CS 301

Lecture 02 – Deterministic Finite Automata (DFAs)

Stephen Checkoway

January 22, 2018

Review from last time

Alphabet Finite, nonempty set of symbols

String Finite-length sequence of symbols from an alphabet

Language Set of strings over an alphabet

	Can be empty	Can be infinite
Alphabet	✗	✗
String	✓	✗
Language	✓	✓

If Σ is an alphabet, then Σ^* is the language consisting of all strings over Σ

State machines

A state machine is a way to structure computation

It consists of

- a fixed set of states
- a fixed initial state
- a specification of what action to take in response to input for each state
- a current “active” state

State machine example: An automatic swinging door

The door has a front and a back sensor

We want to open the door when the front sensor is triggered, as long as it doesn't hit someone (i.e., as long as the back sensor is not triggered)

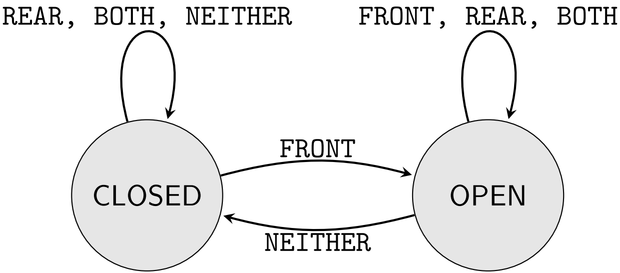
We want to close the door when the front sensor is not triggered, as long as it doesn't hit someone

State machine example: An automatic swinging door

The door can be either OPEN or CLOSED

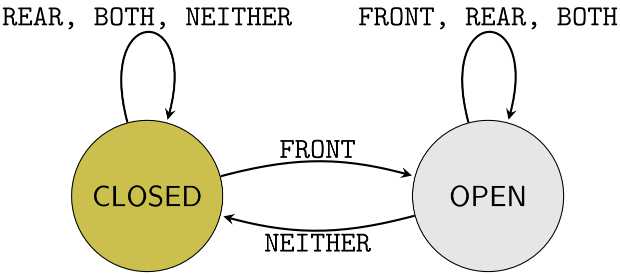
Possible inputs to the state machine:

`FRONT`	Someone is standing on the front sensor
`REAR`	Someone is standing on the rear sensor
`BOTH`	Someone is standing on both sensors
`NEITHER`	No one is standing on either sensor



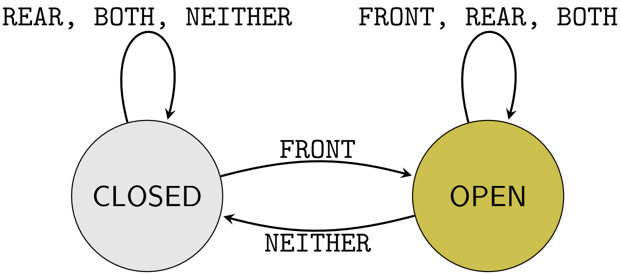
State machine example: An automatic swinging door

1. Initially the door is CLOSED



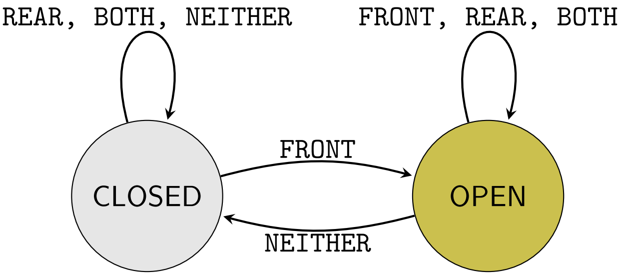
State machine example: An automatic swinging door

1. Initially the door is CLOSED
2. Alice stands on the FRONT sensor and the door changes to OPEN



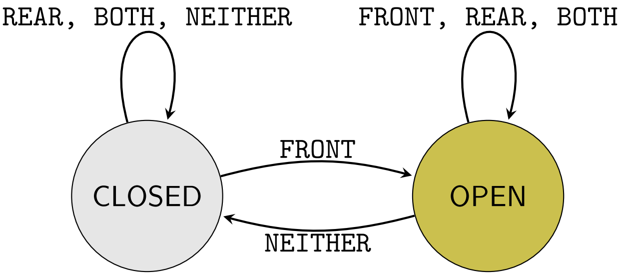
State machine example: An automatic swinging door

1. Initially the door is CLOSED
2. Alice stands on the FRONT sensor and the door changes to OPEN
3. Alice enters as Bob approaches the door so BOTH sensors are triggered and the door stays OPEN



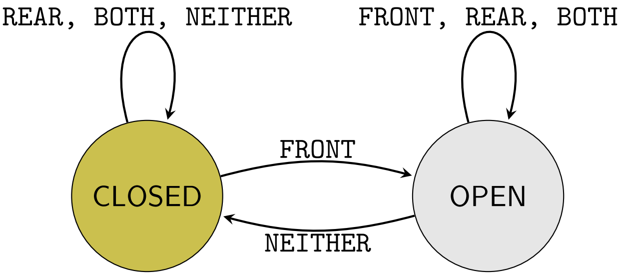
State machine example: An automatic swinging door

1. Initially the door is CLOSED
2. Alice stands on the FRONT sensor and the door changes to OPEN
3. Alice enters as Bob approaches the door so BOTH sensors are triggered and the door stays OPEN
4. Alice moves away as Bob enters so only the REAR sensor is triggered and the door stays OPEN

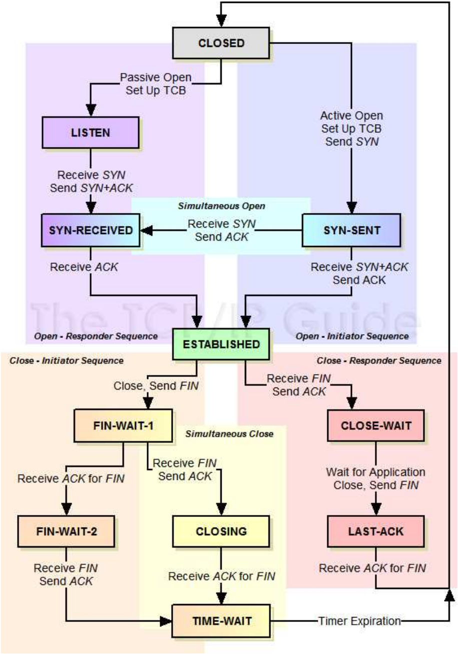


State machine example: An automatic swinging door

1. Initially the door is CLOSED
2. Alice stands on the FRONT sensor and the door changes to OPEN
3. Alice enters as Bob approaches the door so BOTH sensors are triggered and the door stays OPEN
4. Alice moves away as Bob enters so only the REAR sensor is triggered and the door stays OPEN
5. Bob moves away so NEITHER sensor is triggered and the door changes to CLOSED



State machine example: TCP



State machine example: TLS 1.3

```
                              START <----+
               Send ClientHello |        | Recv HelloRetryRequest
          [K_send = early data] |        |
                                v        |
           /                 WAIT_SH ----+
           |                     | Recv ServerHello
           |                     | K_recv = handshake
       Can |                     V
      send |                 WAIT_EE
     early |                     | Recv EncryptedExtensions
      data |            +--------+--------+
           |      Using |                 | Using certificate
           |        PSK |                 v
           |            |            WAIT_CERT_CR
           |            |        Recv |       | Recv CertificateRequest
           |            | Certificate |       v
           |            |             |    WAIT_CERT
           |            |             |       | Recv Certificate
           |            |             v       v
           |            |              WAIT_CV
           |            |                 | Recv CertificateVerify
           |            +> WAIT_FINISHED <+
           |                  | Recv Finished
           \                  | [Send EndOfEarlyData]
                              | K_send = handshake
                              | [Send Certificate [+ CertificateVerify]]
    Can send                  | Send Finished
    app data   -->            | K_send = K_recv = application
    after here                v
                          CONNECTED
```

State machine example: Video games

Input is received from the controller

What does the game do with the input? Depends on what state it's in

- During normal game play: perform an action (jump, run, start a conversation)
- During a cut scene: nothing or maybe end the cut scene
- During a loading screen: nothing
- . . .

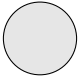
Deterministic finite Automaton (DFA)

DFAs are the simplest model of computation:
Given an input string, the DFA will either accept it or reject it

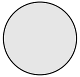
They are state machines

- The (finite set of) states are the DFA's memory
- It starts in a fixed start state
- It processes its input one symbol at a time; for each symbol, it will transition to a new state (or stay in the current state)
- At the end of the input, the state it is in determines if the input is accepted or rejected

DFA notation

The states of a DFA are represented as a circle 

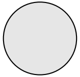
DFA notation

The states of a DFA are represented as a circle 

We will usually give the states short names like q_0 or q_1 

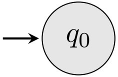
DFA notation

The states of a DFA are represented as a circle

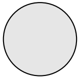


We will usually give the states short names like q_0 or q_1

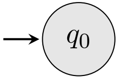
The initial state is represented with an arrow and is frequently named q_0



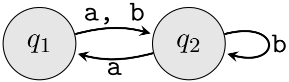
DFA notation

The states of a DFA are represented as a circle 

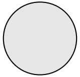
We will usually give the states short names like q_0 or q_1 

The initial state is represented with an arrow and is frequently named q_0 

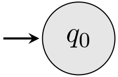
Transitions between states are given by directed edges, labeled by an alphabet symbol and every state must have exactly one transition for each symbol in the alphabet



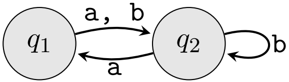
DFA notation

The states of a DFA are represented as a circle 

We will usually give the states short names like q_0 or q_1 

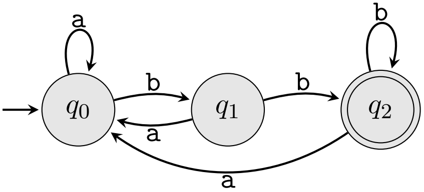
The initial state is represented with an arrow and is frequently named q_0 

Transitions between states are given by directed edges, labeled by an alphabet symbol and every state must have exactly one transition for each symbol in the alphabet



Accepting states are drawn with two circles 

DFA example



States $Q = \{q_0, q_1, q_2\}$

Alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$

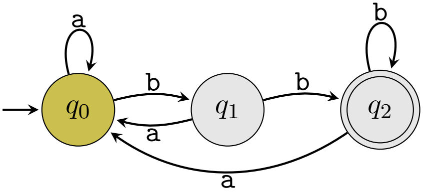
Transitions

δ	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_2

Start state q_0

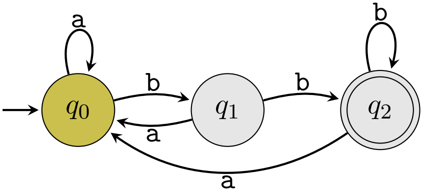
Accepting states $F = \{q_2\}$

DFA example



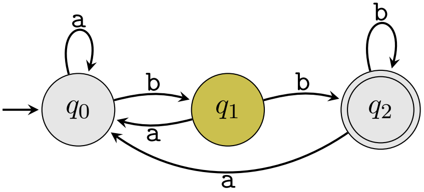
- ababb

DFA example



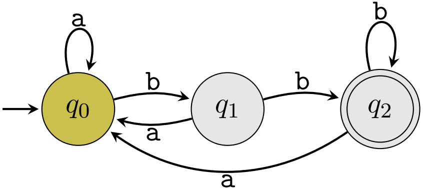
- ababb

DFA example



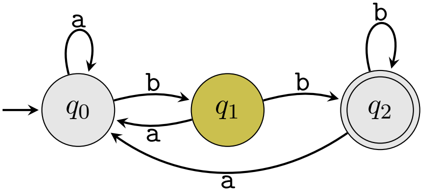
- ababb

DFA example



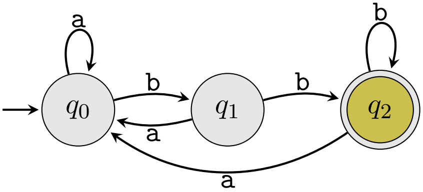
- ababb

DFA example



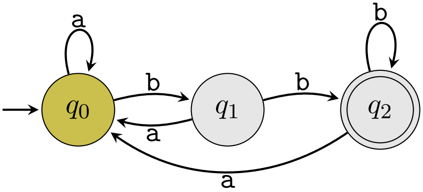
- ababb

DFA example



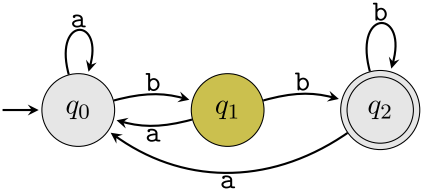
- ababb ✔Accepted

DFA example



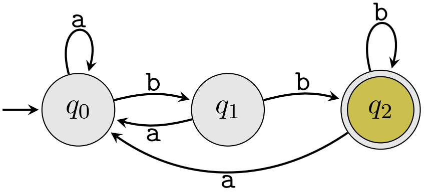
- ababb ✔Accepted
- bbab

DFA example



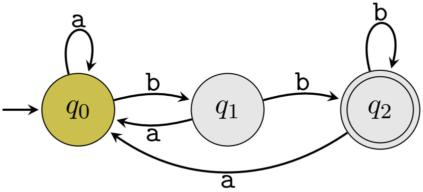
- ababb ✔Accepted
- bbab

DFA example



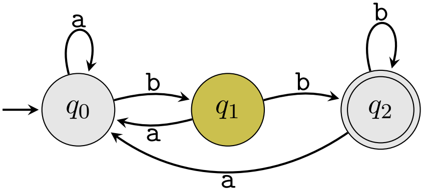
- ababb ✔Accepted
- bbab

DFA example



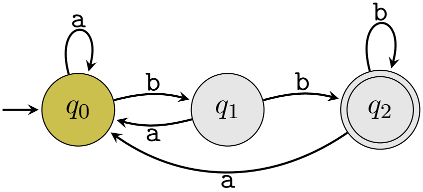
- `ababb` ✔Accepted
- `bbab`

DFA example



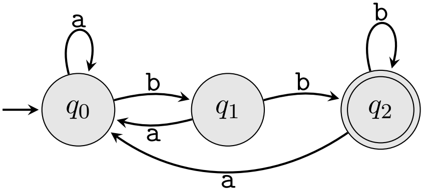
- `ababb` ✔Accepted
- `bbab` ✖Rejected

DFA example



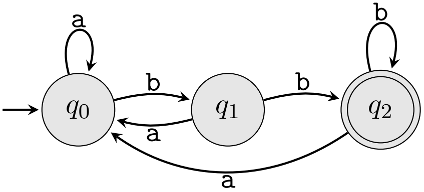
- `ababb` ✔Accepted
- `bbab` ✖Rejected
- ε

DFA example



- ababb ✔Accepted
- bbab ✖Rejected
- ε ✖Rejected

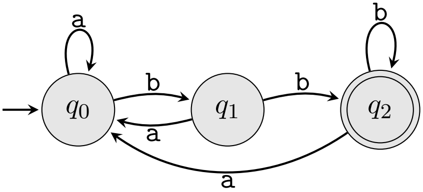
DFA example



- `ababb` ✔ Accepted
- `bbab` ✖ Rejected
- ε ✖ Rejected

What strings does this DFA accept?

DFA example



- `ababb` ✔Accepted
- `bbab` ✖Rejected
- ε ✖Rejected

What strings does this DFA accept?

Strings that end in `bb`

We can write this as a set: $\{w\texttt{bb} \mid w \in \Sigma^*\}$

Formalizing DFAs

A DFA M is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

Formalizing DFAs

A DFA M is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states

Formalizing DFAs

A DFA M is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states
- Σ is an alphabet (finite set of symbols)

Formalizing DFAs

A DFA M is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states
- Σ is an alphabet (finite set of symbols)
- $\delta : Q \times \Sigma \to Q$ is the transition function

Formalizing DFAs

A DFA M is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

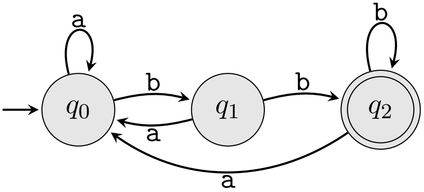
- Q is a finite set of states
- Σ is an alphabet (finite set of symbols)
- $\delta : Q \times \Sigma \to Q$ is the transition function
- $q_0 \in Q$ is the start state

Formalizing DFAs

A DFA M is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states
- Σ is an alphabet (finite set of symbols)
- $\delta : Q \times \Sigma \to Q$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accepting (or final) states

DFA example once again



States $Q = \{q_0, q_1, q_2\}$

Alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$

Transitions

δ	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_2

Start state q_0

Accepting states $F = \{q_2\}$

If we call this DFA M, then $M = (Q, \Sigma, \delta, q_0, F)$ is a complete, mathematical description of the DFA

The diagram is just helpful for humans; it doesn't contain any information not contained in in the 5 components of M

DFA acceptance and rejection

A DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w \in \Sigma^*$ if starting from the start state q_0 and moving from state to state according to the transition function δ on input w, the machine ends in one of the accepting states

If M does not accept w, then it rejects w

Language of a DFA

The language of a DFA M—written $L(M)$—is the set of strings that M accepts

$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$$

We say that M recognizes a set A to mean $L(M) = A$

DFA construction

Let's build a DFA to recognize the language
$A = \{w \mid w \text{ contains exactly one or three } 0\}$ with the alphabet $\Sigma = \{0, 1\}$

If we were writing a Python program to check if a string w has one or three 0s, it might look like this

```
count = 0
for c in w:
    if c == '0':
        count += 1
if count == 1 or count == 3:
    print("ACCEPT")
else:
    print("REJECT")
```

DFA construction

Let's build a DFA to recognize the language
$A = \{w \mid w \text{ contains exactly one or three } 0\}$ with the alphabet $\Sigma = \{0, 1\}$

If we were writing a Python program to check if a string w has one or three 0s, it might look like this

```
count = 0
for c in w:
    if c == '0':
        count += 1
if count == 1 or count == 3:
    print("ACCEPT")
else:
    print("REJECT")
```

states and initial state

DFA construction

Let's build a DFA to recognize the language
$A = \{w \mid w \text{ contains exactly one or three } 0\}$ with the alphabet $\Sigma = \{0, 1\}$

If we were writing a Python program to check if a string w has one or three 0s, it might look like this

```
count = 0
for c in w:
    if c == '0':
        count += 1
if count == 1 or count == 3:
    print("ACCEPT")
else:
    print("REJECT")
```

states and initial state

transition function

DFA construction

Let's build a DFA to recognize the language
$A = \{w \mid w \text{ contains exactly one or three } 0\}$ with the alphabet $\Sigma = \{0, 1\}$

If we were writing a Python program to check if a string w has one or three 0s, it might look like this

```python
count = 0
for c in w:
    if c == '0':
        count += 1
if count == 1 or count == 3:
    print("ACCEPT")
else:
    print("REJECT")
```

states and initial state

transition function

accept states

DFA construction

Let's build a DFA to recognize the language
$A = \{w \mid w \text{ contains exactly one or three } 0\}$ with the alphabet $\Sigma = \{0, 1\}$

Approach:

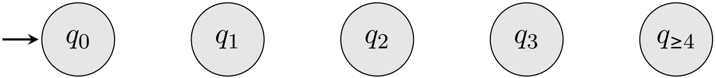
1. We need states to keep track of how many 0s the DFA has seen so far;
   How many states should the DFA have?

DFA construction

Let's build a DFA to recognize the language
$A = \{w \mid w$ contains exactly one or three 0$\}$ with the alphabet $\Sigma = \{0, 1\}$

Approach:

1. We need states to keep track of how many 0s the DFA has seen so far;
   We need five states: corresponding to 0, 1, 2, 3, and ≥ 4 '0' symbols

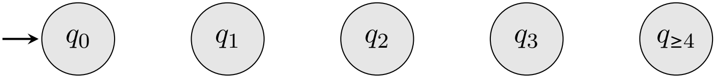


DFA construction

Let's build a DFA to recognize the language
$A = \{w \mid w \text{ contains exactly one or three } 0\}$ with the alphabet $\Sigma = \{0, 1\}$

Approach:

1. We need states to keep track of how many 0s the DFA has seen so far;
   We need five states: corresponding to 0, 1, 2, 3, and ≥ 4 '0' symbols
2. How should the DFA move from state to state?

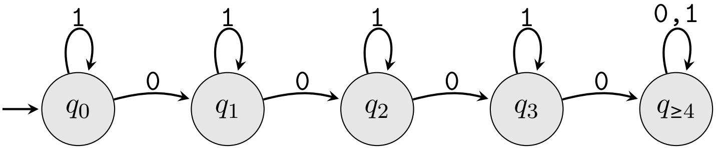


DFA construction

Let's build a DFA to recognize the language
$A = \{w \mid w$ contains exactly one or three $\mathtt{0}\}$ with the alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$

Approach:

1. We need states to keep track of how many 0s the DFA has seen so far;
   We need five states: corresponding to 0, 1, 2, 3, and ≥ 4 '0' symbols
2. On a 1, we should remain in the current state and on a 0, we should move to the next state (or stay in the ≥ 4 state)

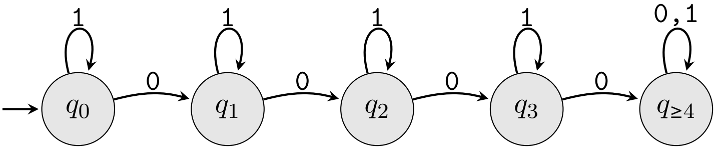


DFA construction

Let's build a DFA to recognize the language
$A = \{w \mid w \text{ contains exactly one or three } 0\}$ with the alphabet $\Sigma = \{0, 1\}$

Approach:

1. We need states to keep track of how many 0s the DFA has seen so far;
   We need five states: corresponding to 0, 1, 2, 3, and ≥ 4 '0' symbols
2. On a 1, we should remain in the current state and on a 0, we should move to the next state (or stay in the ≥ 4 state)
3. Which states should be accepting states?

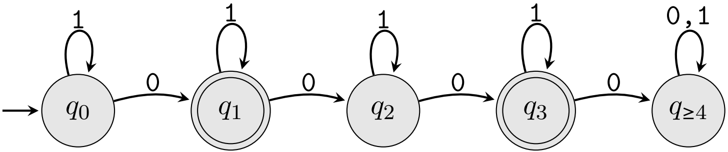


DFA construction

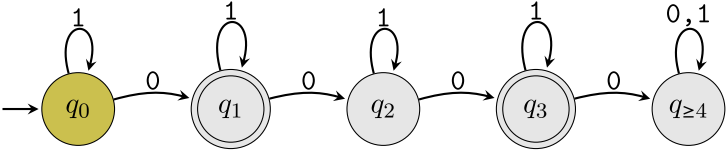
Let's build a DFA to recognize the language
$A = \{w \mid w \text{ contains exactly one or three } 0\}$ with the alphabet $\Sigma = \{0, 1\}$

Approach:

1. We need states to keep track of how many 0s the DFA has seen so far;
   We need five states: corresponding to 0, 1, 2, 3, and ≥ 4 '0' symbols
2. On a 1, we should remain in the current state and on a 0, we should move to the next state (or stay in the ≥ 4 state)
3. The states corresponding to 1 and 3 should be accepting states

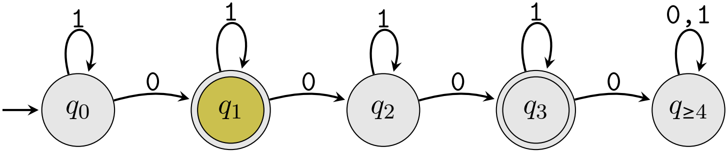


Running our DFA



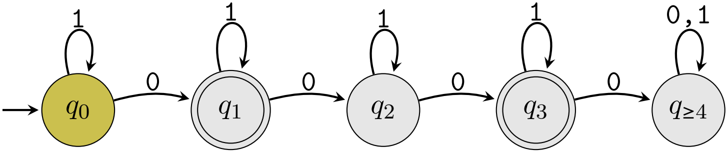
- 0

Running our DFA



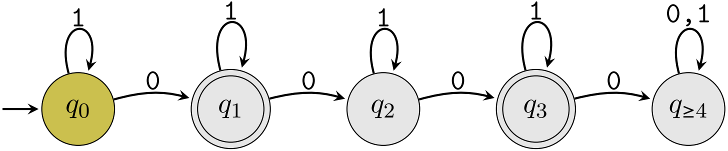
- 0 ✔Accepted

Running our DFA



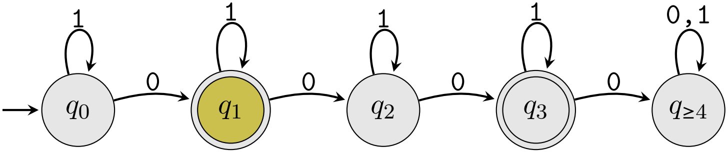
- 0 ✔Accepted
- 10101

Running our DFA



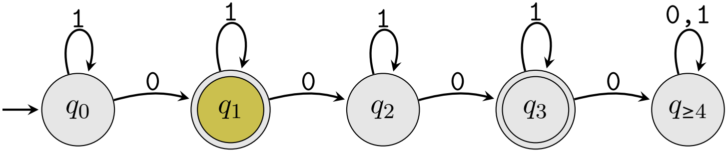
- 0 ✔Accepted
- 10101

Running our DFA



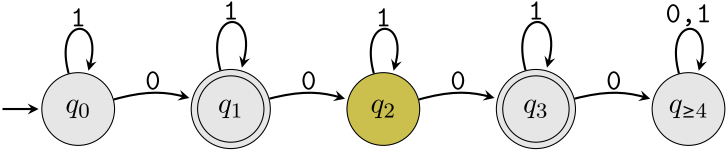
- `0` ✔Accepted
- `10101`

Running our DFA



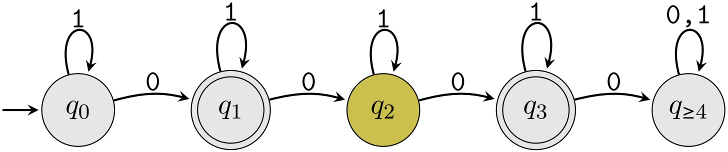
- 0 ✔Accepted
- 10101

Running our DFA



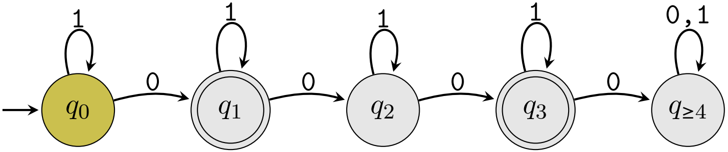
- 0 ✔Accepted
- 10101

Running our DFA



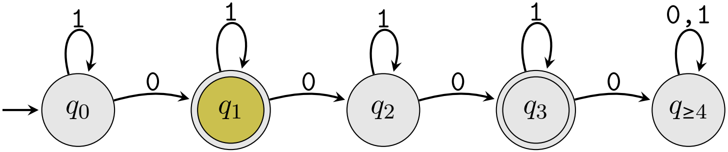
- 0 ✔Accepted
- 10101 ✖Rejected

Running our DFA



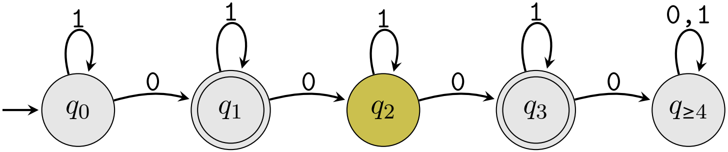
- 0 ✔ Accepted
- 10101 ✘ Rejected
- 000

Running our DFA



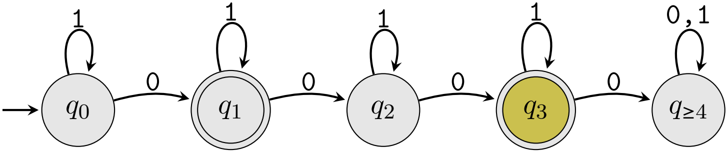
- 0 ✔Accepted
- 10101 ✖Rejected
- 000

Running our DFA



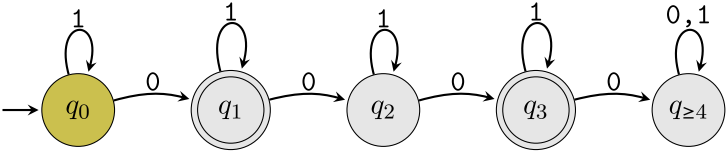
- 0 ✔Accepted
- 10101 ✖Rejected
- 000

Running our DFA



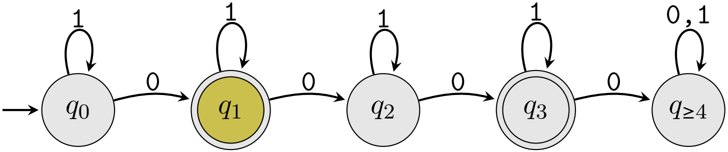
- 0 ✔Accepted
- 10101 ✖Rejected
- 000 ✔Accepted

Running our DFA



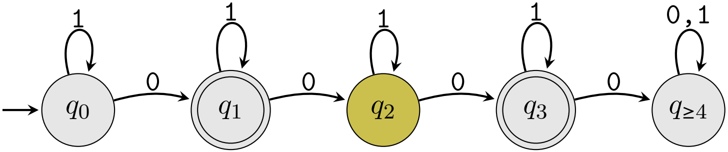
- 0 ✔Accepted
- 10101 ✖Rejected
- 000 ✔Accepted
- 00000

Running our DFA



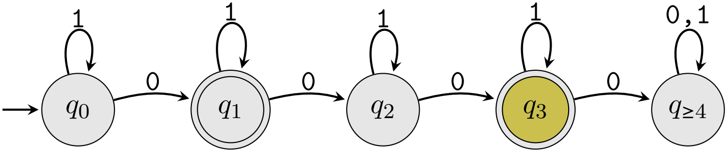
- 0 ✔Accepted
- 10101 ✖Rejected
- 000 ✔Accepted
- 00000

Running our DFA



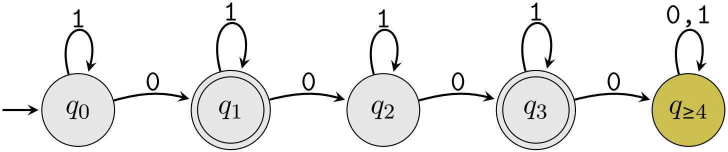
- 0 ✔Accepted
- 10101 ✖Rejected
- 000 ✔Accepted
- 00000

Running our DFA



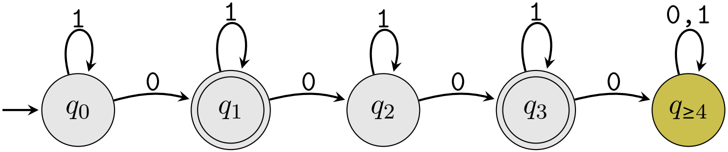
- 0 ✔Accepted
- 10101 ✖Rejected
- 000 ✔Accepted
- 00000

Running our DFA



- 0 ✔Accepted
- 10101 ✖Rejected
- 000 ✔Accepted
- 00000

Running our DFA



- 0 ✔Accepted
- 10101 ✖Rejected
- 000 ✔Accepted
- 00000 ✖Rejected

Formalizing DFA computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $w = w_1 w_2 \cdots w_n$ be a string where $w_i \in \Sigma$

M accepts w if there exist states $r_0, r_1, \ldots, r_n \in Q$ such that

Formalizing DFA computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $w = w_1 w_2 \cdots w_n$ be a string where $w_i \in \Sigma$

M accepts w if there exist states $r_0, r_1, \ldots, r_n \in Q$ such that

1. $r_0 = q_0$
   [The DFA starts in the start state]

Formalizing DFA computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $w = w_1 w_2 \cdots w_n$ be a string where $w_i \in \Sigma$

M accepts w if there exist states $r_0, r_1, \ldots, r_n \in Q$ such that

1. $r_0 = q_0$
   [The DFA starts in the start state]
2. $r_i = \delta(r_{i-1}, w_i)$ for $i \in \{1, 2, \ldots, n\}$
   [The DFA moves from state to state according to δ]

Formalizing DFA computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $w = w_1 w_2 \cdots w_n$ be a string where $w_i \in \Sigma$

M accepts w if there exist states $r_0, r_1, \ldots, r_n \in Q$ such that

1. $r_0 = q_0$
   [The DFA starts in the start state]
2. $r_i = \delta(r_{i-1}, w_i)$ for $i \in \{1, 2, \ldots, n\}$
   [The DFA moves from state to state according to δ]
3. $r_n \in F$
   [The DFA ends in an accepting state]

Formalizing DFA computation

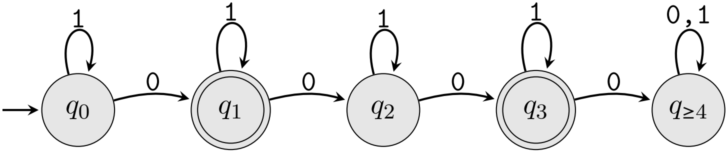
Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $w = w_1 w_2 \cdots w_n$ be a string where $w_i \in \Sigma$

M accepts w if there exist states $r_0, r_1, \ldots, r_n \in Q$ such that

1. $r_0 = q_0$
   [The DFA starts in the start state]
2. $r_i = \delta(r_{i-1}, w_i)$ for $i \in \{1, 2, \ldots, n\}$
   [The DFA moves from state to state according to δ]
3. $r_n \in F$
   [The DFA ends in an accepting state]

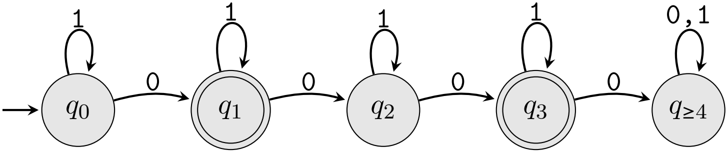
The sequence of $n + 1$ states $r_0, r_1, \ldots, r_n$ are the states that the DFA moves through on input w

Examples



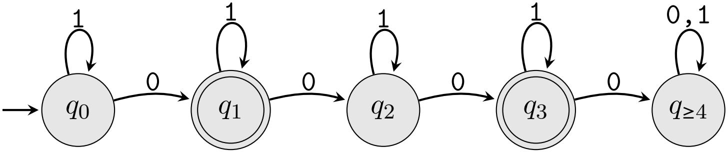
Input	States $r_0, r_1, \ldots, r_n$	Accepted/Rejected
ε	q_0	
0		
10101		
000		
00000		

Examples



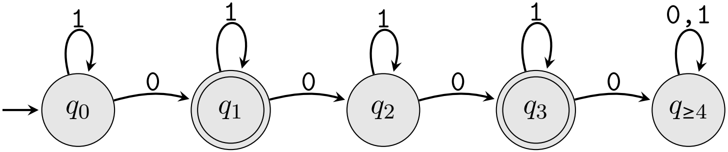
Input	States $r_0, r_1, \ldots, r_n$	Accepted/Rejected
ε	q_0	✖Rejected
0		
10101		
000		
00000		

Examples



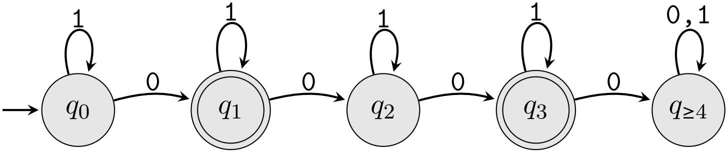
Input	States $r_0, r_1, \ldots, r_n$	Accepted/Rejected
ε	q_0	✖Rejected
0	q_0, q_1	
10101		
000		
00000		

Examples



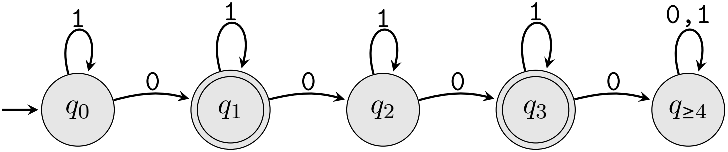
Input	States $r_0, r_1, \ldots, r_n$	Accepted/Rejected
ε	q_0	✖Rejected
0	q_0, q_1	✔Accepted
10101		
000		
00000		

Examples



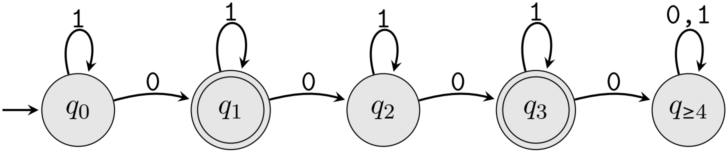
Input	States $r_0, r_1, \ldots, r_n$	Accepted/Rejected
ε	q_0	✖Rejected
0	q_0, q_1	✔Accepted
10101	$q_0, q_0, q_1, q_1, q_2, q_2$	
000		
00000		

Examples



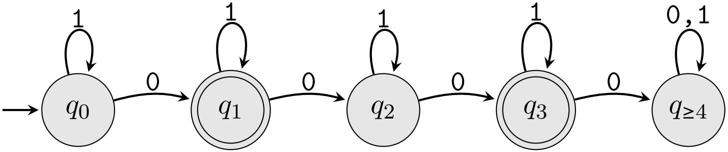
Input	States $r_0, r_1, \ldots, r_n$	Accepted/Rejected
ε	q_0	✖Rejected
0	q_0, q_1	✔Accepted
10101	$q_0, q_0, q_1, q_1, q_2, q_2$	✖Rejected
000		
00000		

Examples



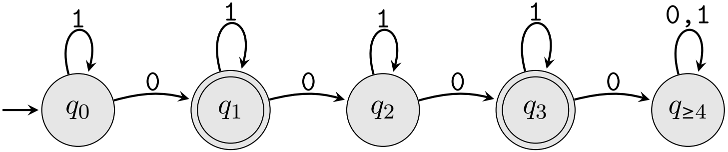
Input	States $r_0, r_1, \ldots, r_n$	Accepted/Rejected
ε	q_0	✖Rejected
0	q_0, q_1	✔Accepted
10101	$q_0, q_0, q_1, q_1, q_2, q_2$	✖Rejected
000	q_0, q_1, q_2, q_3	
00000		

Examples



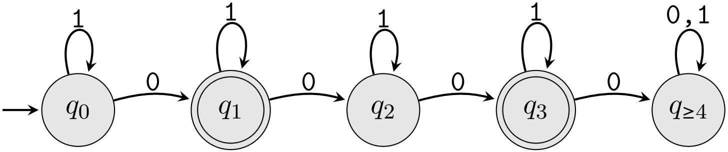
Input	States $r_0, r_1, \ldots, r_n$	Accepted/Rejected
ε	q_0	✗ Rejected
0	q_0, q_1	✓ Accepted
10101	$q_0, q_0, q_1, q_1, q_2, q_2$	✗ Rejected
000	q_0, q_1, q_2, q_3	✓ Accepted
00000		

Examples



Input	States $r_0, r_1, \ldots, r_n$	Accepted/Rejected
ε	q_0	✖Rejected
0	q_0, q_1	✔Accepted
10101	$q_0, q_0, q_1, q_1, q_2, q_2$	✖Rejected
000	q_0, q_1, q_2, q_3	✔Accepted
00000	$q_0, q_1, q_2, q_3, q_{\geq 4}, q_{\geq 4}$	

Examples



Input	States $r_0, r_1, \ldots, r_n$	Accepted/Rejected
ε	q_0	✖Rejected
0	q_0, q_1	✔Accepted
10101	$q_0, q_0, q_1, q_1, q_2, q_2$	✖Rejected
000	q_0, q_1, q_2, q_3	✔Accepted
00000	$q_0, q_1, q_2, q_3, q_{\geq 4}, q_{\geq 4}$	✖Rejected

Regular languages

A language is regular if some DFA recognizes it

Recall: A DFA M recognizes a language A if $A = \{w \mid M \text{ accepts } w\} = L(M)$

Prove some languages are regular

Let's construct some DFAs with JFLAP for the following languages over $\Sigma = \{\mathtt{a}, \mathtt{b}\}$

- $A = \{w \mid w \text{ starts and ends with } \mathtt{a}\}$
- $B = \{\mathtt{a}w\mathtt{a} \mid w \in \Sigma^*\}$
- $C = \{w \mid w \text{ starts and ends with different symbols}\}$
- $D = \Sigma^*$
- $E = \varnothing$
- $F = \{w \mid |w| \text{ is not a multiple of } 4\}$