Automata theory

- Is the study of Abstract Computing Devices or Machines
- Deals with definitions and properties of mathematical models of computation

Automata theory allows us to practice with formal definitions of computation.

An automaton is a construct that possesses all the indispensable features of a digital computer.

Examples where Automata is used

Finite automata are a useful model for important kinds of hardware and software:

- **Software for designing and checking digital circuits.**
- **Lexical analyzer of compilers.**
- **Finding words and patterns in large bodies of text.**
- **Verification of systems with finite number of states,**

  e.g. communication protocols.

Practical Applications

Pattern recognition, intrusion detection, software engineering (UML), GUI programming (event-listeners), network security (protocols), verification (discovering bugs in chips), natural language processing (Amtrak reservation system), computational genomics (DNA sequencing), compilers, document definitions (XML), and many many more.

Text & Reference Books

Text Books:

- Linz, P , An Introduction to Formal Languages and Automata, Narosa Publishing House, 2007
- Hoperoft,J.E. and Ullman,J.D., Introduction to Automata Theory, Languages and Computation, 2 nd edition, Pearson Education.

References:

- Sipser, M., Introduction to the Theory of Computation, Second Edition, Thomson Course Technology, 2007.
- Martin, John C. , Introduction to Languages and the Theory of Computation, 3rd edition, McGraw-Hill, Inc., New York, NY, 2003.
- Moret,B. , The Theory of Computation, Pearson Education, 2001.
- Savage,J.E., Models of Computation, Addison-Wesley, 1998.
- Lewis, Elements of The Theory of Computation, 2nd edition, Pearson Education.
- http://portal.acm.org/

Model a Computer

- To Model Hardware we use the concept of AUTOMATON
- To Model Programming Language, we use Formal Languages as an abstraction.
- Associated with a Language is its GRAMMER.

Modeling Computers

- Input
  - Without it, we can’t describe a problem
- Output
  - Without it, we can’t get an answer
- Processing
  - Need some way of getting from the input to the output
- Memory
  - Need to keep track of what we are doing

Modeling Computer

- A transition Function gives the next state in terms of current state the current input symbol and the information currently in temporary storage.
- Transition from one state to another is called Move.
- Acceptors and Transducers

Languages

- A formal language is a set consisting of strings.
  - In some sense, one can think of English as consisting of all valid English words

    {a, all, back, abandon, … }
  - Probably more accurate to think of English as consisting of valid English *sentences.*
  - C is the set of all compiling C programs

Grammars

- Means of Describing languages
- A grammar is a set of rules which governs what is and isn't *syntactically* correct.
  - English example: "I dog love."

    -- violates English Grammar
  - C example:

```
int Wrong{
      float int;
}
```

    -- violates C's specification

Grammars

- GOAL : Automate the process of finding syntactic errors by magically transforming any grammatical specification to a computer or computer program

Relation between concepts

- Automata, Languages and Grammars are explored simultaneously through three computational models, progressively more complicated:

1. Finite Automata
2. Pushdown Automata
3. Turing Machines

Formal language

Alphabet = finite set of symbols or characters

examples: $\Sigma = \{a,b,\ldots.,z\}$, binary, $\{a,b\}$, ASCII

String = finite sequence of symbols from an alphabet

examples: aab, bbaba, 01101, 111, also computer programs

A **formal language** is a set of strings over an alphabet

Examples of formal languages over alphabet $\Sigma = \{a, b\}$:

$L_1 = \{aa, aba, aababa, aa\}$

$L_2 = \{$all strings containing just two a's and any number of b's$\}$

A formal language can be finite or infinite.

Formal languages (cont…)

- The **empty string**, denoted λ , has some special properties:

$$|\lambda| = 0$$

$$\lambda w = w \lambda = w$$

- If w is a string, then w^n stands for the string obtained by repeating w n times.

$$w^0 = \lambda$$

- Σ^k - set of strings of length k , each of whose symbol is in Σ.

$\Sigma^0 = \{\lambda\}$, if $\Sigma = \{0,1\}$ then

$\Sigma^1 = \{0,1\}$, $\Sigma^2 = \{00,11,01,10\}$

$\Sigma^3 = \{000,111,001,010,011,100,101,110\}$

Formal languages (cont…)

- Σ^* - set of all strings over an alphabet Σ.

e.g. $\{0,1\}^*$ - $\{\lambda, 0, 1, 11, 00, 01, 000, 101, \ldots\ldots\ldots\ldots\ldots\}$

- $\Sigma^+ = \Sigma^* - \{\lambda\}$ or $\Sigma^* = \Sigma^+ \cup \{\lambda\}$
- Language : set of all strings chosen from some Σ^*.

  L is the subset of Σ^*.

  $L^0 = \{\lambda\}$

  $L^1 = L$

L^n = L concatenated with itself n times.

Operations on languages

String operations:

$L^R = \{w^R \mid w \in L\}$ is "reverse of language"

$L_1L_2 = \{xy \mid x \in L_1, y \in L_2\}$ is "concatenation of languages"

$L^* = L^0 \cup L^1 \cup L^2 \cdot \ldots$ is "Kleene star" or "star closure"

$L^+ = L^1 \cup L^2 \cdot \ldots$ is positive closure

$\overline{L} = \Sigma^* - L$ Complement of language

Example

$L = \{a^n b^n : n \geq 0\}$

determine L^2.

$L^2 = \{a^n b^n a^m b^m : n \geq 0 , m \geq 0 \}$ where n and m are unrelated.

e.g. $L^2 = \{ aabbaaabbb \}$

Important example of a formal language

- alphabet: ASCII symbols
- string: a particular C++ program
- formal language: set of all legal C++ programs

Grammars

A grammar G is defined as a quadruple:

$$G = (V, T, S, P)$$

Where

- V is a finite set of objects called variables
- T is a finite set of objects called terminal symbols
- $S \in V$ is a special symbol called the Start symbol
- P is a finite set of productions or "production rules"

Sets V and T are nonempty and disjoint

Grammars

Production rules have the form:

$$X \to y$$

where X is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$

Given a string of the form

$$w = uXv$$

and a production rule

$$X \to y$$

we can apply the rule, replacing x with y, giving

$$z = uyv$$

We can then say that $\qquad w \Rightarrow z$

Read as "w derives z", or "z is derived from w"

Grammars

If $u \Rightarrow v$, $v \Rightarrow w$, $w \Rightarrow x$, $x \Rightarrow y$, and $y \Rightarrow z$, then we say:

$$u \overset{*}{\Rightarrow} z$$

This says that u derives z in an unspecified number of steps.

Along the way, we may generate strings which contain variables as well as terminals. These are called sentential forms.

Grammars

What is the relationship between a language and a grammar?

Let G = (V, T, S, P)

The set

$$L(G) = \{w \in T^* : S \Rightarrow w\}$$

is the language generated by G.

Grammars

Consider the grammar G = (V, T, S, P), where:

V = {S}

T = {a, b}

S = S,

P =

S $\rightarrow$ aSb
S $\rightarrow$ λ

Grammars

These are some of the strings in this language?

$S \Rightarrow aSb \Rightarrow ab$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$

It is easy to see that the language generated by this grammar is:

$L(G) = \{a^n b^n : n \geq 0\}$

Grammars

Let's go the other way, from a description of a language to a grammar that generates it. Find a grammar that generates:

$L = \{a^n b^{n+1} : n \geq 0\}$

So the strings of this language will be:

b (0 a's and 1 b)

abb (1 a and 2 b's)

aabbb (2 a's and 3 b's) . . .

$G = (\{S,A\}, \{a,b\}, S, P)$ where P is

$S \rightarrow Ab$, $A \rightarrow aAb$, $A \rightarrow \lambda$

or $S \rightarrow aSb$, $S \rightarrow b$

Questions

- Generate the grammar corresponding to the following languages :
  - $L = \{1^n0^n \mid n \geq 0\}$
  - $L = \{ 0^n1^n \mid n \geq 1\}$

Problems

- In automata theory, a problem is to decide whether a given string is a member of some particular language.

- This formulation is general enough to capture the difficulty levels of all problems.

Ways of Describing Computation

Machines

<u>Machine</u> = a formal description of a "computer"

- Based on states & transitions between states
- Computes some output from input
- 3 uses of machine, can be related to each other:
  - <u>acceptance</u>: input = string, output = yes/no membership in some language
  - <u>enumeration</u>: no input, output = list of all strings in some language
  - <u>function</u>: input = string, output = string

Finite Automata (or Finite State Machines)

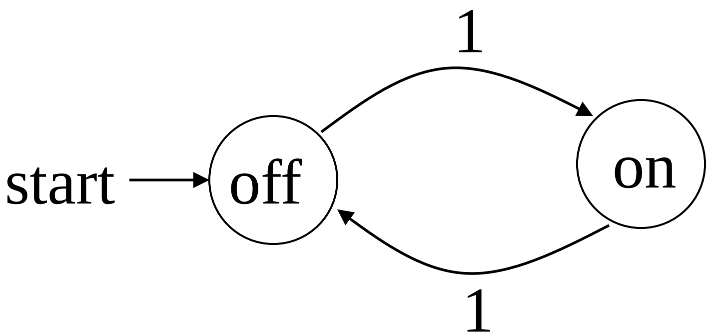
- This is the simplest kind of machine.
- Purpose of state – remember the relevant portion of system's history.
- Finite no of states – implement the system with a fixed set of resources.
- We will study 2 types of Finite Automata:
  - Deterministic Finite Automata (DFA)
  - Non-deterministic Finite Automata (NFA)

Deterministic Finite Automata

- A simplest model for computing
  - **Deterministic**: Machine is in a state. Upon receipt of a symbol will go to a **unique** state.
  - **Finite**: Have a finite number of states
  - **Automata**Self-operating machine
- DFA: finite-state machine without ambiguity
- Example : on/off switch

Deterministic Finite Automata (DFA)

The simplest example is:



There are some <u>states</u> and <u>transitions</u> (edges) between the states. The edge labels tell when we can move from one state to another.

Definition of DFA

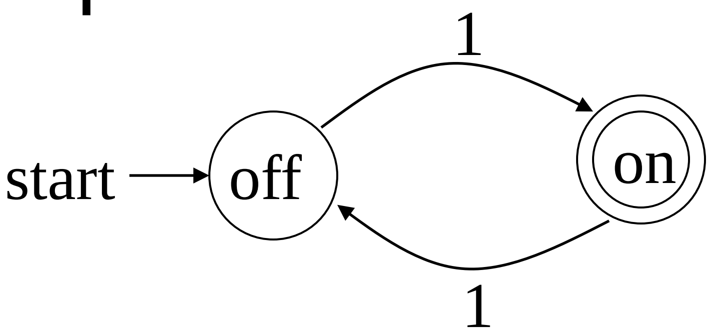
A DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q: Finite set of states
- Σ: Finite input alphabet
- δ:Transition function mapping $Q \times \Sigma \longrightarrow Q$
- $q_0 \in Q$:Initial state (only one)
- $F \subseteq Q$:Set of final states (zero or more)

δ specifies exactly one next state for each possible combination of a state and input symbol. In other words, exactly one transition arrow exits every state for each possible input symbol.

Definition of DFA

For example:



We use double circle to specify a final state.

We use a start marker to specify the initial state

$$Q : \{on, off\}$$

$$\Sigma : \{1\}$$

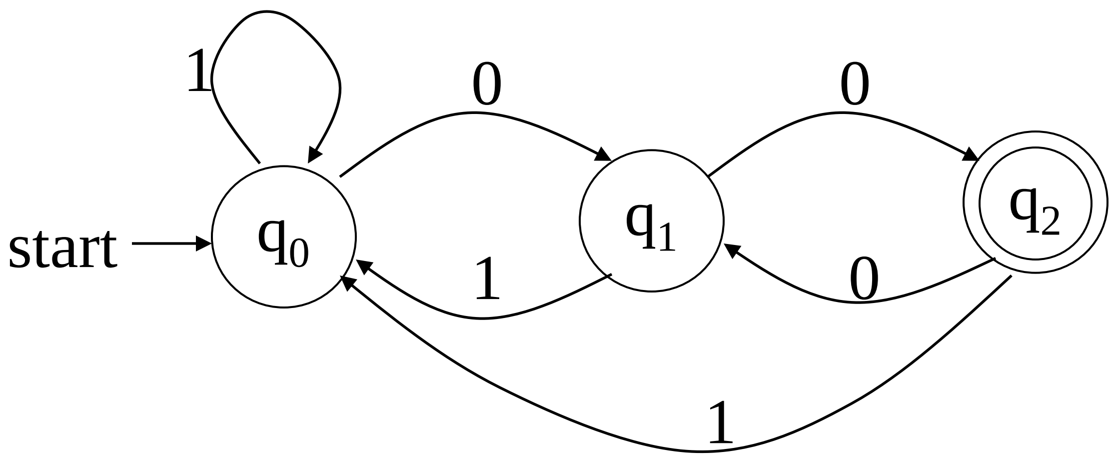
$$\delta : \{off \times 1 \rightarrow on;\ on \times 1 \rightarrow off\}$$

$$q_0 : off$$

$$F : \{on\}$$

Definition of DFA

Another Example:



What are Q, Σ, δ, q_0 and F in this DFA?

Transition Table

The DFA is (Q,Σ,δ,q_0,F) where $Q = \{q_0,q_1,q_2\}$, $\Sigma = \{0,1\}$, $F = \{q_2\}$ and δ is such that

Input →

States ↓

δ	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_0
q_2	q_1	q_0

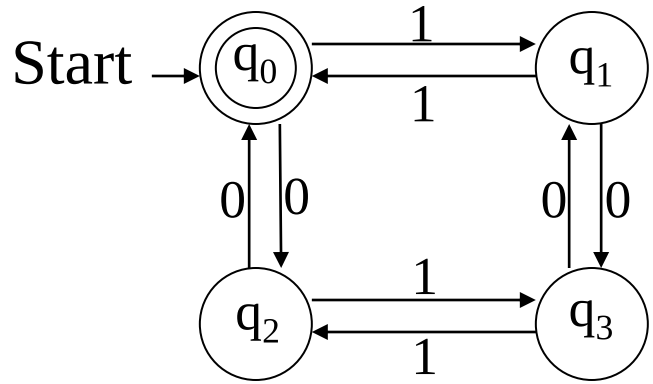
Note that there is <u>one transition only</u> for each input symbol from each state.

DFA Example

Consider the DFA $M=(Q,\Sigma,\delta,q_0,F)$ where $Q = \{q_0,q_1,q_2,q_3\}$, $\Sigma = \{0,1\}$, $F = \{q_0\}$ and δ is:

δ	0	1
$\rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

OR



We can use a <u>transition table</u> or a <u>transition diagram</u> to specify the transitions. What input can take you to the final state in M?

DFA and Strings

- DFA can **recognize** strings
  - String is input
  - If DFA ends at accept state, string is recognized

- Formal def. : A string is accepted by a finite automaton $M=(Q,\Sigma,\delta,q_0,F)$ if

  $\delta(q_0, x) = q$ for some $q \in F$.

Language of a DFA

Given a DFA M, the language accepted (or recognized) by M is the set of all strings that, starting from the initial state, will reach one of the final states after the whole string is traversed.

$L(M) = \{ w \in \Sigma^* \ : \delta^*(q_0, w) \in F \}$

- A machine may accept several strings, but it always recognizes one language. If m/c accepts no strings, the language recognized by such machine is empty language.
- A language is called a **regular language** if some finite automaton recognizes it

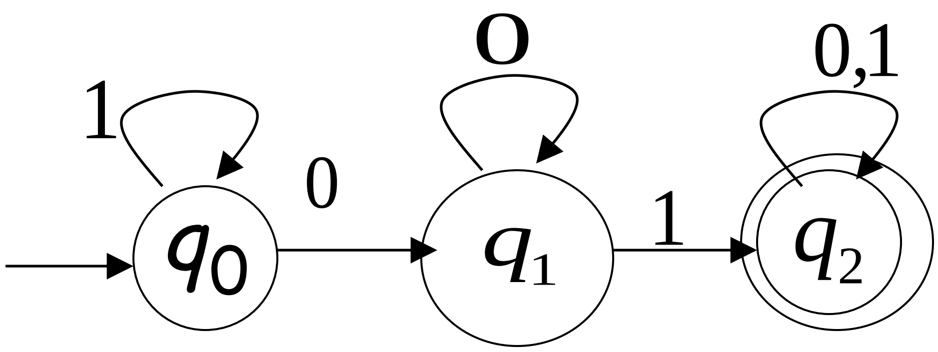
- Language rejected by DFA M:

$$\overline{L(M)} = \{w \in \Sigma^* : \delta^*(q_0, w) \notin F\}$$

Example

- DFA that accepts all strings of 0's and 1's with a substring 01

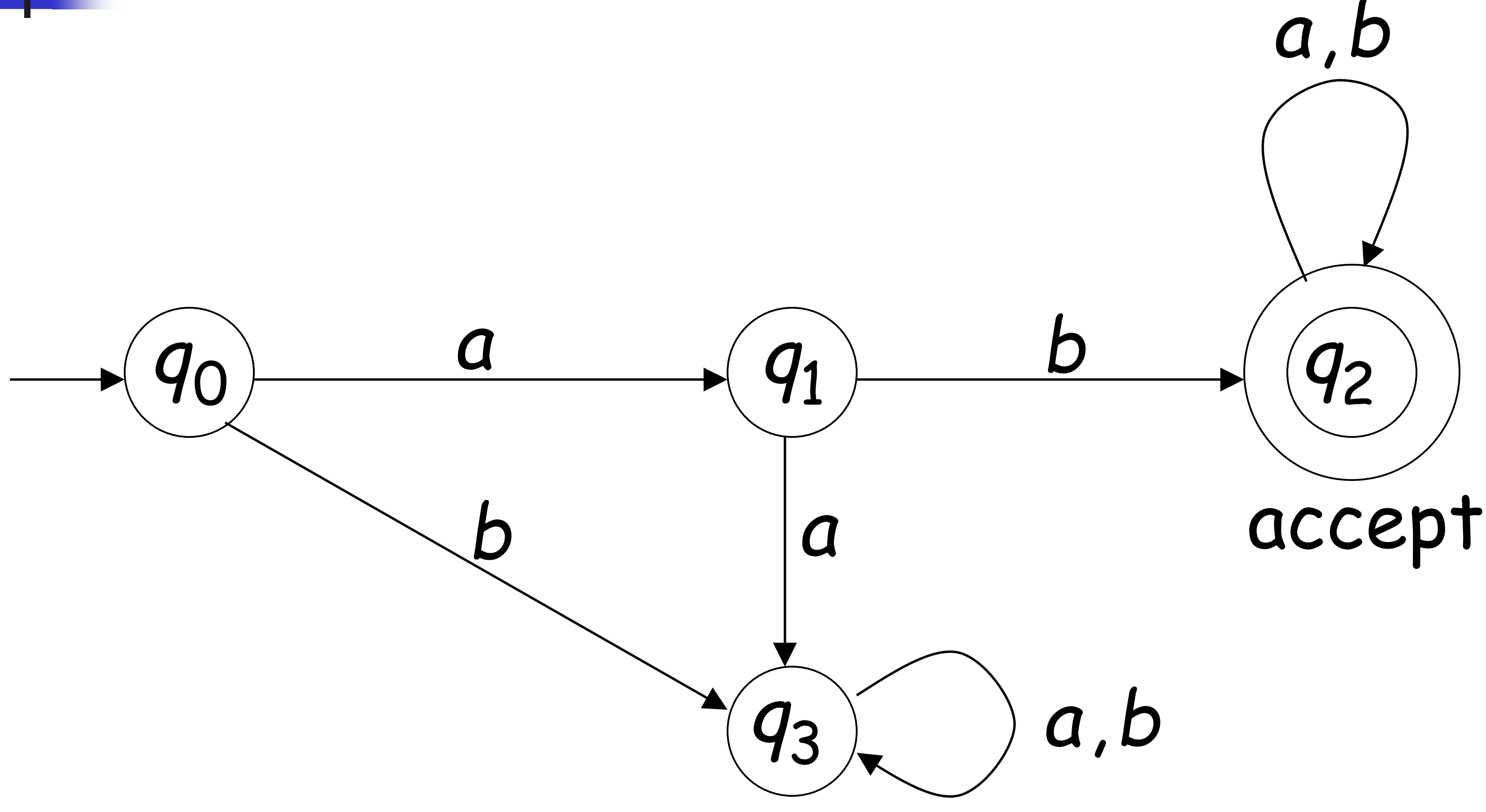
L ={w =x01y , x and y € {0,1}*}



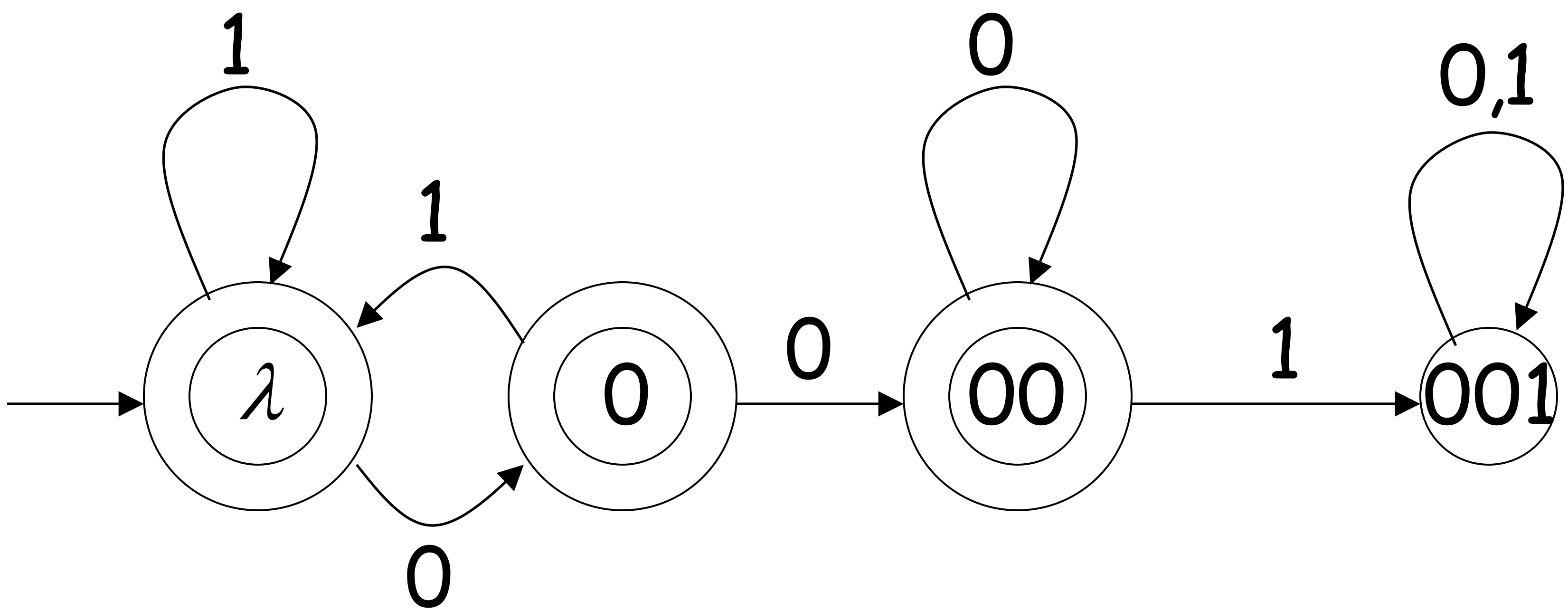
Class discussion - examples

- The language accepted by the previous example (given in slide 47) is the set of strings over input alphabets {1,0} with even number of 0's and 1's.
- Find a DFA that recognizes the set of all strings on {a,b} starting with prefix ab.
- Find a DFA that accepts all the strings on {0,1} except those containing 001.

$L(M)$ = { all strings with prefix ab }



$L(M)$ = { all strings without substring 001 }



$$L(M) = \{a^n b : n \geq 0\}$$