}	{3}
{3}	{1}	D_1
{2,3}	**{1,2,3}**	**{3}**
{1,2,3}	**{1,2,3}**	**{2,3}**

We create a new row to represent this state

Converting NFA to DFA

- **Algorithm to convert NFA to DF**
  - Using Transition Table

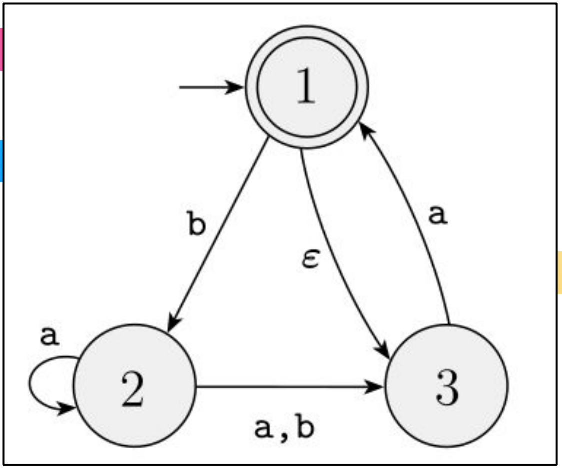


	a	b	ε
1	{1,3}	{2}	{3,1}
2	{2,3}	{3}	{2}
3	{1,3}	Φ	{3}

	a	b
{1,3}	{1,3}	{2}
{1}	{1,3}	{2}
{2}	{2,3}	{3}
{3}	{1}	D_1
{2,3}	**{1,2,3}**	**{3}**
{1,2,3}	**{1,2,3}**	**{2,3}**
D_1	D_1	D_1

We create a new row to represent this state

Converting NFA to DFA

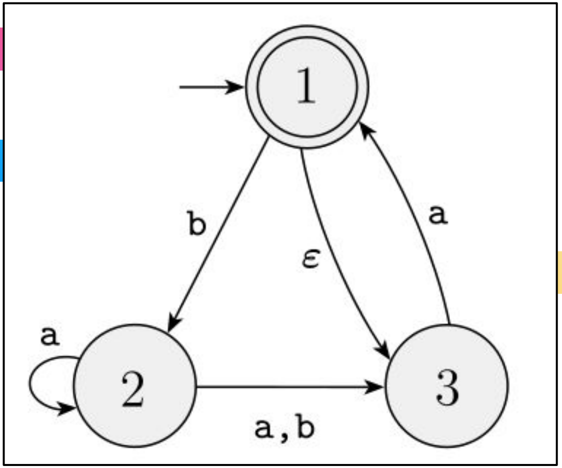


- **Algorithm to convert NFA to DFA : Example 1**

	a	b
{1,3}	{1,3}	{2}
{1}	{1,3}	{2}
{2}	{2,3}	{3}
{3}	{1,3}	D_1
{2,3}	**{1,2,3}**	**{3}**
{1,2,3}	**{1,2,3}**	**{2,3}**

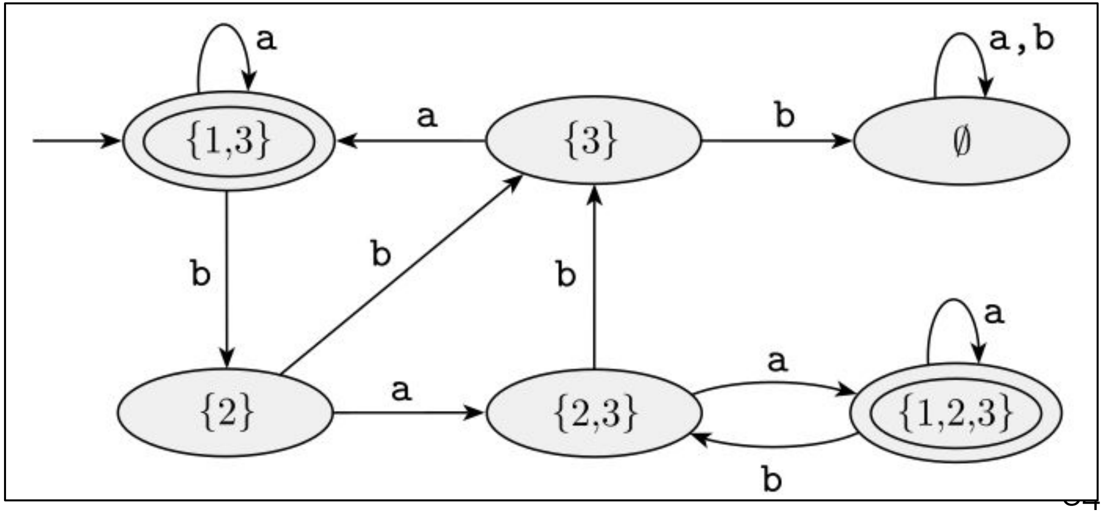
NO incoming transitions to {1} , no need for this state

Converting NFA to DFA



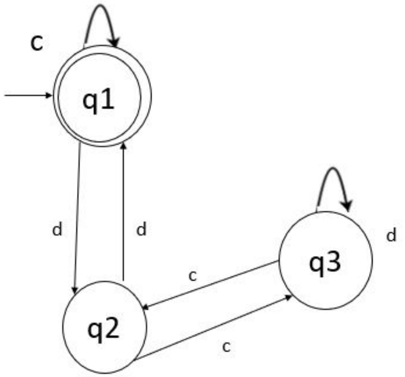
- **Algorithm to convert NFA to DFA : Example 1**

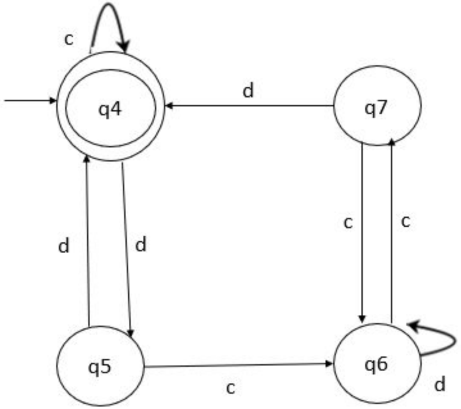
	a	b
{1,3}	{1,3}	{2}
{1}	{1,3}	{2}
{2}	{2,3}	{3}
{3}	{1,3}	D_1
{2,3}	**{1,2,3}**	**{3}**
{1,2,3}	**{1,2,3}**	**{2,3}**



Equivalence and Automata (DFA)

- How to determine that two deterministic finite automata are **equivalent**
  - **Represent the same language**





Equivalence and Automata (DFA)

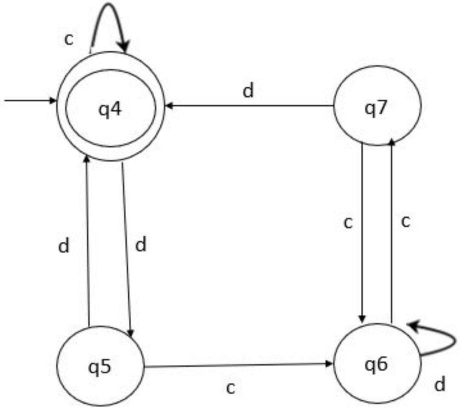
- From initial pair (q_0^a, q_0^b) of the start states of the automata M_a and M_b
  - Start the computing transitions for all symbols
    - **To be equivalent :** for a given symbol**, the resulting states must be an accepting for both or a non-accepting for both machines.**
    - **Otherwise, the two machines are not equivalent**
- Any new pair showing up, do the same recursively until no new pair shows up

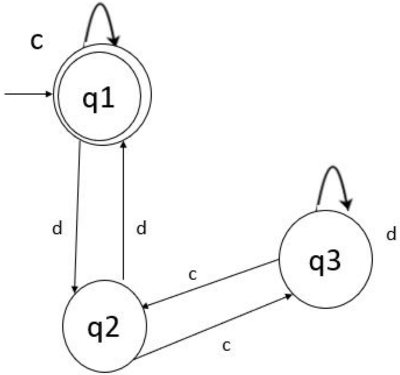
Equivalence and Automata (DFA)

All possible inputs from the Alphabet

	c	d
(q4,q1)		

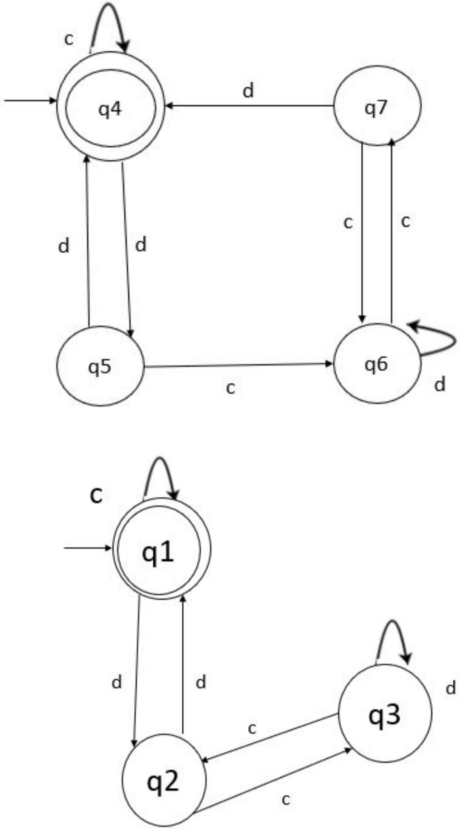
Pair of states from the two machines to compare





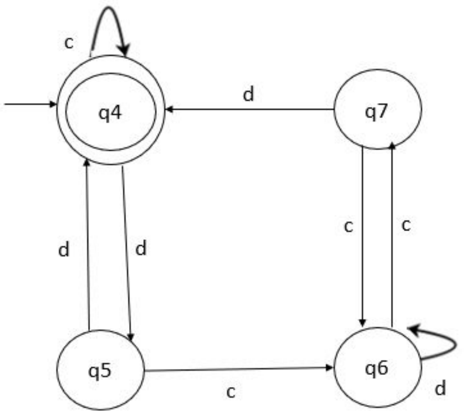
Equivalence and Automata (DFA)

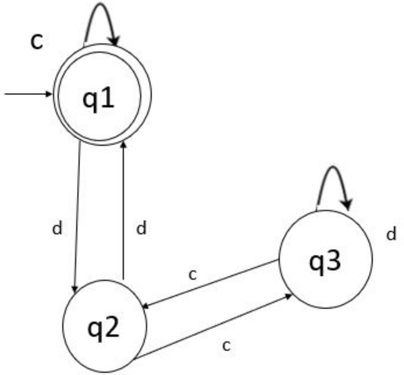
	c	d
(q4,q1)	(q4,q1)	(q5,q2)



Equivalence and Automata (DFA)

	c	d
(q4,q1)	(q4,q1) Both Accept	(q5,q2) Both Non-Accept

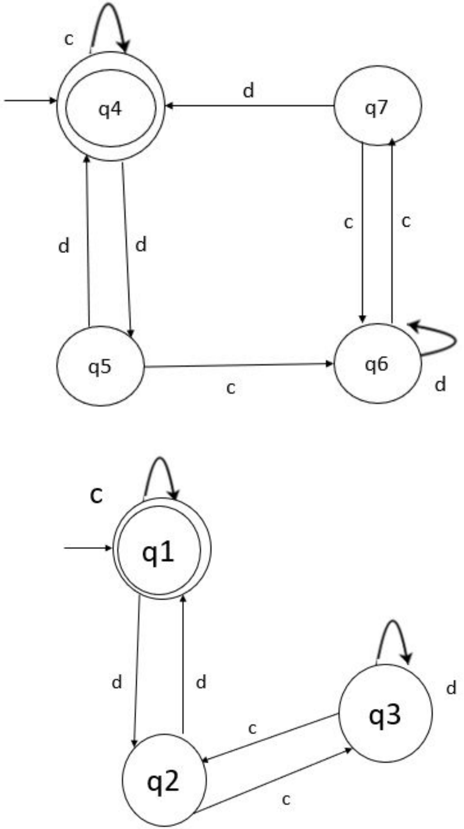




Equivalence and Automata (DFA)

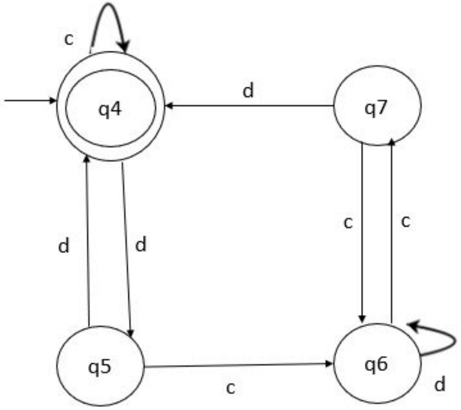
	c	d
(q4,q1)	(q4,q1) Both Accept	**(q5,q2)** Both Non-Accept

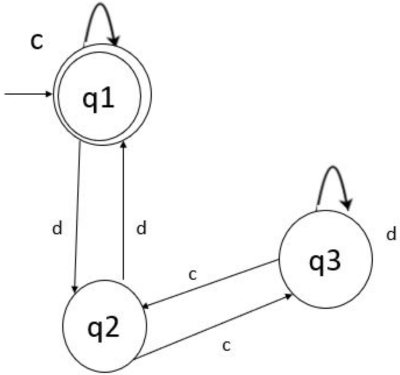
The pair (q5,q2) needs to be assessed in the same way, (q4,q1) already done



Equivalence and Automata (DFA)

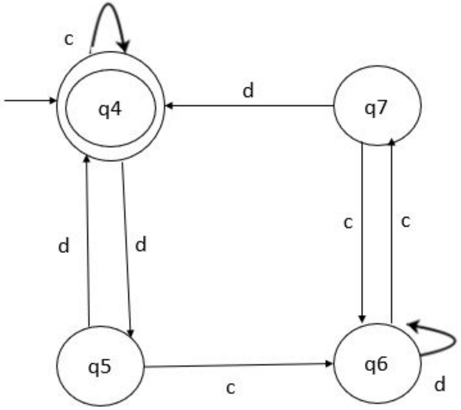
	c	d
(q4,q1)	(q4,q1) Both Accept	(q5,q2) Both Non-Accept
(q5,q2)	**(q6,q3)** Both Non-accept	(q4,q1) Both Accept

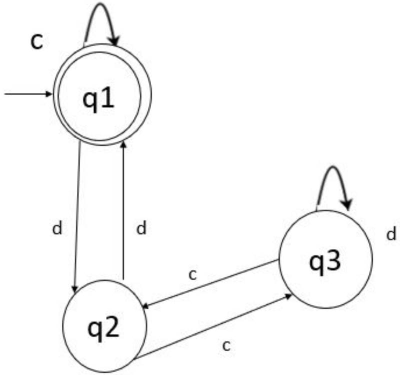




Equivalence and Automata (DFA)

	c	d
(q4,q1)	(q4,q1) Both Accept	(q5,q2) Both Non-Accept
(q5,q2)	**(q6,q3)** Both Non-accept	(q4,q1) Both Accept
(q6,q3)	**..**	..





Questions

- DFA/NFA have finite number of states, can they be used to represent languages with infinite words ?
- Can we construct an NFA for the language of palindromes $L = \{w \mid w = w^R \text{ and } |w| < 5\}$, alphabet is {0,1} ?
- Inferring the NFA for the complement of language ?

Notations

- Language notation
  - a^i : a is being repeated a times.
  - $n_a(x)$: number of occurrences of a in the word x

Constructing NFA

- **Example 2 :**
    - Design the NFA for the following Language L over Σ={1,0}
        - L= { }
            -

Constructing NFA

- **Example 2 :**
  - Design the NFA for the following Language L over Σ={1,0}
    - L= { }
      -

Constructing NFA

- **Example 3 :**
  - Design the NFA for the following Language L over Σ={1,0}
    - L= { }
      -

Constructing NFA

- **Example 3 :**
    - Design the NFA for the following Language L over Σ={1,0}
        - L= { }
            -