Deterministic Finite Automata (DFAs)

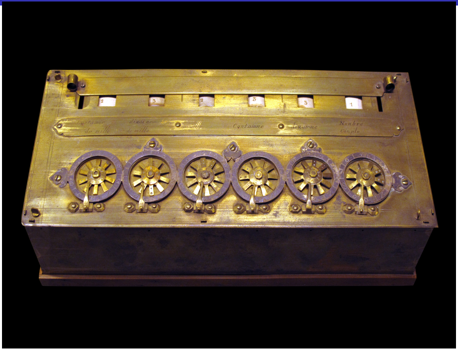
Lecture 3

Jan 28, 2020

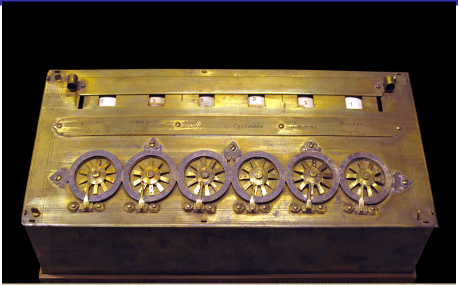
Part I

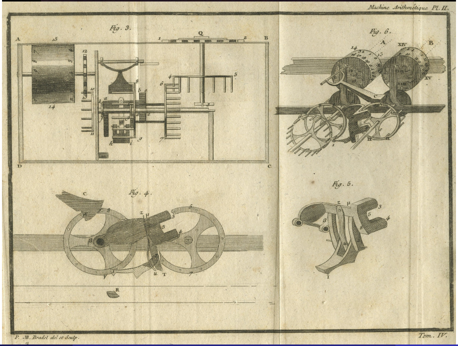
DFA Introduction

Pascaline

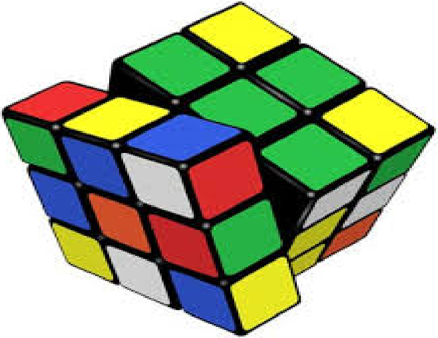


Pascaline





Rubik's Cube



DFAs also called Finite State Machines (FSMs)

Deterministic Finite Automata (DFA)

DFAs also called Finite State Machines (FSMs)

Deterministic Finite Automata (DFA)

Also called Finite State Machines (FSMs)

DFAs also called Finite State Machines (FSMs)

Deterministic Finite Automata (DFA)

Also called Finite State Machines (FSMs)

- State machines with fixed memory: very common in practice.
  - Vending machines
  - Elevators
  - Digital watches
  - Simple network protocols

A simple program

- Q: Finite set of states (encodes fixed memory).

A simple program

- Q: Finite set of states (encodes fixed memory).
- Σ: Finite alphabet set.

A simple program

- Q: Finite set of states (encodes fixed memory).
- Σ: Finite alphabet set. $\{0, 1\}$

A simple program

- Q: Finite set of states (encodes fixed memory).
- Σ: Finite alphabet set. $\{0, 1\}$
- q_0: Start state.

A simple program

- Q: Finite set of states (encodes fixed memory).
- Σ: Finite alphabet set. $\{0, 1\}$
- q_0: Start state. (alternate notation s)

A simple program

- Q: Finite set of states (encodes fixed memory).
- Σ: Finite alphabet set. $\{0, 1\}$
- q_0: Start state. (alternate notation s)
- $A \subseteq Q$: Set of accepting states.

A simple program

- Q: Finite set of states (encodes fixed memory).
- Σ: Finite alphabet set. $\{0, 1\}$
- q_0: Start state. (alternate notation s)
- $A \subseteq Q$: Set of accepting states.(alternate notation F)

A simple program

- Q: Finite set of states (encodes fixed memory).
- Σ: Finite alphabet set. $\{0, 1\}$
- q_0: Start state. (alternate notation s)
- $A \subseteq Q$: Set of accepting states.(alternate notation F)
- $\delta : Q \times \Sigma \to Q$ transition function

A simple program

- Q: Finite set of states (encodes fixed memory).
- Σ: Finite alphabet set. $\{0, 1\}$
- q_0: Start state. (alternate notation s)
- $A \subseteq Q$: Set of accepting states.(alternate notation F)
- $\delta : Q \times \Sigma \rightarrow Q$ transition function

Is the number represented by binary input string w is multiple of 5?

```
MULTIPLEOF5(w[1..n]):
  rem ← 0
  for i ← 1 to n
      rem ← (2 · rem + w[i]) mod 5
  if rem = 0
      return TRUE
  else
      return FALSE
```

State

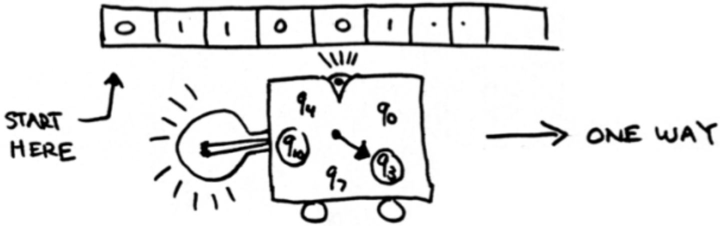
$Q : \{0, 1, 2, 3, 4\}$

$q_0 = 0$

$A = \{0\}$

δ

Machine View



- Machine has input written on a *read-only* tape
- Start in specified start state
- Read input starting from left: scan symbol, change state and move right
- Circled states are *accepting*
- Machine *accepts* input string if it is in an accepting state after scanning the last symbol.

Graphical Representation/State Machine

```
MultipleOf5(w[1..n]):
    rem ← 0
    for i ← 1 to n
        rem ← (2 · rem + w[i]) mod 5
    if rem = 0
        return True
    else
        return False
```

Graphical Representation/State Machine

```
MultipleOf5(w[1..n]):
    rem ← 0
    for i ← 1 to n
        rem ← (2 · rem + w[i]) mod 5
    if rem = 0
        return True
    else
        return False
```

$M = (Q, \Sigma, q_0, A, \delta)$

- Q: States $\{0, 1, 2, 3, 4\}$
- Σ: Alphabet $\{0, 1\}$
- q_0: Start state. '0'
- $A \subseteq Q$: Accepting states. $\{0\}$
- $\delta : Q \times \Sigma \to Q$

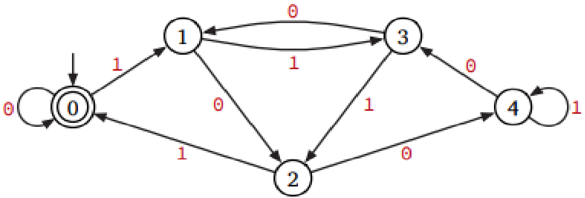
$\delta(q, a) = (2 \cdot q + a) \bmod 5$

Graphical Representation/State Machine

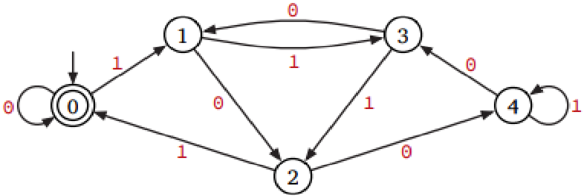
MultipleOf5($w[1..n]$):
    $rem \leftarrow 0$
    for $i \leftarrow 1$ to n
        $rem \leftarrow (2 \cdot rem + w[i]) \bmod 5$
    if $rem = 0$
        return True
    else
        return False

$M = (Q, \Sigma, q_0, A, \delta)$

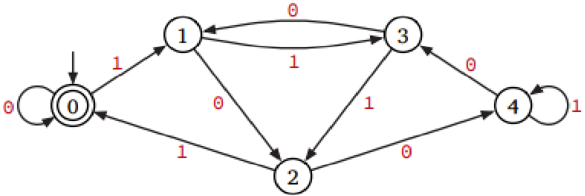
- Q: States
- Σ: Alphabet $\{0, 1\}$
- q_0: Start state.
- $A \subseteq Q$: Accepting states.
- $\delta : Q \times \Sigma \rightarrow Q$



Tabular Representation



Tabular Representation

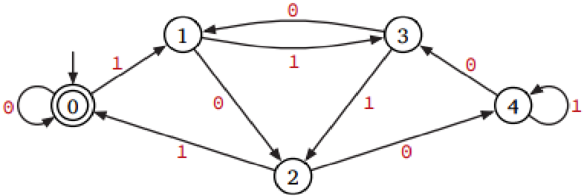


Tabular representation

q	$\delta[q,0]$	$\delta[q,1]$	$A[q]$
0	0	1	TRUE
1	2	3	FALSE
2	4	0	FALSE
3	1	2	FALSE
4	3	4	FALSE

$\delta(q, a) = (2 * q + a) \; mod \; 5$

Tabular Representation



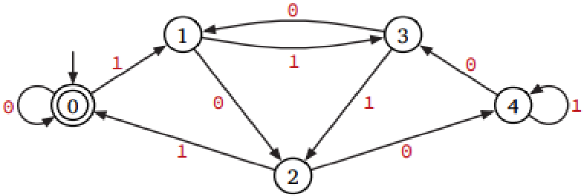
Tabular representation

q	$\delta[q,0]$	$\delta[q,1]$	$A[q]$
0	0	1	TRUE
1	2	3	FALSE
2	4	0	FALSE
3	1	2	FALSE
4	3	4	FALSE

$\delta(q, a) = (2 * q + a) \bmod 5$

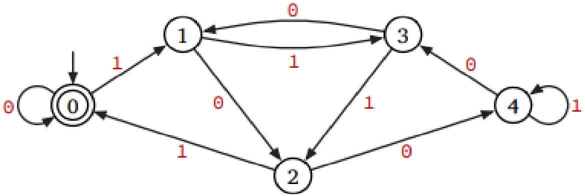
```
DoSomethingCool(q, w):
  if w = ε
      return A[q]
  else
      decompose w = a · x
      return DoSomethingCool(δ(q, a), x)
```

Graphical Representation



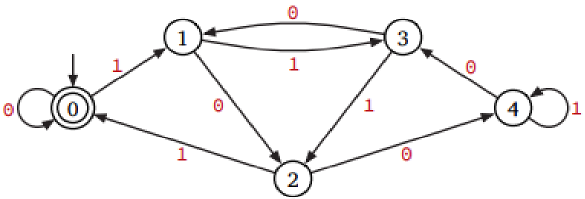
- **Convention:** Machine reads symbols from left to right
- Where does **001** lead? **10010001001 1**?

Graphical Representation



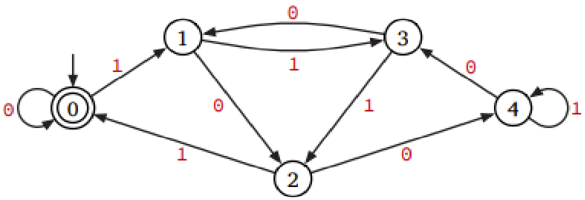
- **Convention:** Machine reads symbols from left to right
- Where does **001** lead? **10010001001 1**?
- Any string you would like to try?

Graphical Representation



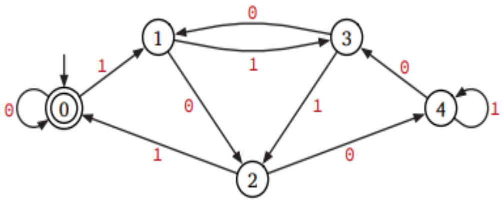
- **Convention:** Machine reads symbols from left to right
- Where does **001** lead? **100100010011**?
- Any string you would like to try?

Graphical Representation



- **Convention:** Machine reads symbols from left to right
- Where does **001** lead? **100100010011**?
- Any string you would like to try?
- Every string w has a unique walk that it follows from a given state q by reading one letter of w from left to right.

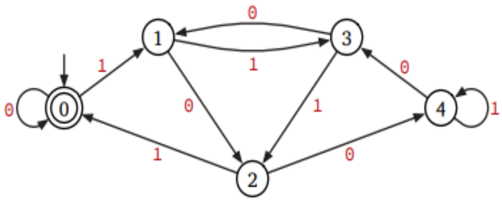
Graphical Representation



Definition

A DFA M accepts a string w iff the unique walk starting at the start state and spelling out w ends in an accepting state.

Graphical Representation



Definition

A DFA M accepts a string w iff the unique walk starting at the start state and spelling out w ends in an accepting state.

Definition

The language accepted (or recognized) by a DFA M is denote by $L(M)$ and defined as: $L(M) = \{w \mid M \text{ accepts } w\}$.

Warning

"M accepts language L" does not mean simply that that M accepts each string in L.

It means that M accepts each string in L and no others. Equivalently M accepts each string in L and does not accept/rejects strings in $\Sigma^* \setminus L$.

Warning

"M accepts language L" does not mean simply that that M accepts each string in L.

It means that M accepts each string in L and no others. Equivalently M accepts each string in L and does not accept/rejects strings in $\Sigma^* \setminus L$.

M "recognizes" L is a better term but "accepts" is widely accepted (and recognized) (joke attributed to Lenny Pitt)

Extending the transition function to strings

Given DFA $M = (Q, \Sigma, \delta, s, A)$, $\delta(q, a)$ is the state that M goes to from q on reading letter a

Useful to have notation to specify the unique state that M will reach from q on reading *string* w

Extending the transition function to strings

Given DFA $M = (Q, \Sigma, \delta, s, A)$, $\delta(q, a)$ is the state that M goes to from q on reading letter a

Useful to have notation to specify the unique state that M will reach from q on reading *string* w

Transition function $\delta^* : Q \times \Sigma^* \to Q$ defined inductively as follows:

- $\delta^*(q, w) = q$ if $w = \epsilon$
- $\delta^*(q, w) = \delta^*(\delta(q, a), x)$ if $w = ax$.

Formal definition of language accepted by M

Definition

The language $L(M)$ accepted by a DFA $M = (Q, \Sigma, \delta, s, A)$ is

$$\{w \in \Sigma^* \mid \delta^*(s, w) \in A\}.$$

Formal definition of language accepted by M

Definition

The language $L(M)$ accepted by a DFA $M = (Q, \Sigma, \delta, s, A)$ is

$$\{w \in \Sigma^* \mid \delta^*(s, w) \in A\}.$$

Kleene (1956): L is regular if and only if it is $L(M)$ for some DFA M.

Formal definition of language accepted by M

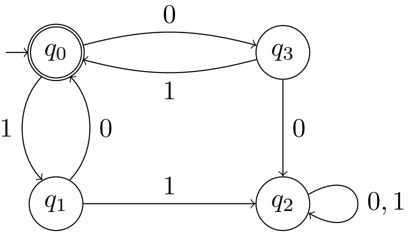
Definition

The language $L(M)$ accepted by a DFA $M = (Q, \Sigma, \delta, s, A)$ is

$$\{w \in \Sigma^* \mid \delta^*(s, w) \in A\}.$$

Kleene (1956): L is regular if and only if it is $L(M)$ for some DFA M.!!!

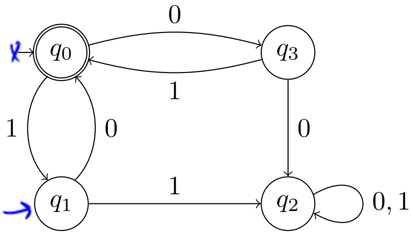
Example



What is:

- $\delta^*(q_1, \epsilon)$
- $\delta^*(q_0, 1011)$
- $\delta^*(q_1, 010)$
- $\delta^*(q_4, 10)$

Example Contd.



- What is $L(M)$?
- What is $L(M)$ if start state is changed to q_1?
- What is $L(M)$ if final/accepte states are set to $\{q_2, q_3\}$ instead of $\{q_0\}$?

Advantages of formal specification

- Necessary for proofs
- Necessary to specify abstractly for class of languages

Exercise: Prove by induction that for any two strings u, v, and any state q,

$$\delta^*(q, uv) = \delta^*(\delta^*(q, u), v)$$

.

Part II

Constructing DFAs

DFA Construction: Example

Assume $\Sigma = \{0, 1\}$
$L = \{$ Strings with 11 as a sub-string $\} = (0+1)^*11(0+1)^*$

DFA Construction: Example

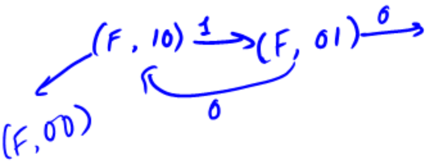
Assume $\Sigma = \{0, 1\}$

$L = \{$ Strings with 11 as a sub-string $\} = (0 + 1)^* 11 (0 + 1)^*$

```
CONTAINS11(w[1..n]):
  found ← FALSE
  for i ← 1 to n
      if i = 1
          last2 ← w[1]
      else
          last2 ← w[i − 1] · w[i]
      if last2 = 11
          found ← TRUE
  return found
```

Q: (found, last2)
2 7

|Q| = 14



DFA Construction: Example

Assume $\mathbf{\Sigma = \{0, 1\}}$

$L = \{$ Strings with 11 as a sub-string $\} = \mathbf{(0+1)^*11(0+1)^*}$

```
CONTAINS11(w[1..n]):
  found ← FALSE
  for i ← 1 to n
      if i = 1
          last2 ← w[1]
      else
          last2 ← w[i − 1] · w[i]
      if last2 = 11
          found ← TRUE
  return found
```

q	$\delta[q,0]$	$\delta[q,1]$
(FALSE, ε)	(FALSE, 0)	(FALSE, 1)
(FALSE, 0)	(FALSE, 00)	(FALSE, 01)
(FALSE, 1)	(FALSE, 10)	(**TRUE**, 11)
(FALSE, 00)	(FALSE, 00)	(FALSE, 01)
(FALSE, 01)	(FALSE, 10)	(**TRUE**, 11)
(FALSE, 10)	(FALSE, 00)	(FALSE, 01)
(FALSE, 11)	(FALSE, 10)	(**TRUE**, 11)

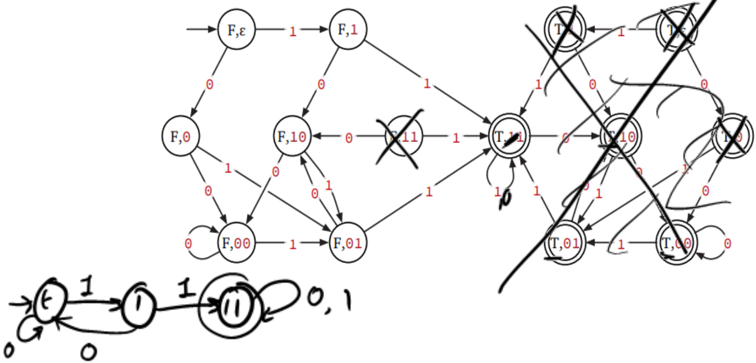
q	$\delta[q,0]$	$\delta[q,1]$
(TRUE, ε)	(TRUE, 0)	(TRUE, 1)
(TRUE, 0)	(TRUE, 00)	(TRUE, 01)
(TRUE, 1)	(TRUE, 10)	(TRUE, 11)
(TRUE, 00)	(TRUE, 00)	(TRUE, 01)
(TRUE, 01)	(TRUE, 10)	(TRUE, 11)
(TRUE, 10)	(TRUE, 00)	(TRUE, 01)
(TRUE, 11)	(TRUE, 10)	(TRUE, 11)

DFA Construction: Example Contd.

L = { Strings with 11 as a sub-string } $= (0+1)^*11(0+1)^*$

q	$\delta[q,0]$	$\delta[q,1]$
(FALSE, ε)	(FALSE, 0)	(FALSE, 1)
(FALSE, 0)	(FALSE, 00)	(FALSE, 01)
(FALSE, 1)	(FALSE, 10)	**(TRUE, 11)**
(FALSE, 00)	(FALSE, 00)	(FALSE, 01)
(FALSE, 01)	(FALSE, 10)	**(TRUE, 11)**
(FALSE, 10)	(FALSE, 00)	(FALSE, 01)
(FALSE, 11)	(FALSE, 10)	**(TRUE, 11)**

q	$\delta[q,0]$	$\delta[q,1]$
(TRUE, ε)	(TRUE, 0)	(TRUE, 1)
(TRUE, 0)	(TRUE, 00)	(TRUE, 01)
(TRUE, 1)	(TRUE, 10)	(TRUE, 11)
(TRUE, 00)	(TRUE, 00)	(TRUE, 01)
(TRUE, 01)	(TRUE, 10)	(TRUE, 11)
(TRUE, 10)	(TRUE, 00)	(TRUE, 01)
(TRUE, 11)	(TRUE, 10)	(TRUE, 11)

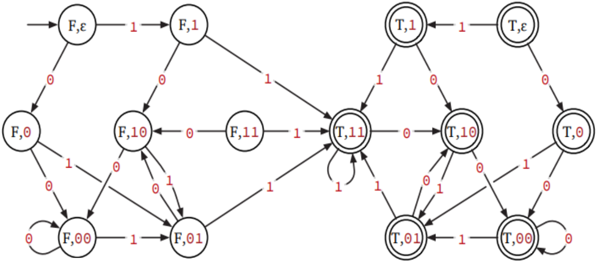


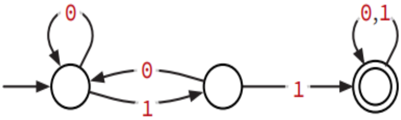
DFA Construction: Example Contd.

$L = \{$ Strings with 11 as a sub-string $\} = (0+1)^*11(0+1)^*$

q	$\delta[q,0]$	$\delta[q,1]$
(FALSE, ε)	(FALSE, 0)	(FALSE, 1)
(FALSE, 0)	(FALSE, 00)	(FALSE, 01)
(FALSE, 1)	(FALSE, 10)	**(TRUE, 11)**
(FALSE, 00)	(FALSE, 00)	(FALSE, 01)
(FALSE, 01)	(FALSE, 10)	**(TRUE, 11)**
(FALSE, 10)	(FALSE, 00)	(FALSE, 01)
(FALSE, 11)	(FALSE, 10)	**(TRUE, 11)**

q	$\delta[q,0]$	$\delta[q,1]$
(TRUE, ε)	(TRUE, 0)	(TRUE, 1)
(TRUE, 0)	(TRUE, 00)	(TRUE, 01)
(TRUE, 1)	(TRUE, 10)	(TRUE, 11)
(TRUE, 00)	(TRUE, 00)	(TRUE, 01)
(TRUE, 01)	(TRUE, 10)	(TRUE, 11)
(TRUE, 10)	(TRUE, 00)	(TRUE, 01)
(TRUE, 11)	(TRUE, 10)	(TRUE, 11)





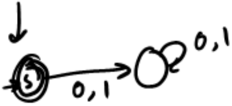
DFAs: State = Memory

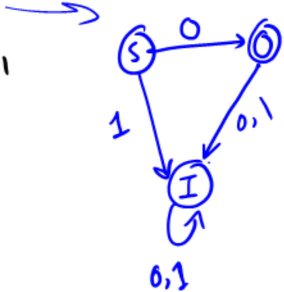
How do we design a DFA M for a given language L? That is $L(M) = L$.

- DFA is a like a program that has fixed amount of memory independent of input size.
- The memory of a DFA is encoded in its states
- The state/memory must capture enough information from the input seen so far that it is sufficient for the suffix that is yet to be seen (note that DFA cannot go back)

DFA Construction: More examples

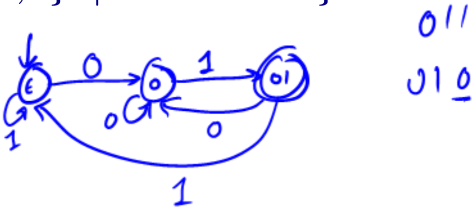
- $L = \emptyset$, $L = \Sigma^*$, $L = \{\epsilon\}$, $L = \{0\}$.





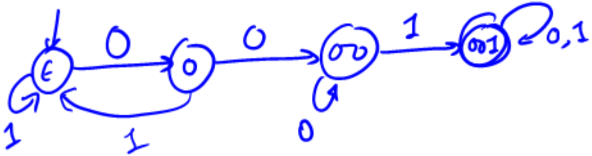
DFA Construction: More examples

- $L = \emptyset$, $L = \Sigma^*$, $L = \{\epsilon\}$, $L = \{0\}$.
- $L = \{w \in \{0,1\}^* \mid w \text{ ends with } 01\}$



DFA Construction: More examples

- $L = \emptyset$, $L = \Sigma^*$, $L = \{\epsilon\}$, $L = \{0\}$.
- $L = \{w \in \{0, 1\}^* \mid w \text{ ends with } 01\}$
- $L = \{w \in \{0, 1\}^* \mid w \text{ contains } 001 \text{ as substring}\}$



DFA Construction: More examples

- $L = \emptyset$, $L = \Sigma^*$, $L = \{\epsilon\}$, $L = \{0\}$.
- $L = \{w \in \{0,1\}^* \mid w$ ends with 01$\}$
- $L = \{w \in \{0,1\}^* \mid w$ contains 001 as substring$\}$
- $L = \{w \in \{0,1\}^* \mid w$ contains 001 or 010 as substring$\}$

DFA Construction: More examples

- $L = \emptyset$, $L = \Sigma^*$, $L = \{\epsilon\}$, $L = \{0\}$.
- $L = \{w \in \{0,1\}^* \mid w \text{ ends with 01}\}$
- $L = \{w \in \{0,1\}^* \mid w \text{ contains 001 as substring}\}$
- $L = \{w \in \{0,1\}^* \mid w \text{ contains 001 or 010 as substring}\}$
- $L = \{w \mid w \text{ has a 1 } k \text{ positions from the end}\}$