

Theory of Computing :

2. Finite Automata : DFA



Professor Imed Bouchrika

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

Outline :

- Revision: Computation & Complexity
- Automaton Process Flow
- Representation of Automata
- DFA Construction and Examples
- Minimizing DFA

Computation & Complexity



- What problems we can solve with a computer ?
- And how efficient we solve them if they are computable ?

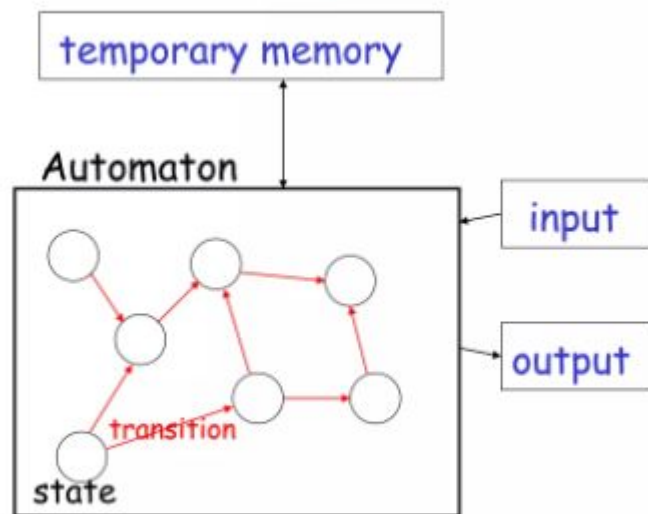
Computation & Complexity

- What problems we can solve with a **computer** ?
- And how efficient we solve them if they are computable ?

What type of a computer ?

Computation & Complexity

- The part for processing the information or solving a given problem is the **Automaton**
- In this case : CPU + Program memory for a physical device is replaced by:
 - States
 - Transitions



Computation & Complexity



- Automaton :
 - Literal Meaning : a device that can do “things” on its own or a self-operating machine
 - Plural (Automata)
 - In this course :
 - An **abstract device** for processing information to solve a given problem. (no labs, only pen and paper)

Computation & Complexity



- Automaton :
 - *are abstract models of machines that perform computations on an input by moving through a series of states or configurations.*
 - *At each state of the computation, a transition function determines the next configuration on the basis of a finite portion of the present configuration.*
 - *As a result, once the computation reaches an accepting configuration, it accepts that input. The most general and powerful automata is the Turing machine*

Computation & Complexity



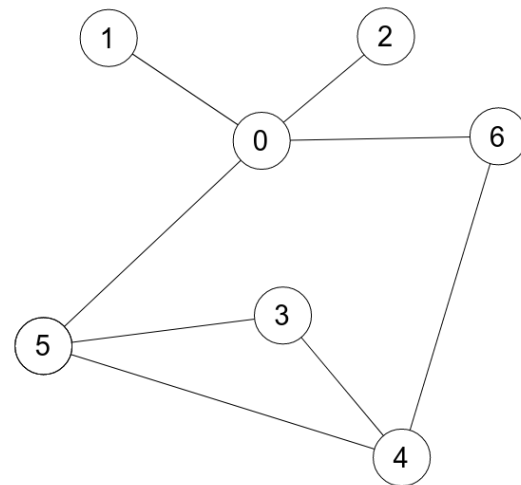
- There are different types of automata including :
 - *Finite Automata (no temporary memory)*
 - *Pushdown Automata (has a stack)*
 - *Turing Machines (Random access memory)*

Computation & Complexity

- Given a problem, given a computational model ?
 - How to feed to the problem into the computational model ?

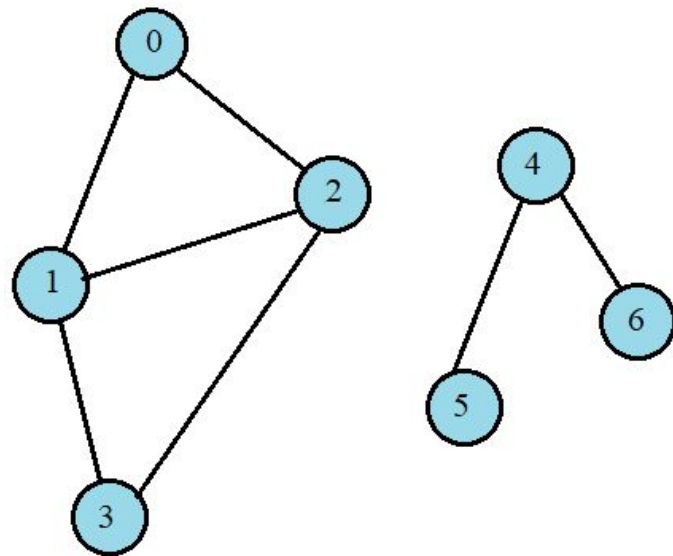
Computation & Complexity

- Given a problem, given a computational model ?
 - Are all nodes connected ?
 - Or, is this a connected graph ?



Computation & Complexity

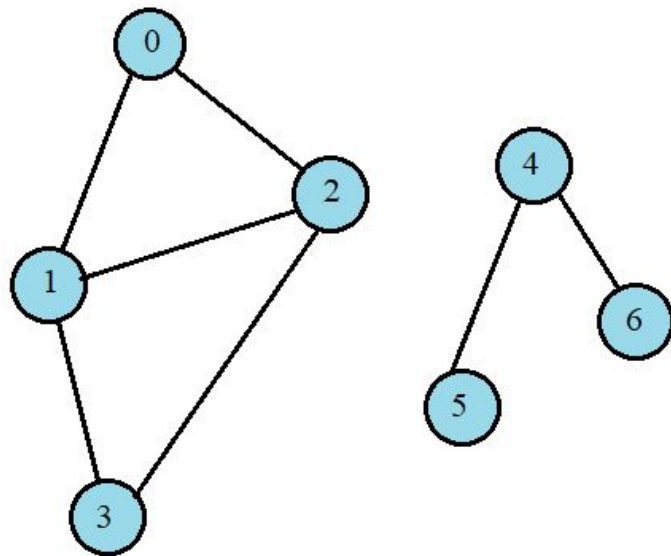
- Given a problem, given a computational model ?
 - Are all nodes connected ?
 - Or, is this a connected graph ?



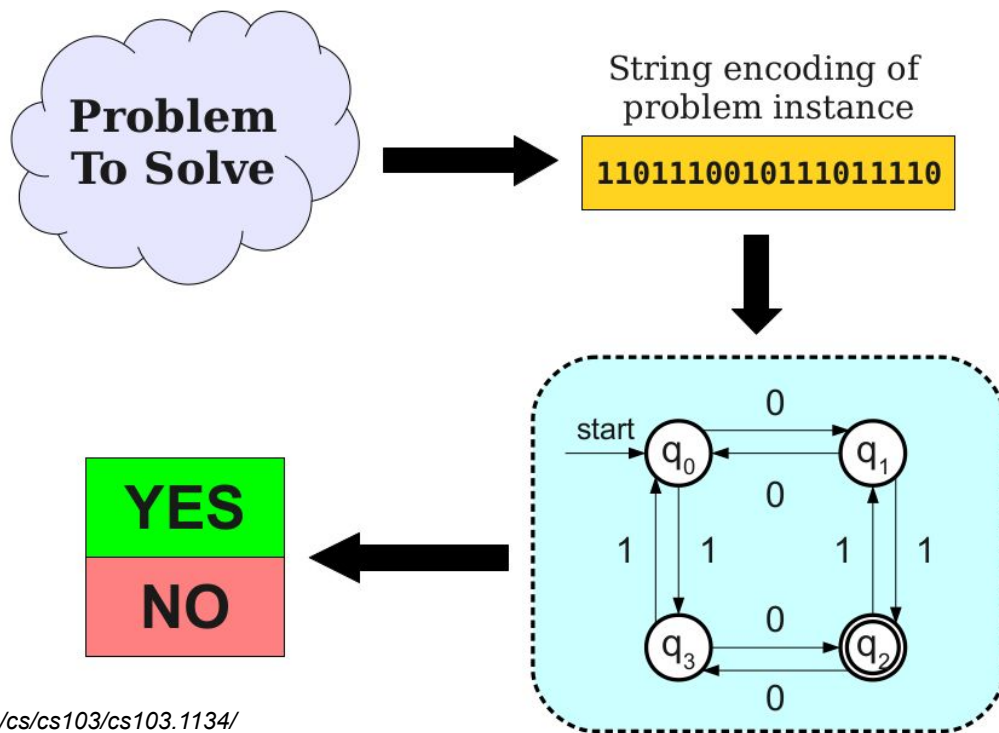
Computation & Complexity

- Given a problem, given a computational model ?
 - Are all nodes connected ?
 - Or, is this a connected graph ?

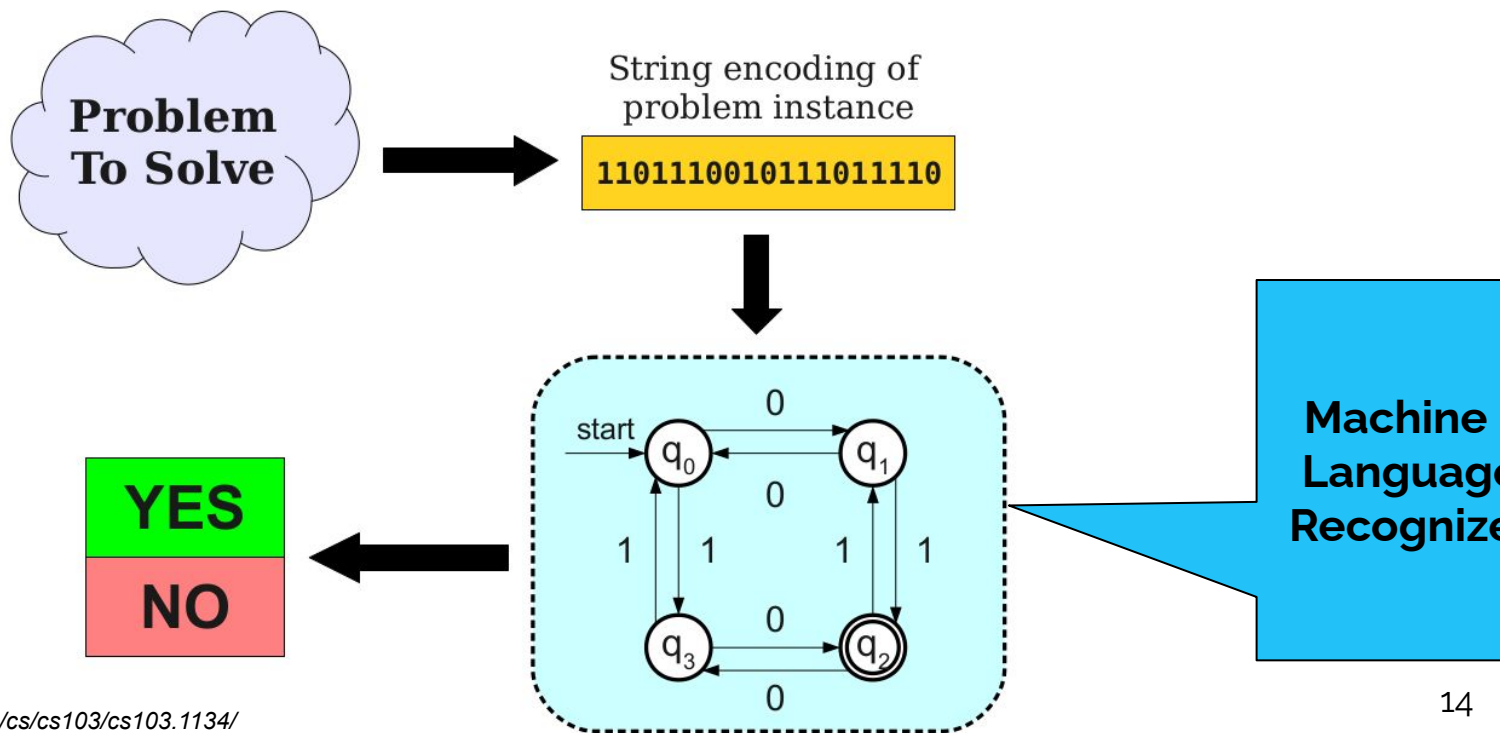
How to automate solving the problem ?



Computation & Complexity



Computation & Complexity



Computation & Complexity

- **Symbol :**
 - Atomic unit such as : a , 1, #, True, False (We cannot split into other units)
- **Alphabet :**
 - **Finite** set of symbols
 - Should not be empty
 - Usually denoted by Sigma : Σ
 - Examples :
 - Binary Alphabet : $\Sigma = \{ 0, 1 \}$
 - English Alphabet : $\Sigma = \{ a...z, A ...Z \}$
 - Binary Alphabet : $\Sigma = \{ a, b \}$
 - Unary Alphabet : $\Sigma = \{ z \}$

Computation & Complexity

- **String :**

- is a word with a **finite** set of symbols taken from the defined set of Alphabet.
- The empty string is denoted by ϵ
- $|x|$ = the length of the string x
- Examples:
 - $x = \text{abababab}$ from : $\Sigma = \{ a, b \}$
 - $x = 1111011$ from $\Sigma = \{ 0, 1 \}$
 - $x = \text{ENSIA}$ $\Sigma = \{ a, \dots, z, A, \dots, Z, \text{SPACE} \}$

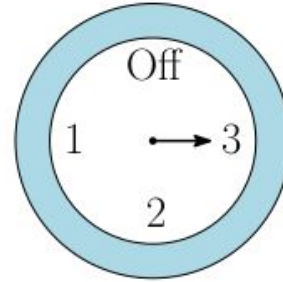
Computation & Complexity

- **Powers of An Alphabet : Sets of Strings :**

- Σ is defined as the alphabet
- Σ^k = is the set of all strings composed from the alphabet Σ and **have a length of k**
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \Sigma^4 \cup \dots$ = the set of all strings from the alphabet Σ
 - Σ^* Called the universal set of all strings
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \Sigma^4 \cup \dots$: All strings excluding the empty string
- Examples for $\Sigma = \{ 1, 0 \}$
 - $\Sigma^2 = \{ 00, 01, 11, 10 \}$
 - $\Sigma^4 = \{ 0011, 1011, 1111, 1110, \dots \}$
 - $\Sigma^* = \{ \epsilon, 0, 1, 00, 01, 11, 000, 001, \dots \}$

Finite Automata

- We have a multi-speed fan and we want to design the logic for its controller.

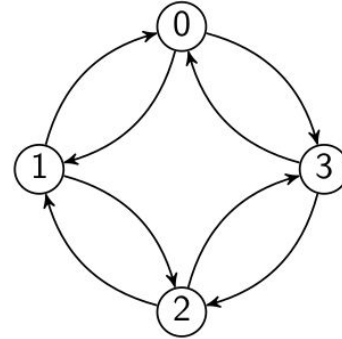
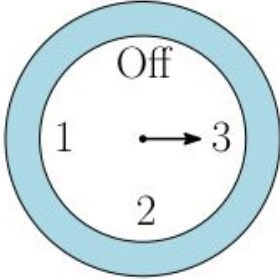


- The equipment has a rotary controller with the inputs { off,1, 2,3,4}
- Does the machine accept the sequence of the following commands/inputs :

Off \rightarrow 1 \rightarrow 3 \rightarrow 2

Finite Automata

- Even though, it is trivial to decide, we draw a **state diagram** for the **automaton** :



- From the state diagram, we can observe that the sequence of commands $\text{Off} \rightarrow 1 \rightarrow 3 \rightarrow 2$ is **not recognized**.

Finite Automata



- Finite Automaton : is a computational model with an extremely limited amount of memory
 - There is a language
 - There are rules
- The basic components for finite automata are :
 - **Finite** number of states
 - Transitions between states
- The finite automaton acts a **Language Recognizer/Acceptor** as a decision problem with “Accept” or “Reject” (Or Yes vs No)

Finite Automata

- Finite Automaton : is a decision problem solver that checks whether a given word is accepted or rejected



Finite Automata



- Finite Automaton :
 - is a mathematical/abstract machine for determining whether a string is contained within some language.

Finite Automata

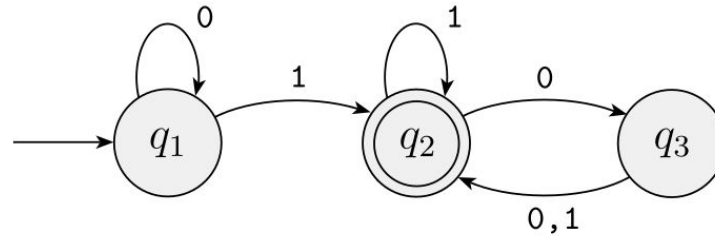


- Finite Automata are everywhere in many applications including :
 - ATMs,
 - Vending Machines
 - Traffic Signal Systems
 - Calculators
 - Washing Machines
 - Compilers
 - Search Engines.
 - ..

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):**

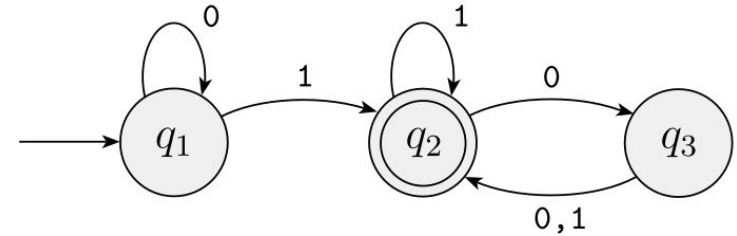
- There must **be one** start state with the inbound arrow

- **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)**

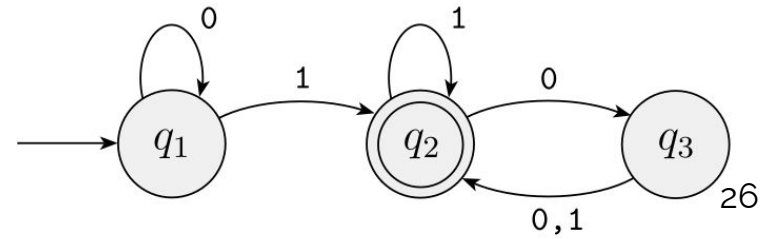
- Drawn as single-line circle
- There can be many or none
- If the machine reaches the end of the word at this state, the word is **rejected**.



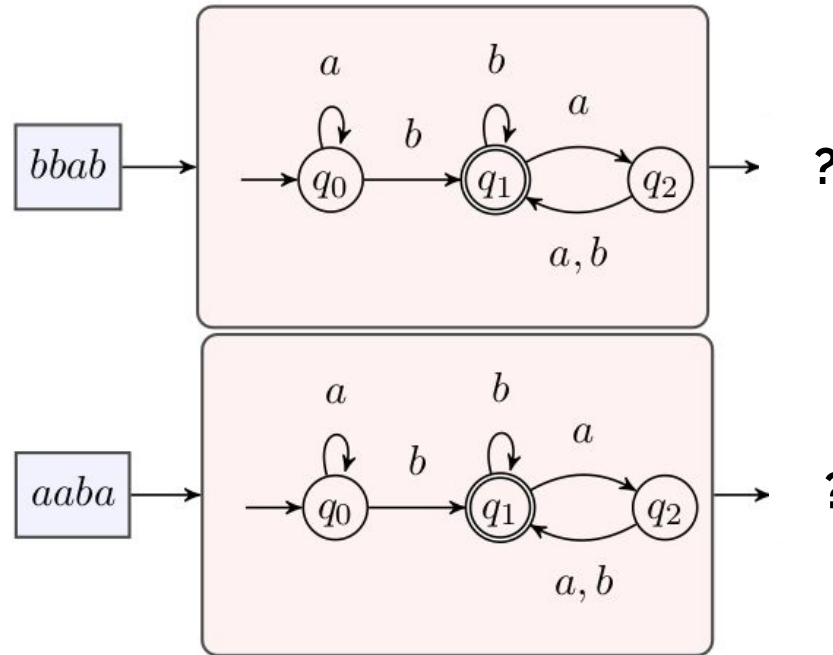
Finite Automata

- **Transitions :**

- Are the labelled one-direction arrows between
 - Two different states
 - To the state itself.
- Each transition should have one or more labels reflecting the given input **symbol** from the alphabet

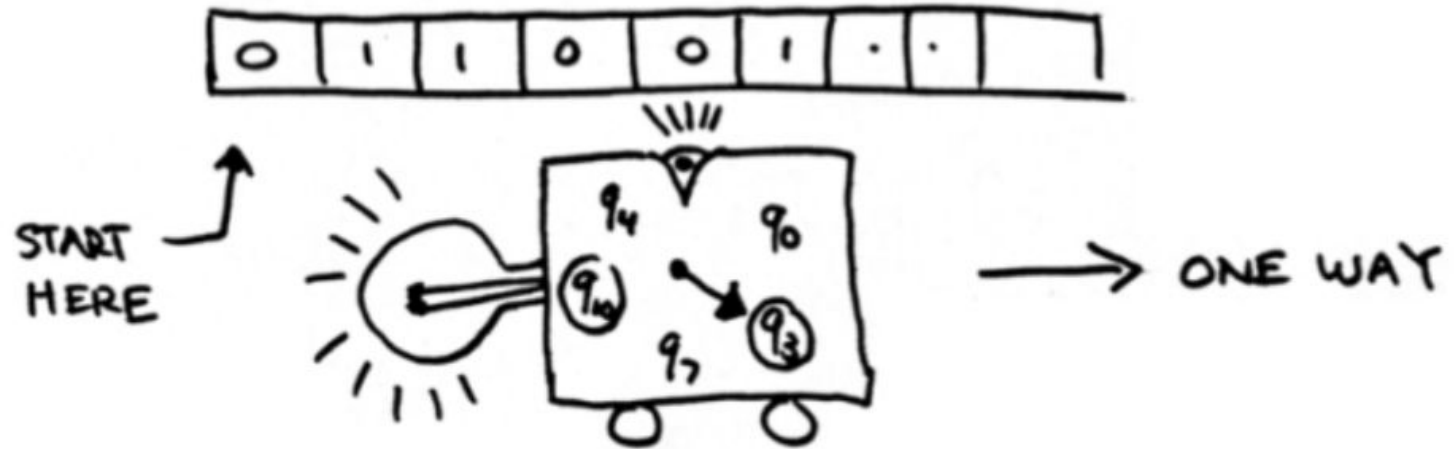


Finite Automata



Finite Automata

- Visualization for a Finite Automaton :



Finite Automata

- **Visualization for a Finite Automaton :**

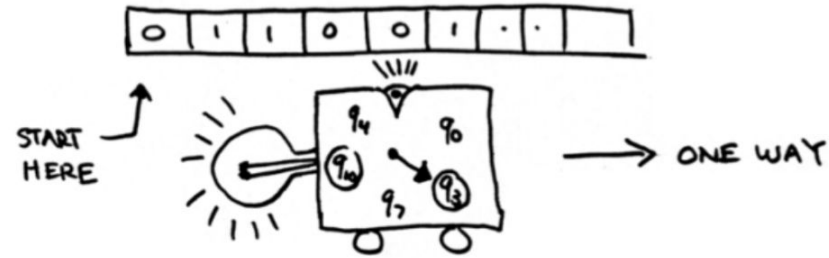
- Automaton:

- *Limited Memory to store the current state.*

- The machine reads sequentially the tape in one way.

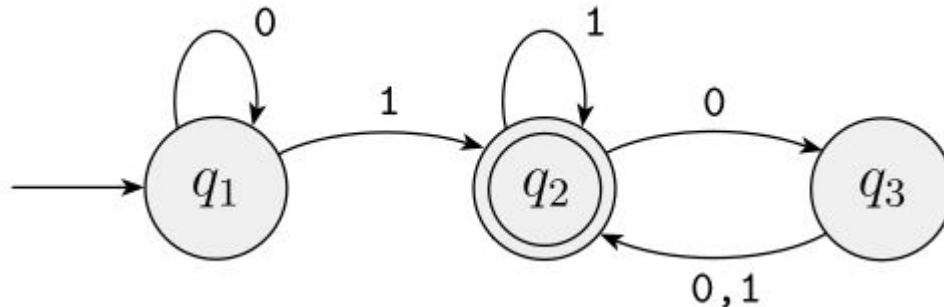
- Every symbol the machine reads, the state is updated based on the transitions defined.

- Whenever, it reaches the end of the tape (word), if the reached state is an accepting, it is an accept, otherwise, it is a reject.

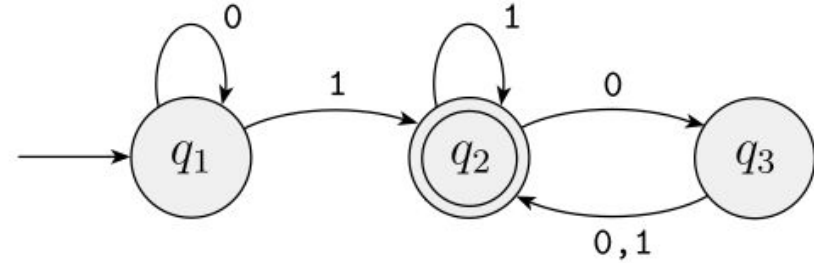


Finite Automata

- **Simulating the processing of a string on an Automaton:**
 - We want to see if the string **1101** is **accepted by** by the following machine:

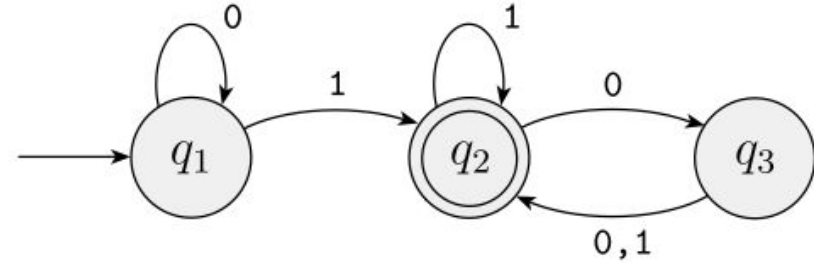


Finite Automata



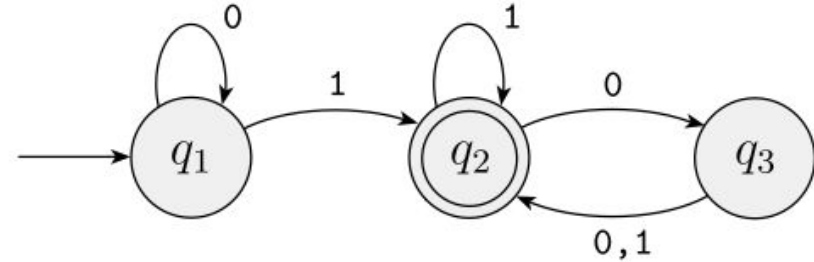
- **Simulating the processing of a string on an Automaton:**
 - We want to see if the string **1101** is **accepted by** by the following machine:
 1. The machine is initialized at **q1** (The starting state)

Finite Automata



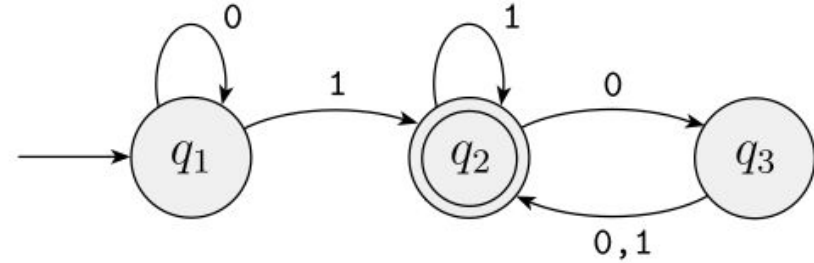
- **Simulating the processing of a string on an Automaton:**
 - We want to see if the string **1101** is **accepted by** by the following machine:
 1. The machine is initialized at **q1** (The starting state)
 2. The machine reads the first **symbol 1** (Remember, current state is q1) → Transition to state **q2** (We follow the transition label). → Current state is updated to **q2** inside the limited memory: **q1→q2**

Finite Automata



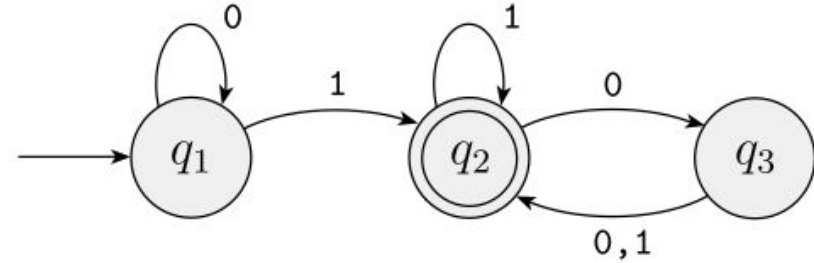
- **Simulating the processing of a string on an Automaton:**
 - We want to see if the string **1101** is **accepted by** by the following machine:
 1. The machine is initialized at **q1** (The starting state)
 2. The machine reads the first **symbol 1** (Remember, current state is q1) → Transition to state **q2** (We follow the transition label). → Current state is updated to **q2** inside the limited memory: **q1→q2**
 3. The machine moves to read the next symbol : **1** , There a transition of input **1** from q2 to **q2** → **q2 : we stay at q2**

Finite Automata



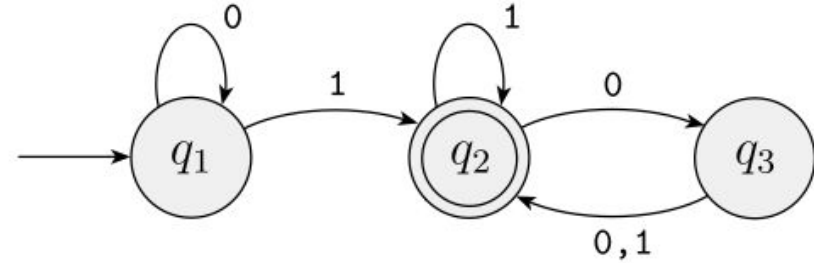
- **Simulating the processing of a string on an Automaton:**
 - We want to see if the string **1101** is **accepted by** by the following machine:
 4. The machine reads the next symbol **0** whilst the current state is **q2**, there is a transition with the label 0 towards q3. The machine updates the current state to **q3** : **q2** → **q3**

Finite Automata



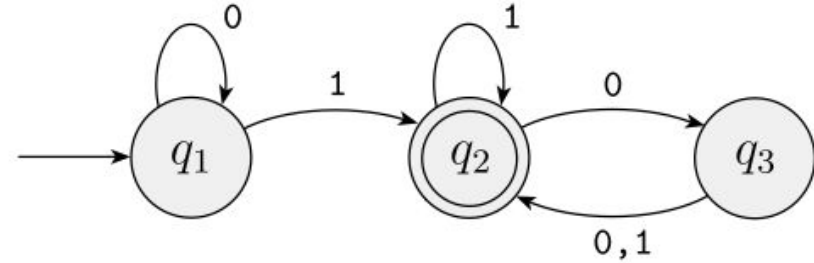
- **Simulating the processing of a string on an Automaton:**
 - We want to see if the string **1101** is **accepted by** by the following machine:
 4. The machine reads the next symbol **0** whilst the current state is **q2**, there is a transition with the label 0 towards q3. The machine updates the current state to **q3** : **q2 → q3**
 5. The machine reads the **final symbol 1**, there is a transition from q3 to q2 with the label 1. The machine moves to state **q2** : **q3 → Q2**

Finite Automata



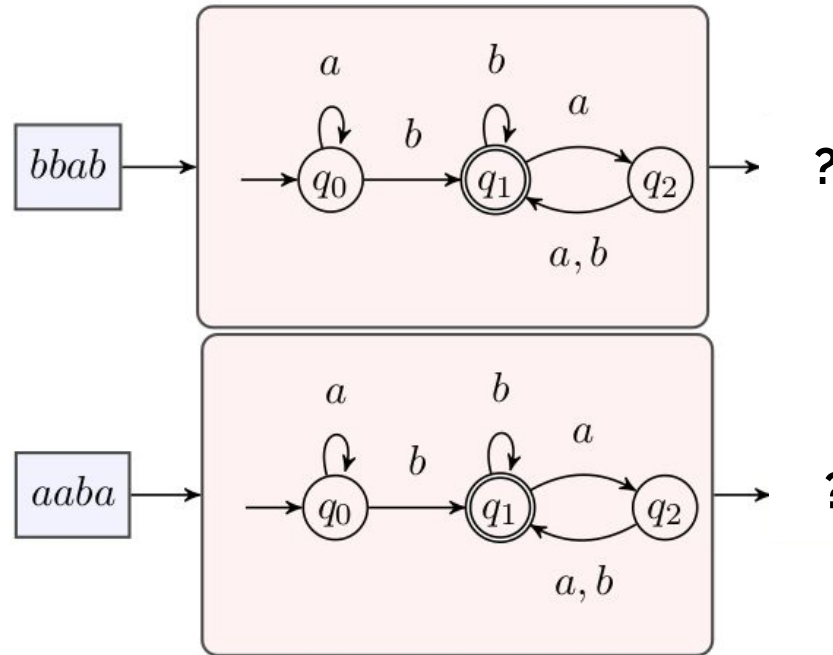
- **Simulating the processing of a string on an Automaton:**
 - We want to see if the string **1101** is **accepted by** by the following machine:
 - The machine read all symbols in the word/string.
 - Terminates at an accepting/final state
 - \Rightarrow The word is recognized by the finite automata

Finite Automata

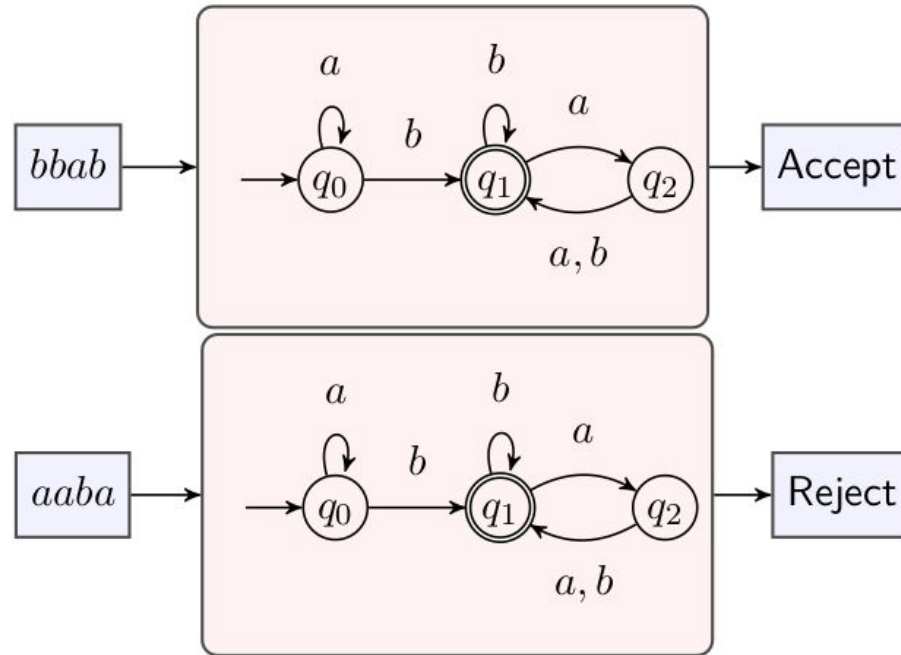


- **Simulating the processing of a string on an Automaton:**
 - Let's see if the word "**10**" is accepted by the machine ?

Finite Automata

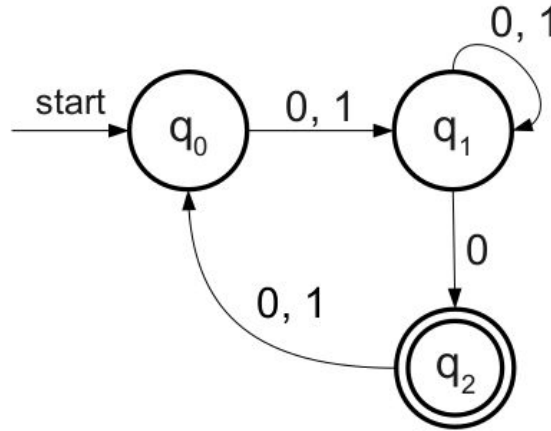


Finite Automata



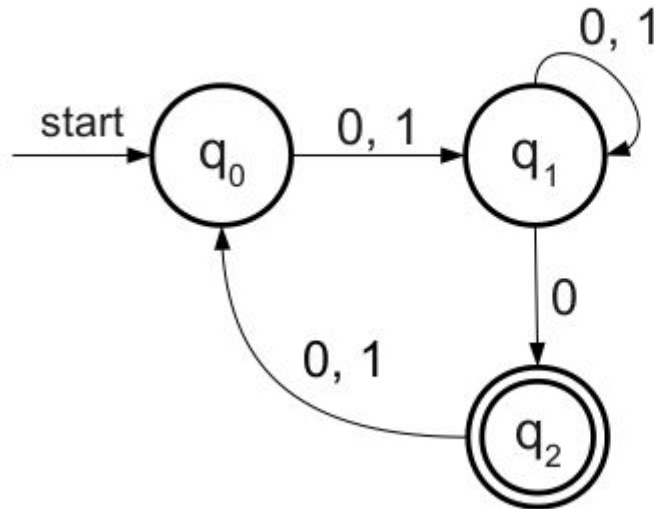
Finite Automata

- **Simulating the processing of a string on an Automaton:**



Finite Automata

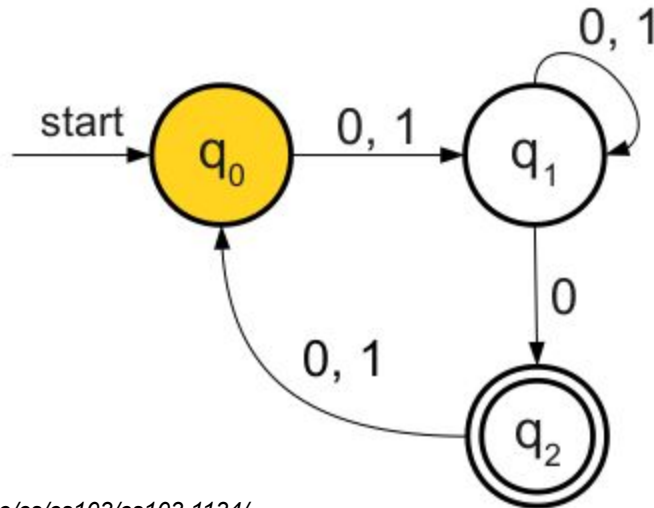
- Simulating the processing of a string on an Automaton:



0 0 0

Finite Automata

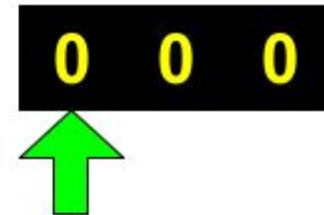
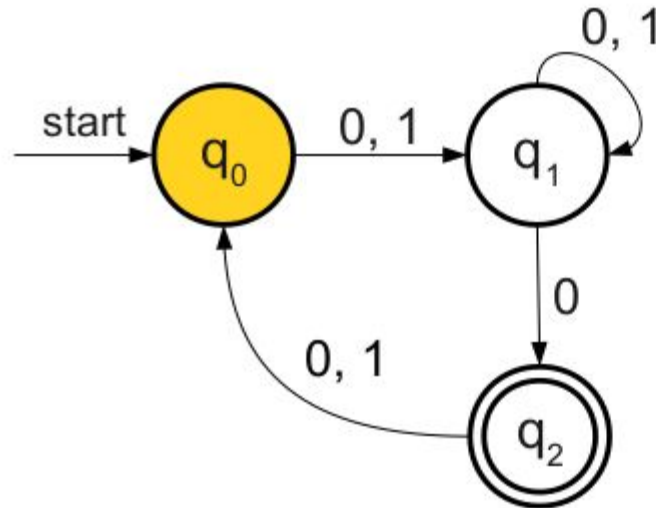
- Simulating the processing of a string on an Automaton:



0 0 0

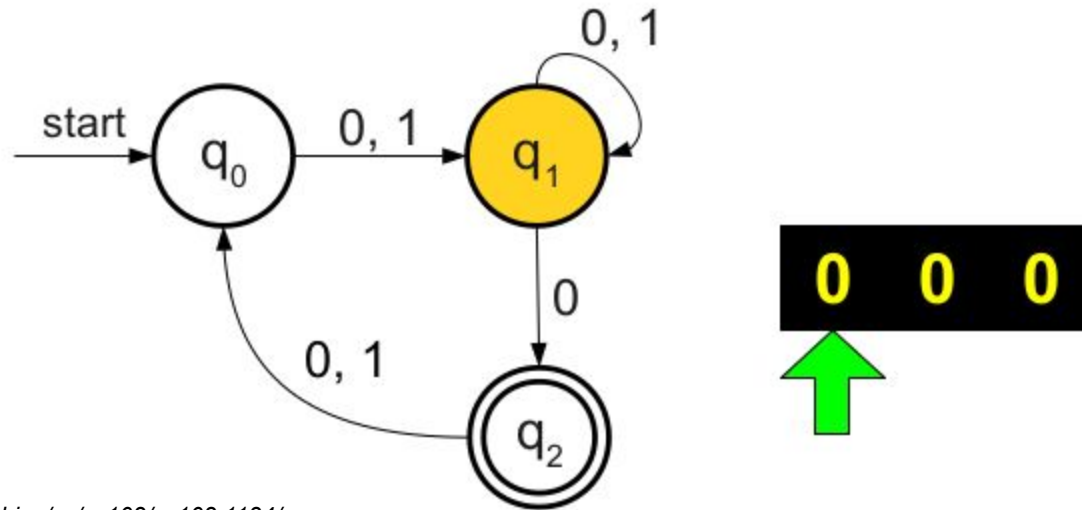
Finite Automata

- Simulating the processing of a string on an Automaton:



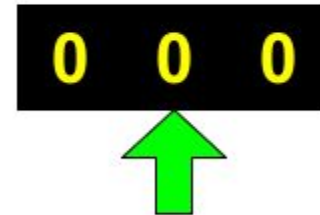
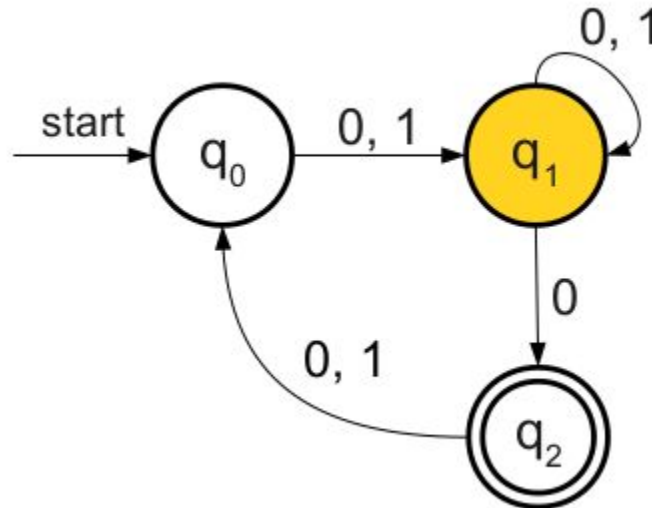
Finite Automata

- Simulating the processing of a string on an Automaton:



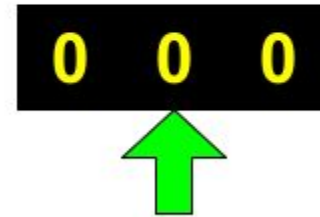
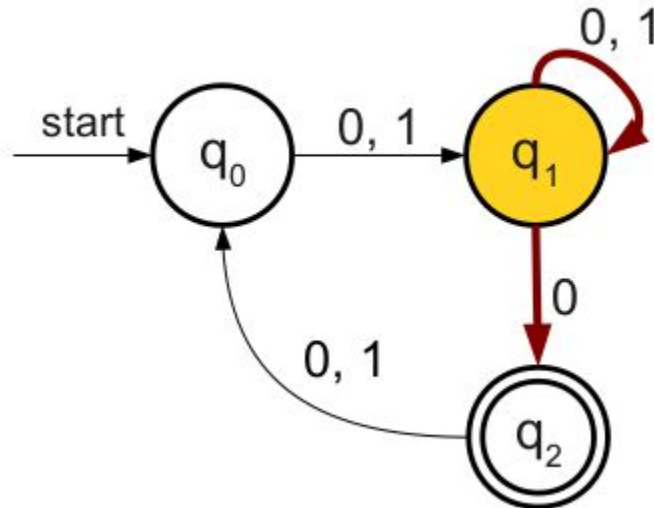
Finite Automata

- Simulating the processing of a string on an Automaton:



Finite Automata

- Simulating the processing of a string on an Automaton:



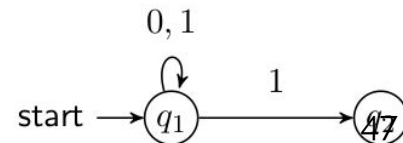
Finite Automata

- **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



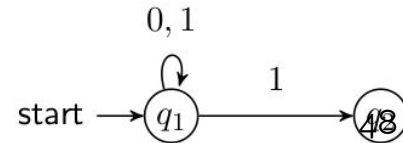
Finite Automata

- **Deterministic Finite Automaton :**

- Deterministic = Events can be determined precisely
- Finite = Finite and small amount of space used
- Automaton = Computing machine

- **Deterministic ?**

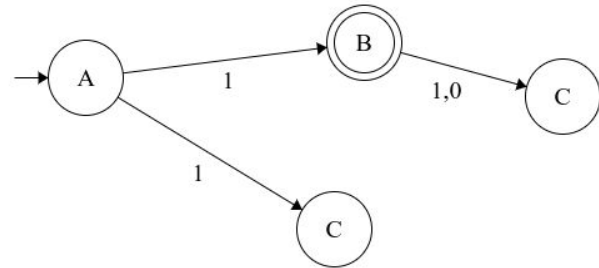
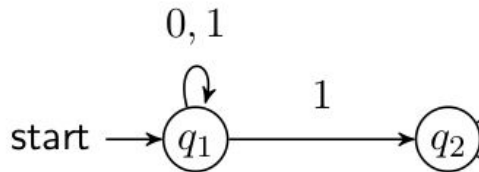
- From a given state there is only one transition for each symbol + Each transition must have non-empty string symbol.
- The q_1 state : There are two transitions for the same symbol 1.
Therefore, this is not a deterministic finite automaton



Finite Automata

- **Deterministic Finite Automaton :**

- The following Automata are not deterministic because:
 - Left Automata : q_1 : has two outgoing transitions with input 1
 - Right Automata : A : has two transitions with input 1



Finite Automata

- **Formal Definition of Deterministic Finite Automaton :**

- Usually Automata 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*.

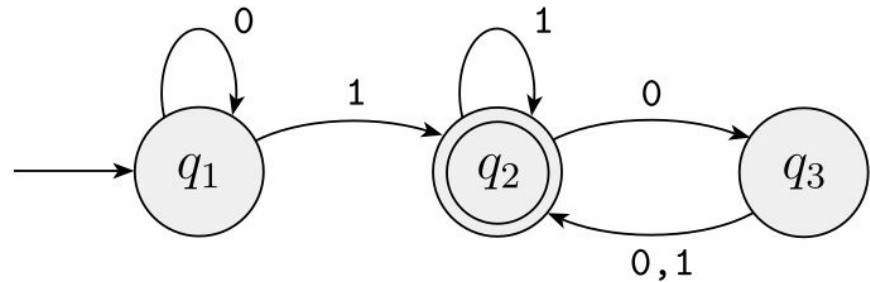
Finite Automata

- **Transitions Table :**

- Finite Automata can be additionally represented by a Transition Table showing all possible transitions at different configuration:

- For automaton shown $M = (Q, \Sigma, \delta, q_1, F)$ such that

- $Q = \{ q_1, q_2, q_3 \}$
- $F = \{ q_2 \}$
- $\Sigma = \{ 0, 1 \}$



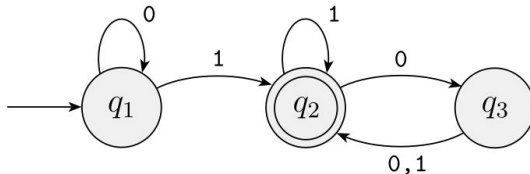
Finite Automata

- **Transition Table :**

- Finite Automata can be additionally represented by a Transition Table showing all possible transitions at different configuration:

- For automaton shown $M = (Q, \Sigma, \delta, q_1, F)$ such that

- $Q = \{ q_1, q_2, q_3 \}$
- $F = \{ q_2 \}$
- $\Sigma = \{ 0, 1 \}$



**Current
Configuration**

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

Finite Automata

- **Formal Definition of Computation using Automata :**

- Let:

- $\mathbf{M} = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton

- $w = w_1 w_2 \cdots w_n$ be a string/word where each symbol w_i is a member of the alphabet Σ .

- Then \mathbf{M} accepts \mathbf{w} if a **sequence** of states r_0, r_1, \dots, r_n in \mathbf{Q} exists with three conditions:

1. $r_0 = q_0$,

2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n - 1$, and

3. $r_n \in F$.

Finite Automata

- **Formal Definition of Computation using Automata :**

- The notation $\delta(r,s)$: means the transition from state r when reading the symbol s
- Remember : δ is a function, it would map to a state.
- What about the notation :
 - $\delta^*(q,w)$

Finite Automata

- **Formal Definition of Computation using Automata :**

- The notation $\delta(r,s)$: means the transition from state r when reading the symbol s
- Remember : δ is a function, it would map to a state.
- What about the notation :
 - $\delta^*(q,w)$
 - Means : the **set** of transitions starting with state **q** upon reading a **string** w (Set of symbols)

Finite Automata

- **Regular Language**

- Let **M** a finite Automata:
- We say that M recognizes language L if $L = \{ w \mid M \text{ accepts/recognizes } w \}$

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

Finite Automata



- Designing Automata

DESIGNING FINITE AUTOMATA

Whether it be of automaton or artwork, design is a creative process. As such, it cannot be reduced to a simple recipe or formula. However, you might find a particular approach helpful when designing various types of automata. That is, put *yourself* in the place of the machine you are trying to design and then see

Finite Automata

Important !

When you design your abstract machine for a given language :

- 1. The machine should not miss words from the language**
- 2. Accept words not from the Language**

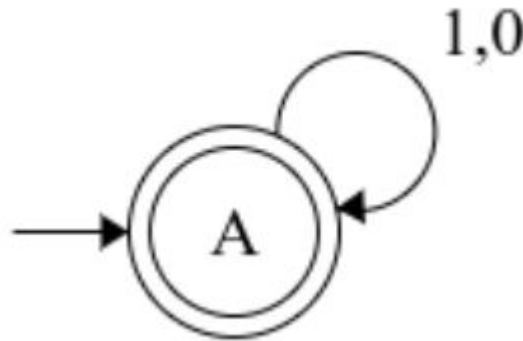
Finite Automata

- **Designing Automata :**
 - Alphabet $\Sigma = \{ 0 , 1 \}$
 - Design M which recognizes $L = \Sigma^*$

Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \Sigma^*$



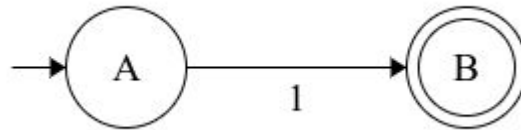
Finite Automata

- **Designing Automata :**
 - Alphabet $\Sigma = \{ 0 , 1 \}$
 - Design M which recognizes $L = \{ 1 \}$

Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ 1 \}$



- **What happens if 0 is given as input ?**

Finite Automata

- **Trap or Dead States:**

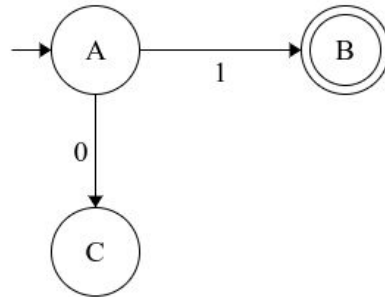
- Is a state that once entered, you can never leave.
- Used either :
 - to reject partly read strings that will never be accepted.
 - To accept partly read strings that will definitely be accepted

Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ 1 \}$

- **We create a trap case C to deal with the rejected string 0**



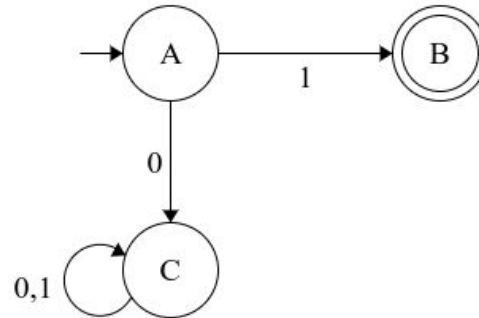
- What about the string 01 or 00 or 001

Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ 1 \}$

- **The trap case now can handles other rejected string starting with zero**

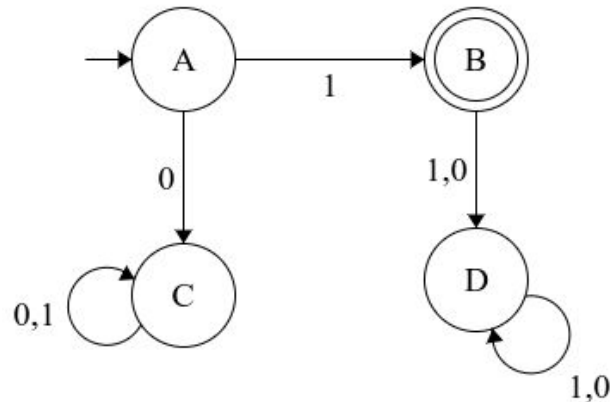


- What about the string 11 or 10 or 1001

Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ 1 \}$



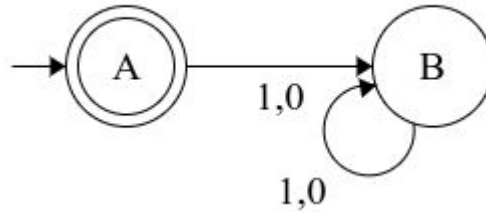
Finite Automata

- **Designing Automata :**
 - Alphabet $\Sigma = \{ 0, 1 \}$
 - Design M which recognizes $L = \{ \epsilon \}$

Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ \epsilon \}$

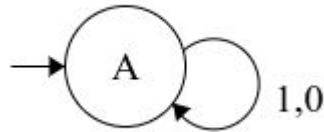


Finite Automata

- **Designing Automata :**
 - Alphabet $\Sigma = \{ 0, 1 \}$
 - Design M which recognizes $L = \emptyset$

Finite Automata

- **Designing Automata :**
 - Alphabet $\Sigma = \{ 0, 1 \}$
 - Design M which recognizes $L = \emptyset$



Finite Automata

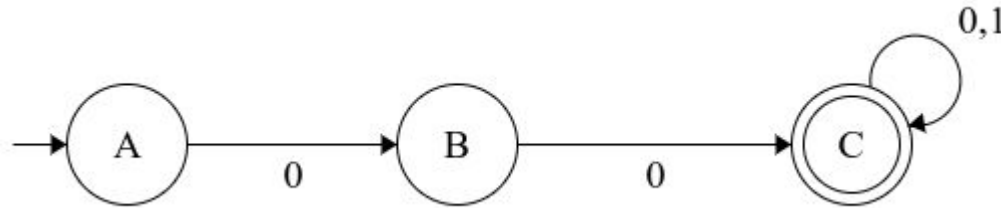
- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ starts with } 00 \}$

Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ starts with } 00 \}$

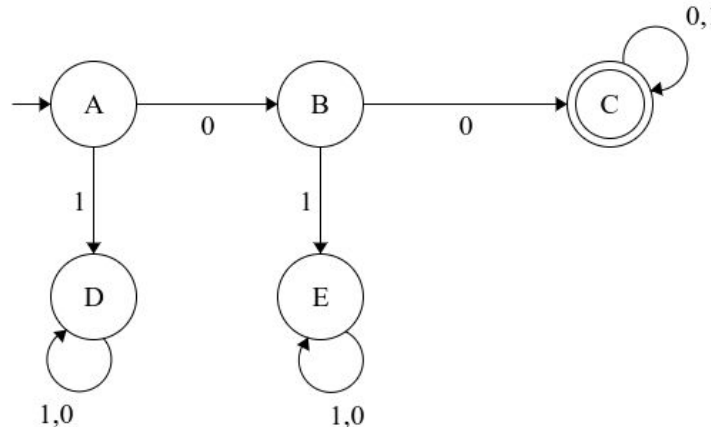


- Let's add the **dead states**

Finite Automata

- **Designing Automata :**

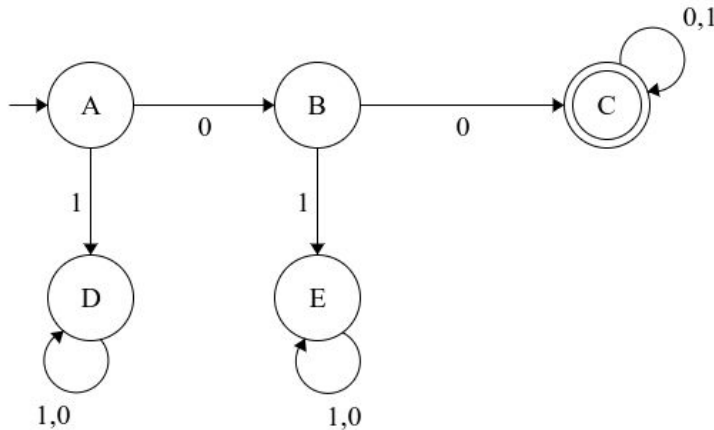
- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ starts with } 00 \}$



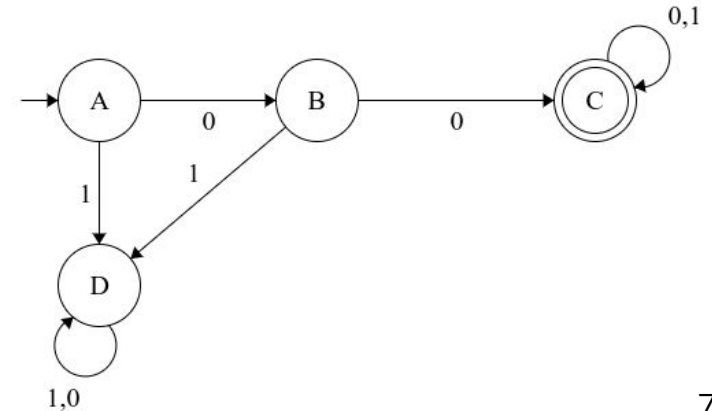
Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ starts with } 00 \}$



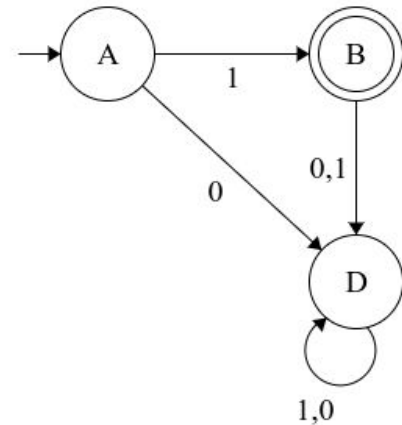
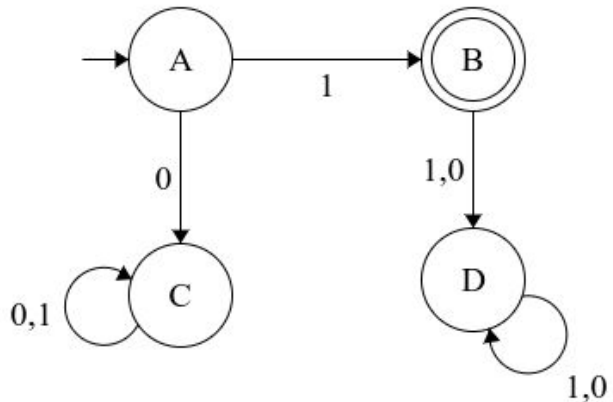
VS



Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ 1 \}$



Finite Automata

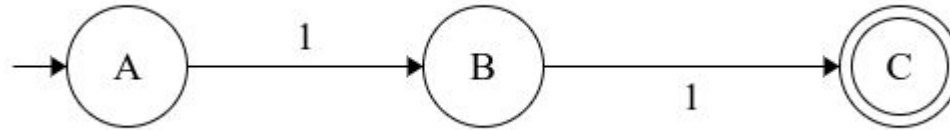
- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 11 \}$

Finite Automata

- **Designing Automata :**

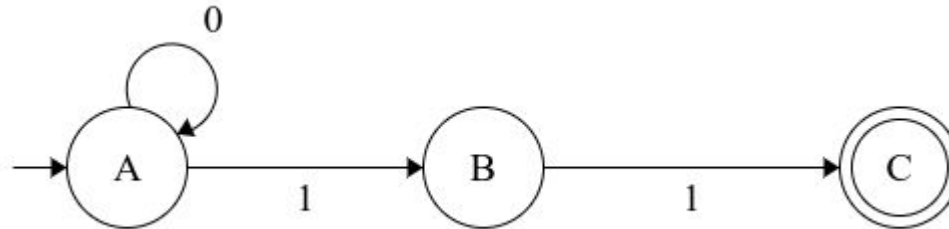
- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 11 \}$
 - *The following accepts 11*
 - *What about : **011** ? what to do ?*



Finite Automata

- **Designing Automata :**

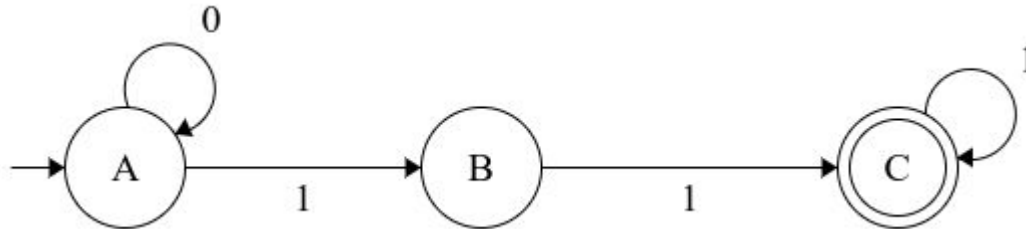
- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 11 \}$
 - *The following accepts 011*
 - *What about : **111** ? what to do ?*



Finite Automata

- **Designing Automata :**

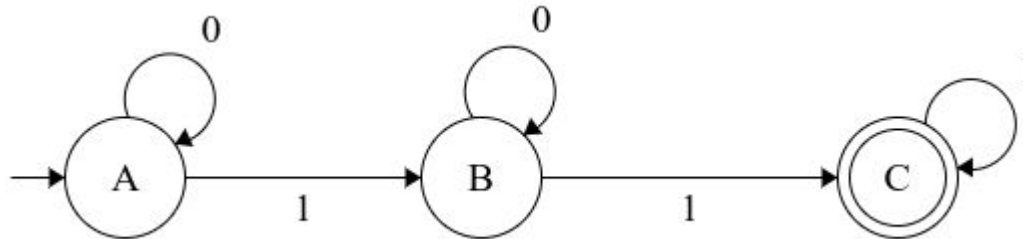
- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 11 \}$
 - *The following accepts 111*
 - *What about : **01011** ?*



Finite Automata

- **Designing Automata :**

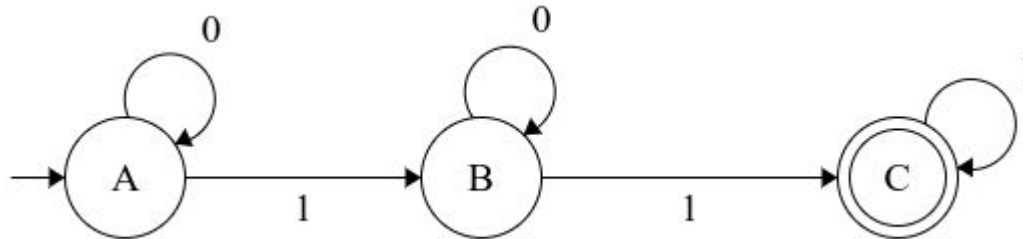
- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 11 \}$
 - *The following accepts **01011***
 - *What about : **000011100011** ?*



Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 11 \}$
 - *The following accepts **01011***
 - *What about : **000011100011** ?*

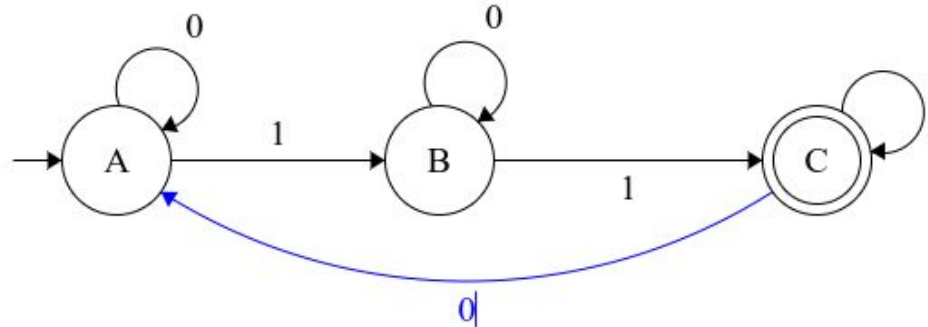


**Does not
work**

Finite Automata

- **Designing Automata :**

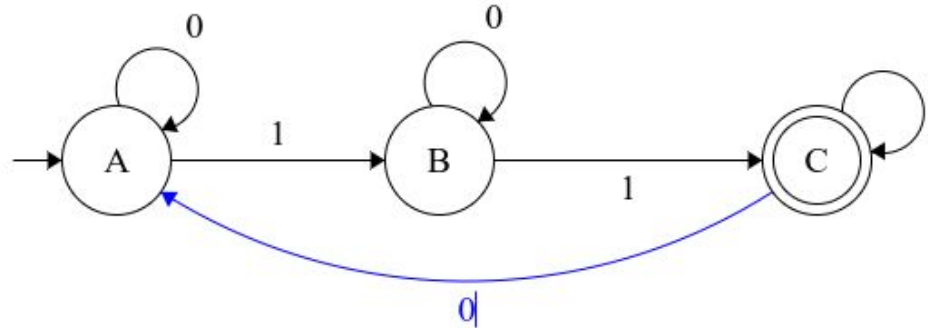
- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 11 \}$
 - *The following accepts **01011***
 - *What about : **000011100011** ? Check for further cases ?*



Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 11 \}$
 - *What about : **000011100011** ?*
 - *Let's Check further cases ? back to the simple word : **101** ?*

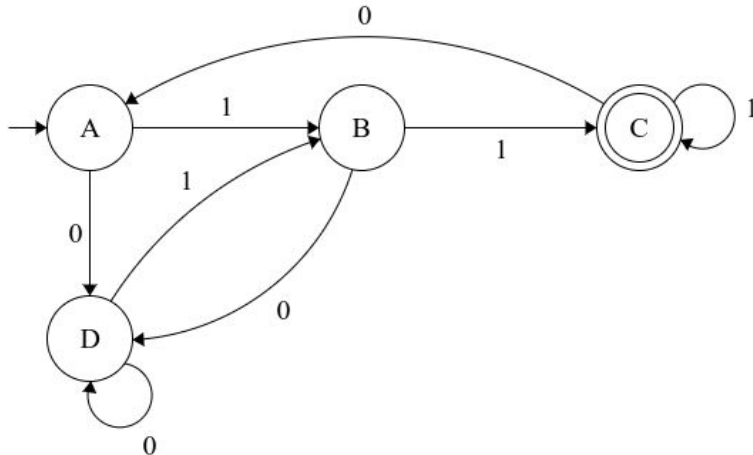


101 : is accepted by the machine but it is not part of the language

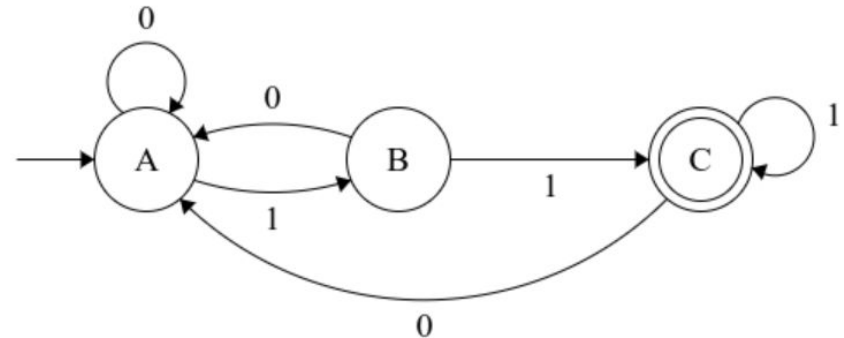
Finite Automata

- **Designing Automata :**

- Alphabet $\Sigma = \{ 0, 1 \}$
- Design M which recognizes $L = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 11 \}$



VS



Finite Automata

- **Designing Automata :**
 - Alphabet $\Sigma = \{ 0 , 1 \}$
 - Language L such that ..

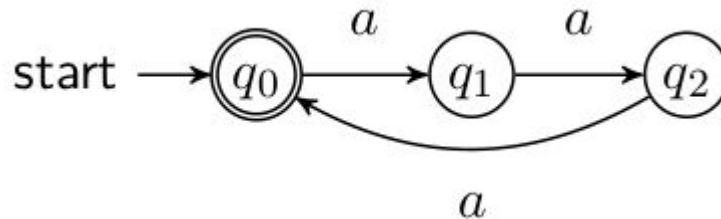
Finite Automata

- **Designing Automata :**
 - Alphabet $\Sigma = \{ 0 , 1 \}$
 - Language L such that ..solution

Finite Automata

- **DFA for a Complement of a Language :**

- M is the automaton for the language $L = \{ w \mid \text{the length of } w \text{ is divisible by } 3 \}$
- Alphabet is $\{ a \}$
- Language L: $\{, aaa, aaaaaa, aaaaaaaaaa, \dots \}$



Finite Automata

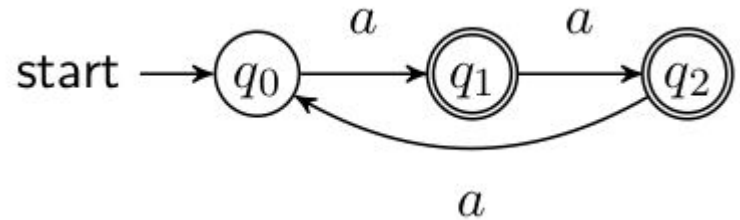
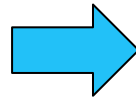
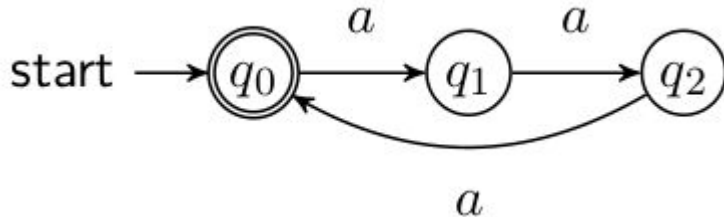
- **DFA for a Complement of a Language :**

- What's the automaton for the complement $L' = \{ w \mid \text{the length of } w \text{ is **not** divisible by 3} \}$
- Alphabet is $\{ a \}$
- Language $L = \{ a, aa, aaaaaa, aaaaaaaaaa, \dots \}$

Finite Automata

- **DFA for a Complement of a Language :**

- What's the automaton for the complement $L' = \{ w \mid \text{the length of } w \text{ is not divisible by } 3 \}$
- Alphabet is $\{ a \}$
- Language $L = \{ a, aa, aaaaaa, aaaaaaaaaa, \dots \}$



Finite Automata

- **DFA for a Complement of a Language :**
 - Given
 - $\mathbf{M} = (Q, \Sigma, \delta, q_0, F)$ over the alphabet Σ ,
 - its complement is : $\mathbf{M}' = (Q, \Sigma, \delta, q_0, Q-F)$ (Another notation of difference : \setminus)
 - Let $w \in \Sigma^*$, then $w \in L(M')$ **iff** $\delta^*(q_0, w) \in Q-F$
 - iff** $\delta^*(q_0, w) \in Q-F$
 - iff** $\neg (\delta^*(q_0, w) \in F)$
 - iff** $\neg (w \in L(M))$
 - iff** $w \in \Sigma^* - L(M)$

Minimization of Finite Automata

- The steps or the Algorithm to minimize deterministic finite automaton:
 - Find Unreachable states and remove them
 - *Any state that cannot be reached directly or indirectly from the start state*
 - For other remaining states: Create two groups : Accept vs Non-Accept:
 - *For each set W :*
 - *For each pair (B, V) of states in W :*
 - *See if B and V are **distinguishable** = their transitions for some symbol belongs to different **sets**.*
 - *If **distinguishable** → **create a new set** for the alien state(s)*
(Group alien states into a single set if their transition states belong to the same set)
 - Find Dead states and merge them if possible on the same input.

Minimization of Finite Automata

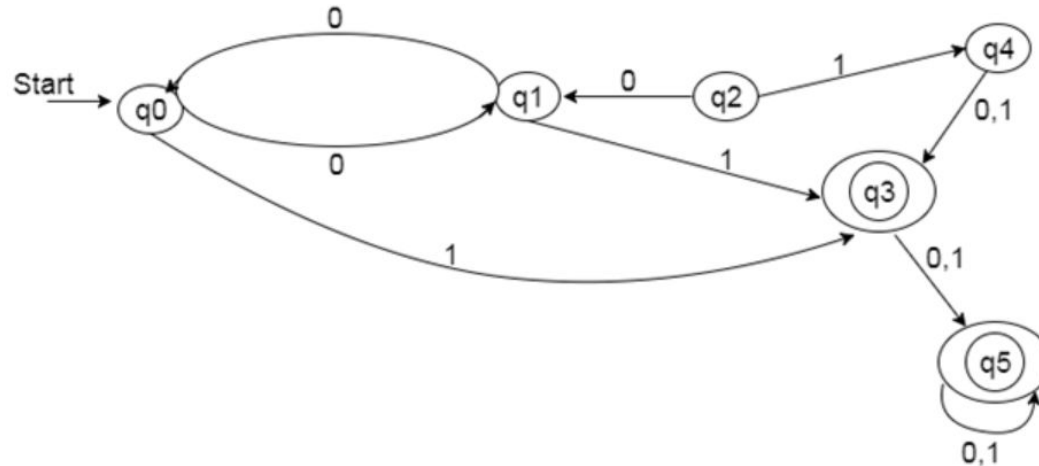


- Simpler version in plain English:
 - Group “equivalent” states into a single “region” or set:
 - Equivalent =
 - States equivalent if they lead to the same “region”
 - Region = set of equivalent states.
 - If elements in a grouped set/region are not equivalent:
 - We create a separate “region” for them.
 - We keep splitting regions until we are no able to split = all elements in each region are equivalent.

Minimization of Finite Automata

- **Finding Unreachable States**

- Q2 and Q4 are not reachable from Q0



Minimization of Finite Automata

- **Partition Algorithm**

- Two sets :

- $N = \{q_0, q_3, q_5\}$, $A = \{q_1, q_2, q_4\}$

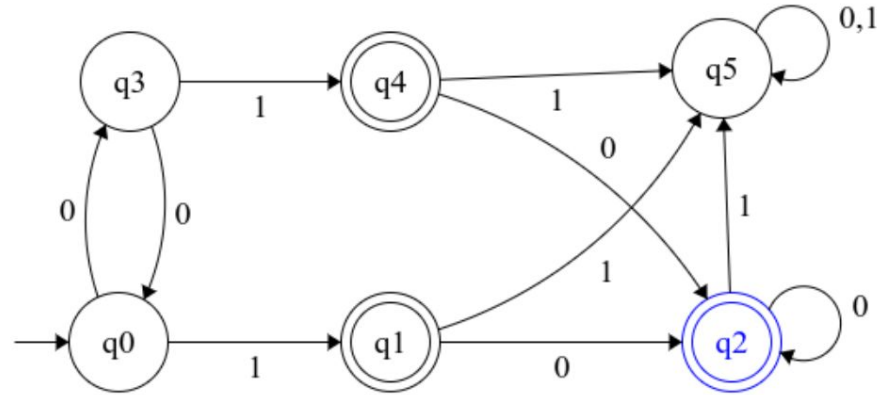
- Partitioning round 1:

- For $N = \{q_0, q_3, q_5\}$:

- For q_0 : { for 0 $\rightarrow q_3$, for 1 $\rightarrow q_1$ } $q_3 \in N$, $q_1 \in A$
- For q_3 : { for 0 $\rightarrow q_0$, for 1 $\rightarrow q_4$ } $q_0 \in N$, $q_4 \in A$
- For q_5 : { for 0 $\rightarrow q_5$, for 1 $\rightarrow q_2$ } $q_5 \in N$, $q_2 \in A$

- Equivalence Check : Finding states which are **not distinguishable**
- q_0 vs q_3 : for both, at 0 , results $\in N$, at 1, results $\in A \rightarrow$ equivalent.
- q_0 vs q_5 : at 0, both results $\in N$, **but for 1**, they belong to different sets \rightarrow **partitioning to a different set for q_5**

- $N \rightarrow N_1 = \{q_0, q_3\}$ and $N_2 = \{q_5\}$



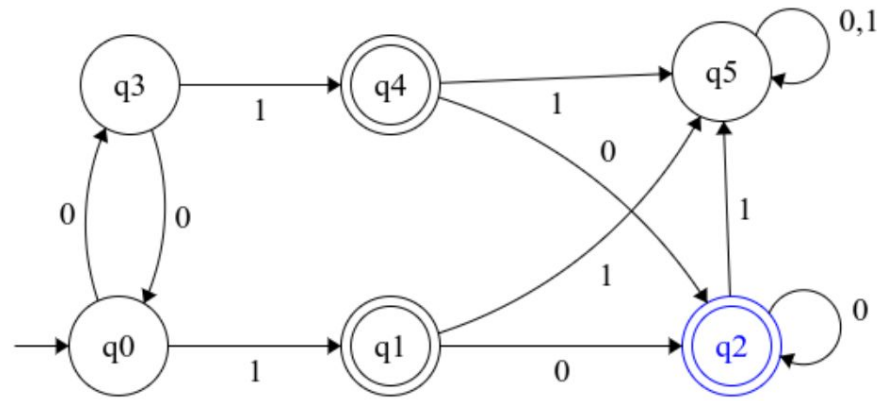
Minimization of Finite Automata

- **Partition Algorithm**

- Two sets :
 - $N1 = \{q0, q3\}$, $N2 = \{q5\}$, $A = \{q1, q2, q4\}$
- Partitioning round 1:
 - For $A = \{q1, q2, q4\}$:

- For $q1$: { for 0 $\rightarrow q2$, for 1 $\rightarrow q5$ } $q2 \in A$, $q5 \in N2$
- For $q2$: { for 0 $\rightarrow q2$, for 1 $\rightarrow q5$ } $q2 \in A$, $q5 \in N2$
- For $q4$: { for 0 $\rightarrow q2$, for 1 $\rightarrow q5$ } $q2 \in A$, $q5 \in N2$
 - Equivalence Check : Finding states which are **not distinguishable**
 - $q1$ vs $q2$: for both, at 0 , results $\in A$, at 1, results $\in N2 \rightarrow$ equivalent.
 - $q1$ vs $q4$: for both, at 0 , results $\in A$, at 1, results $\in N2 \rightarrow$ equivalent.
 - $q2$ vs $q4$: for both, at 0 , results $\in A$, at 1, results $\in N2 \rightarrow$ equivalent.

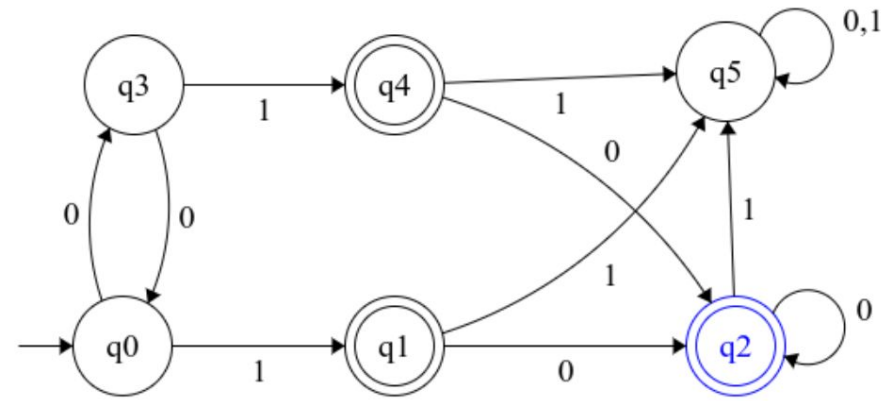
■ **No partitioning is needed**



Minimization of Finite Automata

- **Partition Algorithm**

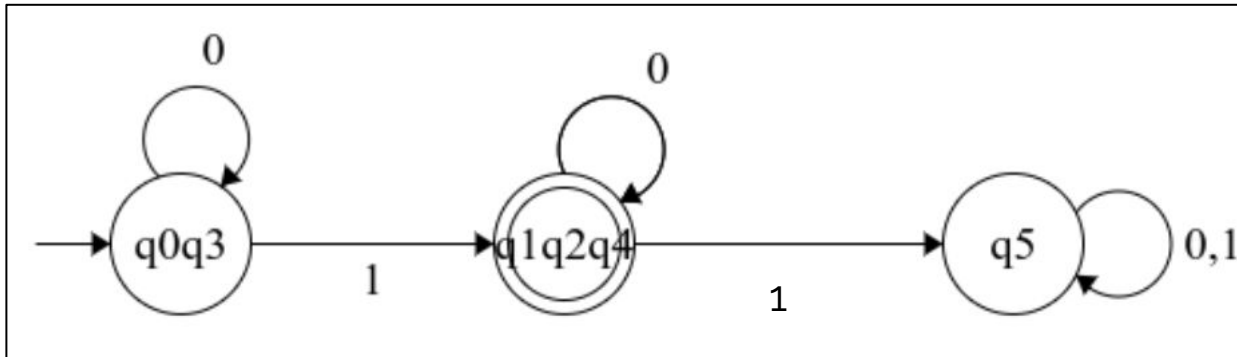
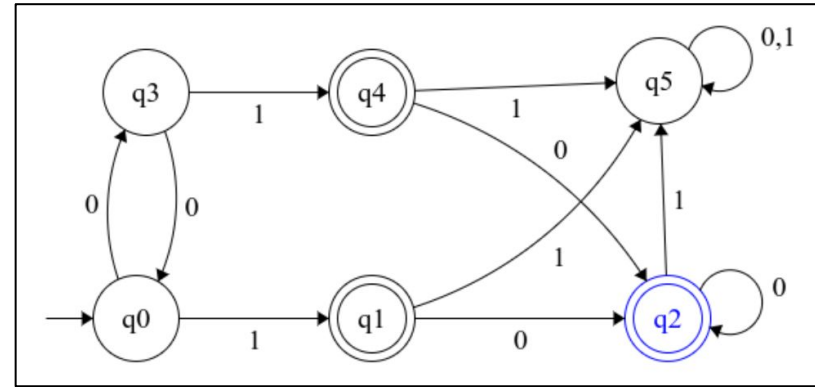
- Two sets :
 - $N_1 = \{q_0, q_3\}$, $N_2 = \{q_5\}$, $A = \{q_1, q_2, q_4\}$
 - Partitioning round 2:
 - For $N_1 = \{q_0, q_3\}$:
 - For q_0 : { for 0 $\rightarrow q_3$, for 1 $\rightarrow q_1$ } $q_3 \in N$, $q_1 \in A$
 - For q_3 : { for 0 $\rightarrow q_0$, for 1 $\rightarrow q_4$ } $q_0 \in N$, $q_4 \in A$
 - Equivalence Check : Finding states which are **not distinguishable**
 - q_0 vs q_3 : for both, at 0 , results $\in N$, at 1, results $\in A \rightarrow$ equivalent.
 - **No partitioning is needed**
 - For N_2 , no need, it a single set..
- Partitioning round 3 : no need as the no further partitioning is conducted.



Minimization of Finite Automata

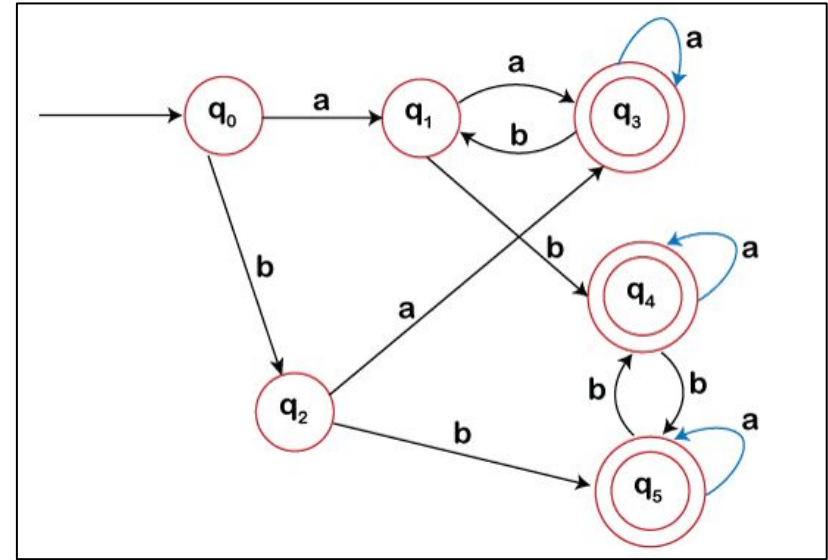
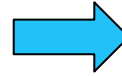
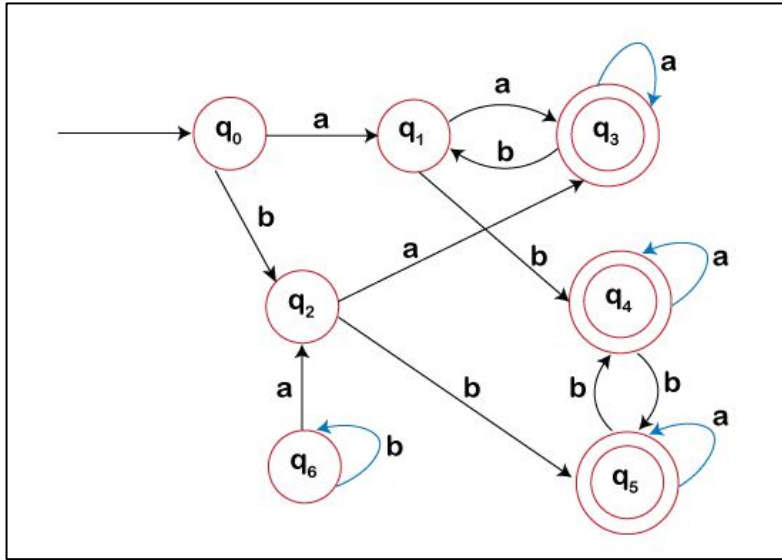
- **Partitioning Algorithm**

- The new states:
 - $\{q_0, q_3\}$ by merging : q_0 and q_3
 - $\{q_5\}$
 - $\{q_1, q_2, q_4\}$ by merging three states:



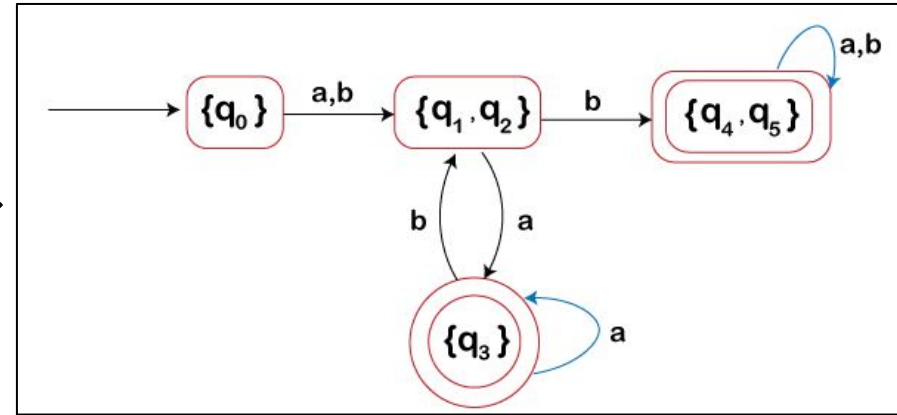
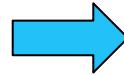
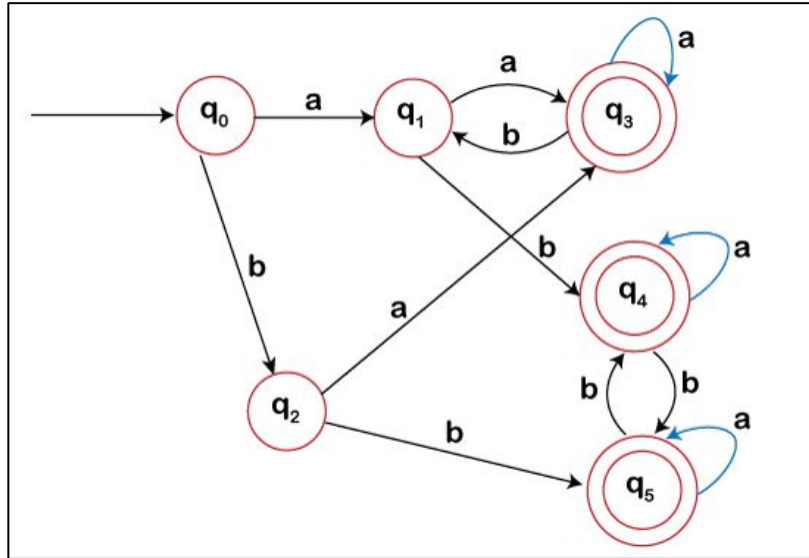
Minimization of Finite Automata

- Another Example



Minimization of Finite Automata

- Another Example





Questions

- How to tell if two different automata correspond to the same language ?
- Provided that you have designed the finite automaton for validating email address. How to put it into action to extract all valid emails from a given text