CS 302.1 - Automata Theory

Shantanav Chakraborty

Center for Quantum Science and Technology (CQST)

Center for Security, Theory and Algorithms (CSTAR)

IIIT Hyderabad

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Introduction

- Obtain models of computation : simple to more powerful ones.

- What are the limits of computational models?

Consider an (extremely) simple robot which

- has a button that turns it ON and OFF

- once turned on, can either move forward or backwards

- has a sensor that recognizes an obstacle and reverses the direction of the robot.

Introduction

Consider an (extremely) simple robot which

- has a button that turns it ON and OFF
- once turned on, can either move forward or backwards
- has a sensor that recognizes an obstacle and reverses the direction of the robot.

States : {OFF, FORWARD, BACKWARD}

Inputs : {BUTTON, SENSOR}

Initial state: OFF

By accepting an INPUT (signal), the robot TRANSITIONS from one state to another

Introduction

States : {OFF, FORWARD, BACKWARD}

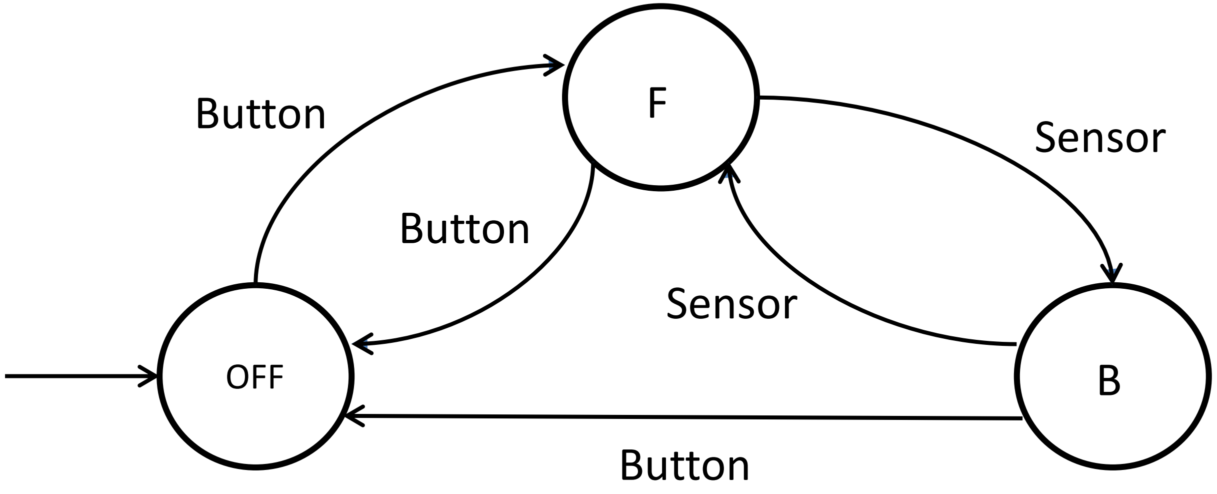
Inputs : {BUTTON, SENSOR}

Initial state: OFF

By accepting an INPUT (signal), the robot TRANSITIONS from one state to another

	BUTTON	SENSOR
OFF	F	X
F	OFF	B
B	OFF	F

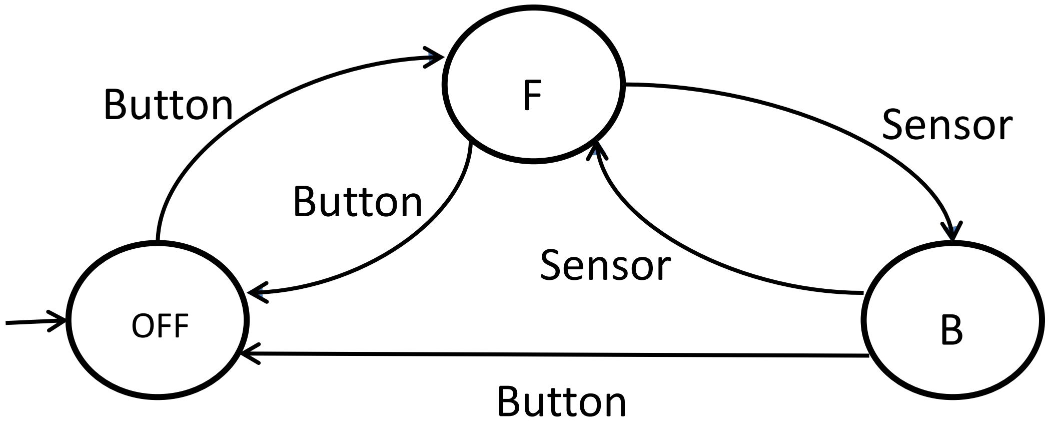
State Transition Table



State diagram for the robot

Introduction

	BUTTON	SENSOR
OFF	**F**	**X**
F	**OFF**	**B**
B	**OFF**	**F**



- Often computational tasks do not require an all powerful computer
- Examples: this robot, elevators, automatic doors, vending machines, ATMs etc.
- Design computational models with varying degrees of power and classify them.
- For a particular computational model, try to classify all the *problems* that can be solved by the model and those that can't be.

Introduction

In this course, we will ask questions such as :

- Can a given problem be computed by a particular computational model?

Let us explore what is meant by this.

Problem	Problem Instance
$\int f(x)dx$	$\int \sin x\, dx$
Sorting	$\frac{\pi}{3}, \frac{1}{2}, 2,$................

Problem vs a specific instance of a problem

Problem vs decision problem: In order to answer these questions, we will always convert a given problem into a *decision* (YES-NO) *problem.* We will always do this!

Introduction

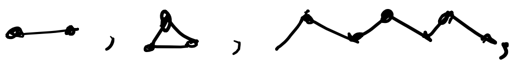
- Can a given problem be computed by a particular computational model?

Problem vs decision problem: In order to answer these questions, we will always convert a given problem into a *decision* (YES-NO) *problem.* We will always do this!

Problem	Decision problem
Sorting	Is the array sorted?
Graph connectivity	Is the graph connected?

By converting a problem into a decision problem is that we obtain two sets :

A YES set containing all the *instances* where the answer is YES.
A NO set containing all the *instances* where the answer is NO.

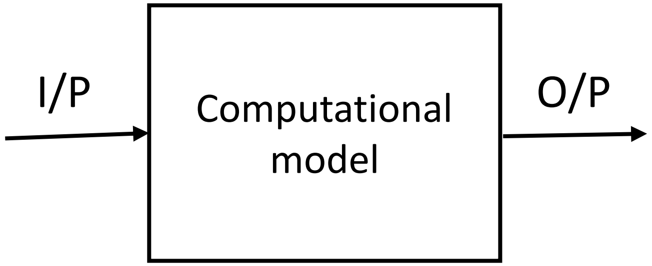
Problem: Graph Connectivity
YES set : {  ,.......}
NO set : {  ……}

Given an input instance, the computer can simply check to which set it belongs to and output accordingly.

Introduction

In this course, we will also ask questions such as :

- Can a given problem be computed by a particular computational model?



A computational model solves a problem P if,

(i) For all inputs belonging to the YES instance of P, the device outputs **YES**

AND

(ii) For all inputs belonging to the NO instance of P, the device outputs **NO**.

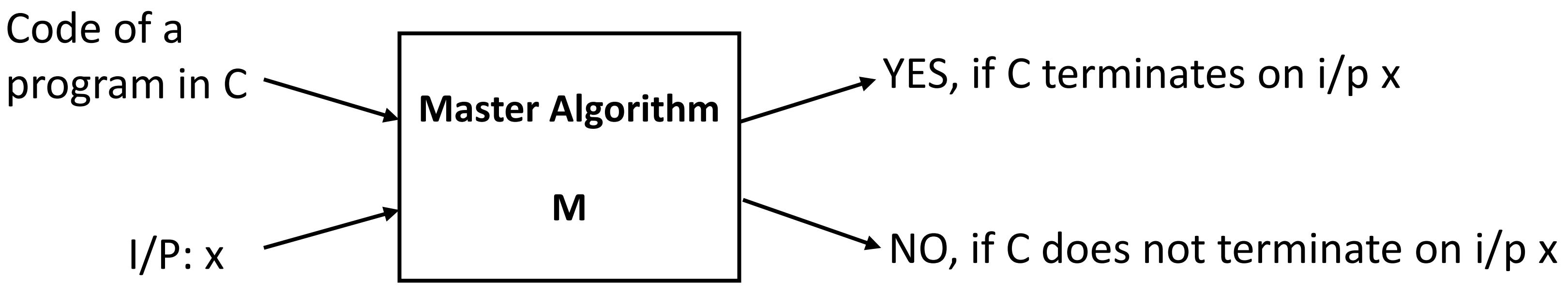
If (i) and (ii) hold, we say that the problem **P is computable** by this computational model.

Introduction

What are the limits of computability?

Can we have problems that cannot be solved by ANY computer, no matter how powerful?

Example 1: Master Algorithm

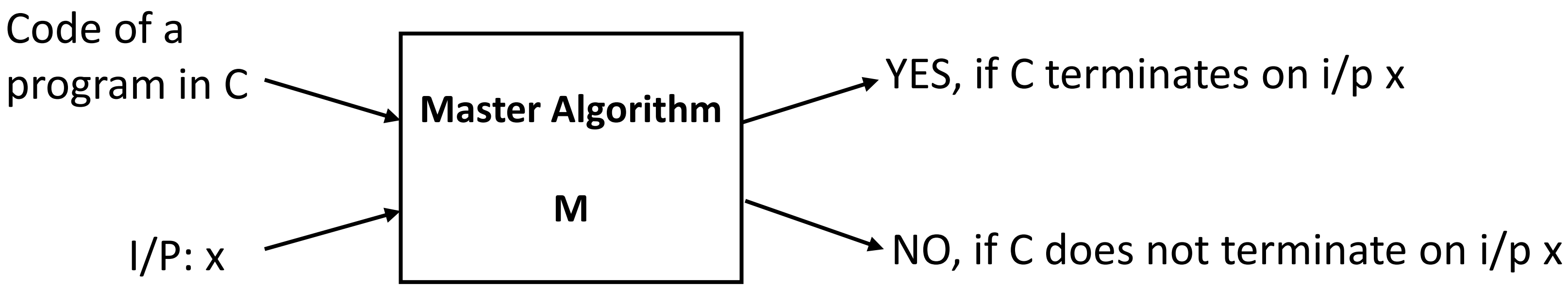


Introduction

What are the limits of computability?

Can we have problems that cannot be solved by ANY computer, no matter how powerful?

Example 1: Master Algorithm

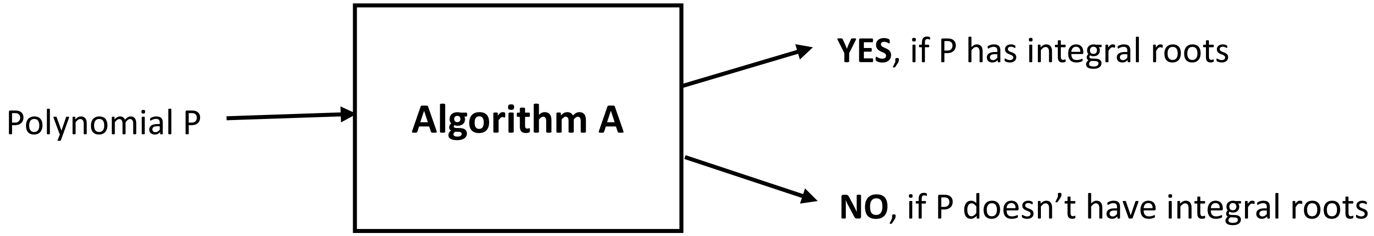


- **M terminates and outputs NO even if C(x) runs infinitely!**
- No such Algorithm M can be written. **Undecidable problem!**

Key takeaway: There are problems that are **not computable**.

Introduction

Example 2: Does a polynomial P(x,y) with integral coefficients have integral roots?



Eg: Input Polynomial P: $x^3y^2 + xy^2 + 3x - 5 = 0$ O/P : YES as (-1,1) are solutions to P

- The algorithm A proceeds by checking whether for integers $0, \pm 1, \pm 2, \cdots$. It terminates and outputs YES, whenever it finds the roots.
- What if P does not have integral roots? Algorithm A will run forever and will never terminate to output NO.
- **Undecidable problem! Key takeaway:** There are problems that are **not computable.**

Introduction

In this course we will:

- We will consider different computational models and classify them based on the problems they can solve
- Start from simple models and gradually increase their power to accommodate real computers
- Identify the problems that are not computable.

In this course we will not:

- Deal with how much time or space (memory) an algorithm would need to solve a certain problem
- Classifying the hardness of computable problems falls under the purview of Complexity Theory

Course Structure

- ❖ 12 Lectures in all

- ❖ Final Exam at the end (**35% weightage**)

- ❖ Two theory assignments (**20% weightage**)
  - Assignment 1 – released after Lec 3 (Deadline: 10 days)
  - Assignment 2 – released after Lec 9 (Deadline: 15 days)

- ❖ Programming assignment **(25% weightage)**
  - Assignment 1 – Released after Lec 3 (Deadline: End of sem)
  - Assignment 2 – Released after Lec 5 (Deadline: 10 days)

- ❖ Quiz (**20% weightage**)

Tutorials and TAs

- Tutorial sessions weekly: Thursday 3:30 PM – 5 PM.
- Teaching Associates:
  - **Aditya Morolia** (aditya.morolia@research.iiit.ac.in)
  - **Aakash Aanegola** (aakash.aanegola@students.iiit.ac.in)
  - **Ashwin Mittal** (ashwin.mittal@students.iiit.ac.in)
  - **Zeeshan Ahmed** (zeeshan.ahmed@research.iiit.ac.in)
  - **Aakash Jain** (aakash.jain@students.iiit.ac.in)
  - **Shaurya Dewan** (shaurya.dewan@students.iiit.ac.in)
  - **Alapan Chaudhuri** (alapan.chaudhuri@research.iiit.ac.in)

- Tutorial sessions **are not** just going to be doubt clearing sessions.
- Several **interesting topics** will be covered.
- **My email:** shchakra@iiit.ac.in
- **Lecture slides available at my homepage:** https://sites.google.com/view/shchakra/teaching

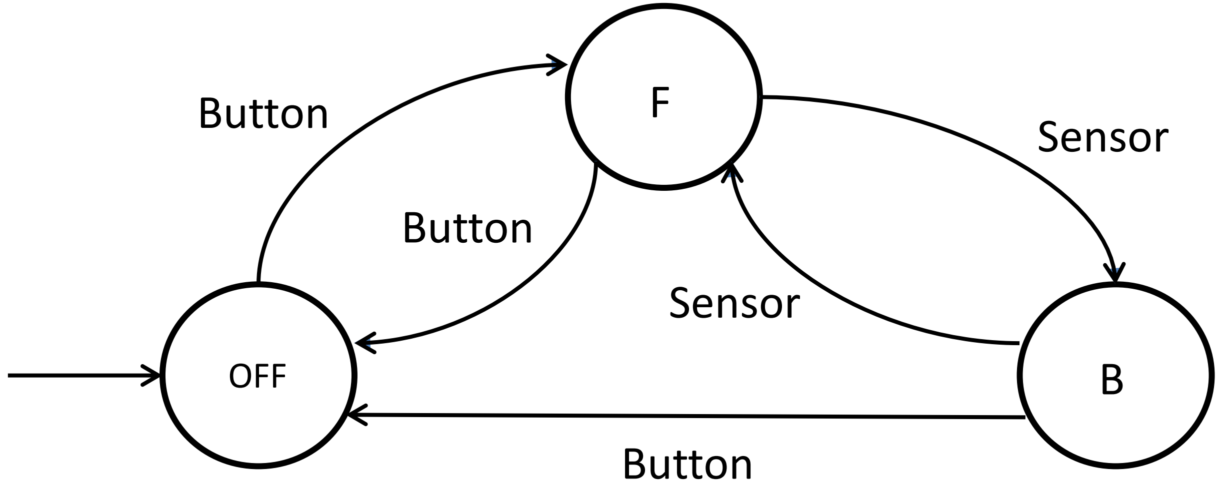
Some terminology

Alphabet	Strings/Words	Language
Any finite, non-empty set of symbols	Finite sequence of symbols from an alphabet.	Set of words/strings from the current alphabet
$\Sigma_1 = \{0,1\}$	0110, 000, 10, 10000,……	Even numbers
$\Sigma_2 = \{a, b, c, \dots, z\}$	any, word, revolution,…..	English

Generally the empty string is denoted by ϵ

Models of computation

- Deterministic Finite Automata (DFA) Model

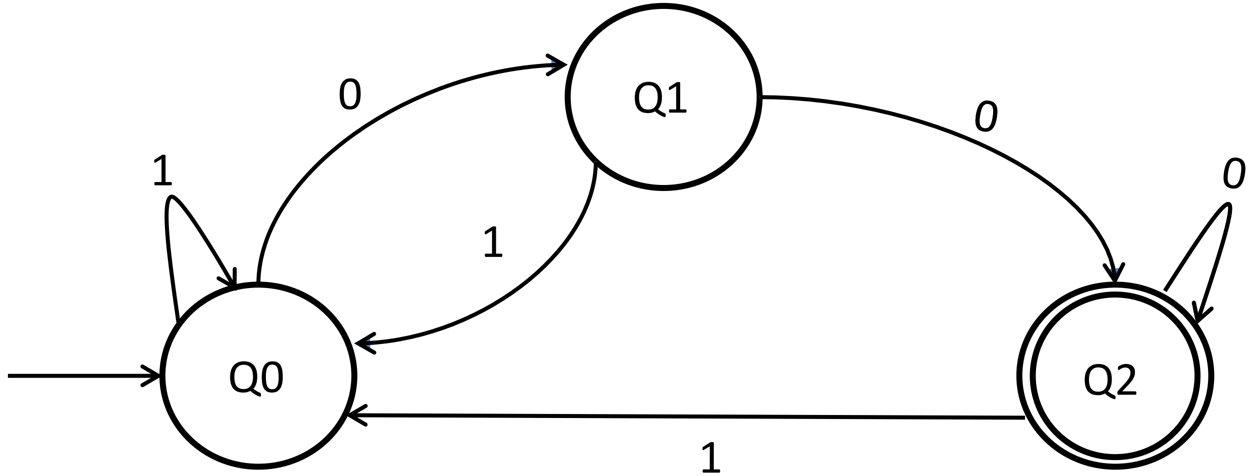


Characteristics: (i) Single start State, (ii) Unique Transitions, (iii) Zero or more final states

Models of computation

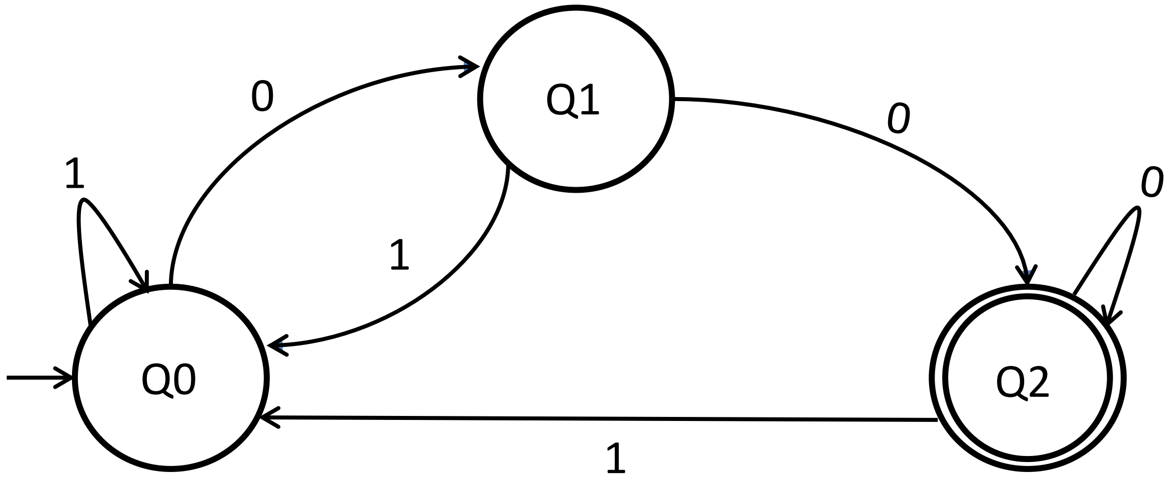
- Deterministic Finite Automata (DFA) Model

Q0: Start state, Q2: Final state



State transition diagram of the Finite State Machine

Deterministic Finite Automata (DFA)

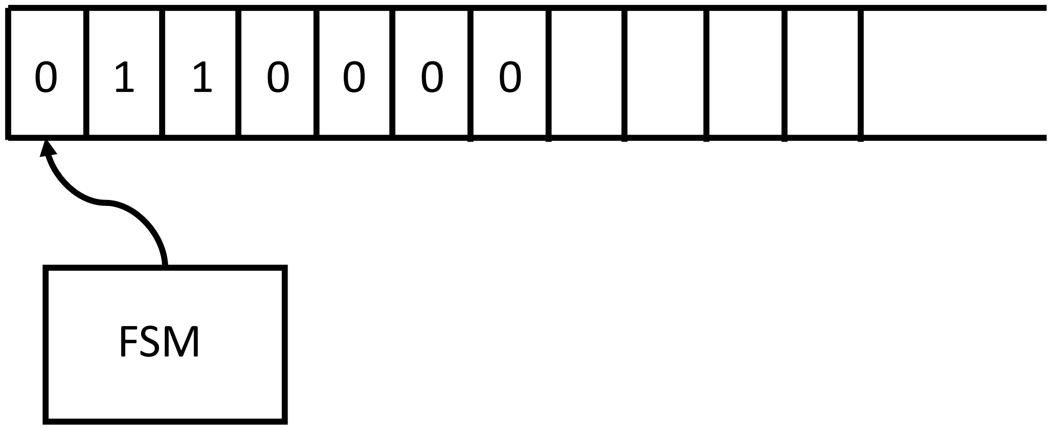


State transition diagram of the Finite State Machine

Input: Strings from alphabet $\Sigma = \{0,1\}$

Q0: Start state, Q2: Final state

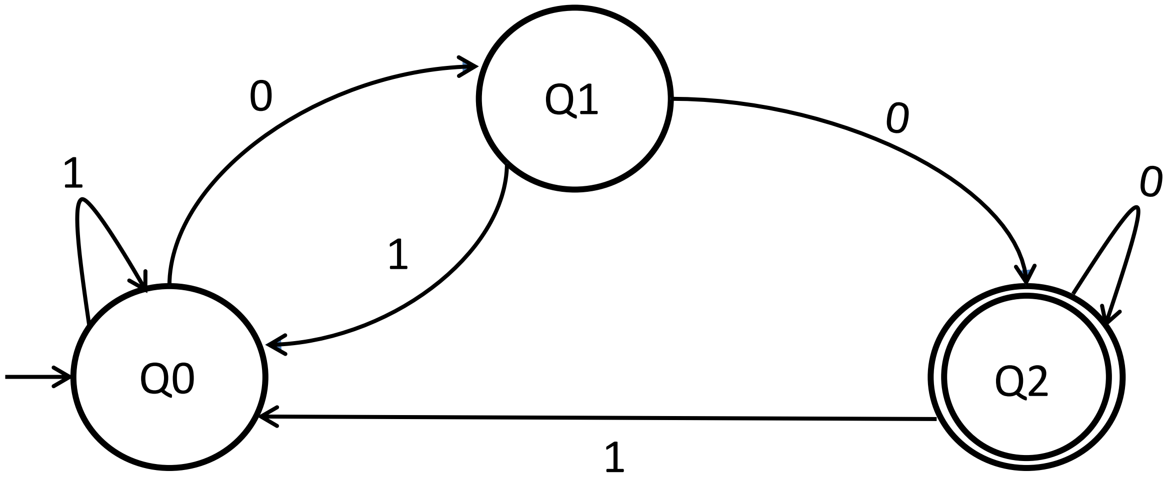
One-way infinite tape



Run:

$$\boldsymbol{Q0} \xrightarrow{0} Q1 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{0} Q2 \xrightarrow{0} Q2 \xrightarrow{0} \boldsymbol{Q2}$$

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

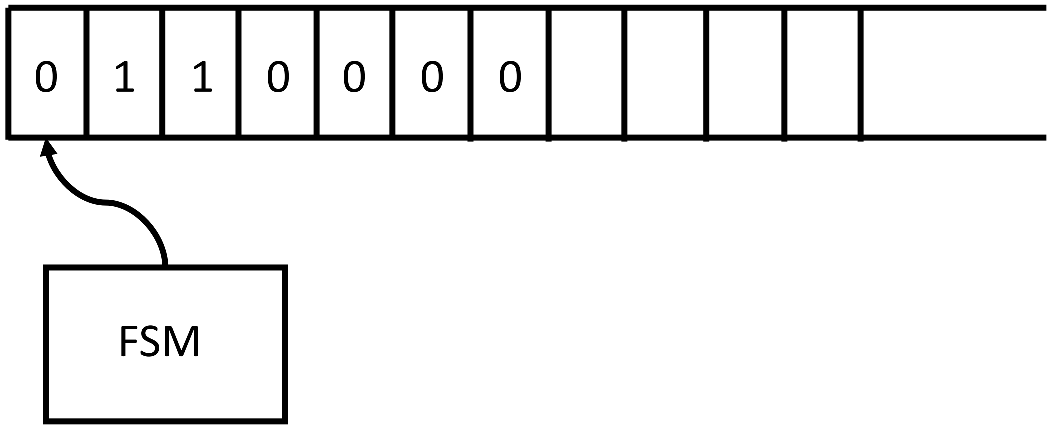
Input: Strings from alphabet $\Sigma = \{0,1\}$

Q0: Start state, Q2: Final state

The DFA "accepts" an input string, if it corresponds to a *run* that ends up in the final state Q2. **(Accepting Run)**

The DFA "rejects" an input string, if it corresponds to a *run* that ends up in any non-final state. **(Rejecting Run)**

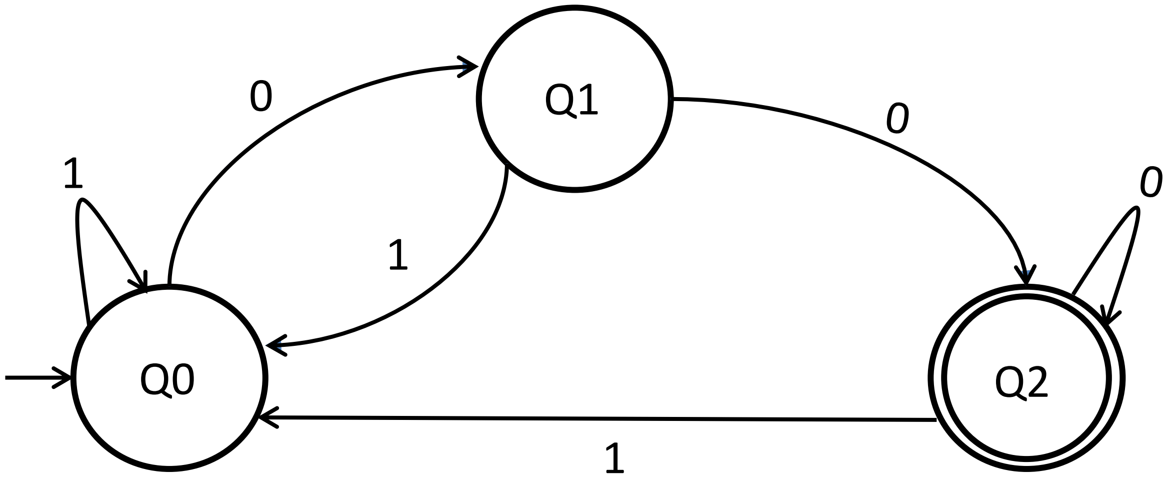
One-way infinite tape



Run:

$$\boldsymbol{Q0} \xrightarrow{0} Q1 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{0} Q2 \xrightarrow{0} Q2 \xrightarrow{0} \boldsymbol{Q2}$$

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

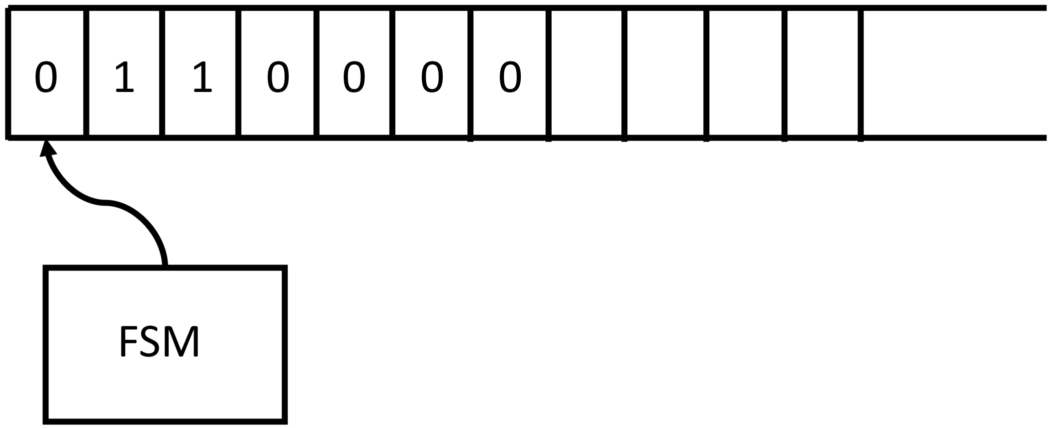
Input: Strings from alphabet $\Sigma = \{0,1\}$

Q0: Start state, Q2: Final state

The DFA “accepts” an input string, if it corresponds to a *run* that ends up in the final state Q2. **(Accepting Run)**

The DFA “rejects” an input string, if it corresponds to a *run* that ends up in any non-final state. **(Rejecting Run)**

One-way infinite tape

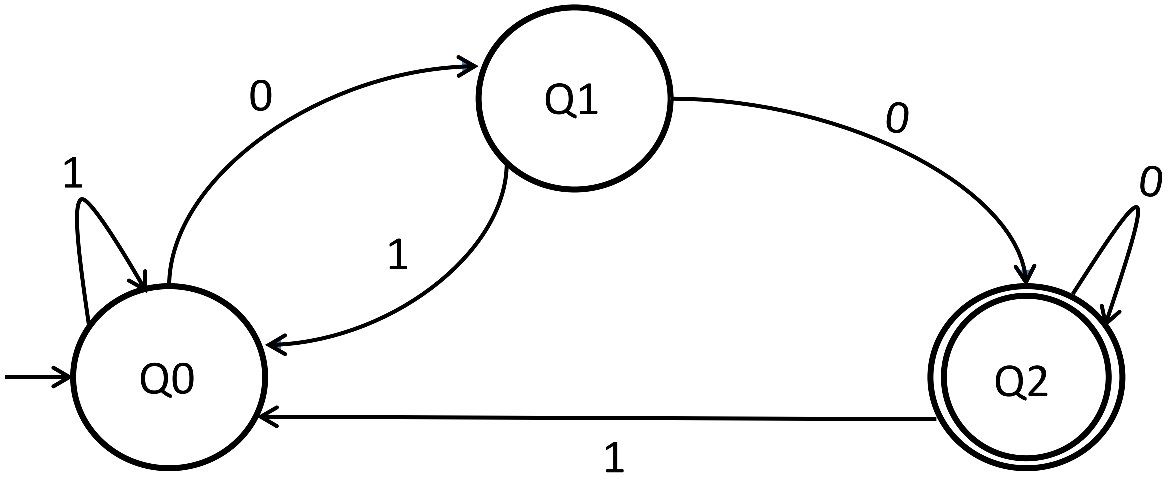


Run:

$$\boldsymbol{Q0} \xrightarrow{0} Q1 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{0} Q2 \xrightarrow{0} Q2 \xrightarrow{0} \boldsymbol{Q2}$$

ACCEPT = {0111000, }
REJECT = {}

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

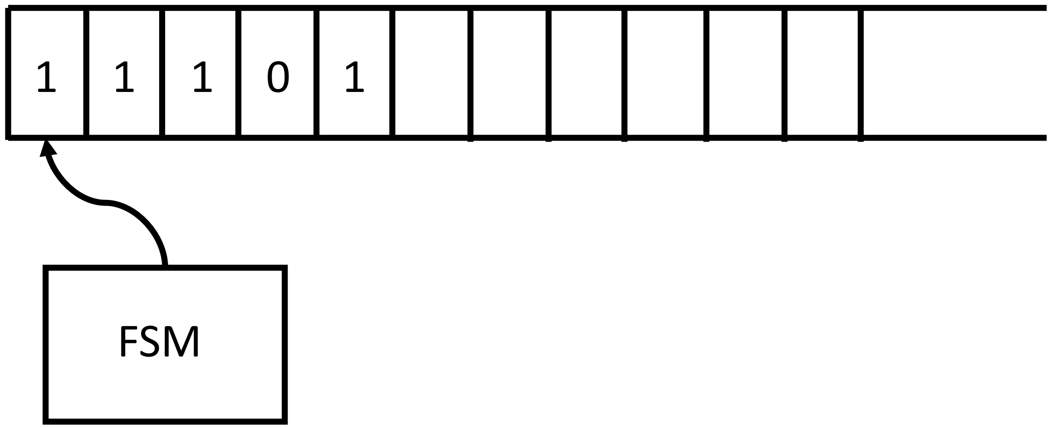
Input: Strings from alphabet $\Sigma = \{0,1\}$

Q0: Start state, Q2: Final state

The DFA "accepts" an input string, if it corresponds to a *run* that ends up in the final state Q2. **(Accepting Run)**

The DFA "rejects" an input string, if it corresponds to a *run* that ends up in any non-final state. **(Rejecting Run)**

One-way infinite tape

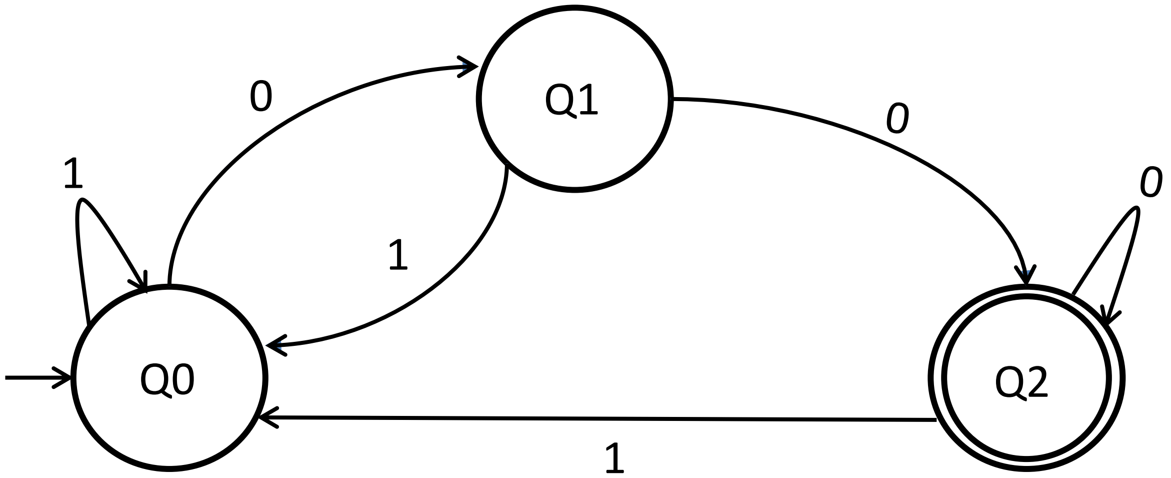


Run:

$$\boldsymbol{Q0} \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{1} \boldsymbol{Q0}$$

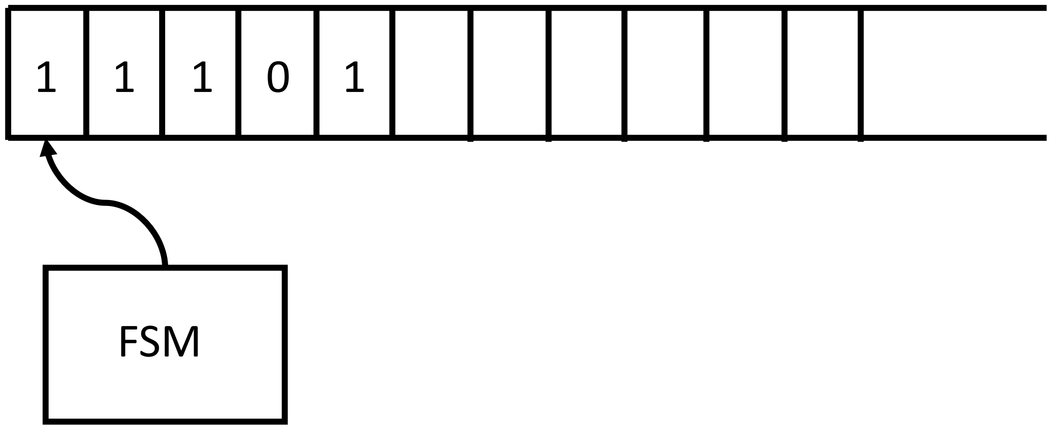
ACCEPT = {0111000, }
REJECT = {}

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



Input: Strings from alphabet $\Sigma = \{0{,}1\}$

Q0: Start state, Q2: Final state

The DFA “accepts” an input string, if it corresponds to a *run* that ends up in the final state Q2. **(Accepting Run)**

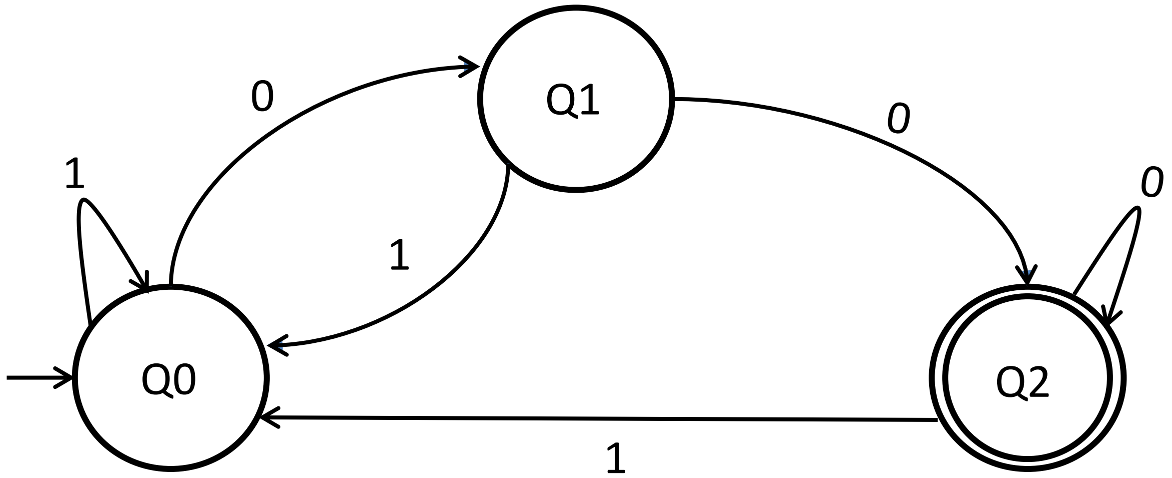
The DFA “rejects” an input string, if it corresponds to a *run* that ends up in any non-final state. **(Rejecting Run)**

Run:

$$\boldsymbol{Q0} \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{1} \boldsymbol{Q0}$$

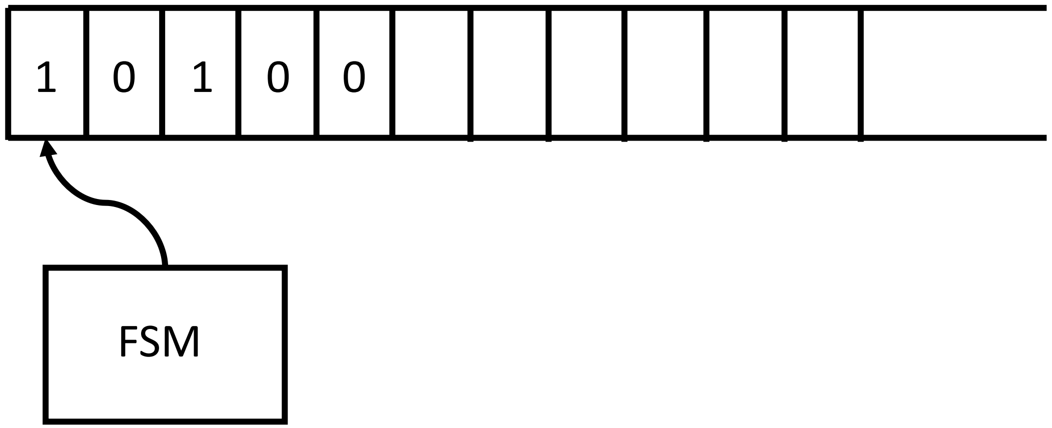
ACCEPT = {0111000, }
REJECT = {11101, }

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



Input: Strings from alphabet $\Sigma = \{0,1\}$

Q0: Start state, Q2: Final state

The DFA "accepts" an input string, if it corresponds to a *run* that ends up in the final state Q2. **(Accepting Run)**

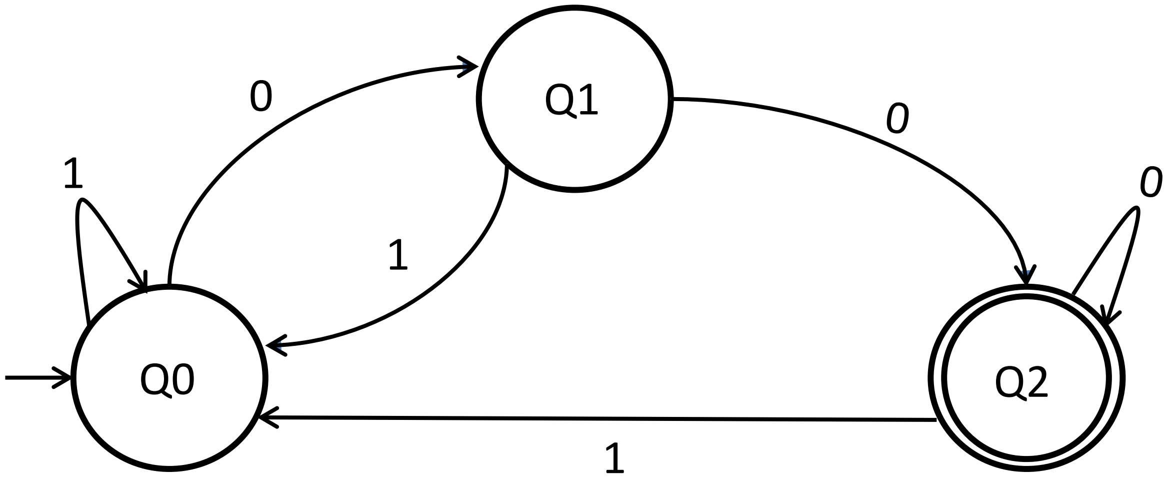
The DFA "rejects" an input string, if it corresponds to a *run* that ends up in any non-final state. **(Rejecting Run)**

Run:

$$\boldsymbol{Q0} \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{0} \boldsymbol{Q2}$$

ACCEPT = {0111000, 10100,….}
REJECT = {11101, …….}

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

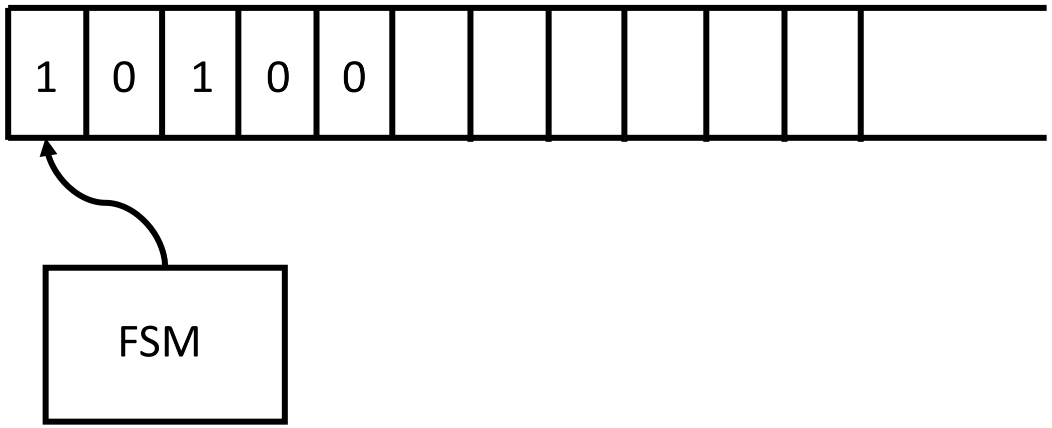
Input: Strings from alphabet $\Sigma = \{0,1\}$

Q0: Start state, Q2: Final state

The DFA "accepts" an input string, if it corresponds to a *run* that ends up in the final state Q2. **(Accepting Run)**

The DFA "rejects" an input string, if it corresponds to a *run* that ends up in any non-final state. **(Rejecting Run)**

One-way infinite tape

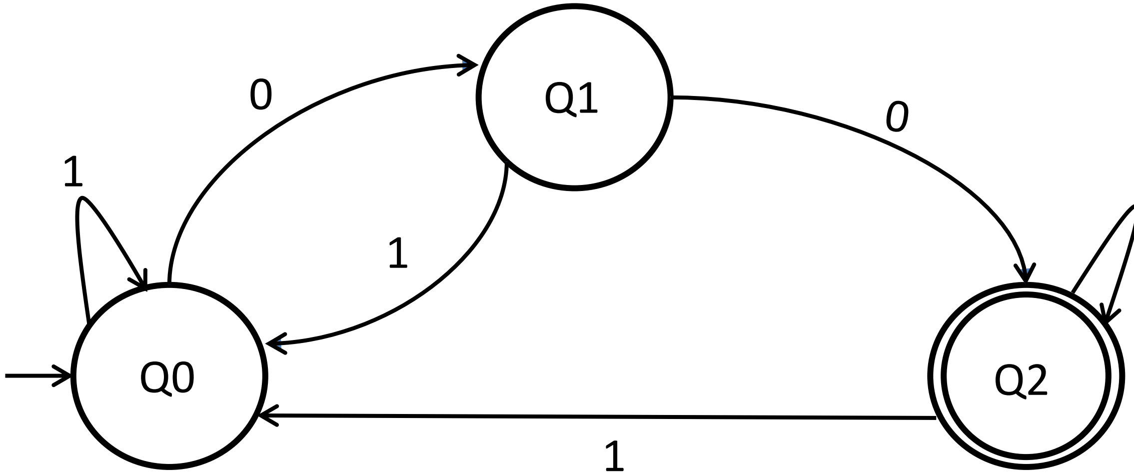


Run:

$$\boldsymbol{Q0} \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{0} \boldsymbol{Q2}$$

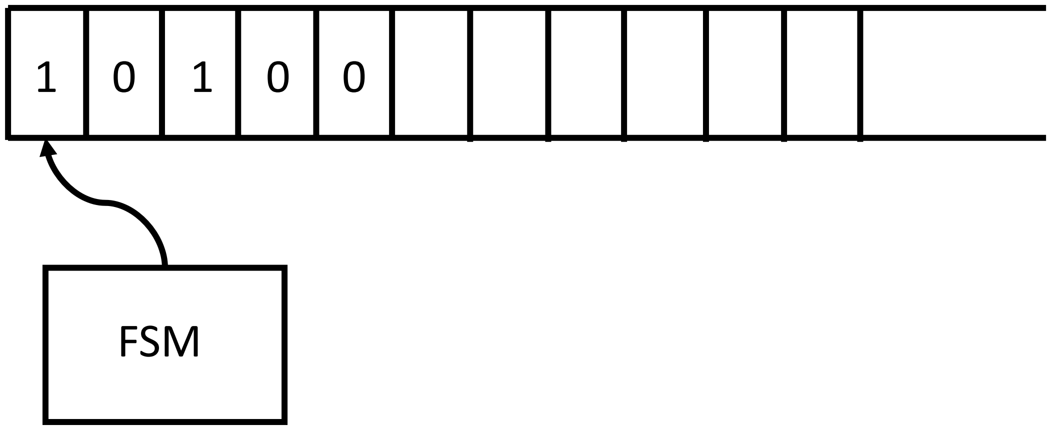
ACCEPT = {0111000, 10100, 0100, 00, 10000….}
REJECT = {11101, 0, 1, 11, 001,…….}

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



ACCEPT = {0111000, 10100, 0100, 00, 10000….}
REJECT = {11101, 0, 1, 11, 001,…….}

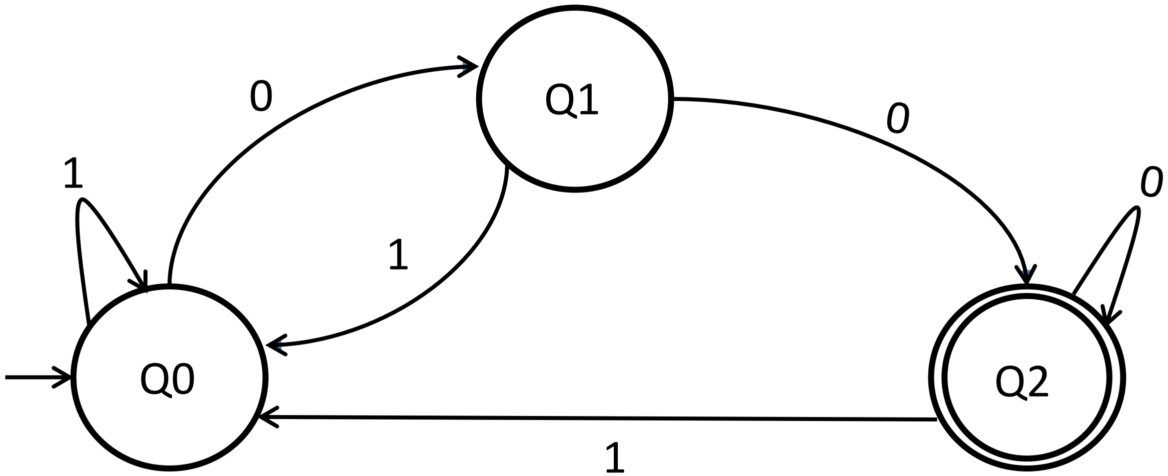
Let the DFA be M. Then, **the problem M solves/ language M accepts is**

L(M) = {$\omega|\omega$ results in an accepting run}

Equivalently, **M recognizes L**

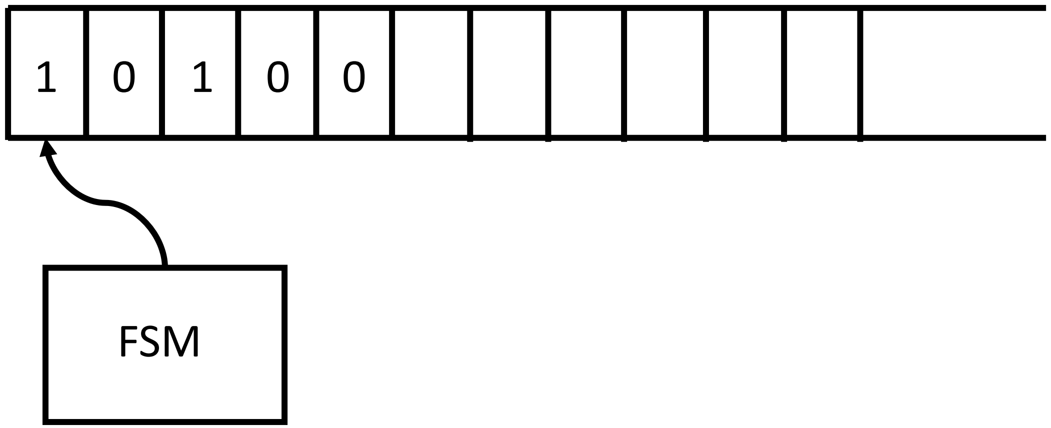
For the example above, **L(M) = {$\omega|\omega$ ends in “00”}**

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



Characteristics of DFA : (i) Single start state (ii) Unique transitions (iii) Zero or more final states

Formally, a finite automaton M is a 5-tuple $(\mathrm{Q}, \Sigma, \delta, q_0, F)$ where

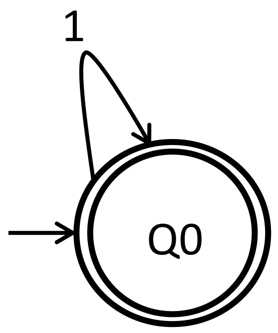
- Q is a finite set called the ***states.*** Q = {Q0, Q1, Q2}
- Σ is a finite set called the ***alphabet***. $\Sigma = \{0,1\}$
- $\delta: Q \times \Sigma \mapsto Q$ is the ***transition function*** *(unique)*. $(Q0,0) \mapsto Q1;\ (Q0,1) \mapsto Q0, \ldots\ldots, (Q2,1) \mapsto Q0$
- $q_0 \in Q$ is the ***start state***. $q_0 = Q0$
- $F \subseteq Q$ are the ***final/accepting states***. $F = Q2$

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma = \{0, 1\}}$, **L(M)={$\omega|\omega$ has an even number of 0's}**

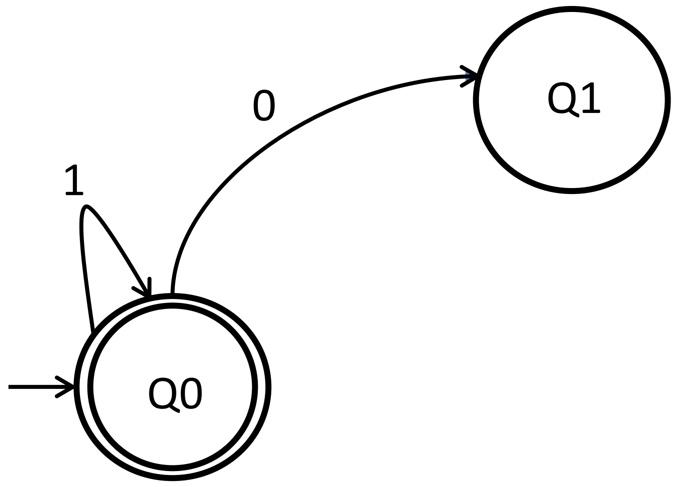
Constructing DFA for a language

Examples: $\boldsymbol{\Sigma = \{0, 1\}}$, **L(M)={$\boldsymbol{\omega | \omega}$ has an even number of 0's}**



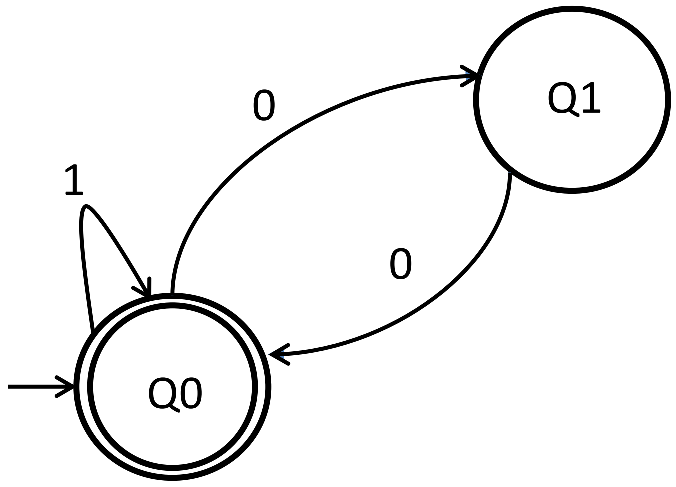
Constructing DFA for a language

Examples: $\mathbf{\Sigma = \{0, 1\}}$, **L(M)={$\omega|\omega$ has an even number of 0's}**



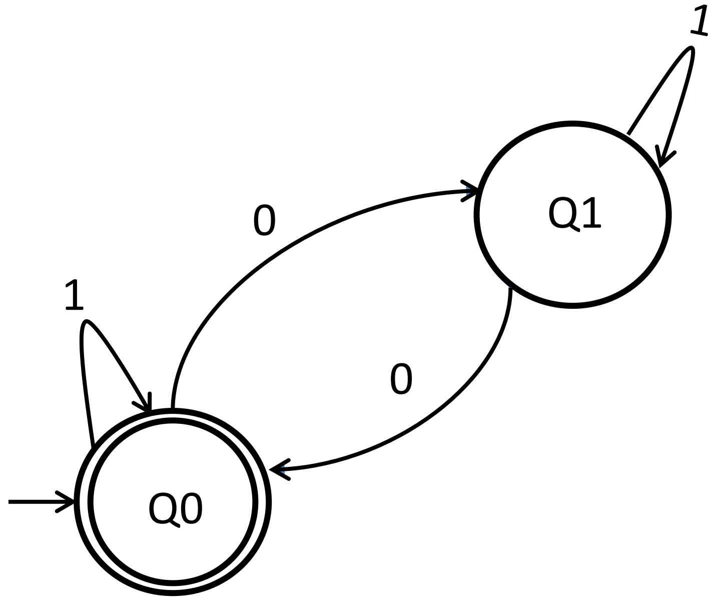
Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}|\boldsymbol{\omega}$ has an even number of 0's}**



Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\omega | \omega$ has an even number of 0's}**



	0	**1**
Q0	**Q1**	**Q0**
Q1	**Q0**	**Q1**

Constructing DFA for a language

Examples: $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}$|$\boldsymbol{\omega}$ is divisible by 3}**

Any input string would leave three remainders: $0, 1$ or 2.

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}$|$\boldsymbol{\omega}$ is divisible by 3}**

Any input string would leave three remainders: $0, 1$ or 2.

Intuition: Let ω be any substring of the input string divisible by 3, i.e. $\omega = 0(mod\ 3)$

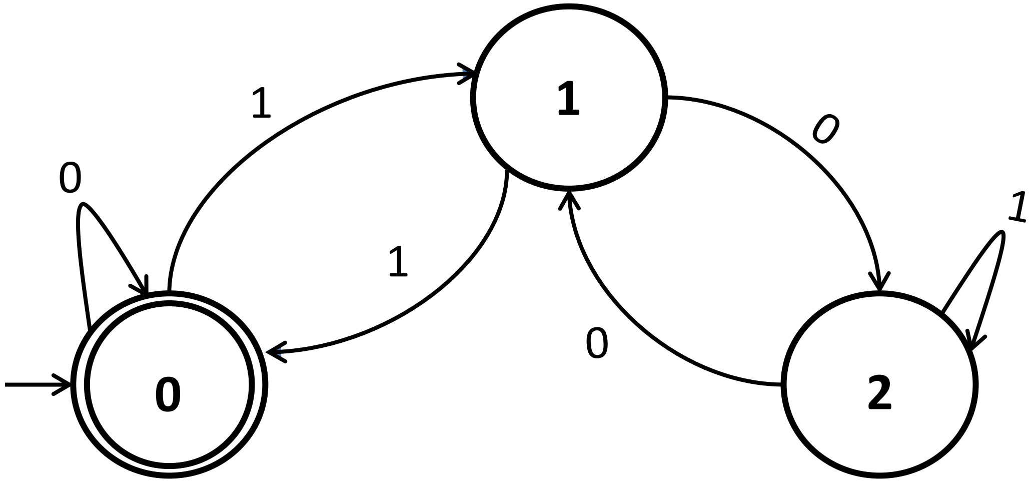
$$\omega\ 0 = 2 \times value\ (\omega) = 0\ (mod\ 3)$$
$$\omega\ 1 = 2 \times value(\omega) + 1 = 1(mod\ 3)$$
$$\omega\ 10 = 2 \times value(\omega 1) = 2(mod\ 3)$$
$$\omega\ 11 = 2 \times value(\omega 1) + 1 = 0(mod\ 3)$$

…. And so on

- The DFA will have three states, each corresponding to the remainder of $value(\omega)/3$.
- The final state $= 0(mod\ 3)$ – the string ω is accepted if after reading it, the DFA ends in this state.

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\omega|\omega$ is divisible by 3}**



Any input string would either leave remainders 0, 1 or 2.

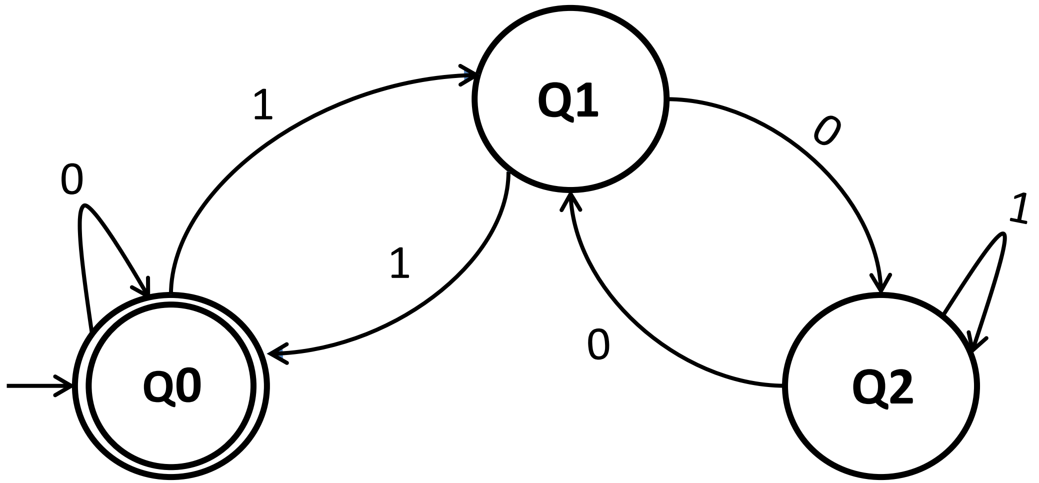
Intuition: Let ω be any substring of the input string divisible by 3, i.e. $\omega = 0(mod\ 3)$

$$\omega\ 0 = 2 \times value\ (\omega) = 0\ (\text{mod } 3)$$
$$\omega\ 1 = 2 \times value(\omega) + 1 = 1(mod\ 3)$$
$$\omega\ 10 = 2 \times value(\omega 1) = 2(mod\ 3)$$
$$\omega\ 11 = 2 \times value(\omega 1) + 1 = 0(mod\ 3)$$

…. And so on

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\omega|\omega$ is divisible by 3}**



	0	1
Q0	Q0	Q1
Q1	Q2	Q0
Q2	Q1	Q2

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma = \{0, 1\}}$, **L(M)={$\boldsymbol{\omega|\omega}$ is NOT divisible by 3}**

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma = \{0, 1\}}$, **L(M)={$\boldsymbol{\omega|\omega}$ is NOT divisible by 3}**

Intuition - Construct a **Toggled DFA:** Toggle the final states and the non-final states!

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}$|$\boldsymbol{\omega}$ is NOT divisible by 3}**

Intuition - Construct a **Toggled DFA:** Toggle the final states and the non-final states!

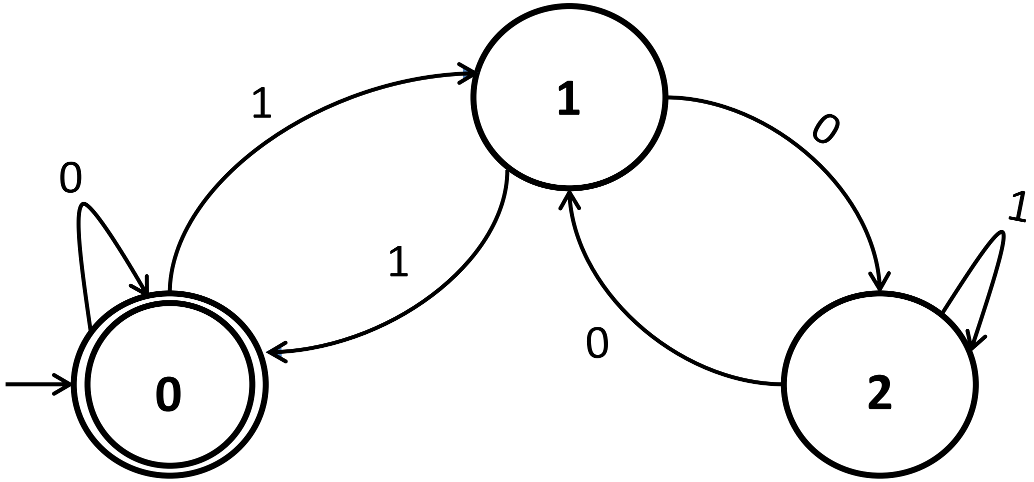
In fact if any DFA accepts $\boldsymbol{L}$, the toggled DFA accepts $\overline{\boldsymbol{L}}$, the complement of $\boldsymbol{L}$

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}|\boldsymbol{\omega}$ is NOT divisible by 3}**

Intuition - Construct a **Toggled DFA:** Toggle the final states and the non-final states!

In fact if any DFA accepts $\boldsymbol{L}$, the toggled DFA accepts $\overline{\boldsymbol{L}}$, the complement of $\boldsymbol{L}$

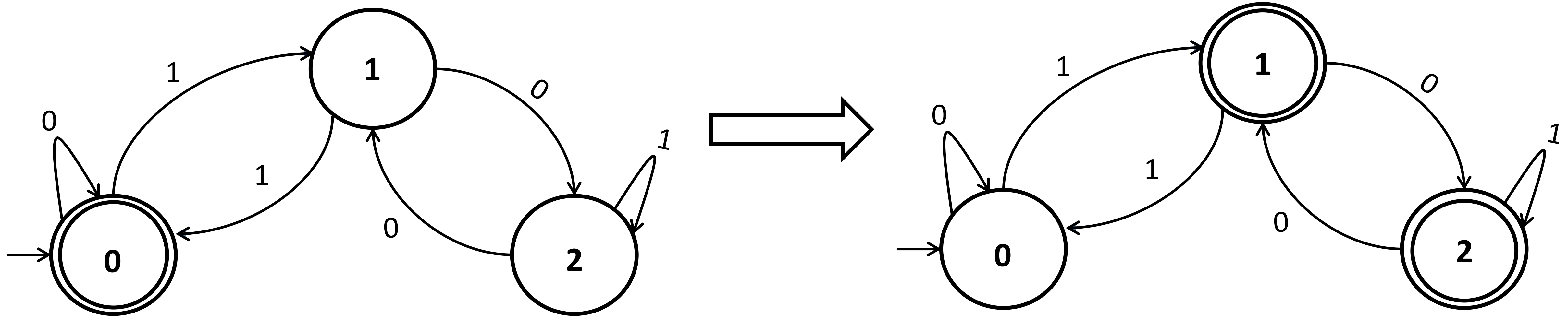


Constructing DFA for a language

Examples: $\boldsymbol{\Sigma = \{0, 1\}}$, **L(M)={$\boldsymbol{\omega | \omega}$ is NOT divisible by 3}**

Intuition - Construct a **Toggled DFA:** Toggle the final states and the non-final states!

In fact if any DFA accepts $\boldsymbol{L}$, the toggled DFA accepts $\boldsymbol{\overline{L}}$, the complement of $\boldsymbol{L}$



Non-deterministic Finite Automata (NFA)

Characteristics of DFA : (i) Single start state (ii) Unique transitions (iii) Zero or more final states

Non-deterministic Finite Automata (NFA)

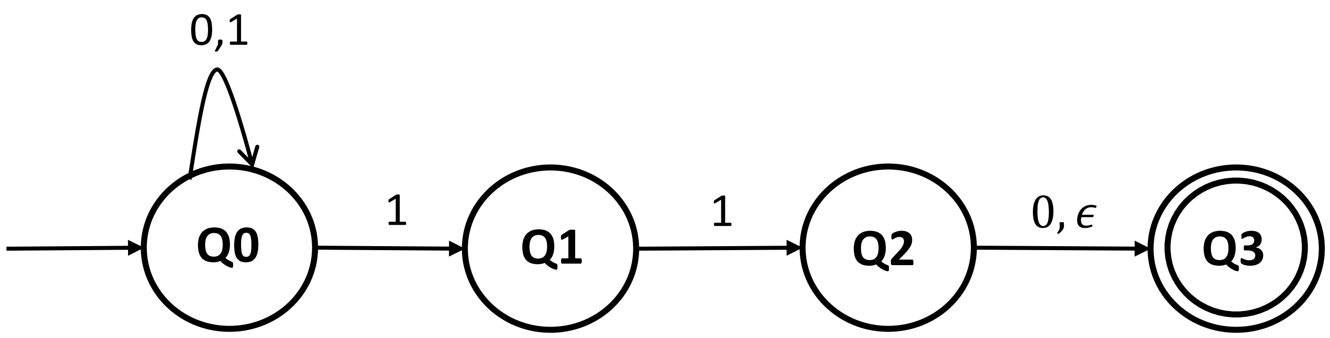
Characteristics of DFA : (i) Single start state (ii) Unique transitions (iii) Zero or more final states

Characteristics of NFA : (i) Single start state (ii) Zero or more final states

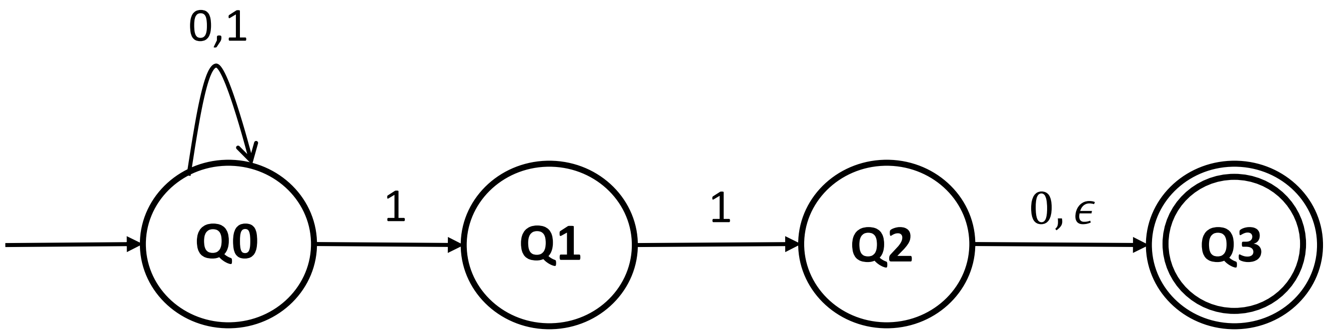
(iii) Multiple transitions are possible on the same input for a state

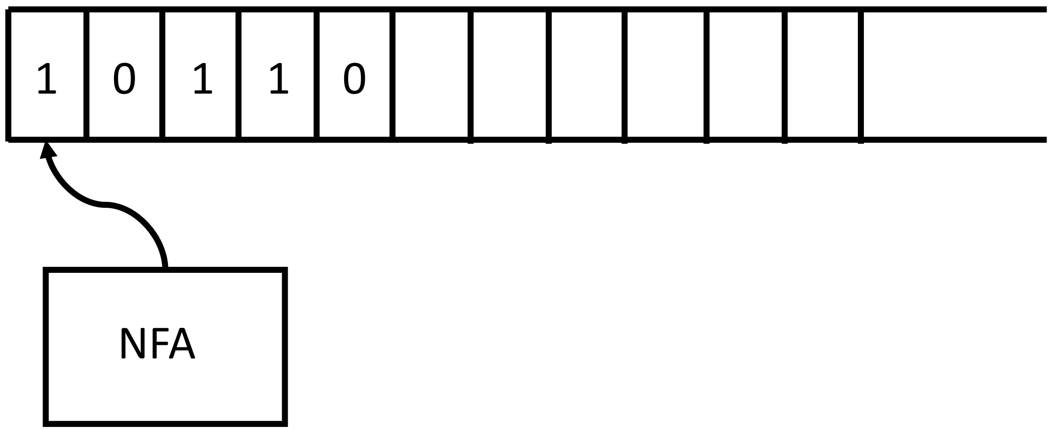
(iv) Some transitions might be missing

(v) ϵ - transitions



Non-deterministic Finite Automata (NFA)



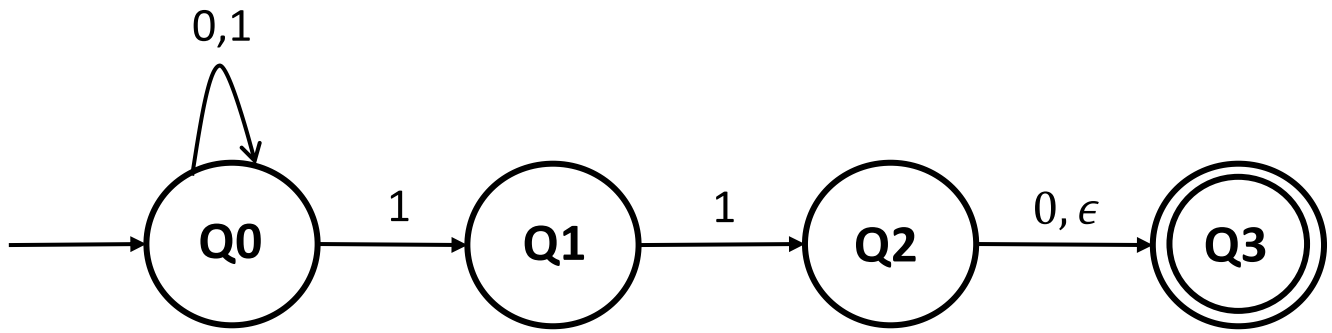


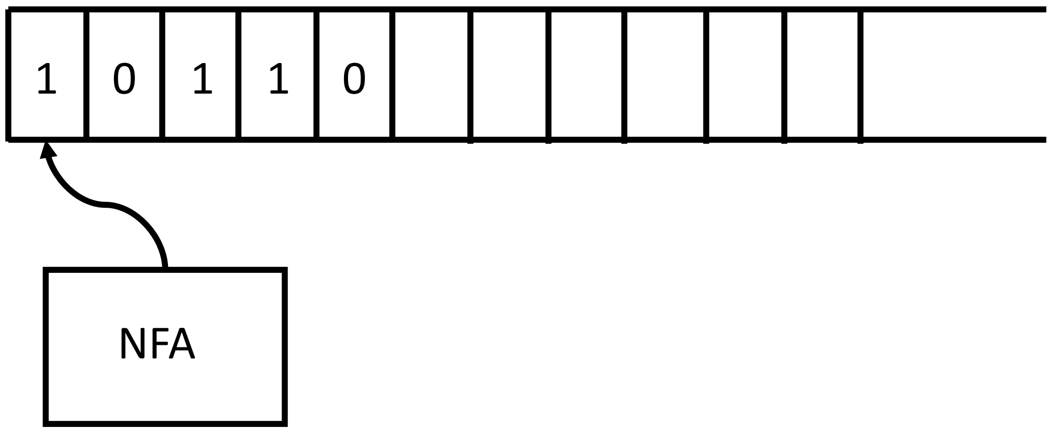
Run 1: $Q0 \overset{1}{\rightarrow} Q0 \overset{0}{\rightarrow} Q0 \overset{1}{\rightarrow} Q0 \overset{1}{\rightarrow} Q0 \overset{0}{\rightarrow} Q0$ (**REJECT**)

Run 2: $Q0 \overset{1}{\rightarrow} Q0 \overset{0}{\rightarrow} Q0 \overset{1}{\rightarrow} Q1 \overset{1}{\rightarrow} Q2 \overset{0}{\rightarrow} Q3$ (**ACCEPT**)

Multiple **runs** per input is possible

Non-deterministic Finite Automata (NFA)





Run 1: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0$ (**REJECT**)

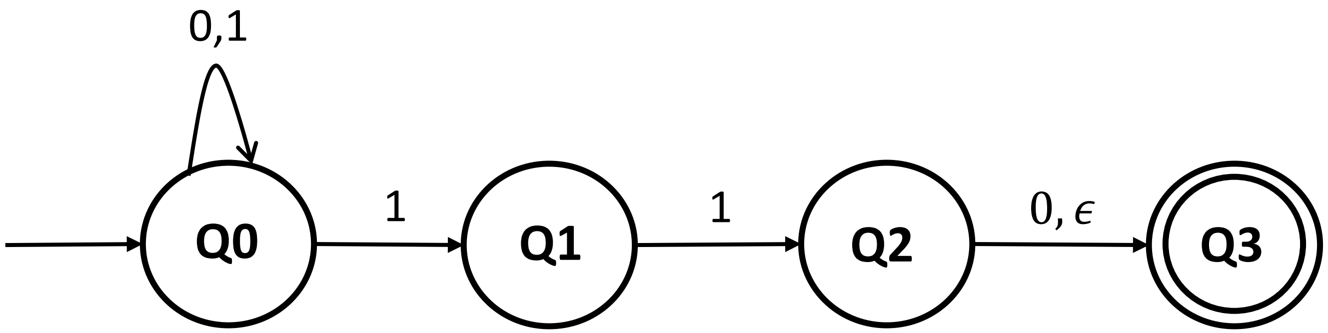
Run 2: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q1 \xrightarrow{1} Q2 \xrightarrow{0} Q3$ (**ACCEPT**)

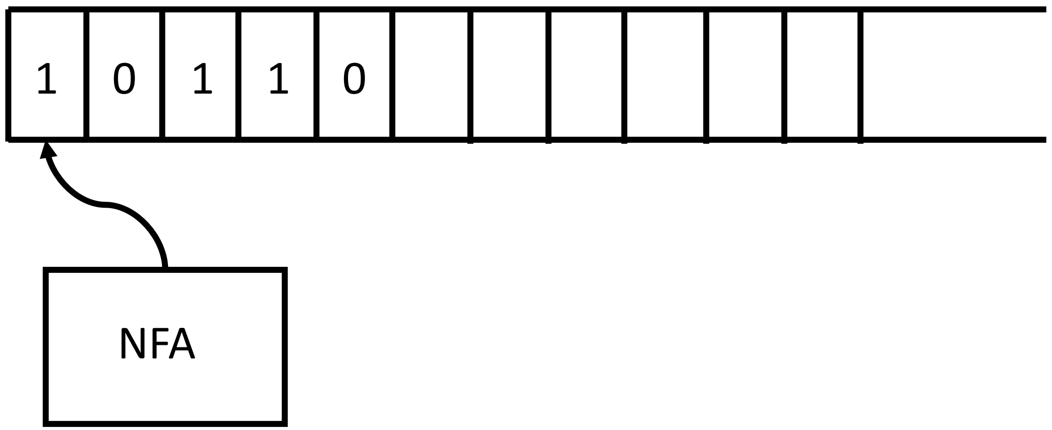
Run 3: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q0 \xrightarrow{1} Q1 \xrightarrow{0}$ **CRASH**

Run 4: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q1 \xrightarrow{1} Q2 \xrightarrow{\epsilon} Q3 \xrightarrow{0}$ **CRASH**

CRASH is a Rejecting Run

Non-deterministic Finite Automata (NFA)





Run 1: $Q0 \overset{1}{\to} Q0 \overset{0}{\to} Q0 \overset{1}{\to} Q0 \overset{1}{\to} Q0 \overset{0}{\to} Q0$ (**REJECT**)

Run 2: $Q0 \overset{1}{\to} Q0 \overset{0}{\to} Q0 \overset{1}{\to} Q1 \overset{1}{\to} Q2 \overset{0}{\to} Q3$ (**ACCEPT**)

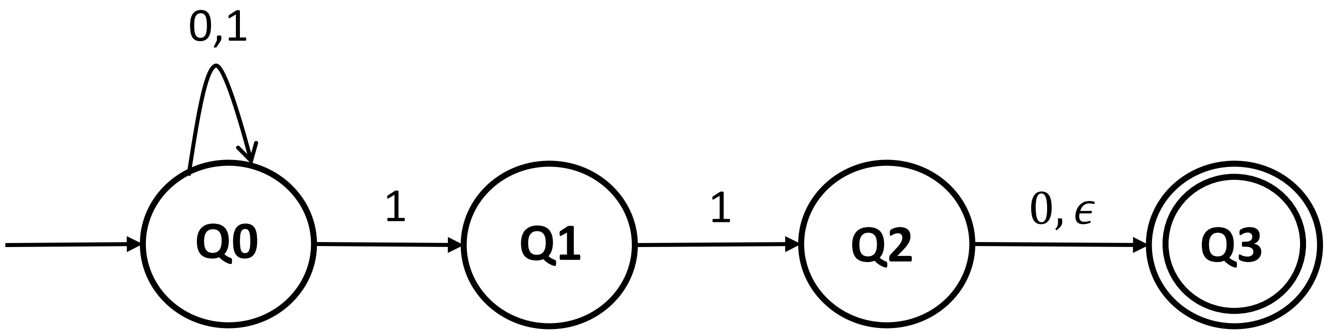
Run 3: $Q0 \overset{1}{\to} Q0 \overset{0}{\to} Q0 \overset{1}{\to} Q0 \overset{1}{\to} Q1 \overset{0}{\to}$ CRASH **(REJECT)**

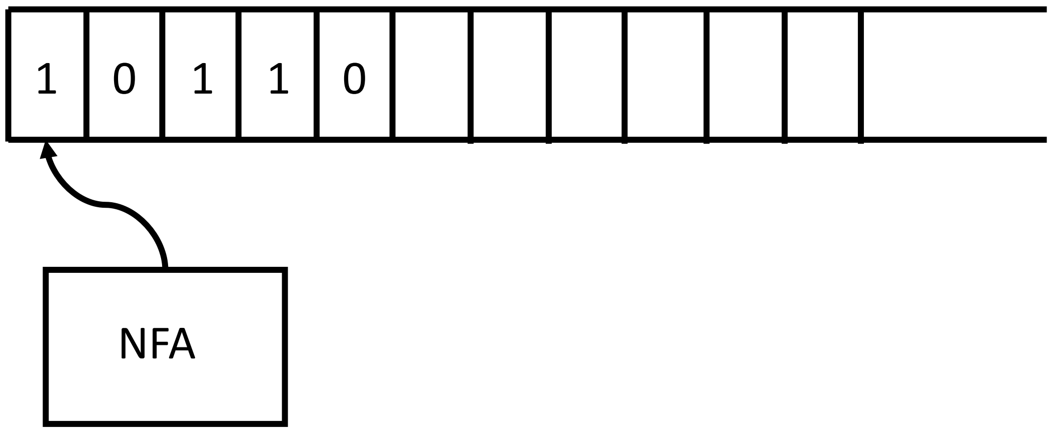
Run 4: $Q0 \overset{1}{\to} Q0 \overset{0}{\to} Q0 \overset{1}{\to} Q1 \overset{1}{\to} Q2 \overset{\epsilon}{\to} Q3 \overset{0}{\to}$ CRASH **(REJECT)**

The NFA “accepts” an input string, if it at **least one *run* ends up in the final state**. **(Accepting Run)**

The NFA “rejects” an input string, if there are **no runs that end up in a final state**. **(Rejecting Run)**

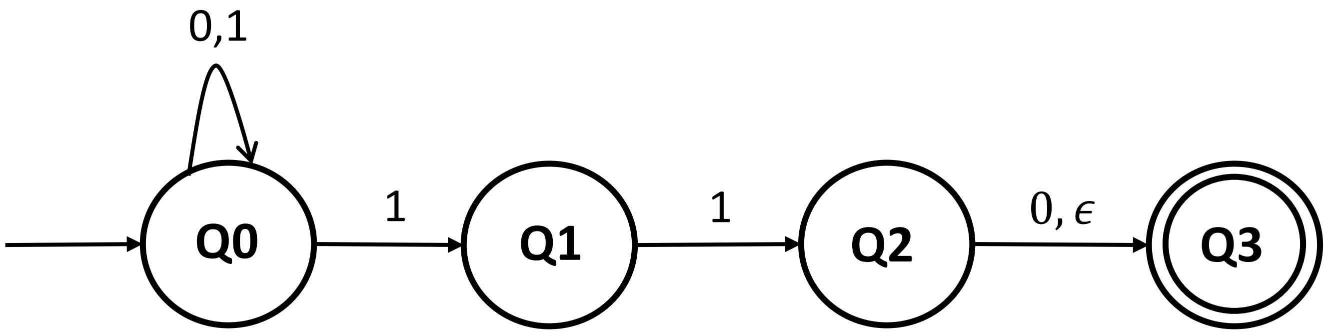
Non-deterministic Finite Automata (NFA)

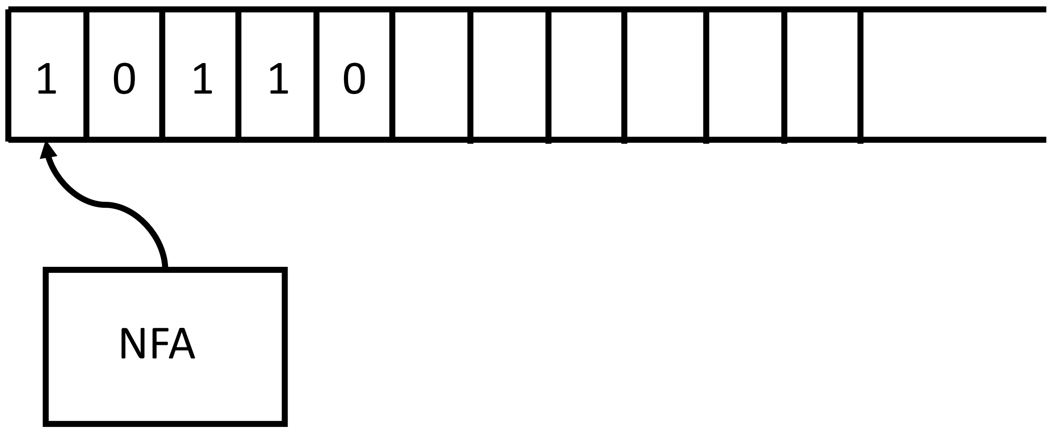




	0	**1**	**ϵ**
Q0	**Q0**	**Q0, Q1**	
Q1		**Q2**	
Q2	**Q3**		**Q3**
Q3			

Non-deterministic Finite Automata (NFA)





Formally, a finite automaton M is a 5-tuple $(Q, \Sigma, \delta, q_0, F\ )$ where

- Q is a finite set called the ***states.***
- Σ is a finite set called the ***alphabet***.
- $\delta: Q \times \Sigma \mapsto P(Q)$ is the ***transition function***. $P(Q)$ is the power set of Q
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***final/accepting states***.

	0	**1**	$\boldsymbol{\epsilon}$
Q0	**Q0**	**Q0, Q1**	
Q1		**Q2**	
Q2	**Q3**		**Q3**
Q3			

NFA vs DFA

- Are NFAs more powerful than DFAs?
- Intuitively, non-determinism seems to be adding more “power”.

NFA vs DFA

- Are NFAs more powerful than DFAs?
- Intuitively, non-determinism seems to be adding more “power”.
- Let L_1 be the language accepted by NFAs and L_2 be the language accepted by DFAs
- Is $L_2 \subseteq L_1$?

NFA vs DFA

- Are NFAs more powerful than DFAs?
- Intuitively, non-determinism seems to be adding more “power”.
- Let L_1 be the language accepted by NFAs and L_2 be the language accepted by DFAs
- Is $L_2 \subseteq L_1$? Clearly true, because a DFA is just a special case of an NFA.

NFA vs DFA

- Are NFAs more powerful than DFAs?
- Intuitively, non-determinism seems to be adding more “power”.
- Let L_1 be the language accepted by NFAs and L_2 be the language accepted by DFAs
- Is $L_2 \subseteq L_1$? Clearly true, because a DFA is just a special case of an NFA.
- Surprisingly, what we will show next is that $\boldsymbol{L_1 \subseteq L_2}$!
- That is, **given an NFA, we can convert it to a DFA that accepts the same language**.
- Such a DFA is called a “**Remembering DFA**”. We will learn about this in the next lecture.

Thus, DFAs and NFAs are completely equivalent and $\boldsymbol{L_1 = L_2}$!

Thank You!