

# CS 302.1 - Automata Theory

## Lecture 01

Shantanav Chakraborty

Center for Quantum Science and Technology (CQST)

Center for Security, Theory and Algorithms (CSTAR)

IIIT Hyderabad



INTERNATIONAL INSTITUTE OF  
INFORMATION TECHNOLOGY

H Y D E R A B A D

# Introduction

In this course, we will look at:

- **Which problems are computable?**
  - Can we characterize them?
  - What about natural problems? Computers are exotic physics experiments after all.
- **Design abstract models of computation and try to understand what problems can be solved by them.**
  - Small models that are limited in power and can solve a subset of computable problems.
  - We will build increasingly powerful computational models as we go along.
- **What are the limits of computation?**
  - Are there problems that cannot be solved on the most powerful computers that will exist in the future.

# Introduction

In this course, we will look at:

- Which problems are computable?
- Design abstract models of computation and try to understand what problems can be solved by them.
- 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}

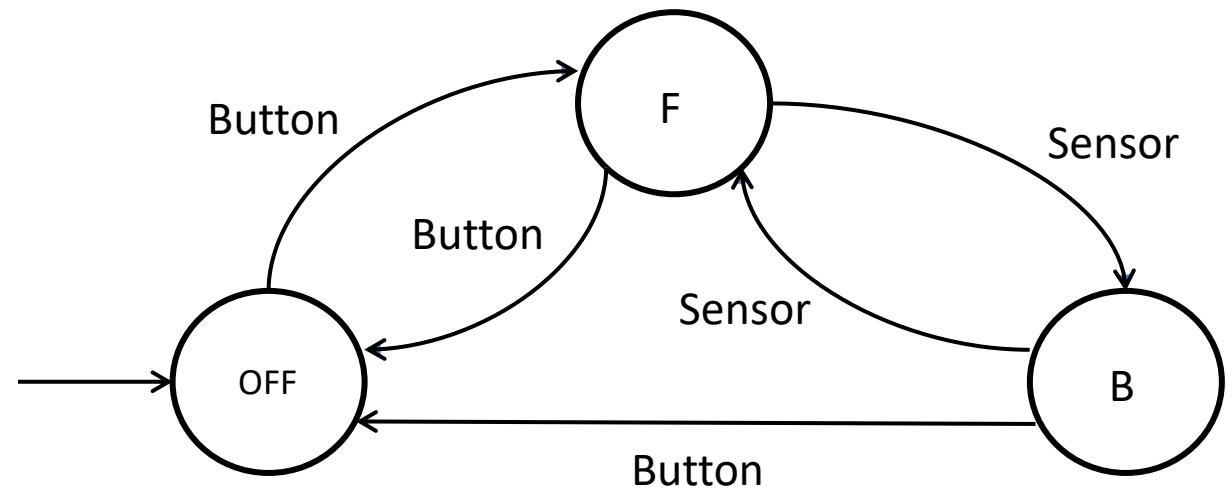
Inputs : {BUTTON, SENSOR}

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

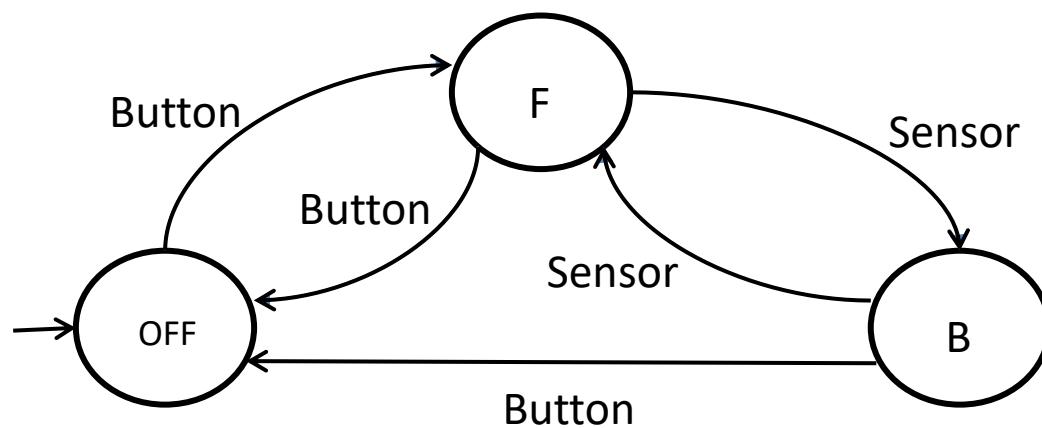


State diagram for the robot

# 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	$\frac{\pi}{3}, \frac{1}{2}, 2, \dots$

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*.

# 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*.

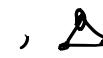
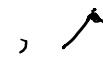
Problem	Decision problem
Sorting	Is the array sorted?
Graph connectivity	Is the graph connected?

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 : {  ,  ,  , ..... }

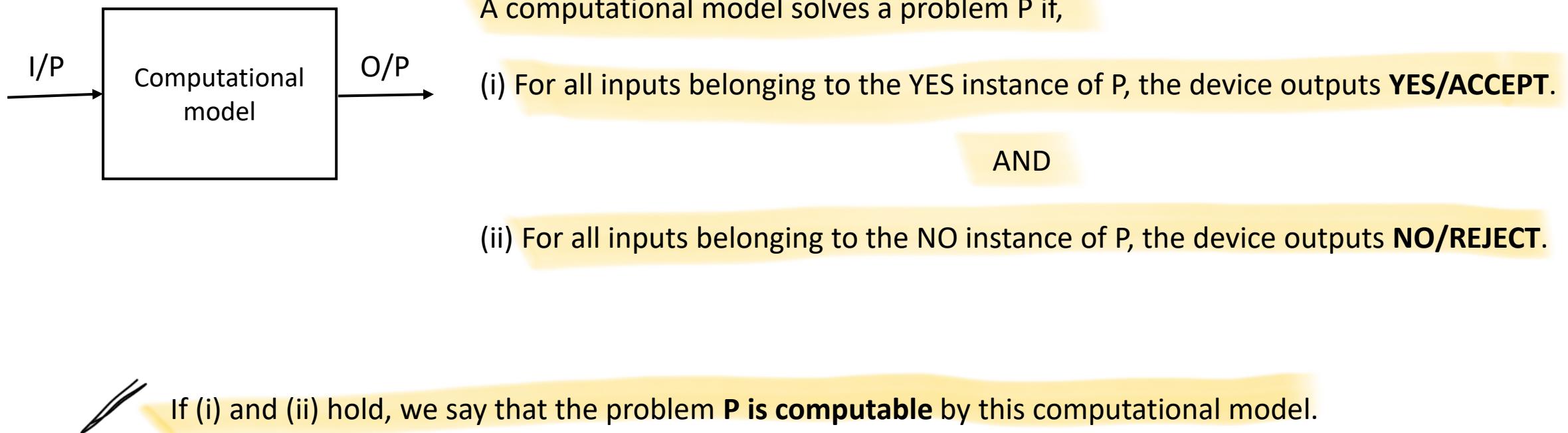
NO set : {  ,  ,  , ..... }

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?

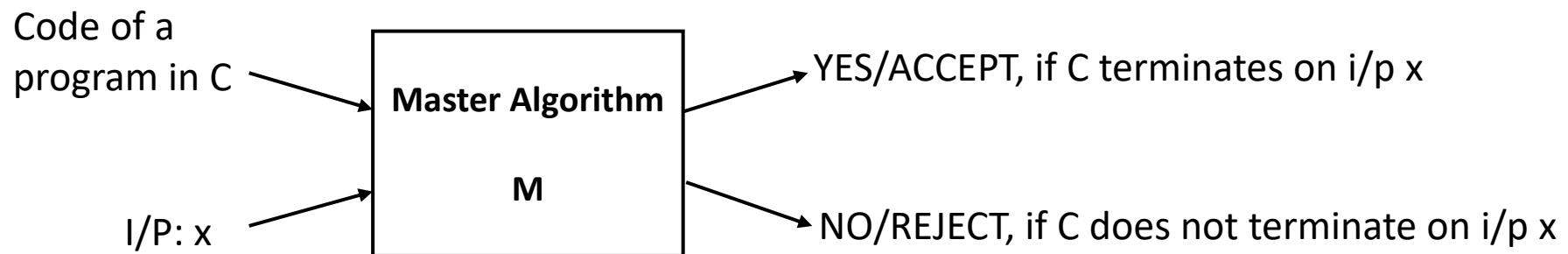


# Introduction

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

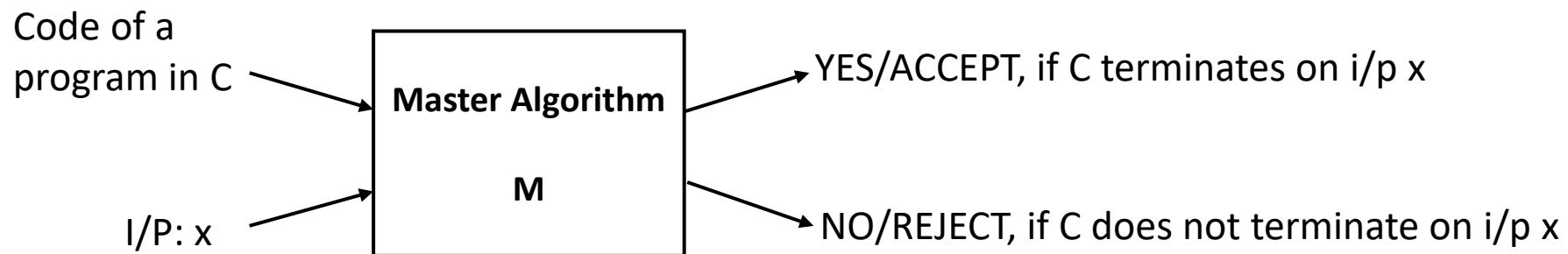


# Introduction

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

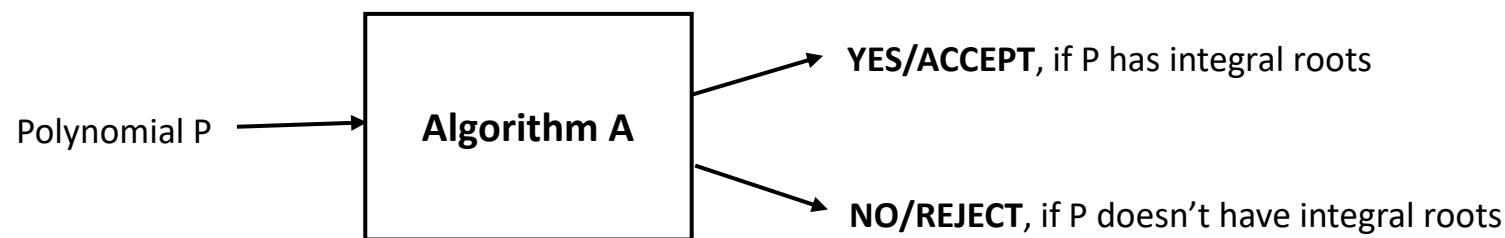


- **M terminates and outputs NO even if  $C(x)$  runs infinitely!**
- No such Algorithm M can be written. **Undecidable problem!**

**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/ACCEPT as (-1,1) are solutions to P

- The algorithm A proceeds by checking whether for integers  $0, \pm 1, \pm 2, \dots$ . 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

- ❖ 13 Lectures in all
- ❖ Final Exam at the end (**35% weightage**)
- ❖ Two theory assignment (**25% weightage**)
  - Assignment 1 – will be released after Lec 3/4 (Deadline: Before Quiz 1)
  - Assignment 2 – will be released after Quiz 1 (Deadline: the week before Final exam)
- ❖ Programming assignment (**20% weightage**)
  - Released after Lec 3/4 (Deadline: End of midsem)
- ❖ Quiz (**20% weightage**)

# Tutorials and TAs

- Tutorial sessions weekly: **Wednesdays, 8:30 AM – 9:55 AM**
- Teaching Associates:
  - Shreyas S ([sreyas.saminathan@research.iiit.ac.in](mailto:sreyas.saminathan@research.iiit.ac.in))
  - S Rajendraprasad ([rajendraprasad.s@research.iiit.ac.in](mailto:rajendraprasad.s@research.iiit.ac.in))
  - Kandi Jayanth Reddy ([kandi.reddy@students.iiit.ac.in](mailto:kandi.reddy@students.iiit.ac.in))
  - Mayank Goel ([mayank.goe@research.iiit.ac.in](mailto:mayank.goe@research.iiit.ac.in))
  - Gargi Shroff ([gargi.shroff@research.iiit.ac.in](mailto:gargi.shroff@research.iiit.ac.in))
  - Rijul Saigal ([rijul.saigal@research.iiit.ac.in](mailto:rijul.saigal@research.iiit.ac.in))
  - Anish Joishy ([anish.joishy@research.iiit.ac.in](mailto:anish.joishy@research.iiit.ac.in))
  - Aadi Prasad ([aadi.prasad@research.iiit.ac.in](mailto:aadi.prasad@research.iiit.ac.in))
  - Devika Umesh Bej ([devika.bej@research.iiit.ac.in](mailto:devika.bej@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](mailto:shchakra@iiit.ac.in)
- **Lecture slides available at my homepage:** <https://sites.google.com/view/shchakra/teaching/m24-automata-theory>

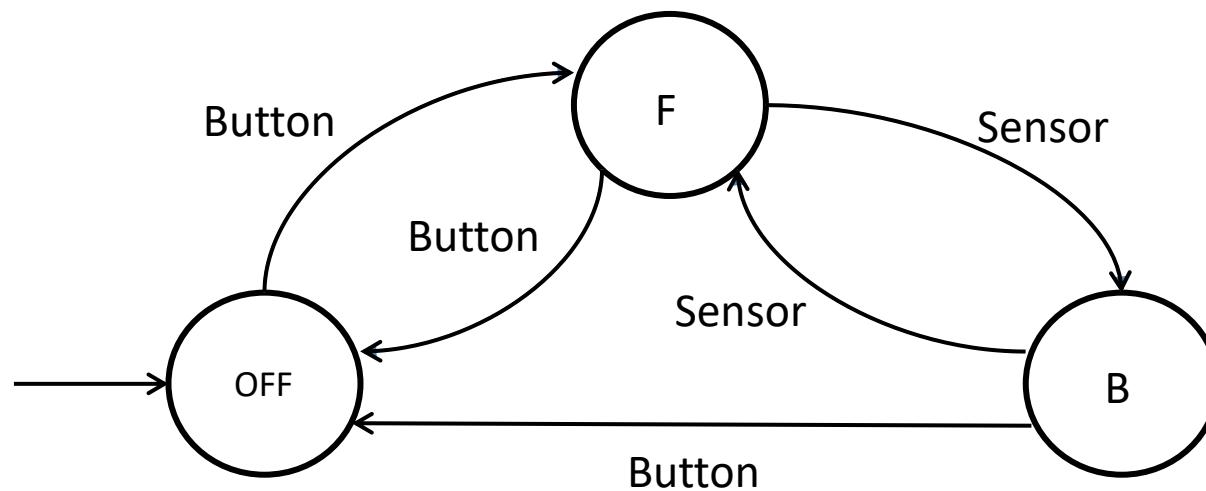
# Some terminology

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  $\epsilon$

# Models of computation

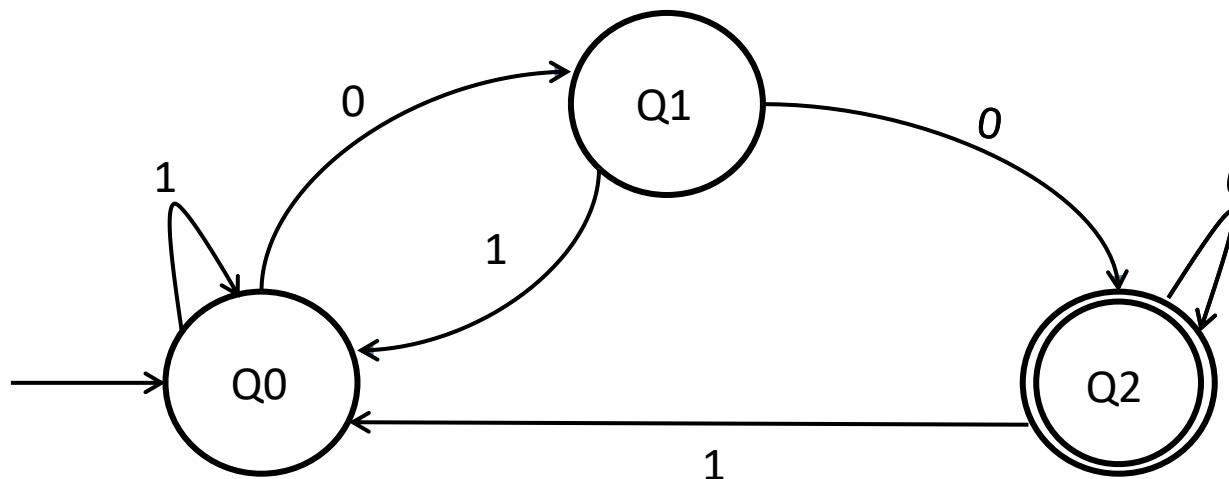
- Deterministic Finite Automata (DFA) Model



# Models of computation

- Deterministic Finite Automata (DFA) Model

Q0: Start state, Q2: Final state

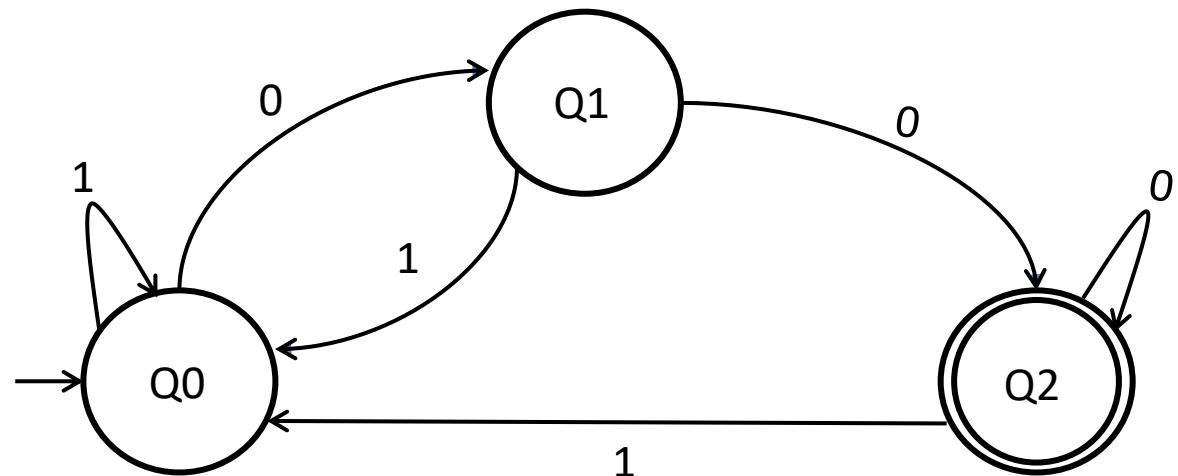


State transition diagram of the Finite State Machine



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

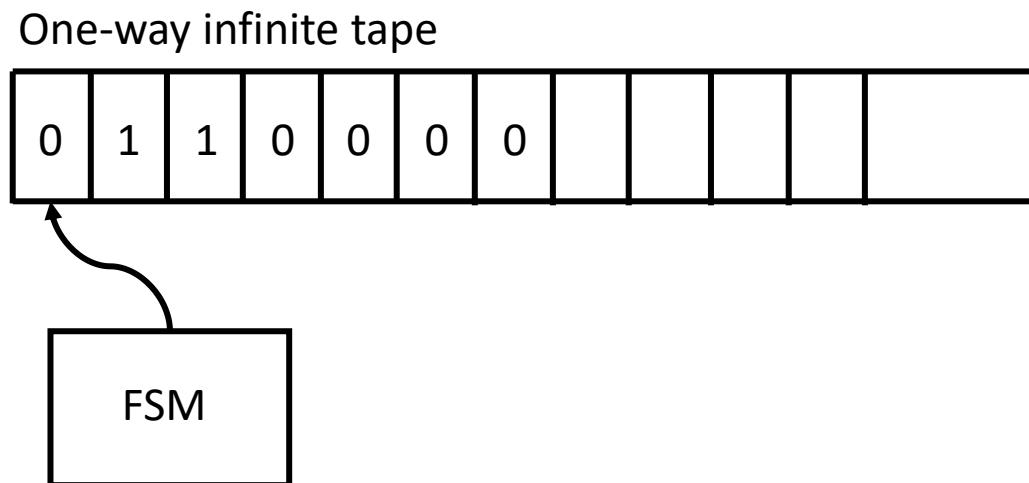
# Deterministic Finite Automata (DFA)



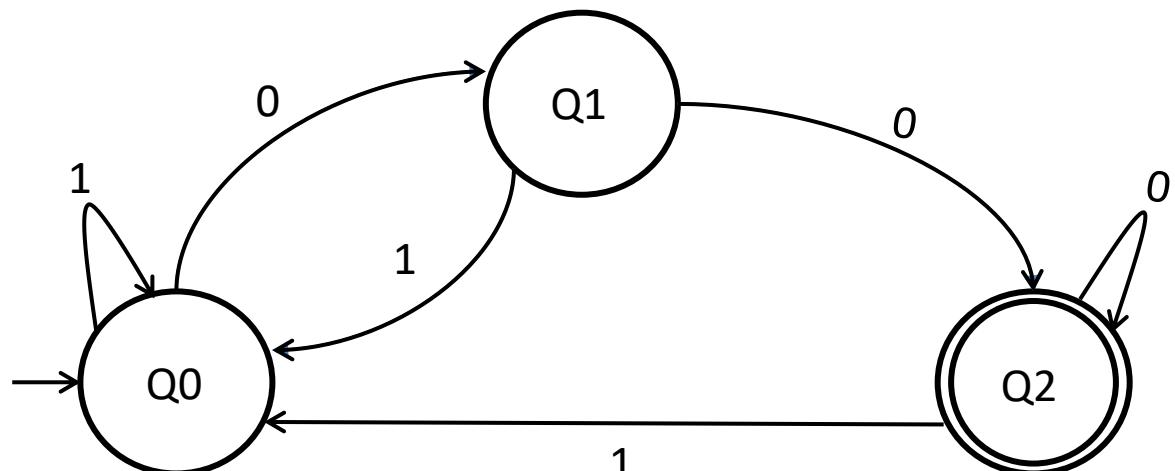
State transition diagram of the Finite State Machine

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

Q0: Start state, Q2: Final state

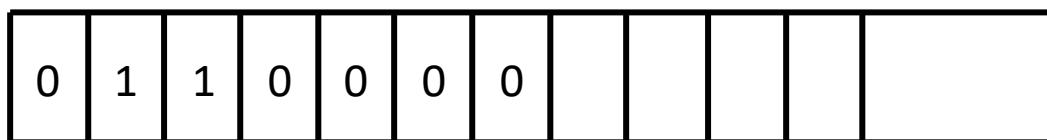


# Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



FSM

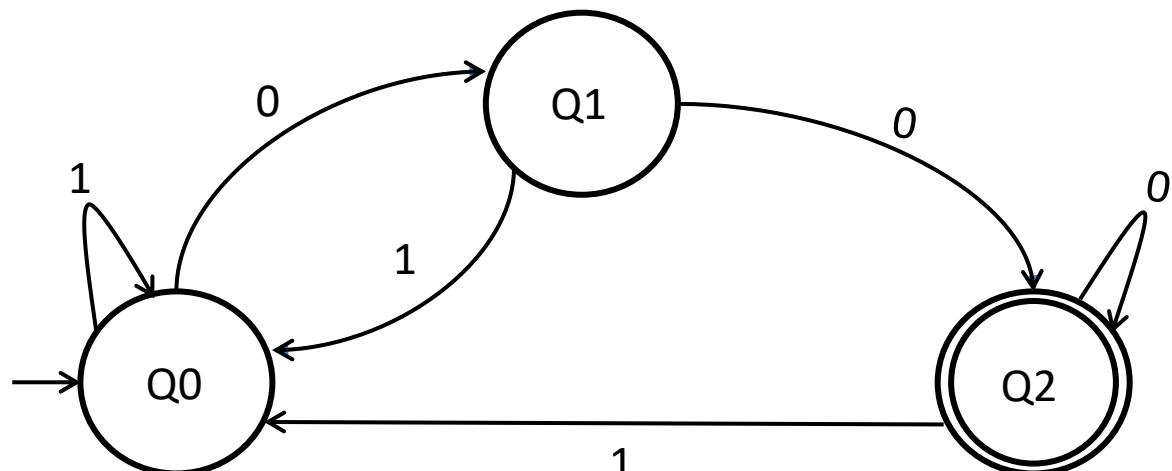
Input: Strings from alphabet  $\Sigma = \{0,1\}$

Q0: Start state, Q2: Final state

**Run:**

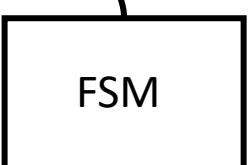
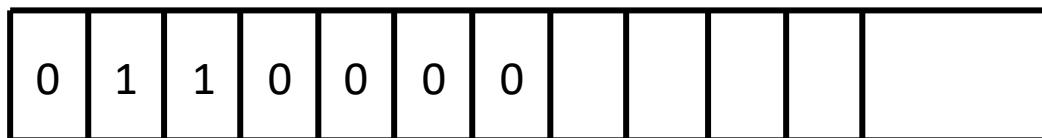
$Q0 \xrightarrow{0} Q1 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{0} Q2 \xrightarrow{0} Q2 \xrightarrow{0} Q2$

# 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

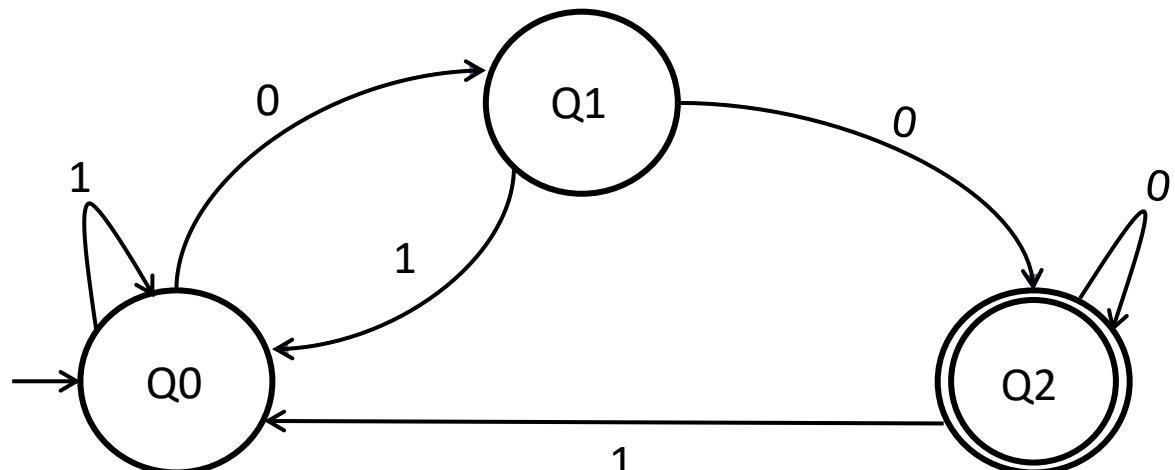
The DFA “accepts” an input string, if it corresponds to a *run* that ends up in the final state Q2. **(Accepting Run)**

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

Run:

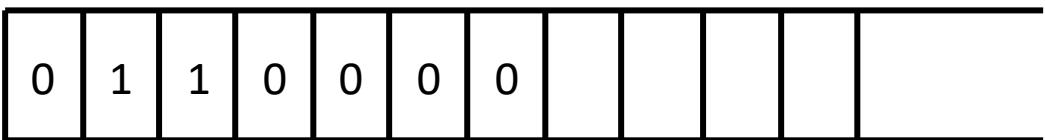
$$Q0 \xrightarrow{0} Q1 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{0} Q2 \xrightarrow{0} Q2 \xrightarrow{0} Q2$$

# Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



FSM

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**)

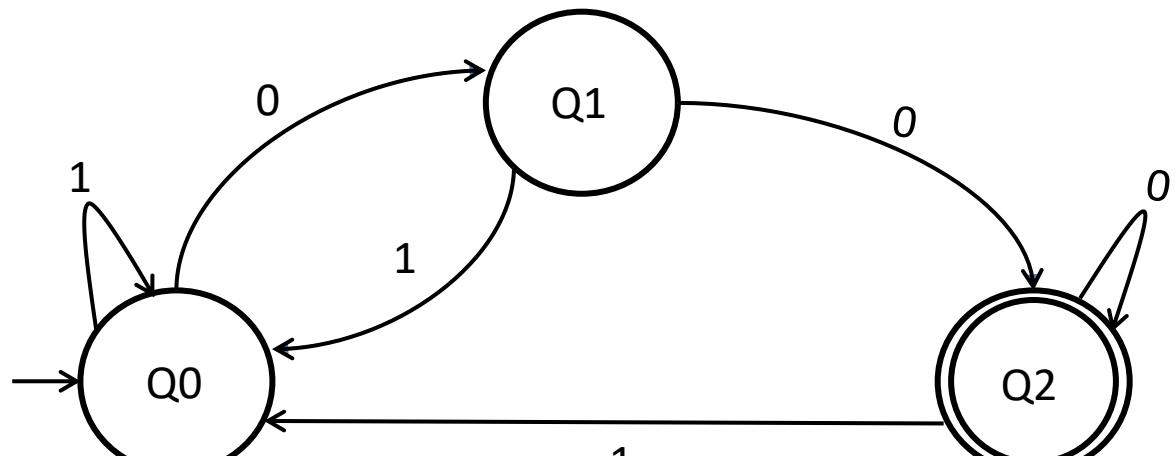
Run:

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

ACCEPT = {0110000, }

REJECT = {}

# Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



FSM

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**)

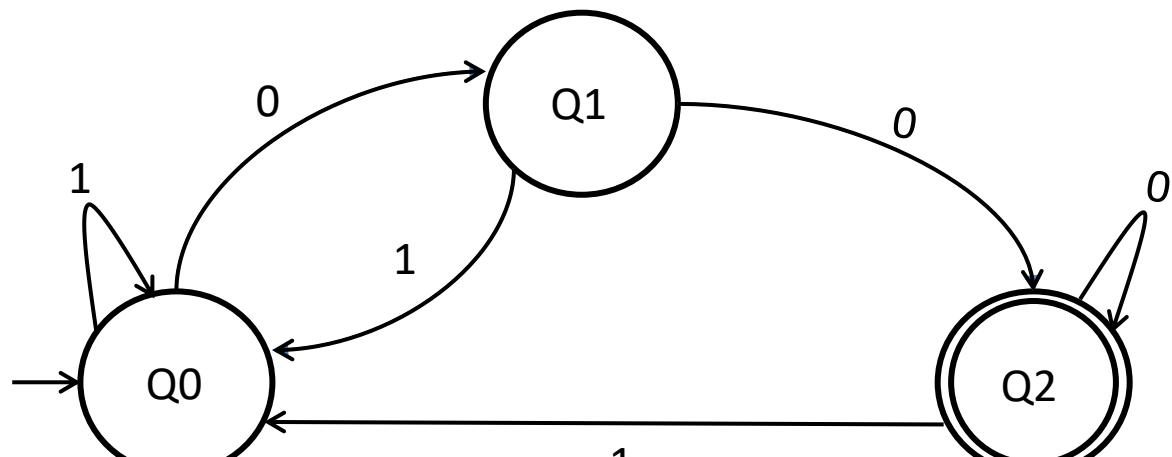
Run:

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

ACCEPT = {0111000, }

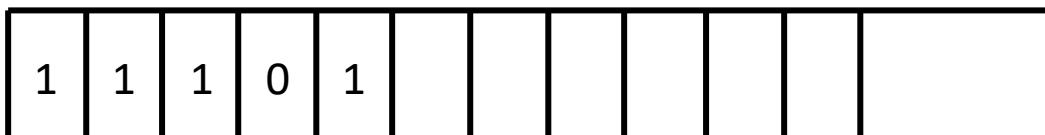
REJECT = {}

# Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



FSM

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**)

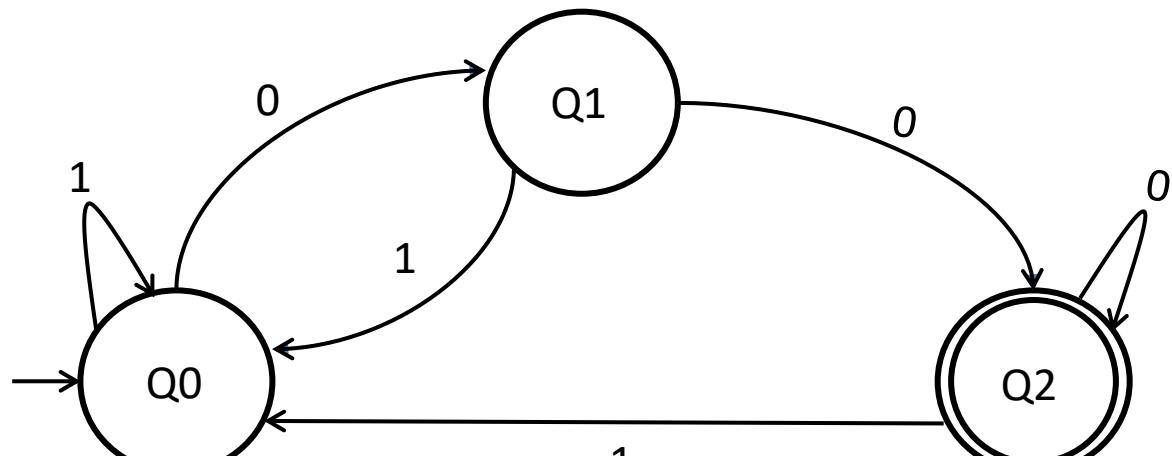
Run:

$$Q0 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{1} \textcolor{red}{Q0}$$

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**)

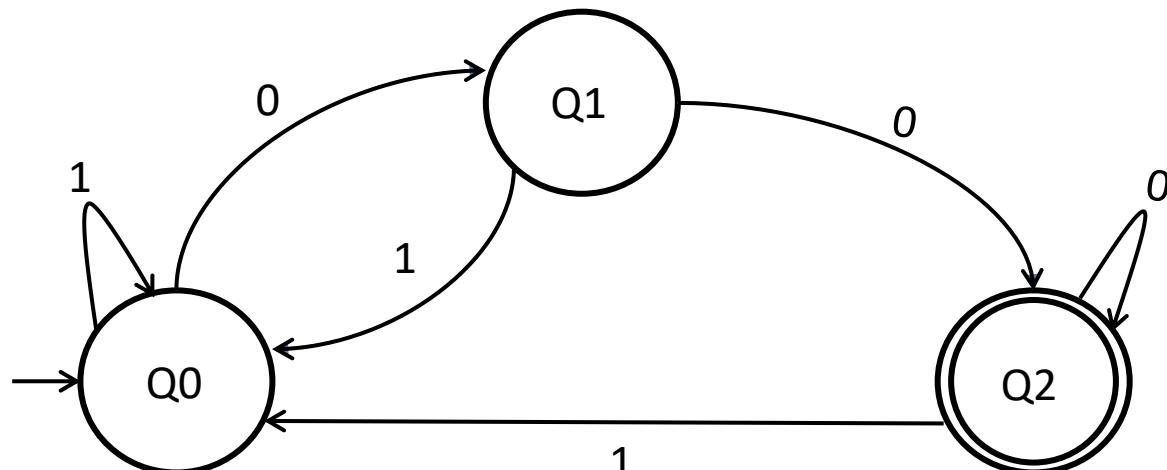
**Run:**

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

ACCEPT = {0111000, 10100, ...}

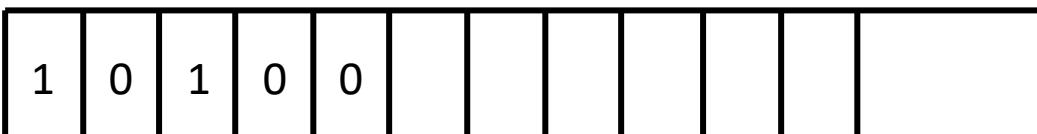
REJECT = {11101, ....}

# Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



FSM

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**)

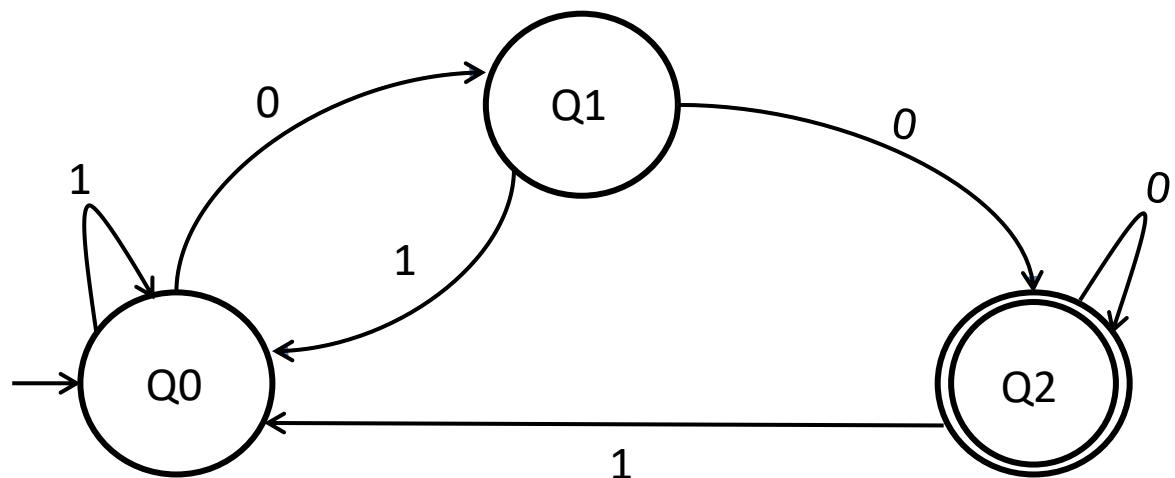
**Run:**

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

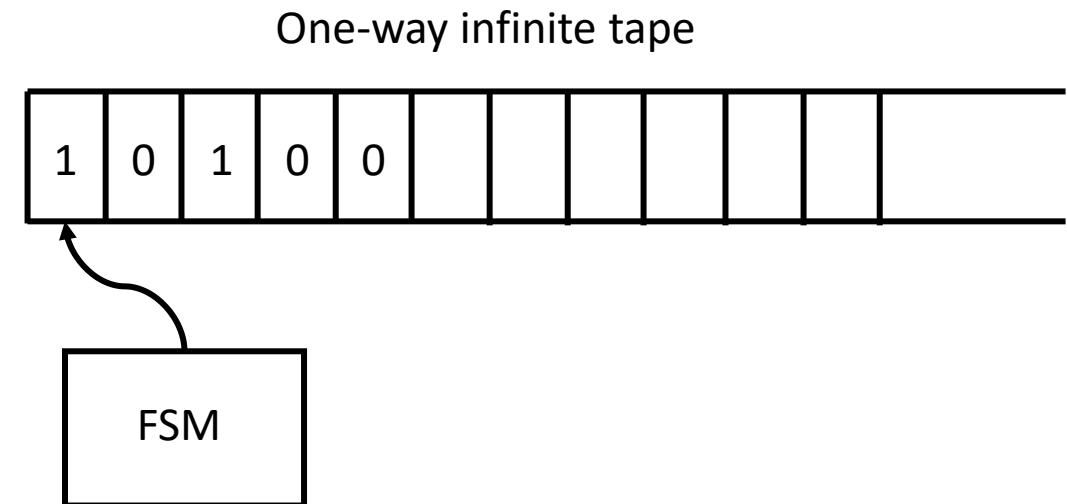
ACCEPT = {0111000, 10100, 0100, 00, 10000,...}

REJECT = {11101, 0, 1, 11, 001,.....}

# Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine



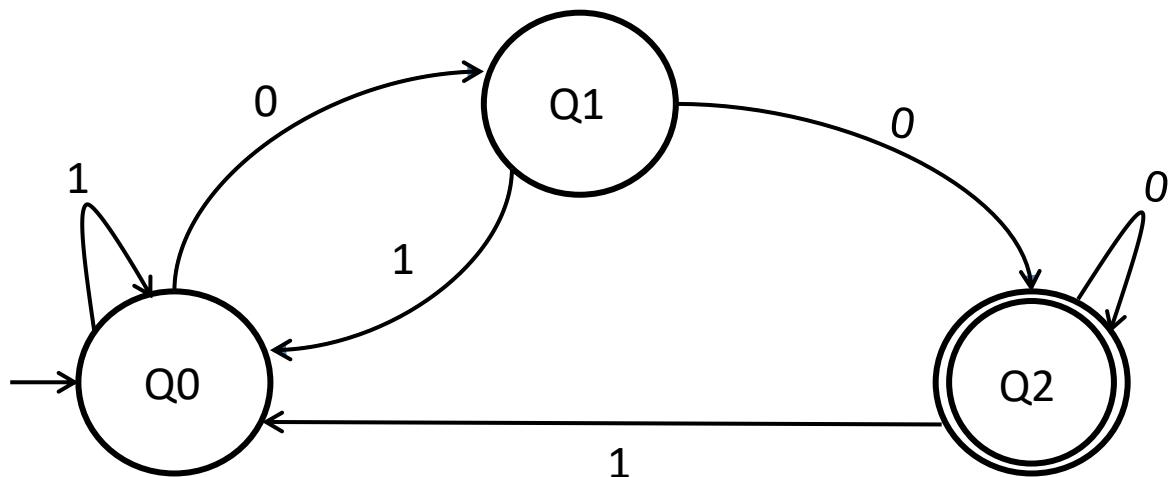
ACCEPT = {0111000, 10100, 0100, 00, 10000,...}  
REJECT = {11101, 0, 1, 11, 001,.....}

Let the DFA be M. Then, language M accepts is

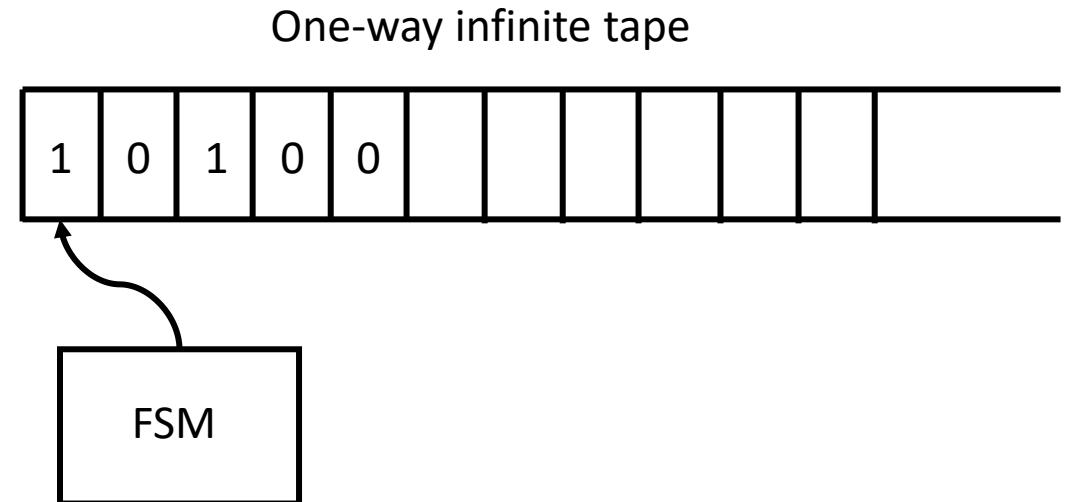
  $L(M) = \{\omega \mid \omega \text{ results in an accepting run}\}$ , i.e. the set of all strings  $\omega$  such that  $M(\omega)$  accepts

For the example above,  $L(M) = \{\omega \mid \omega \text{ ends in "00"}\}$

# Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine



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

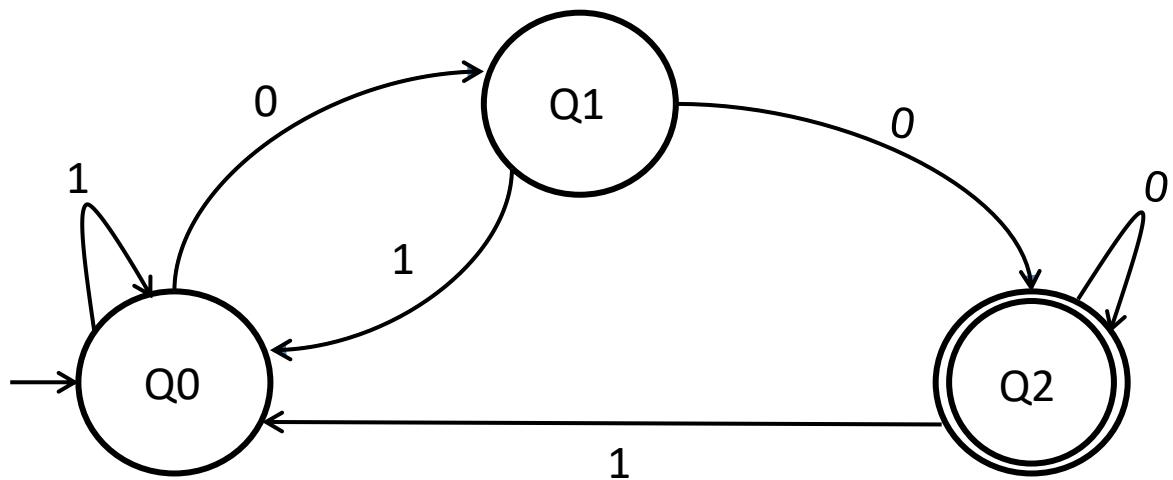
For any language  $L$ , we say  $M$  recognizes  $L$  if



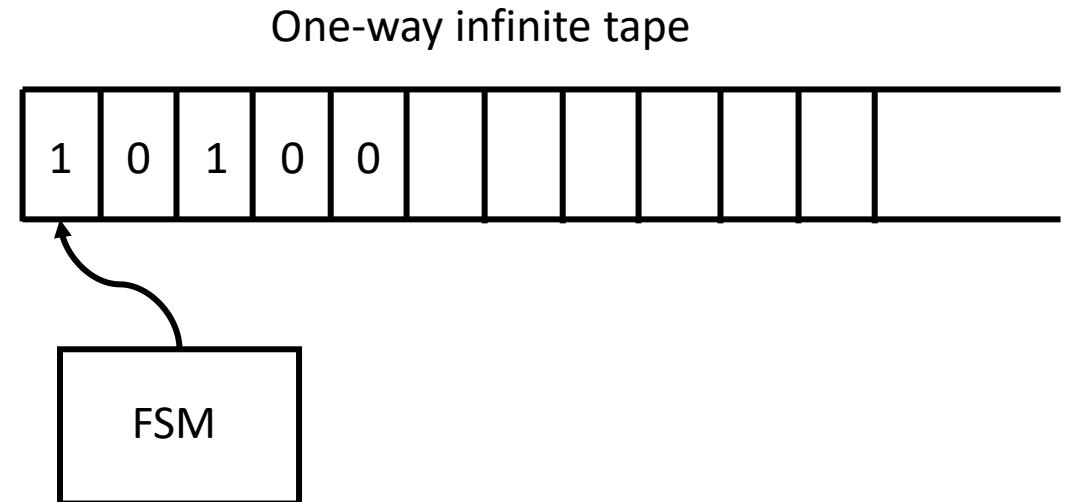
$\forall \omega \in L, M(\omega) \text{ accepts}$

For the example above,  $M$  recognizes  $L = \{\omega | \omega \text{ ends in "00"}\}$

# Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine



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

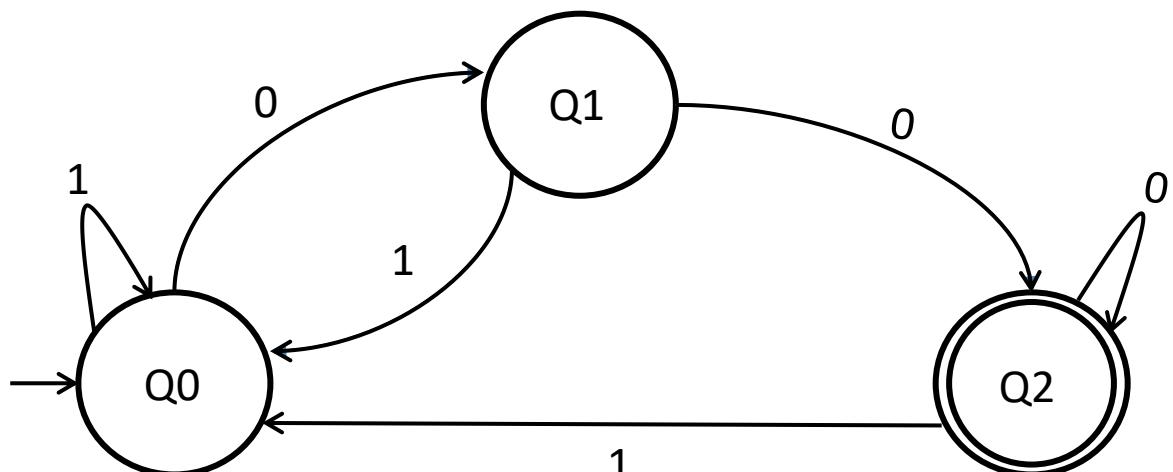
For any language  $L$ , we say **M solves L or M decides L if**



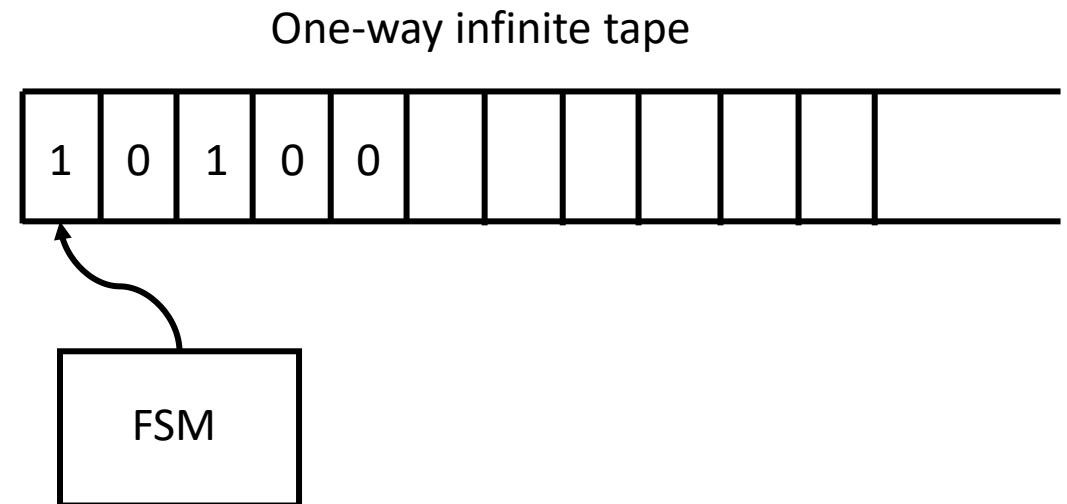
$\forall \omega \in L, M(\omega) \text{ accepts}$   
 $\forall \omega \notin L, M(\omega) \text{ rejects}$

For the example above, **M decides L=  $\{\omega | \omega \text{ ends in "00"}\}$**

# Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine



For any language  $L$ , we say  $M$  **recognizes**  $L$  if

$\forall \omega \in L, M(\omega) \text{ accepts}$

For any language  $L$ , we say  $M$  **decides**  $L$  if

$\forall \omega \in L, M(\omega) \text{ accepts}$   
 $\forall \omega \notin L, M(\omega) \text{ rejects}$

For a DFA, the notions of **deciding a language** and **recognizing a language** are equivalent, but this may not be true for other, more powerful computational models

**Thank You!**