Finite Automata

- ➔ *Useful model for computers having an extremely limited amount of memory.*
- ➔ *Small electromechanical devices*
- ➔ *Example: switch, an automatic door*

Finite Automata

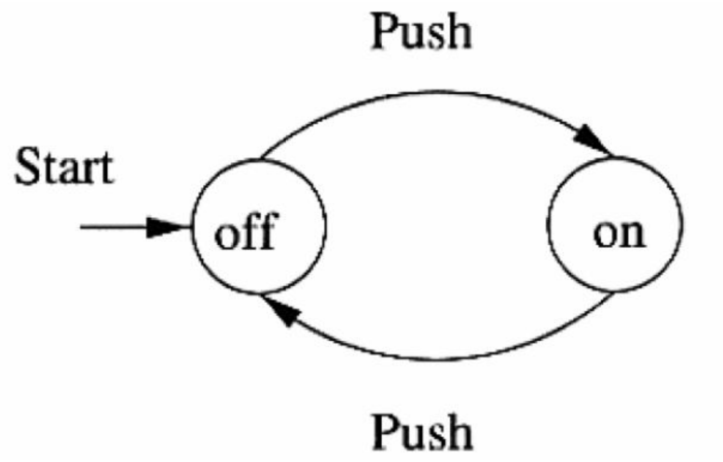


***FIGURE 1.0** Modeling of a switch*

Finite Automata (Door Automation)

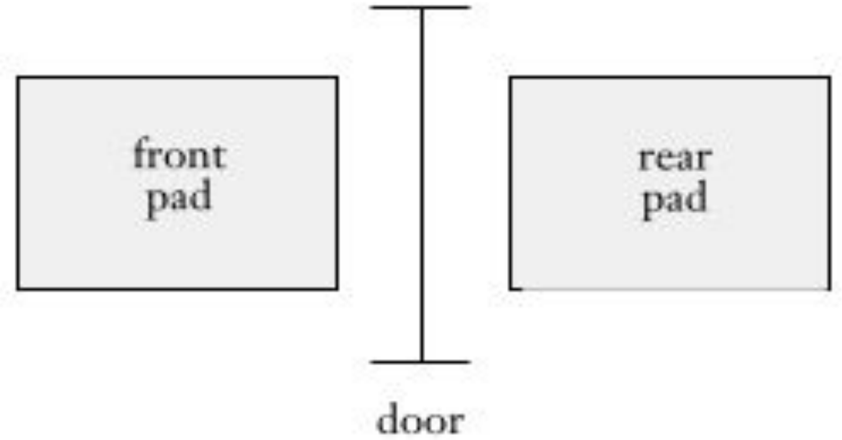


FIGURE 1.1 *Top view of an automatic door*

Finite Automata (Door Automation)

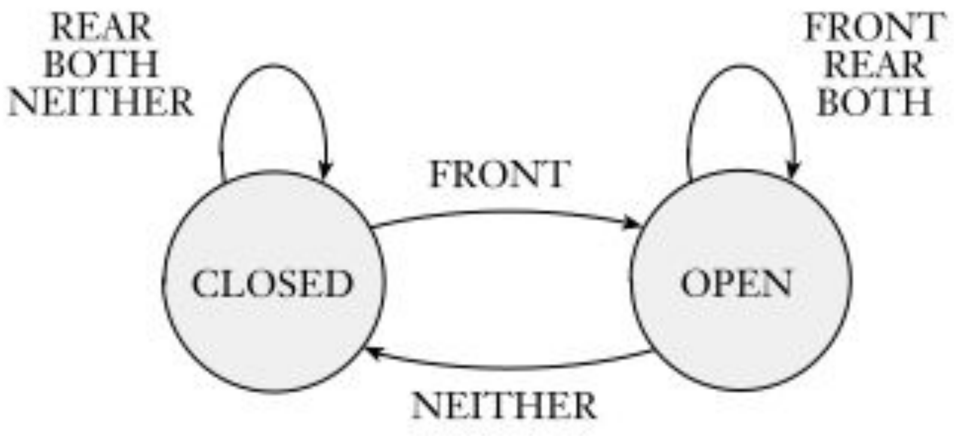


FIGURE 1.2 *State diagram for an automatic door controller*

Finite Automata (Door Automation)

		input signal			
		NEITHER	FRONT	REAR	BOTH
state	CLOSED	CLOSED	OPEN	CLOSED	CLOSED
	OPEN	CLOSED	OPEN	OPEN	OPEN

FIGURE 1.3 *State transition table for an automatic door controller*

Finite Automata

➔ ***Notice the following terms***

- ✓ *state diagram*
- ✓ *states*
- ✓ *start state*
- ✓ *accept state*
- ✓ *Transitions*

Designing Finite Automata

Need to know

➔ *How many states ?*

➔ *What are the inputs?*

➔ *What will be transition table?*

Designing Finite Automata

➔ ***State*** **-** *to remember*

- ✓ ***Start State***
  - *at the initial stage*

- ✓ ***Accepting State***
  - *if the automaton is in this state when finished, the string is accepted otherwise rejected*

Designing Finite Automata

- ➔ *Lets design an automaton*
  - ✓ *Consists of {0,1}*
  - ✓ *Has even length*

- ➔ *'100111'* *-does this string belong to the language?*
- ➔ *'10000'* *-does this string belong to the language?*

Designing Finite Automata Example

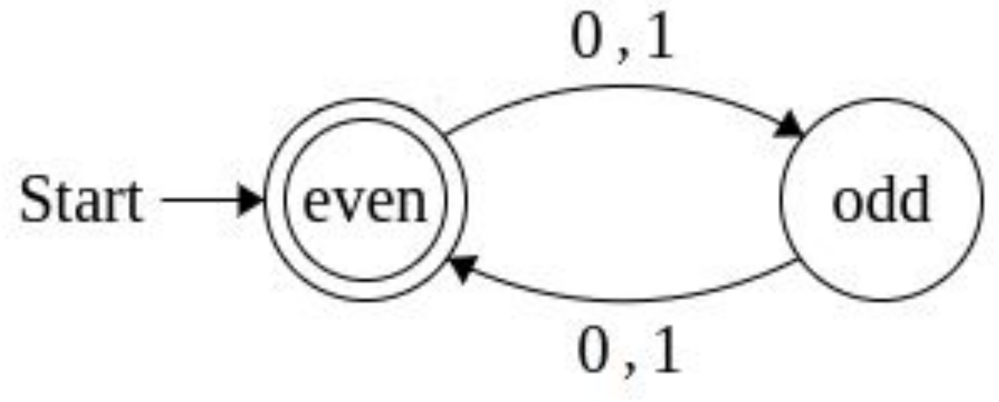


FIGURE: *2-state finite automaton* M_0

Designing Finite Automata

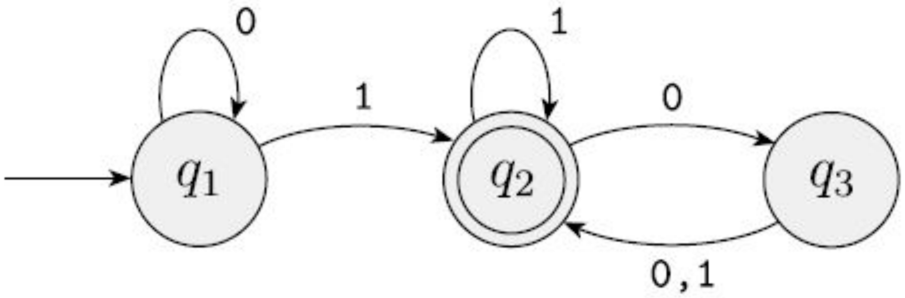


FIGURE 1.4 *A finite automaton called **M**1 that has three states*

Finite Automata

- ➔ ***Deterministic***
  - for each input there must be one and only one state where the automaton can transition from its current state

- ➔ ***Non-deterministic***
  - can be in several states at once

- ➔ *Deterministic Finite Automata (DFA)*
- ➔ *Non-deterministic Finite Automata (NFA)*

Formal Definition of Finite Automata

DEFINITION 1.5

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

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.[2]

Formal Definition of Finite Automata

Let's define automaton M_0 and M_1 formally using 5-tuple.

A = {w| w has even length}
Language of machine M_0 is A, written as $L(M_0) = A$, or equivalently, M_1 recognizes A

$L(M_1) = L_1 = A$ = {w| w contains at least one 1 and an even number of 0s follow the last 1}.

DFA 2-state *(Sipser, M2)*

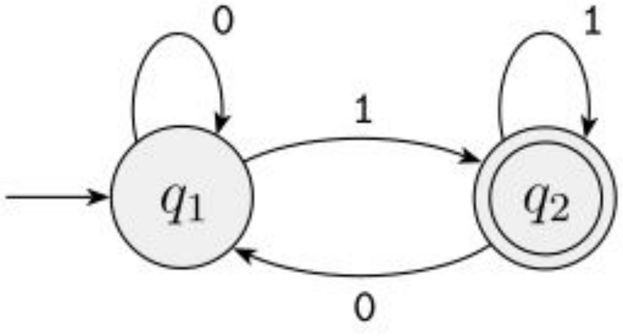


FIGURE 1.8 *State diagram of the **two-state** finite automaton **M2***

DFA 2-state *(Sipser M2)*

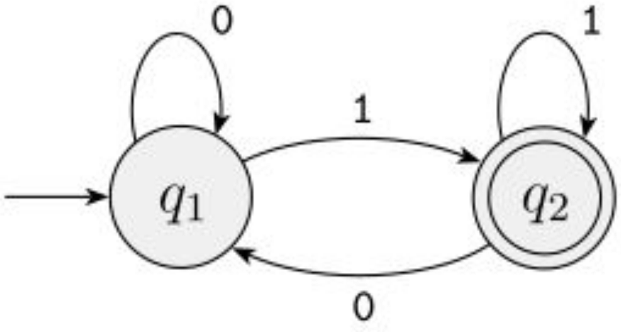


FIGURE 1.8 *State diagram of the* ***two-state*** *finite automaton* ***M_2***
L(M_2) = *{w| w ends in a 1}*

DFA 2-state *(Sipser, M3)*

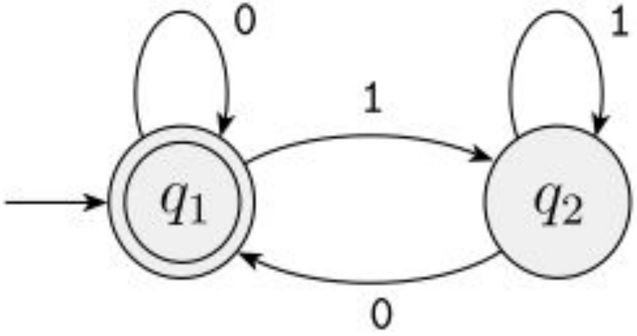


FIGURE 1.10 *State diagram of the two-state finite automaton **M3***

DFA 2-state *(Sipser, M3)*

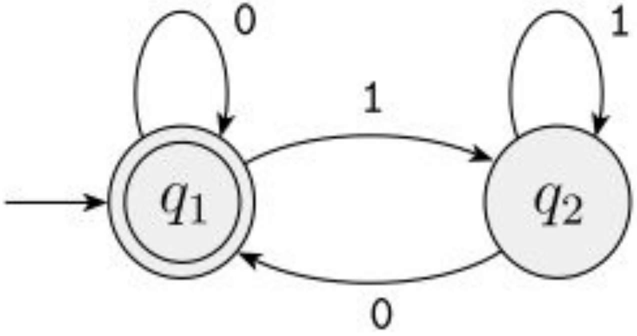


FIGURE 1.10 *State diagram of the two-state finite automaton M_3*

$L(M_3) = \{ w \mid w$ consists of 0,1 and w ends in 0, that includes the empty string, $\varepsilon\}$

DFA 2-state *(Sipser M2)*

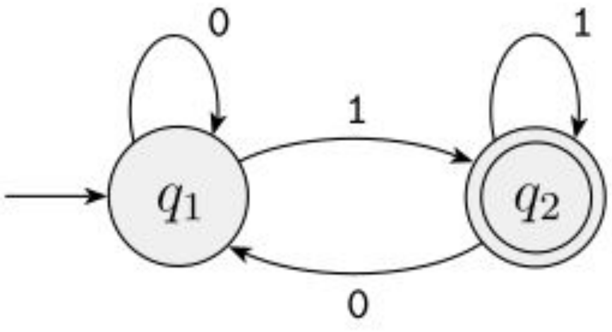


FIGURE 1.8 *State diagram of the* ***two-state*** *finite automaton* ***M2***

L(M2) = *{w| w ends in a 1}*

w *is a set of string.*

DFA 5-state *(Sipser, M4)*

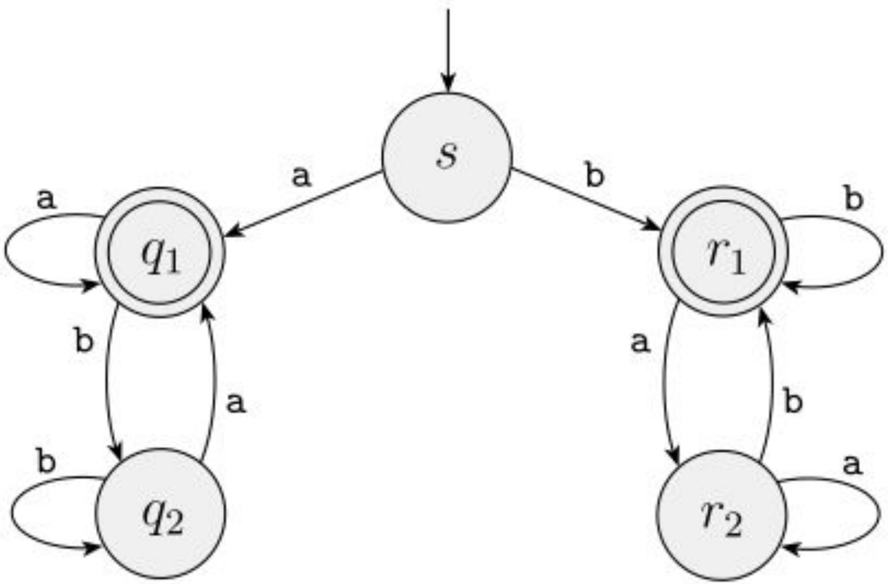


FIGURE 1.12 *Finite automaton* ***M4***

DFA 3-state *(Sipser, M5)*

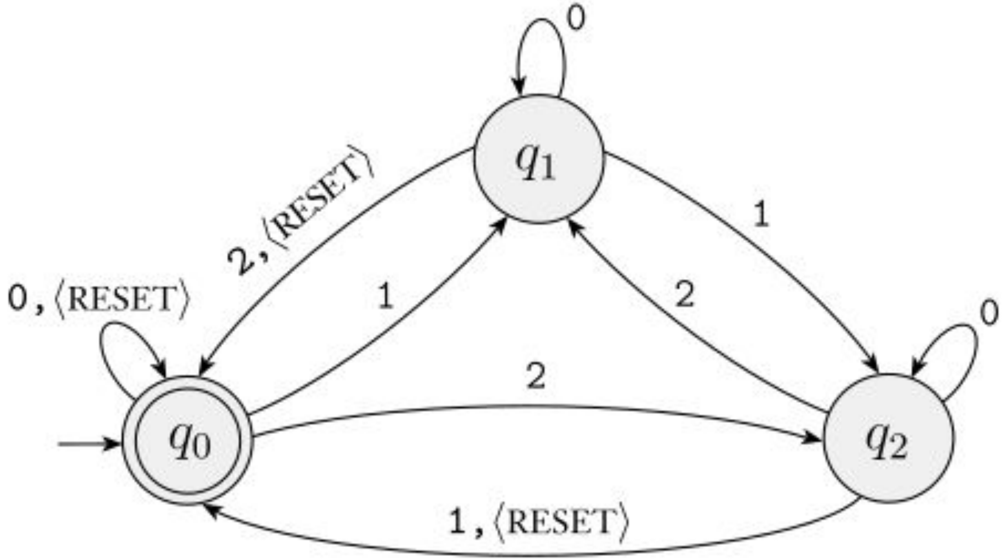


FIGURE 1.14 *Finite automaton* ***M5***

DFA 3-state *(Sipser, M5)*

➔ *Let's explore a generalization of automaton* ***M5***

DFA 3-state *(Sipser, M5) -generalization*

- *For each i ≥ 1 and alphabet Σ = {0,1,2,<RESET>}*
- *Ai ={ w | w is the sum of the numbers is a multiple of i}*
- *Bi = (Qi , Σ, δi , q0 , {q0 })*

$$\delta_i(q_j, 0) = q_j,$$
$$\delta_i(q_j, 1) = q_k, \text{ where } k = j + 1 \text{ modulo } i,$$
$$\delta_i(q_j, 2) = q_k, \text{ where } k = j + 2 \text{ modulo } i, \text{ and}$$
$$\delta_i(q_j, \langle\text{RESET}\rangle) = q_0.$$

DFA 3-state *(Sipser, M5) -generalization*

➔ *What can be possible solutions for previous problem if the alphabets are as followings:*

- ✓ $\Sigma_1 = \{1,2,4,\text{<RESET>}\}$
- ✓ $\Sigma_2 = \{2,5,8,\text{<RESET>}\}$

Formal Definition of Computation

Let,

$M = (Q, \Sigma, \delta, q\ 0, F)$ be a finite automaton and

$w = w_1 w_2 \cdots w_n$ be a string where each $w\ i$ is a member of the alphabet Σ.

Then M accepts w if a sequence of states $r\ 0, r\ 1, \ldots, r\ n$ in Q exists with three

➔ ***Conditions:***

- ✓ $r_0 = q_0$,
- ✓ $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n - 1$, and
- ✓ $r_n \in F$.

Regular Language

DEFINITION 1.16

A language is called a ***regular language*** if some finite automaton recognizes it.

Regular Language

- *Consider the following string* ***w****,*

  10⟨ RESET ⟩22⟨ RESET ⟩012.

- *$L(M_5) = \{w \mid$ the sum of the symbols in w is 0 modulo 3, except that ⟨ RESET ⟩ resets the count to 0$\}$.*

Designing DFA

- ➔ *We have to figure out what you need to remember about the string as you are reading it.*
- ➔ *Suppose that the alphabet is {x, y} and that the language consists of all strings with an odd number of y's.*
- ➔ *We want to construct a finite automaton E_1 to recognize this language.*

Designing DFA

- ➔ *What we need to remember to design this automaton?*
  - ✓ *Remember whether the number of y's seen so far is even or odd for every scanned symbol*
- ➔ *Who will remember?*
  - ✓ *States*
- ➔ *Our states need to remember for E are:*
  - ✓ *even so far, and*
  - ✓ *odd so far.*

Designing DFA

FIGURE 1.18 *The two states qeven and qodd*

Designing DFA

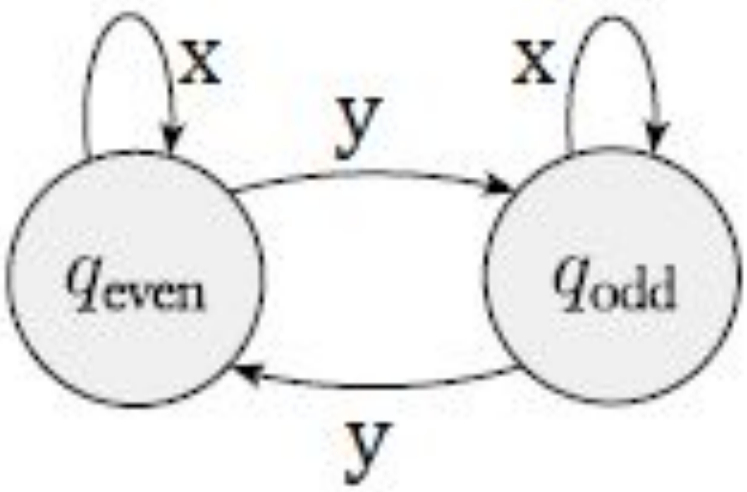


FIGURE 1.19 *Transitions telling how the possibilities rearrange*

Designing DFA

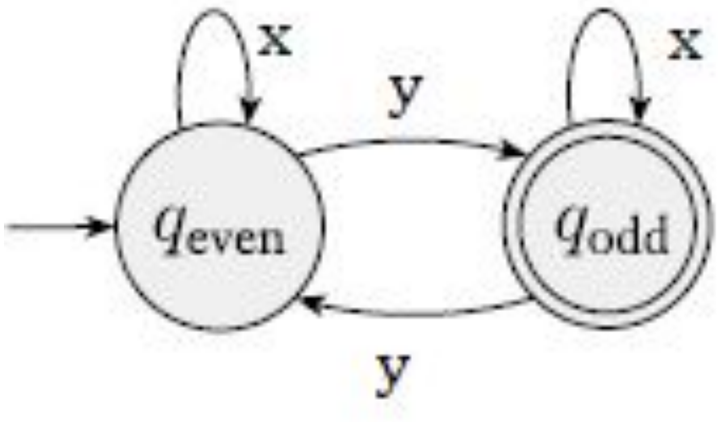


FIGURE 1.20 *Adding the start and accept states*

Designing DFA

➔ *Let's design a finite automaton* ***E2*** *to recognize the regular language of all strings that contain the string* ***001*** *as a* ***substring***

✓	*0010*	*accepted*
✓	*1001*	*accepted*
✓	*001*	*accepted*
✓	*11111110011111*	*accepted*
✓	*0101011010010110101*	*accepted*
✓	*11 0000*	*not accepted*
✓	ε	*not accepted*
✓	*101011101*	*not accepted*

Designing DFA

- ➔ *There are four possibilities:*

  - ✓ *haven't just seen any symbols of the pattern,*
  - ✓ *have just seen a 0,*
  - ✓ *have just seen 00, or*
  - ✓ *have seen the entire pattern 001.*

Designing DFA

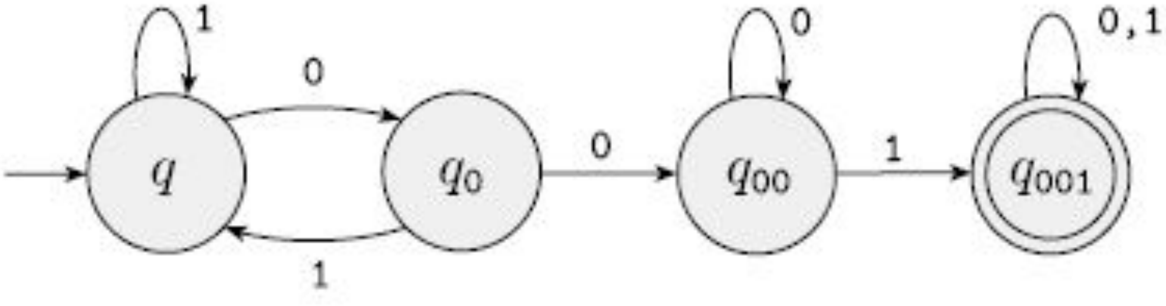


FIGURE 1.22 *Accepts strings containing 001*

Designing DFA (Hopcroft, Motwani, and Ullman, Example-2.1)

- *Let us formally specify a DFA that accepts all and only the strings of 0's and 1's that have the sequence 01 somewhere in the string.*

- *We can write this language L as:*
  L= {w | w is of the form x01y for some strings x and y consisting of 0's and 1's only.}

Designing DFA (Hopcroft, Motwani, and Ullman, Example-2.1)

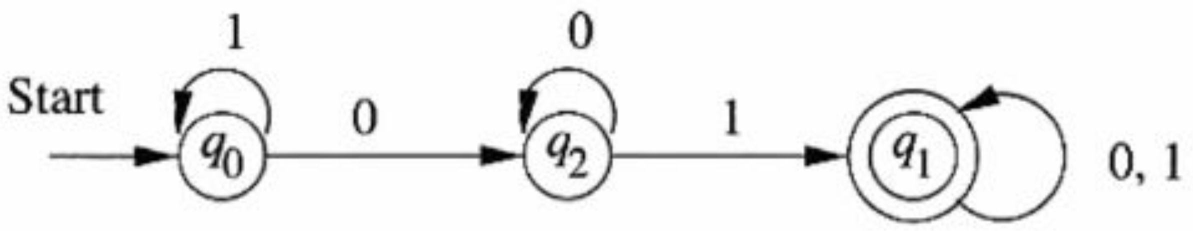


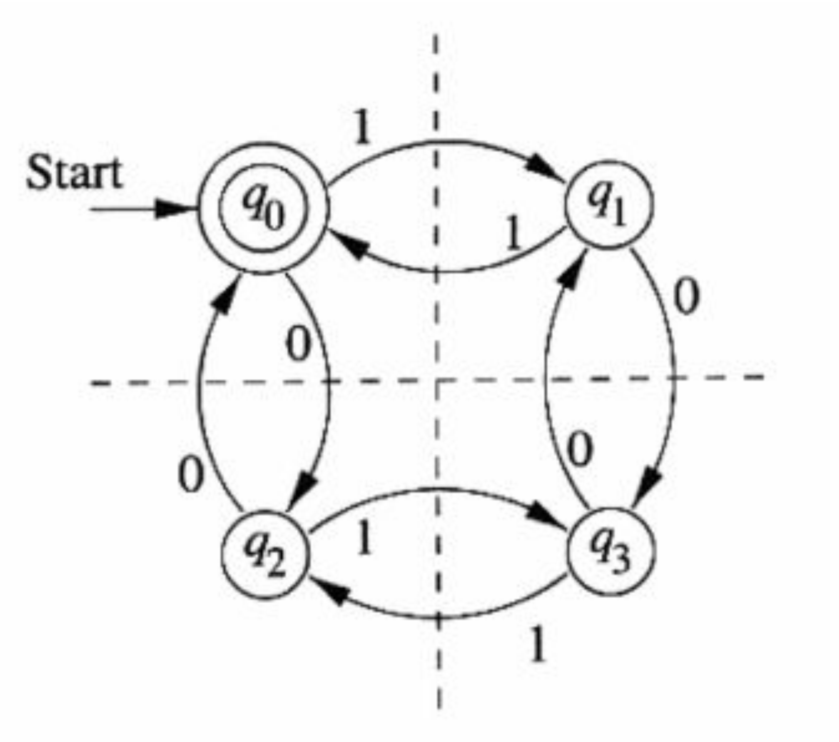
Figure 2.4: The transition diagram for the DFA accepting all strings with a substring 01

Designing DFA

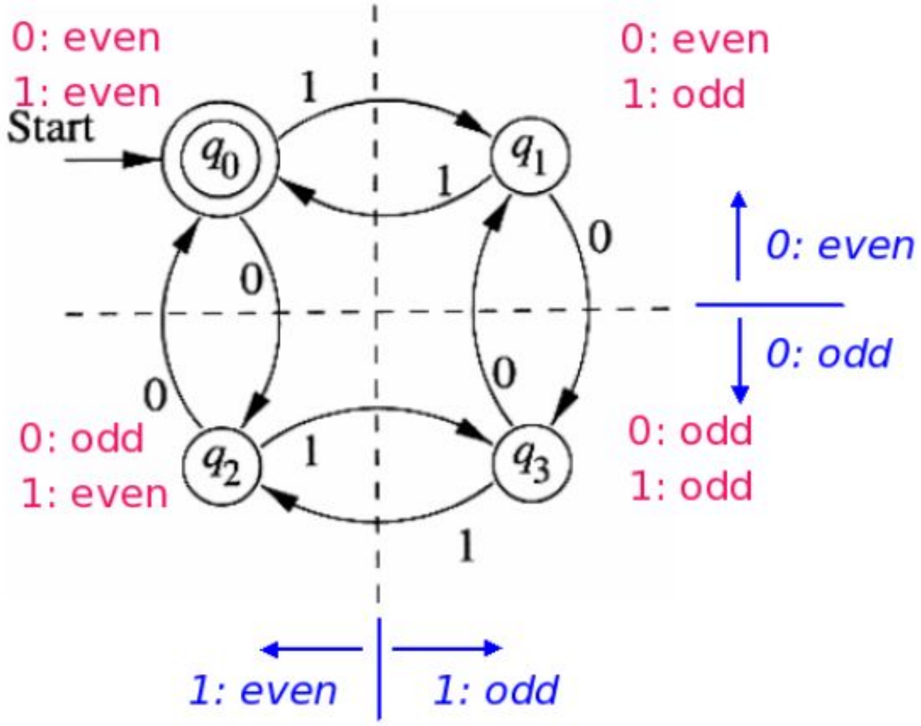
➔ *Design a DFA to accept the language,*

L = {w | w has both an even number of 0's and an even number of 1's}

Designing DFA



Designing DFA



Designing DFA *(Lewis and Papadimitriou, Example 2.1.2)*

➔ *Design a DFA, M that accepts the language,*

$L(M) = \{w \in \{a,b\}^* : w \text{ does not contain three consecutive } b\text{'s}\}$

Designing DFA (*Lewis and Papadimitriou, Example 2.1.2*)

➔ *Design a DFA, M that accepts the language,*

$L(M) = \{w \in \{a,b\}^* : w$ does not contain three consecutive b's$\}$

➔ ***Observation:***
$w \in \{a,b\}^*$ means any symbol can be used any times including 0 times

➔ ***Remember:***
Number of b's appeared one after another in any string.

Designing DFA *(Lewis and Papadimitriou, Example 2.1.2)*

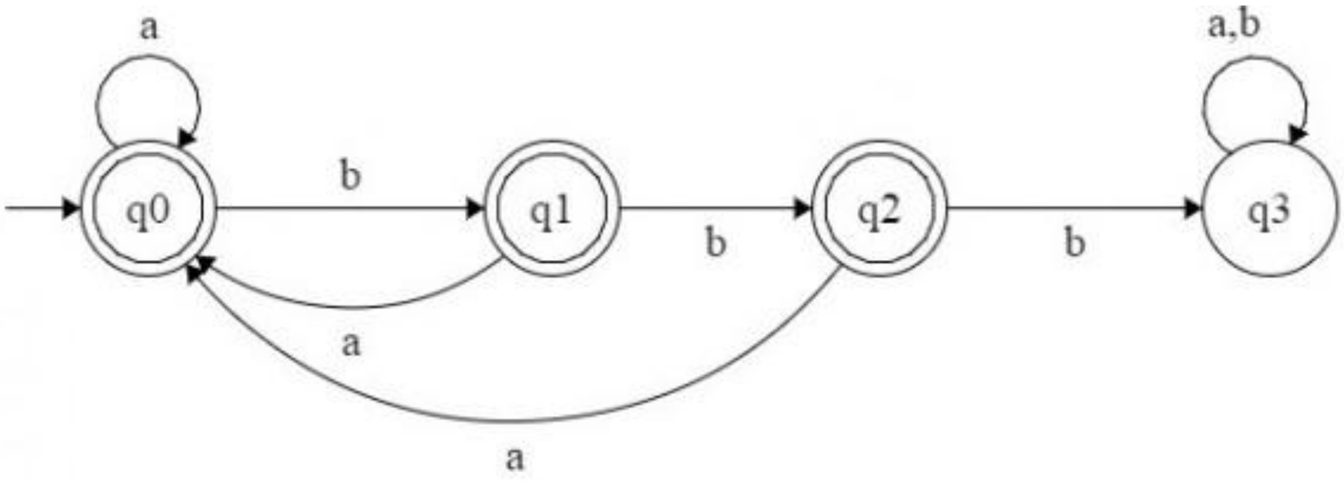


Figure: *State diagram of L(M) where w does not contain three consecutive b's.*

Designing DFA

➔ *Design a DFA that accepts **binary** numbers that are **divisible by three**.*

➔ ***Observations:***

✓ *binary numbers*

✓ *divisible by three*

Designing DFA

➔ *Design a DFA that accepts **binary** numbers that are **divisible by three**.*

➔ ***Observations:***
- ✓ *binary numbers*
- ✓ *divisible by three*

➔ *Thumb rule of binary number:*
- ✓ $X0 = 2 * X$
- ✓ $X1 = 2 * X + 1$

Designing DFA (Hopcroft, Motwani, and Ullman, Example-2.4)

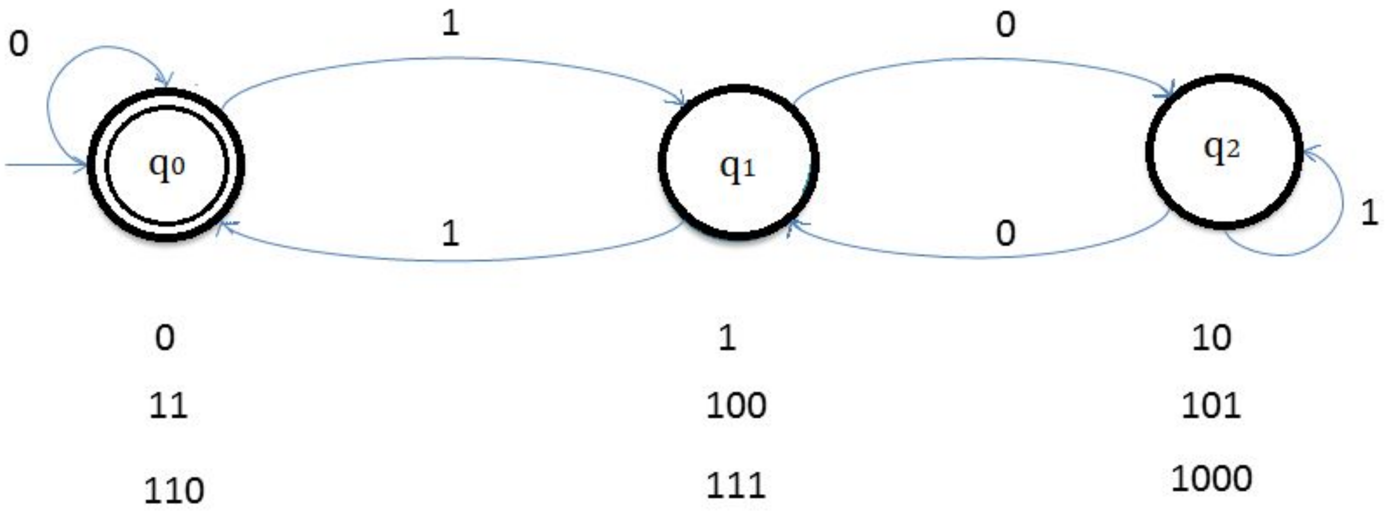


Figure: *DFA accepts binary number divisible by 3*

The Regular Operations

- ➔ *Lets begin to investigate properties of regular languages which is recognized by some finite automaton.*
- ➔ *In arithmetic: objects=numbers and the tools like + and ×*
- ➔ *In the* ***theory of computation****:*
  - ✓ *objects = languages*
  - ✓ *Tools = operations specifically designed for manipulating them.*

- ➔ *Three operations on languages, called the* ***regular operations.***

The Regular Operations *(sipser)*

DEFINITION 1.23

Let A and B be languages. We define the regular operations ***union***, ***concatenation***, and ***star*** as follows:

- **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**: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

The Regular Operations *(sipser)*

- *Alphabet, Σ = {a, b, . . . , z}*
- *Language, A = {good, bad}*
- *Language, B = {boy, girl}*

Union:

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

Concatenation:

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

Star:

$A*$ = *{ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, goodbadgood, goodbadbad, . . . }*

The Regular Operations *(sipser)*

➔ *$N = \{1, 2, 3, \ldots\}$ be the set of natural numbers.*

We say that N is closed under multiplication.
We mean that for any x and y in N , the product $x \times y$ also is in N .
In contrast, N is not closed under division.
1 and 2 are in N but 1/2 is not.

The Regular Operations *(sipser)*

- *A collection of objects is closed under some operation if applying that operation to members of the collection returns an object still in the collection.*

- *We will show that the collection of regular languages is closed under all three of the regular operations.*

The Regular Operations *(sipser)*

THEOREM 1.25

The class of regular languages is closed under the union operation.

The Regular Operations *(sipser)*

Proof Idea:

- ➔ *Languages are A_1 and A_2*
- ➔ *Corresponding Machine M_1 and M_2*
- ➔ *Language of union is $A_1 \cup A_2$*
- ➔ *To prove that $A_1 \cup A_2$ is regular, we demonstrate a finite automaton, call it M, that recognizes $A_1 \cup A_2$*
- ➔ *Once the symbols of the input have been read and used to simulate M1, we can't "rewind the input tape"*
- ➔ *Simulate both M_1 and M_2 simultaneously, as the input symbols arrive one by one.*
- ➔ *String is accepted if either state of pair of state is in accepting state of the individual machines.*

The Regular Operations *(sipser)*

PROOF

➔ *Let* M_1 *recognize* A_1*, where* $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$
M_2 *recognize* A_2*, where* $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

➔ *Construct* M *to recognize* $A_1 \cup A_2$*, where* $M = (Q, \Sigma, \delta, q_0, F)$.

The Regular Operations *(sipser)*

Design of DFA:

1. *$Q = \{(r1, r2) | r1 \in Q1$ and $r2 \in Q2\}$.*
2. *Σ, the alphabet, is the same as in M1 and M2.*
3. *δ, the transition function, is defined as follows.*
   For each $(r1, r2) \in Q$ and each $a \in \Sigma$,
   let $(\ \delta(r1, r2), a\ ) = (\delta_1(r_1, a), \delta2(r2, a))$
4. *q0 is the pair (q1, q2).*
5. *$F = \{(r1, r2) | r1 \in F1$ or $r2 \in F2\}$.*

Thus, it is proved that $A_1 \cup A_2$ is being recognized by a DFA M. That means $A_1 \cup A_2$ is regular. - "Regular language is closed under union operation"

The Regular Operations *(sipser)*

Design of DFA:

1. *$Q = \{(r1, r2) | r1 \in Q1$ and $r2 \in Q2\}$.*
2. *Σ, the alphabet, is the same as in M1 and M2.*
3. *δ, the transition function, is defined as follows.*
   For each $(r1, r2) \in Q$ and each $a \in \Sigma$,
   let $(\ \delta(r1, r2), a\ ) = (\delta_1(r_1, a), \delta 2(r2, a))$
4. *q0 is the pair (q1, q2).*
5. *$F = \{(r1, r2) | r1 \in F1$ or $r2 \in F2\}$.*

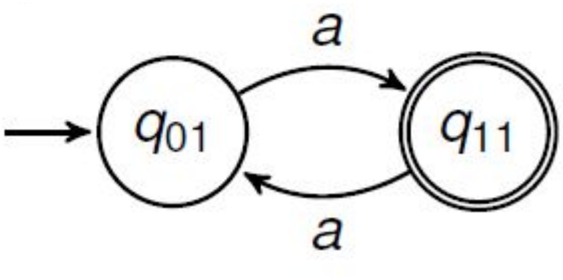
Thus, it is proved that $A_1 \cup A_2$ is being recognized by a DFA M. That means $A_1 \cup A_2$ is regular. - "Regular language is closed under union operation"

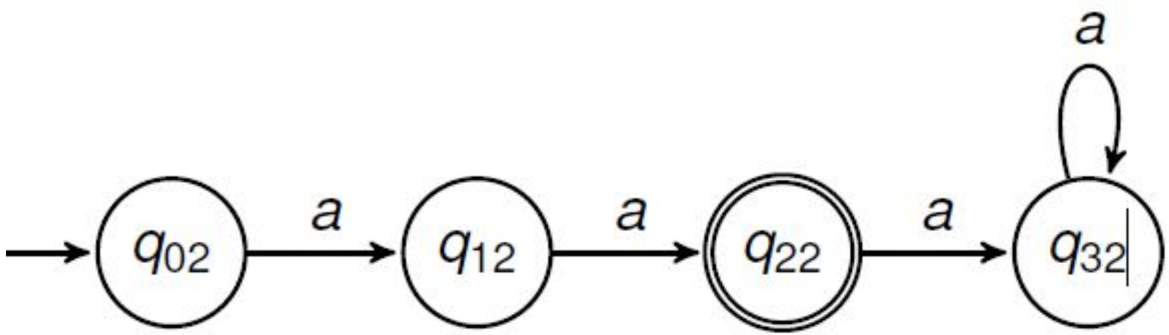
The Regular Operations *(sipser)*

Example: 1

A1 = { contains an odd number of a's }

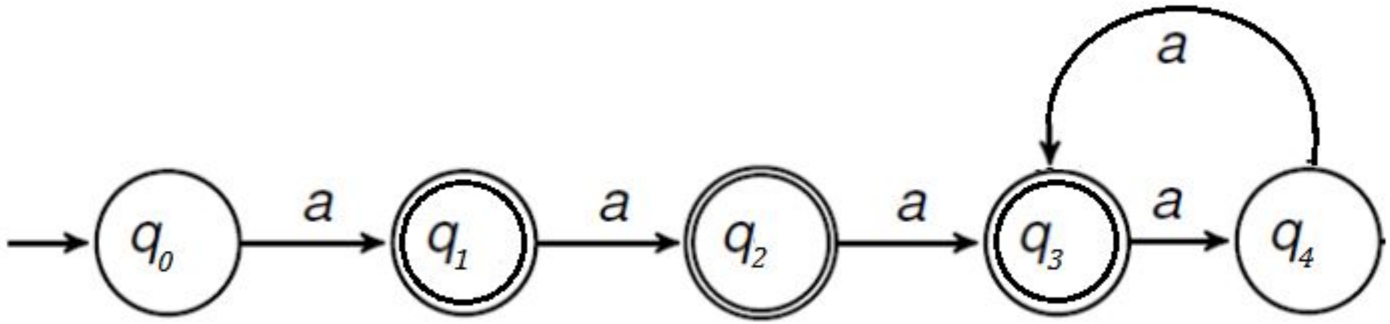
A2 = {aa}





The Regular Operations *(sipser)*

Following Machine M, recognizes $A_1 \cup A_2$



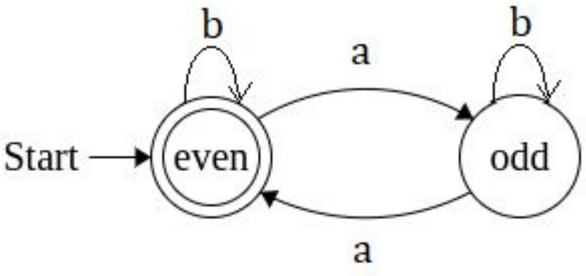
The Regular Operations *(sipser)*

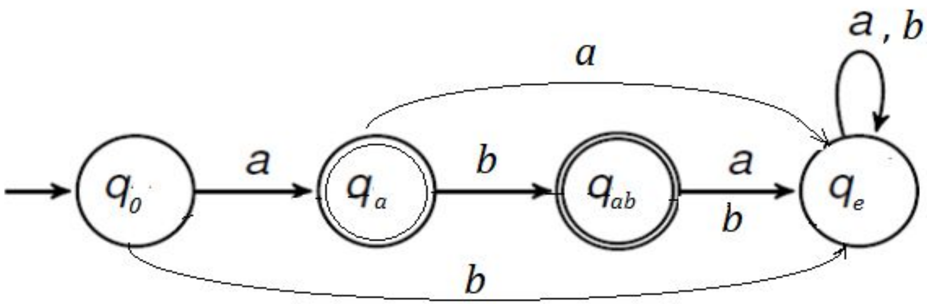
Example: 2

$\Sigma_1 = \{a, b\}$

L1 = { contains an even number of a's }

L2 = {a, ab}





The Regular Operations *(sipser)*

Following Machine , $M_{L1 \cup L2}$ recognizes $L_1 \cup L_2$