Finite Automata

Lin Chen

Email: Lin.Chen@ttu.edu

TEXAS TECH
UNIVERSITY

Basic model for machines

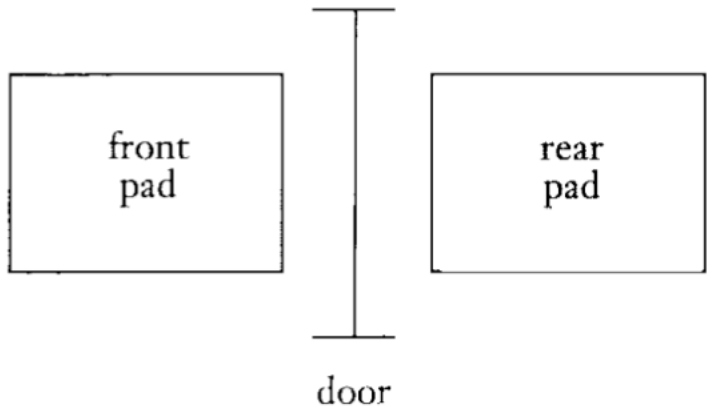
- An automatic door opens in one direction

Basic model for machines

- An automatic door opens in one direction

  -- Open to let people in

  -- Do not knock people on the opening side



How to design a machine fulfilling the function?

Basic model for machines

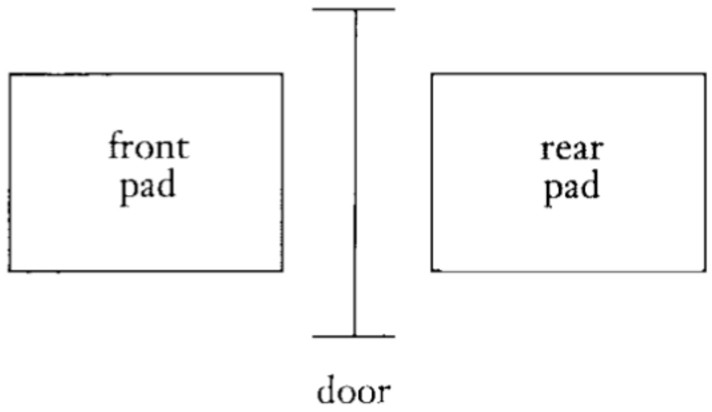
- An automatic door opens in one direction

  -- Check front people: Yes/No

  -- Check rear people: Yes/No

  -- Check current status: Open/Close

  -- Determine the next status: Open/Close



Basic model for machines

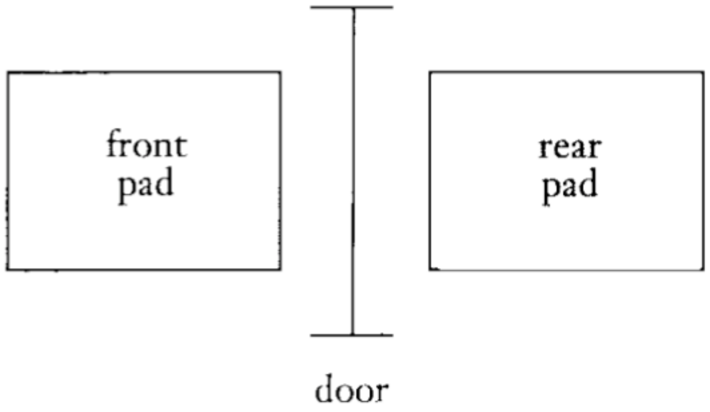
- An automatic door opens in one direction

  -- Check front people: Yes/No

  -- Check rear people: Yes/No

  -- Check current status: Open/Close

  -- Determine the next status: Open/Close



input signal

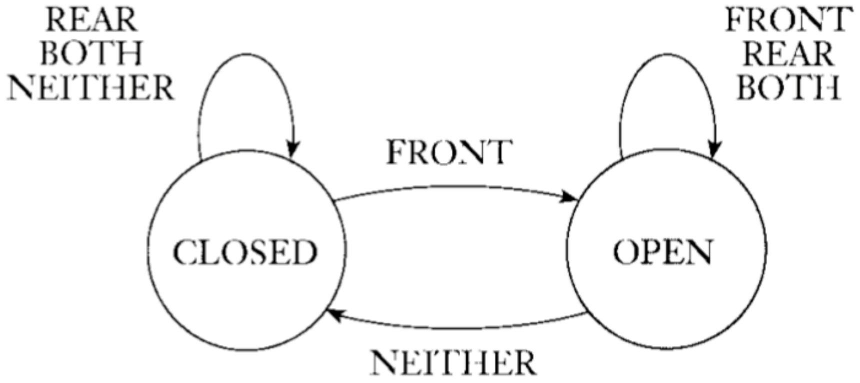
state		NEITHER	FRONT	REAR	BOTH
	CLOSED	CLOSED	OPEN	CLOSED	CLOSED
	OPEN	CLOSED	OPEN	OPEN	OPEN

Basic model for machines

- An automatic door opens in one direction

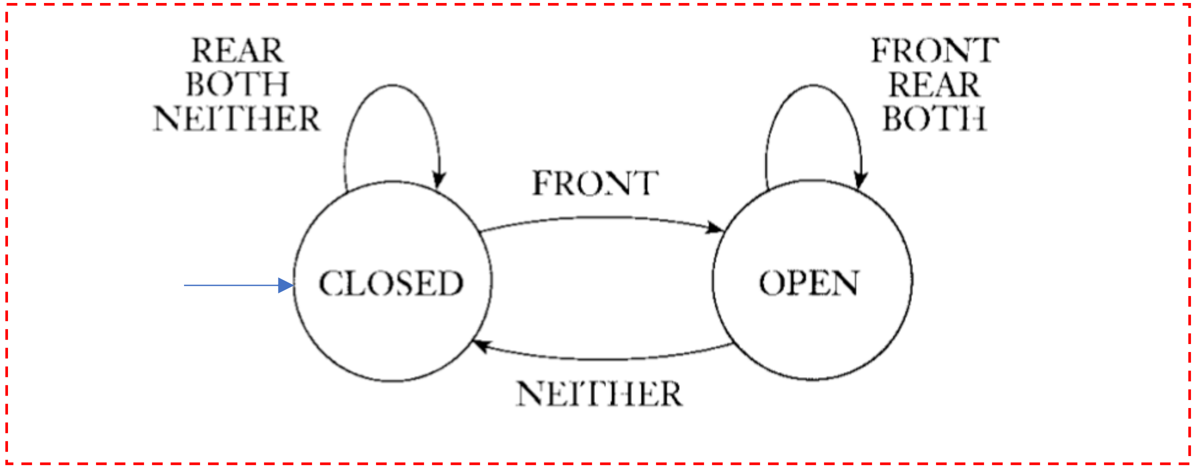
input signal

		NEITHER	FRONT	REAR	BOTH
state	CLOSED	CLOSED	OPEN	CLOSED	CLOSED
	OPEN	CLOSED	OPEN	OPEN	OPEN



Finite Automata -- informal

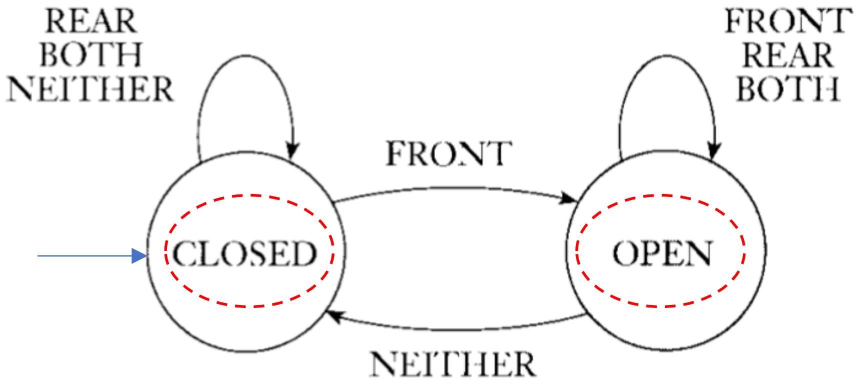
- An automatic door opens in one direction



State Diagram

Finite Automata -- informal

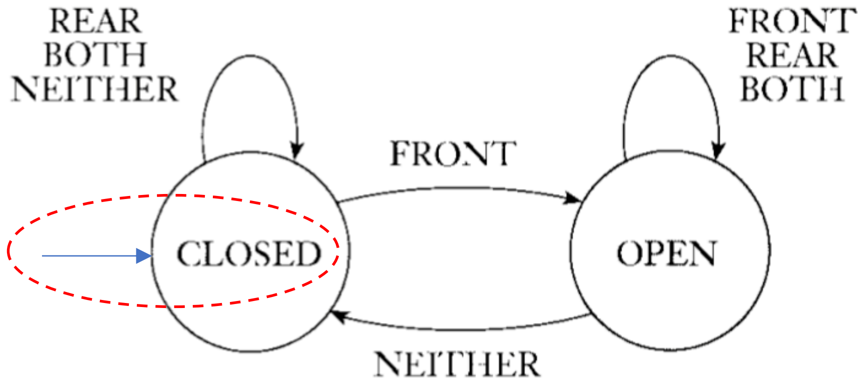
- An automatic door opens in one direction



Two states: Open/Closed

Finite Automata -- informal

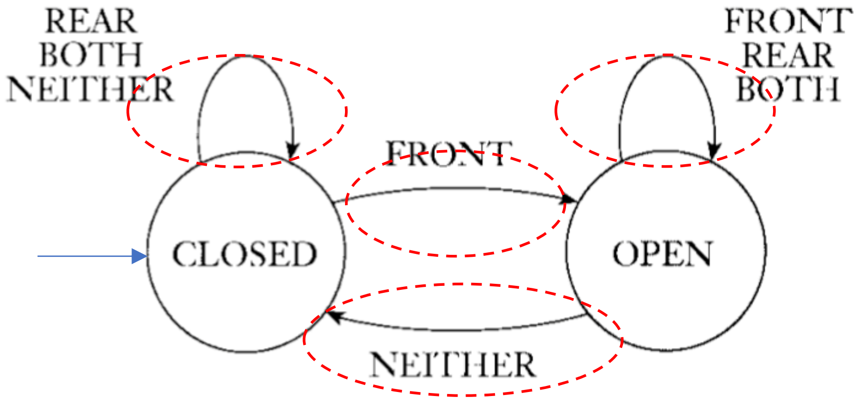
- An automatic door opens in one direction



Start states: Closed

Finite Automata -- informal

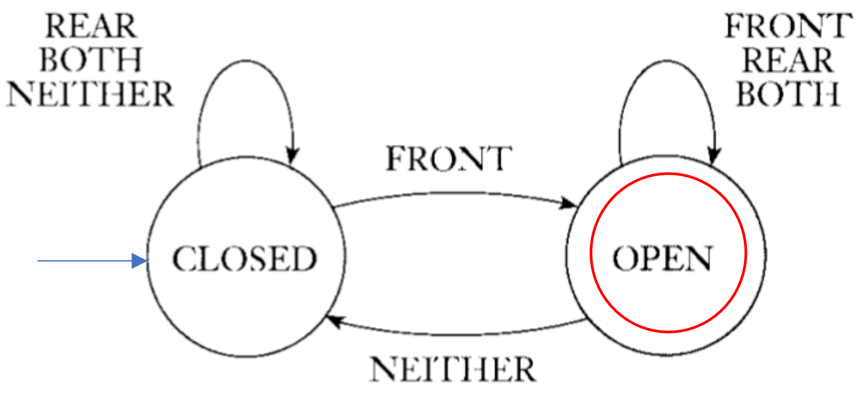
- An automatic door opens in one direction



Transitions: arrows that bring the machine from one state to another

Finite Automata -- informal

- An automatic door opens in one direction

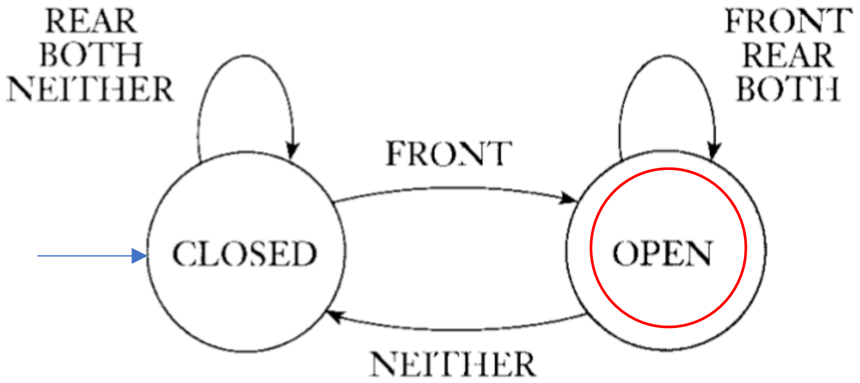


Accept state/final state: state with a double cycle

We want to restrict our attention to a simplified scenario where machines are used to " compute" Yes/No

Finite Automata -- informal

- An automatic door opens in one direction

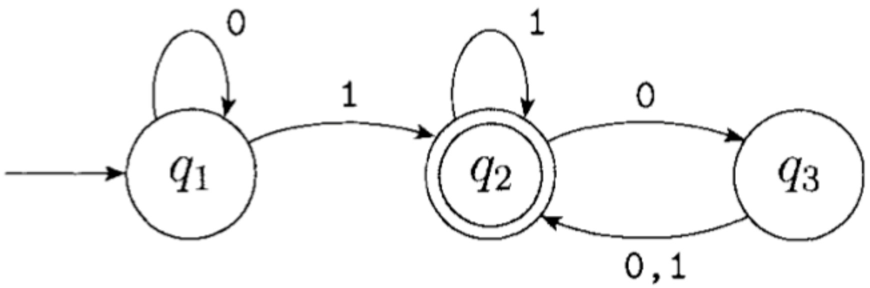


Accept state/final state: state with a double cycle

We want to restrict our attention to a simplified scenario where machines are used to " compute" Yes/No

Finite Automata -- informal

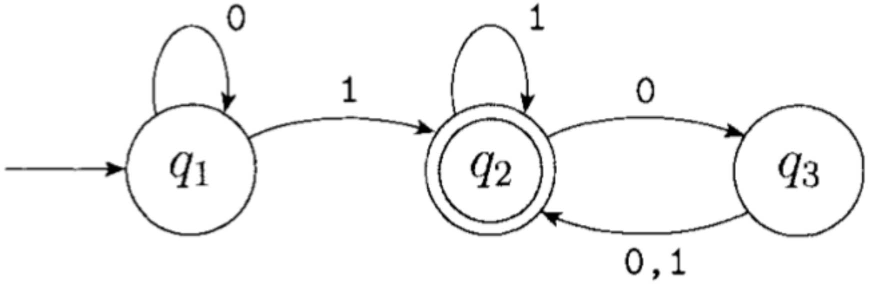
- Example



Start states *Final/Accept states*

Finite Automata -- informal

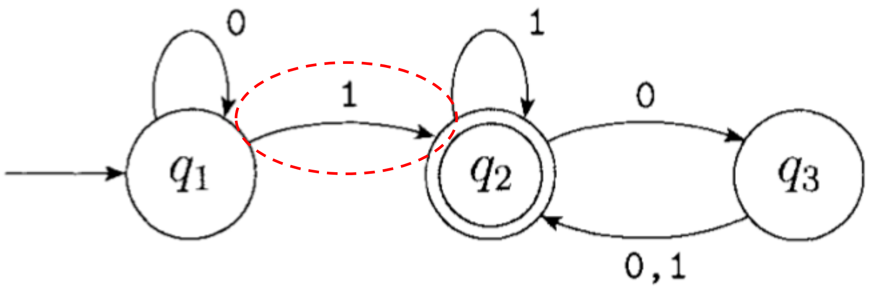
- Example



What happens when we input 1101?

Finite Automata -- informal

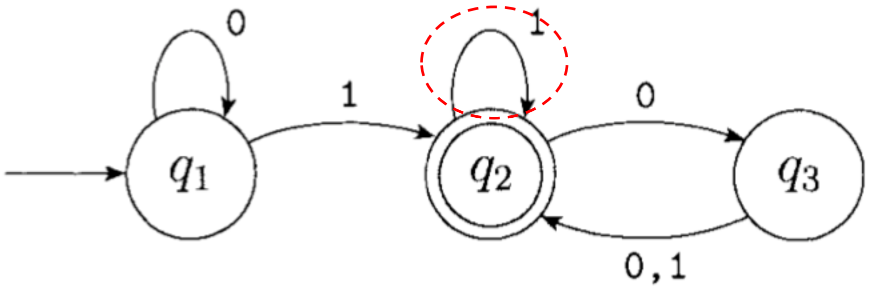
- Example



What happens when we input 1101?

Finite Automata -- informal

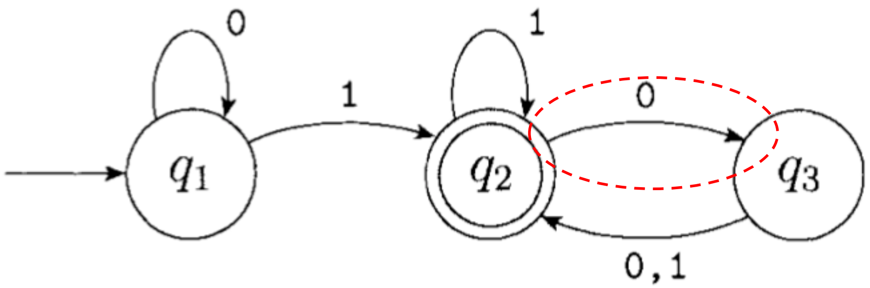
- Example



What happens when we input 1101?

Finite Automata -- informal

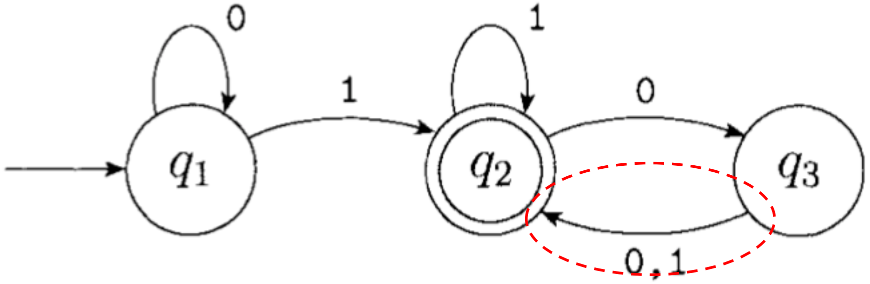
- Example



What happens when we input 1101?

Finite Automata -- informal

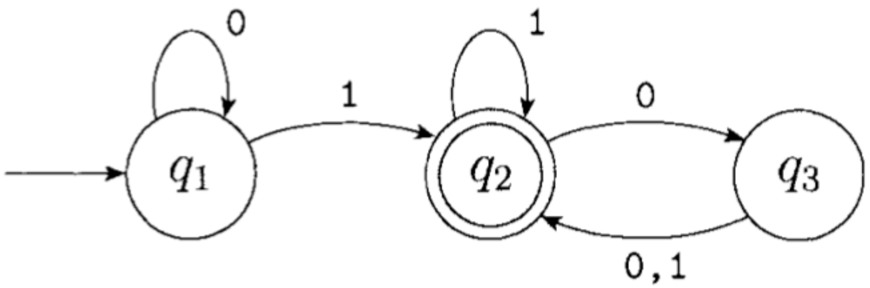
- Example



What happens when we input 1101?

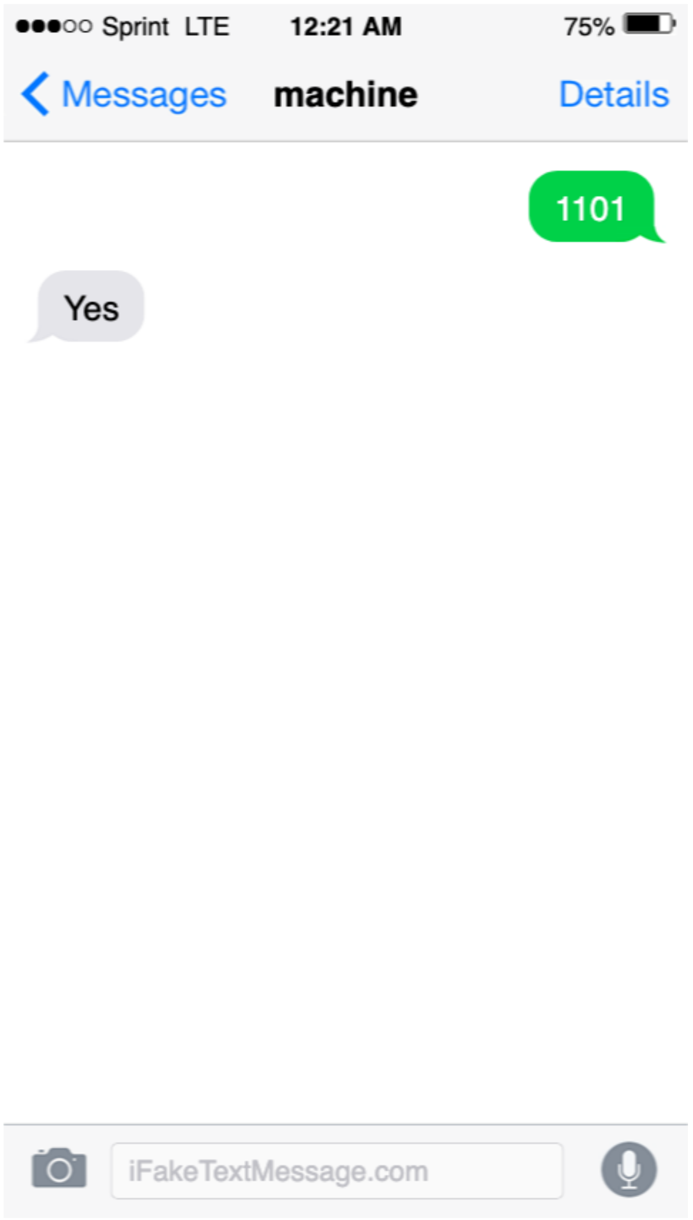
Finite Automata -- informal

- Example



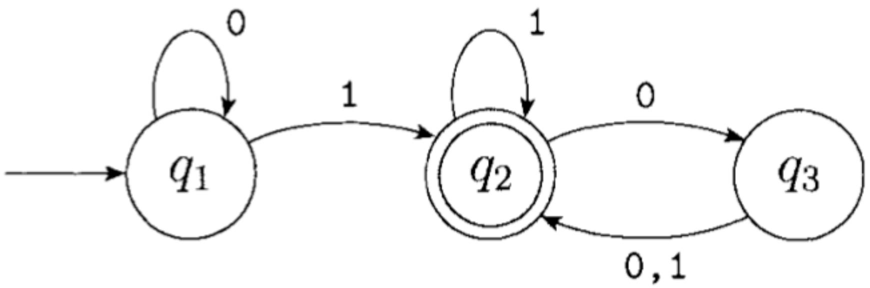
What happens when we input 1101?

Machines ends at an accept state, i.e., machine output: Yes



Finite Automata -- informal

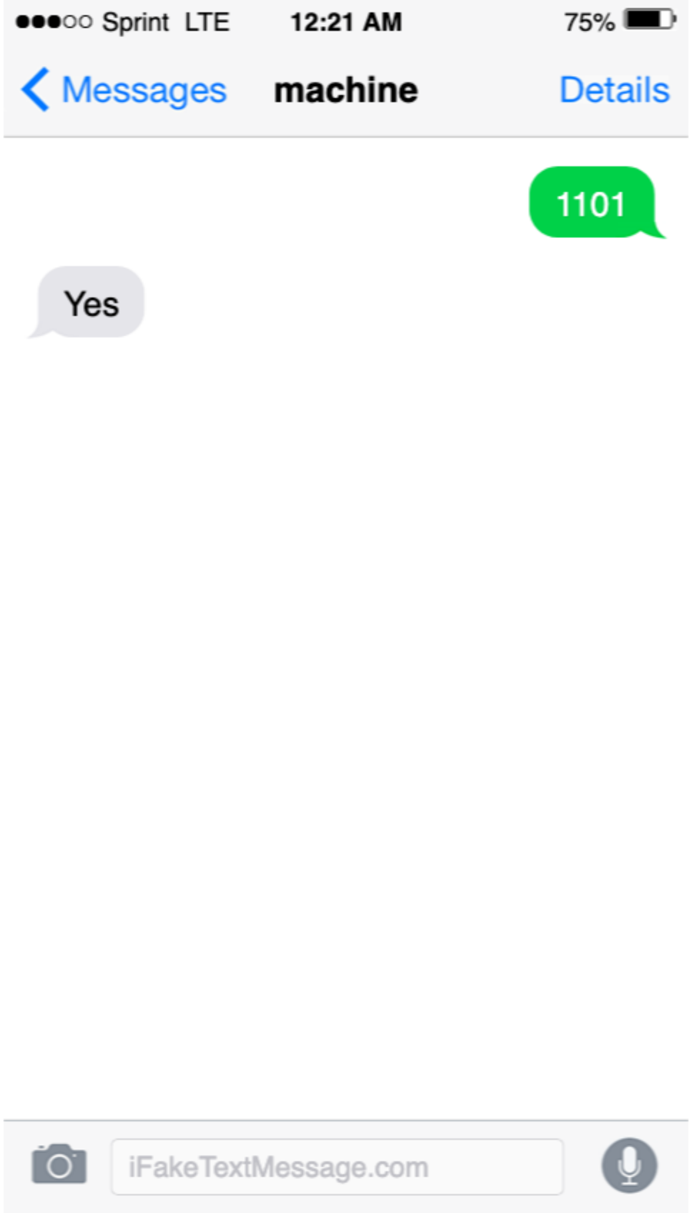
- Example



What happens when we input 1101?

Machines ends at an accept state, i.e., machine output: Yes

What happens when we input 101000?

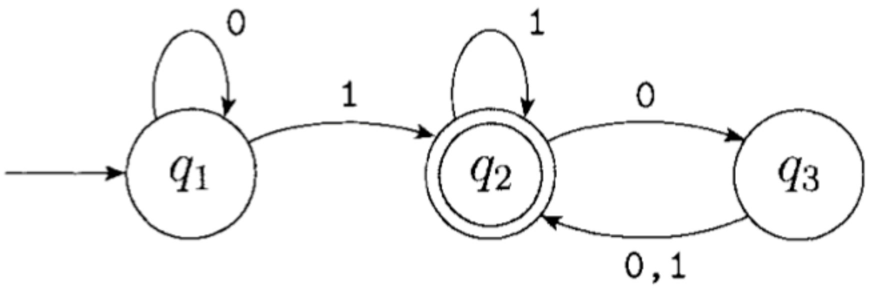


Deterministic Finite Automata -- Formal

- A quintuple $M = (Q, \Sigma, \delta, q_0, F)$ where
  - Q is a finite set of states
  - Σ is an alphabet
  - $q_0 \in Q$ is the initial state
  - $F \subseteq Q$ is the set of final/accept states (can be multiple)
  - δ, the transition function, a function from $Q \times \Sigma$ to Q

Deterministic Finite Automata

- Example



1. $Q = \{q_1, q_2, q_3\}$,

2. $\Sigma = \{0,1\}$,

3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 is the start state, and

5. $F = \{q_2\}$.

Deterministic Finite Automata

- A deterministic finite automaton M can be viewed as a classifier that filters out all the strings it accepts

  -- The set A of all the strings M accepts is the language of machine M

  -- Denote as $L(M) = A$

  -- M recognizes/accepts A

Deterministic Finite Automata

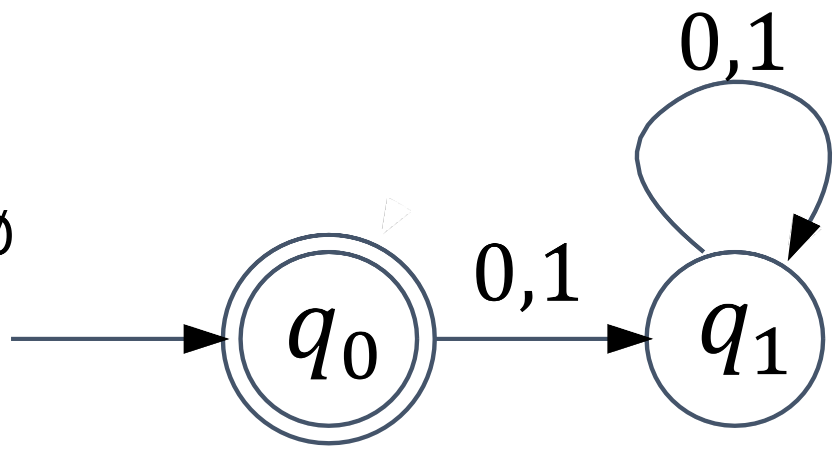
- A deterministic finite automaton M can be viewed as a classifier that filters out all the strings it accepts

-- The set A of all the strings M accepts is the language of machine M

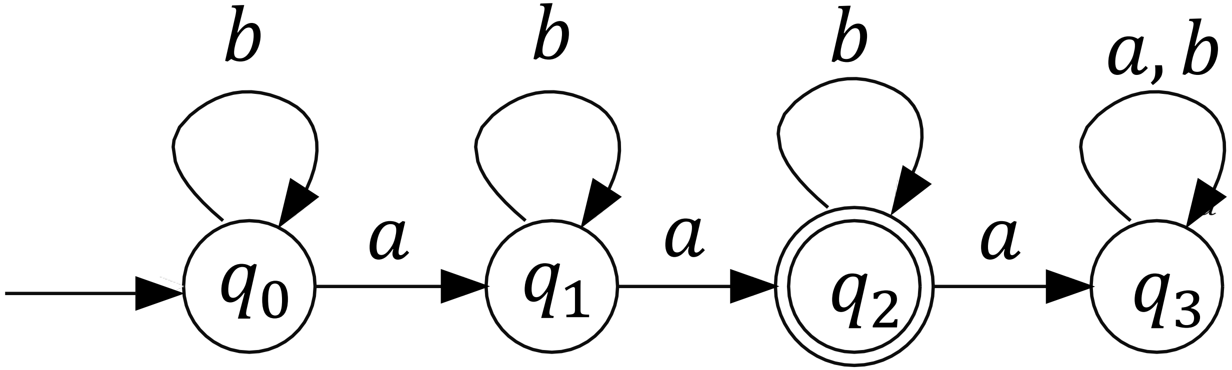
-- Denote as $L(M) = A$

-- M recognizes/accepts A

What if M rejects all inputs? M accepts $\emptyset$



DFA Examples



- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{a, b\}$
- $\delta = \{((q_0, a), q_1), ((q_0, b), q_0), ((q_1, a), q_2), ((q_1, b), q_1), ((q_2, a), q_3), ((q_2, b), q_2), ((q_3, a), q_3), ((q_3, b), q_3)\}$
- start state $= q_0$
- $F = \{q_2\}$

DFA Configurations

- Deterministic finite automata are

  - Deterministic: given the current state and next input symbol, it moves deterministically to a next state.

DFA Configurations

- Deterministic finite automata are

  - Deterministic: given the current state and next input symbol, it moves deterministically to a next state.

  - Finite: consists of finite number of states

  - Automata: machine

DFA Configurations

- We have learned two ways of describing a DFA
  - A quintuple $M = (Q, \Sigma, \delta, q_0, F)$
  - A state diagram
- How do we characterize the computation of a DFA?

DFA Configurations

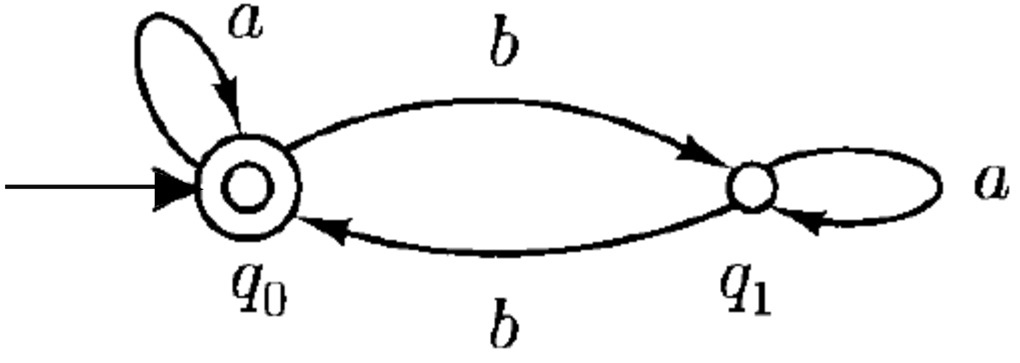
- We have learned two ways of describing a DFA
  - A quintuple $M = (Q, \Sigma, \delta, q_0, F)$
  - A state diagram
- How do we characterize the computation of a DFA?
  - the computation of a DFA has to be defined on a specific input
  - use a sequence of configurations to represent the computation

DFA Configurations

- Configuration for a DFA $M = (K, \Sigma, \delta, s, F)$
  - any element of $K \times \Sigma^*$
  - the state the DFA currently in
  - the remaining part of the string to be processed

DFA Configurations

- Configuration for a DFA $M = (K, \Sigma, \delta, s, F)$
  - any element of $K \times \Sigma^*$
  - the state the DFA currently in
  - the remaining part of the string to be processed

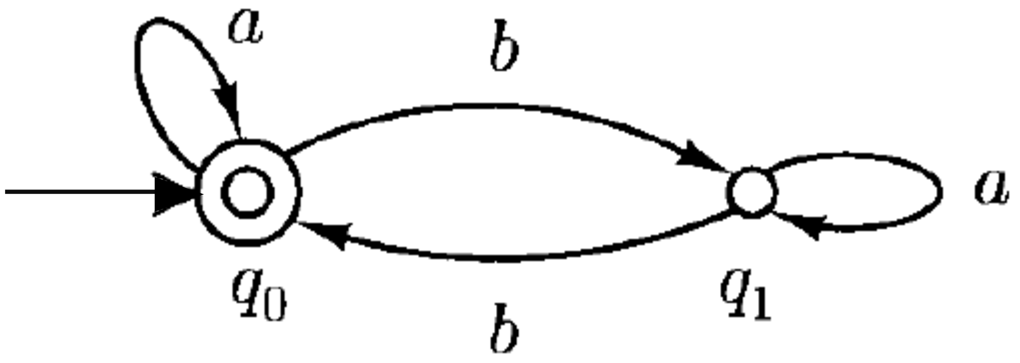


Input string: $aabba$

$$
\begin{aligned}
(q_0, aabba) &\vdash_M (q_0, abba) \\
&\vdash_M (q_0, bba) \\
&\vdash_M (q_1, ba) \\
&\vdash_M (q_0, a) \\
&\vdash_M (q_0, e)
\end{aligned}
$$

DFA Configurations

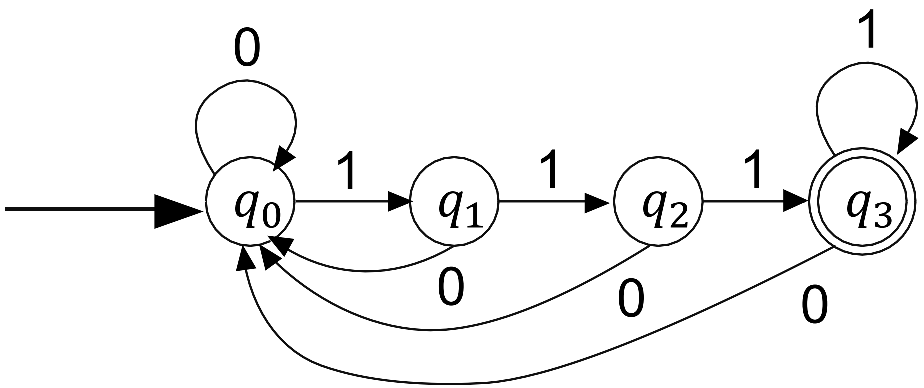
- We use a binary relation $\vdash_M$ to denote that DFA pass from one state to another state as a result of a single move
- $\vdash_M$ is a function from $K \times \Sigma^*$ to $K \times \Sigma^+$ $(\mathrm{L}^+ = LL^*)$



Input string: $aabba$

$$
\begin{aligned}
(q_0, aabba) &\vdash_M (q_0, abba) \\
&\vdash_M (q_0, bba) \\
&\vdash_M (q_1, ba) \\
&\vdash_M (q_0, a) \\
&\vdash_M (q_0, e)
\end{aligned}
$$

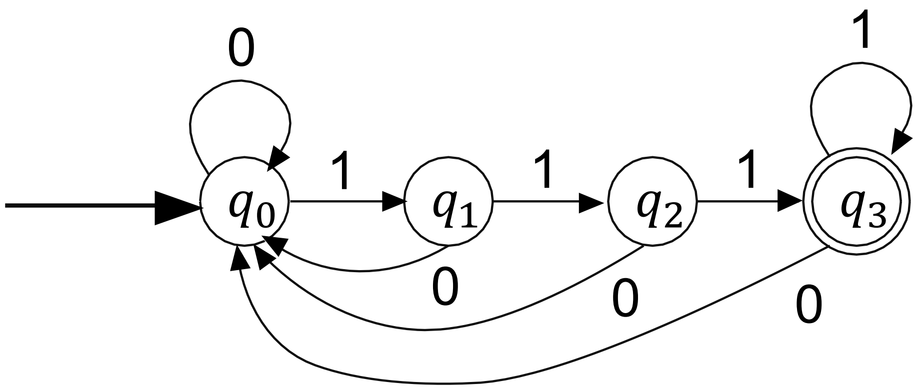
DFA Configurations



$$
\begin{aligned}
(q_0, 11101) &\vdash_M (q_1, 1101)\\
&\vdash_M (q_2, 101)\\
&\vdash_M (q_3, 01)\\
&\vdash_M (q_0, 1)\\
&\vdash_M (q_1, e)
\end{aligned}
$$

$(q_0, 11101) \vdash_M^* (q_1, e)$ $\quad$ $\vdash_M^*$: yields in 0 or more steps

DFA Configurations



$$
\begin{aligned}
(q_0, 10111) &\vdash_M (q_1, 0111) \\
&\vdash_M (q_0, 111) \\
&\vdash_M (q_1, 11) \\
&\vdash_M (q_2, 1) \\
&\vdash_M (q_3, e)
\end{aligned}
$$

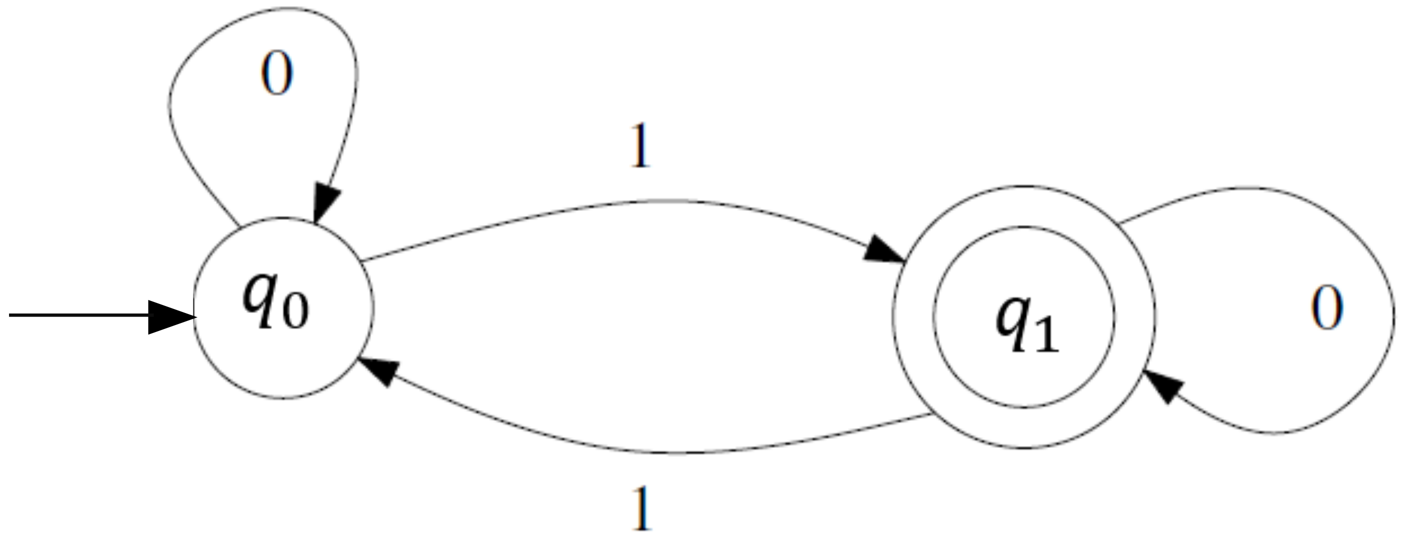
$(q_0, 10111) \vdash_M^* (q_3, e)$ $\vdash_M^*$: yields in 0 or more steps

Regular language

- M accepts a string w if $(q_0, w) \vdash_M^* (q, e)$ for some $q \in F$
- M recognize language A if $A = \{w: M \text{ accepts } w\}$
- A language is regular if some finite automaton recognizes it.

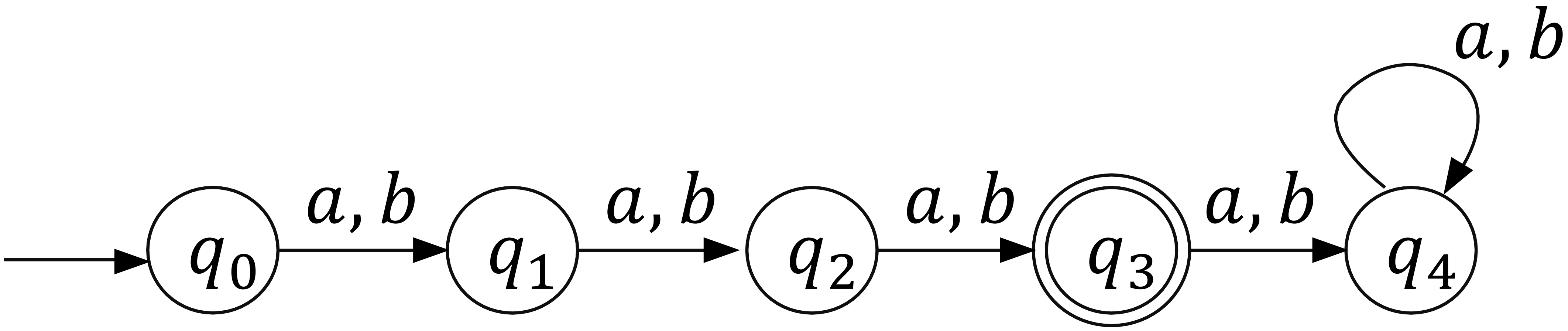
DFA –> regular language

- All binary strings containing an odd number of 1s



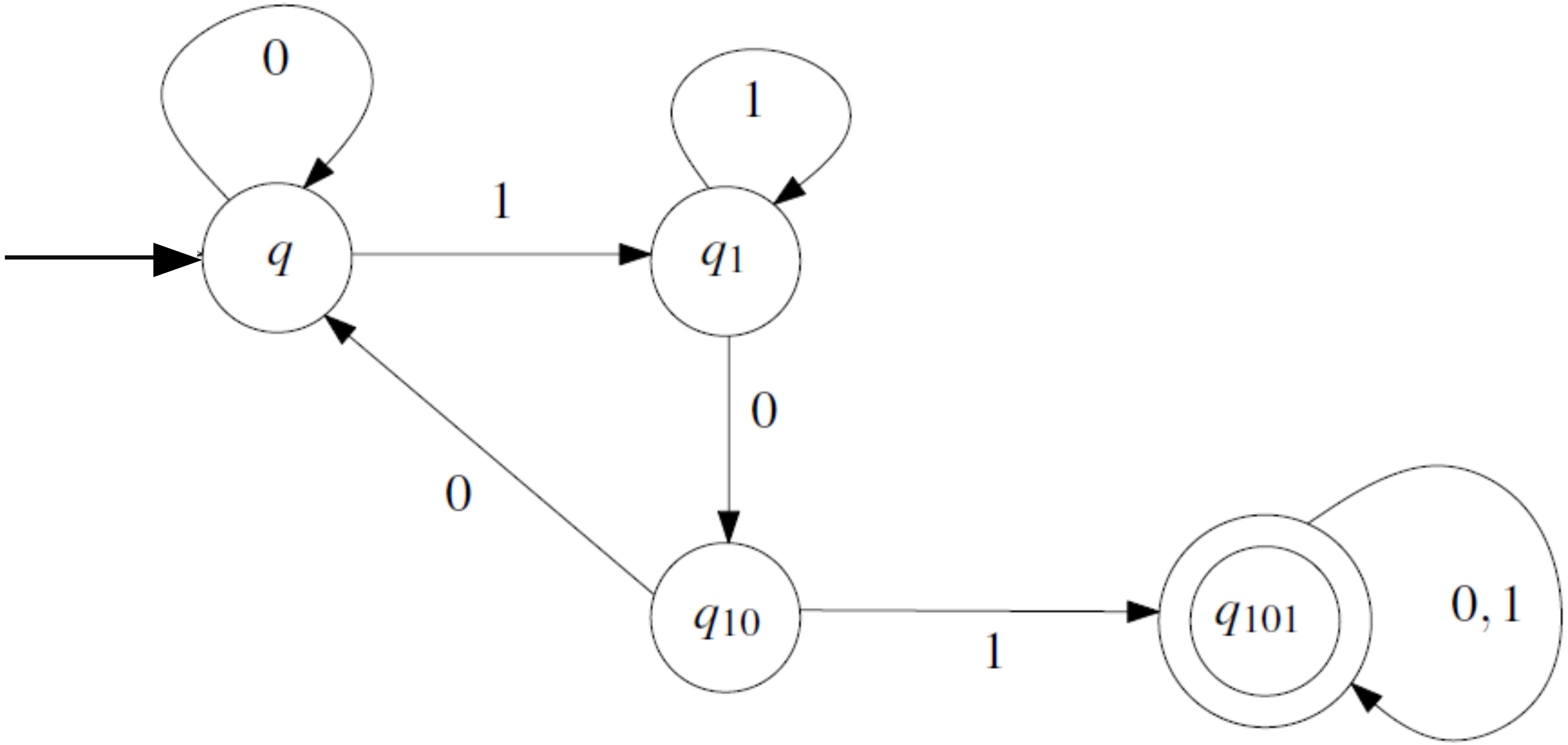
DFA –> regular language

- Example Deterministic Finite Automaton
  - All strings over $\{a, b\}$ that have length 3.



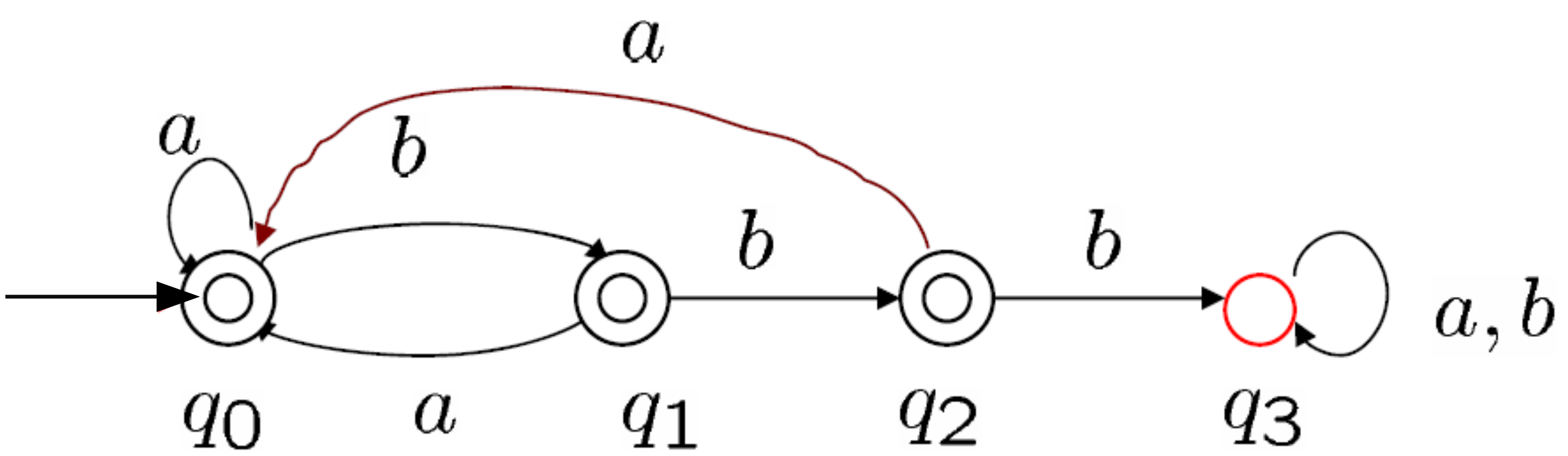
DFA –> regular language

- All strings over $\{0, 1\}$ that contain the substring 101



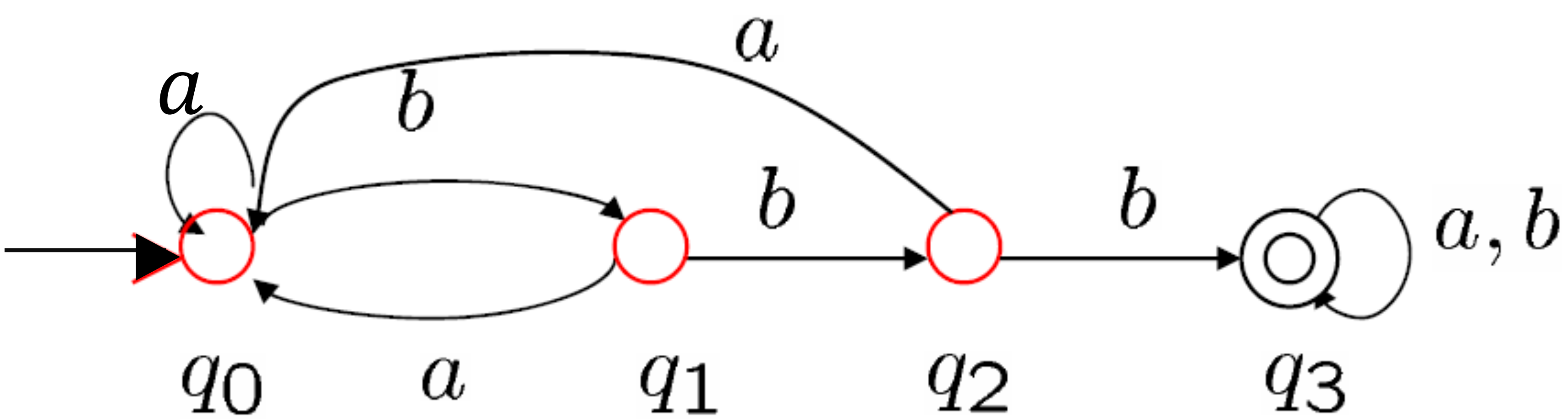
DFA –> regular language

- All strings over $\{a, b\}$ that does not contain three consecutive b' s



DFA Examples

- All strings over $\{a, b\}$ that contains three consecutive b' s



Regular language

- So far we have learned: given an automaton, determine the language it accepts
- Given a regular language, can we design an automaton recognizing it?

Regular language -> DFA

- All strings consisting of $\{0,1\}$ and have an odd number of 1s.

Regular language -> DFA

- All strings consisting of $\{0,1\}$ and have an odd number of 1s.

  -- how would you design an algorithm achieving this?

  1. read every symbol and check if it is 1;
  2. set up a counter, counter -> counter +1 if the symbol is 1;
  3. check odd/even of the counter

Regular language -> DFA

- All strings consisting of $\{0,1\}$ and have an odd number of 1s.

  -- how would you design an algorithm achieving this?

  1. read every symbol and check if it is 1;
  2. set up a counter, counter -> counter +1 if the symbol is 1;
  3. check odd/even of the counter

  Can we do better without memory?

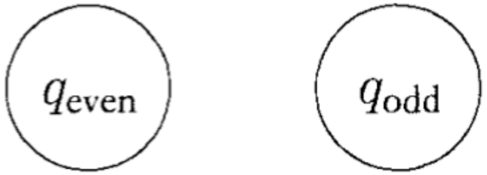
Regular language -> DFA

- All strings consisting of $\{0,1\}$ and have an odd number of 1s.

  -- how would you design an algorithm achieving this?

  1. read every symbol and check if it is 1;
  2. set up a counter, counter -> counter +1 if the symbol is 1;
  3. check odd/even of the counter

Identify states



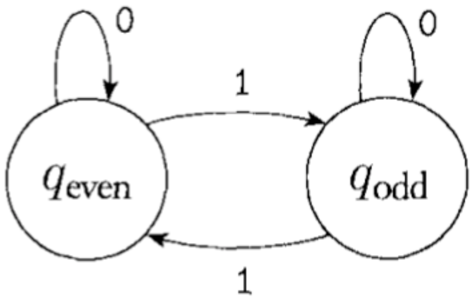
Regular language -> DFA

- All strings consisting of $\{0,1\}$ and have an odd number of 1s.

  -- how would you design an algorithm achieving this?

  1. read every symbol and check if it is 1;
  2. set up a counter, counter -> counter +1 if the symbol is 1;
  3. check odd/even of the counter

Identify transitions



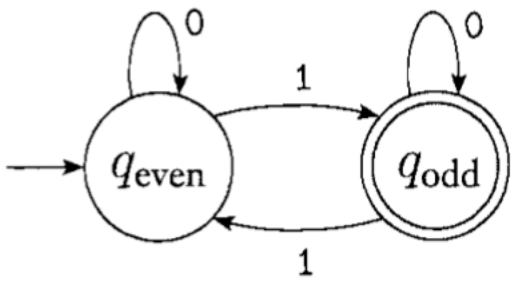
Regular language -> DFA

- All strings consisting of $\{0,1\}$ and have an odd number of 1s.

  -- how would you design an algorithm achieving this?

  1. read every symbol and check if it is 1;
  2. set up a counter, counter -> counter +1 if the symbol is 1;
  3. check odd/even of the counter

Identify start and final states



Regular language -> DFA

- All strings consisting of $\{0,1\}$ and have 001 as a substring.

Identify states: what are the states?

Regular language -> DFA

- All strings consisting of $\{0,1\}$ and have 001 as a substring.

Identify states: what are the states?

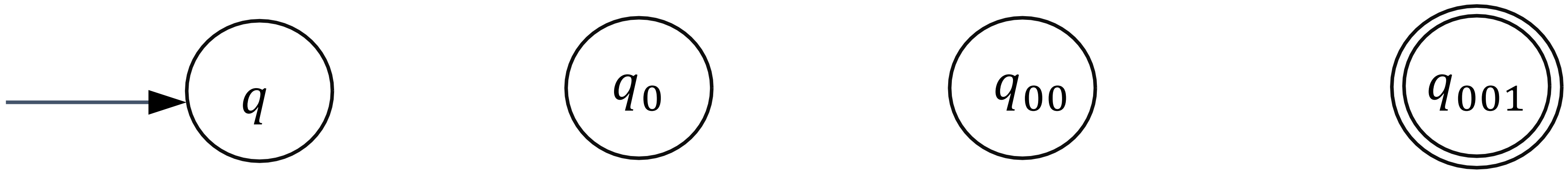
1. haven't just seen any symbols of the pattern,
2. have just seen a 0,
3. have just seen 00, or
4. have seen the entire pattern 001.

q $\quad$ q_0 $\quad$ q_{00} $\quad$ q_{001}

Regular language -> DFA

- All strings consisting of $\{0,1\}$ and have 001 as a substring.

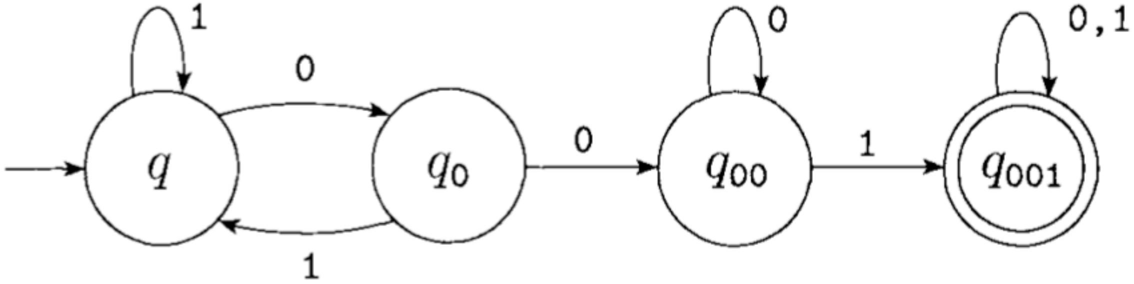
Start and final states?



Regular language -> DFA

- All strings consisting of $\{0,1\}$ and have 001 as a substring.

Transitions



Regular operations

- How to design DFA for very complicated regular language? Is there a systematic way (or "Algorithm" or designing DFA)?

Regular operations

- **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

Regular operations

- **Union**: $A \cup B = \{x | x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy | x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \ldots x_k | k \geq 0 \text{ and each } x_i \in A\}$.

$$A = \{\text{good}, \text{bad}\}, B = \{\text{boy}, \text{girl}\}$$

$A \cup B = \{\texttt{good}, \texttt{bad}, \texttt{boy}, \texttt{girl}\}$,

$A \circ B = \{\texttt{goodboy}, \texttt{goodgirl}, \texttt{badboy}, \texttt{badgirl}\}$, and

$$A^* = \{\varepsilon, \text{good}, \text{bad}, \text{goodgood}, \text{goodbad}, \text{badgood}, \text{badbad}, \\ \text{goodgoodgood}, \text{goodgoodbad}, \text{goodbadgood}, \text{goodbadbad}, \ldots\}.$$

Regular operations

- How to design DFA for very complicated regular language? Is there a systematic way (or "Algorithm" or designing DFA)?
- Regular language is closed under union, concatenation and star, but how can we prove it? We will need a more flexible version of automata…