Theory of Computing SE-2112

Lecture-1

Dr. Naushin Nower
Professor, IIT DU

Textbook

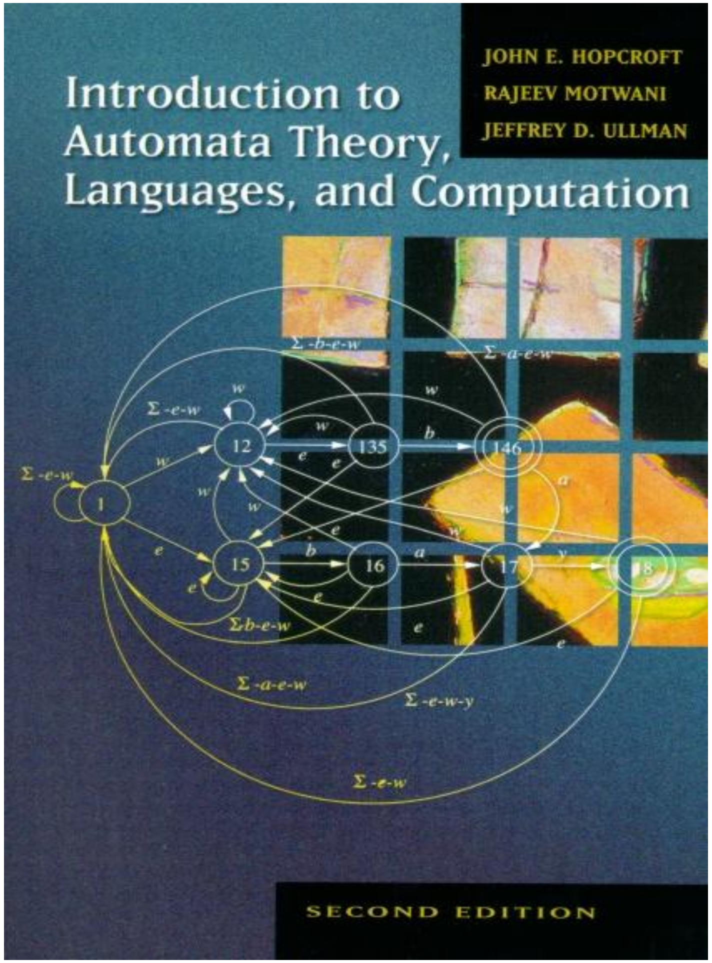
Introduction to Automata Theory, Languages, and Computation

John E. Hopcroft,

Rajeev Motwani,

Jeffrey D. Ullman,

(2nd Ed. Addison-Wesley, 2001)



Topics to be Covered

- Finite automata
- Regular languages, Regular grammars
- Properties of Regular Languages
- Context-free Grammars and Languages
- Pushdown Automata
- Properties of Context-Free Languages
- Induction to Turing Machine and etc.. ☺

Grading

Assignment

Midterm 1

Attendance

Final Exam

Theory of Computation

Theory of computation

Automata theory (also known as **Theory of Computation**) is a theoretical branch of Computer Science and Mathematics

is the branch that deals with how efficiently problems can be solved on a model of computation, using an algorithm

deals with the logic of computation with respect to simple machines, referred to as automata

Automata* enables the scientists to understand how machines compute the functions and solve problems

"What are the fundamental capabilities and limitations of computers?

What is Automata Theory?

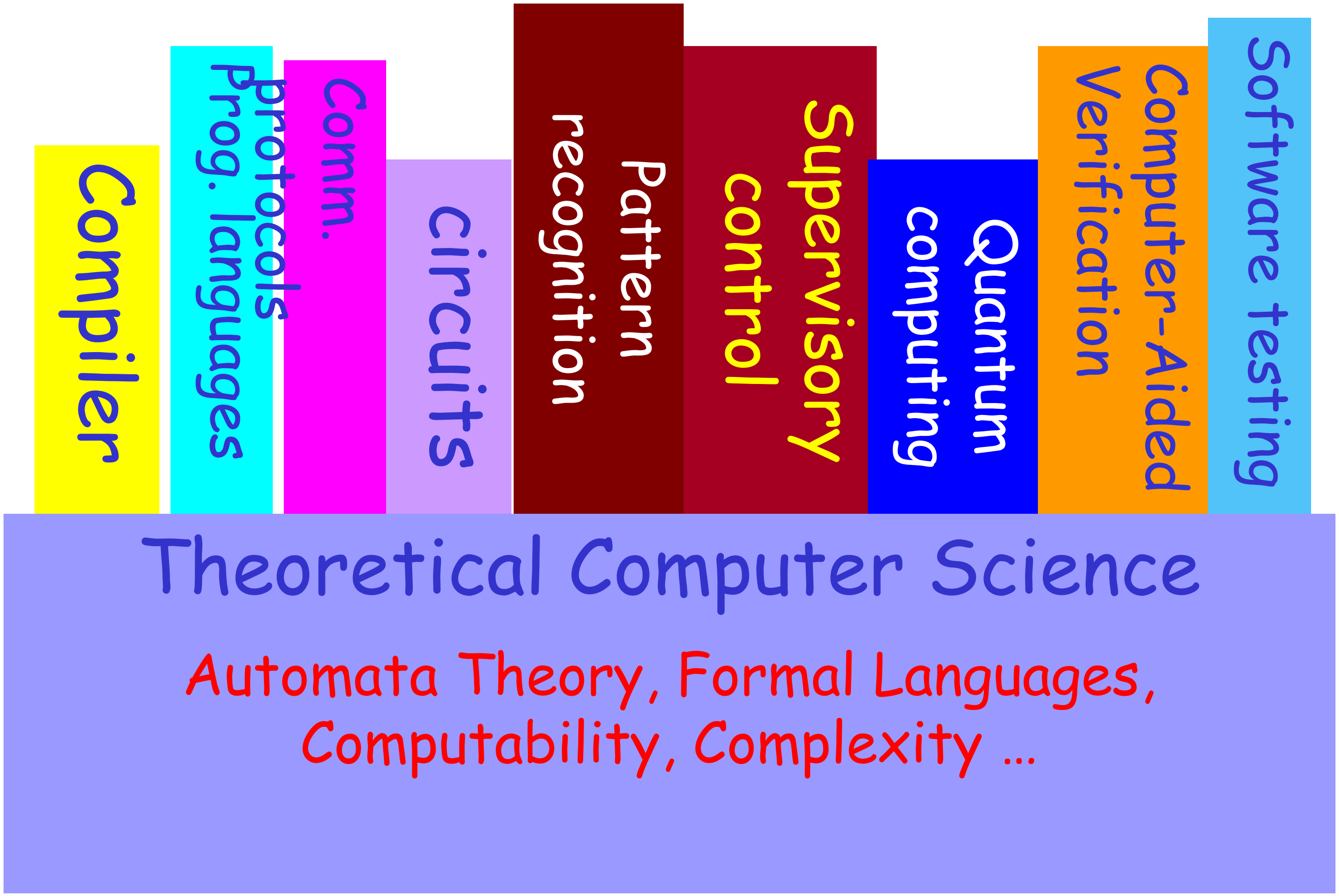
- Study of abstract computing devices or machines
- Automaton=an abstract computing device
  - a device need not even be a physical hardware
- Abstract devices are (simplified) models of real computations
- Computations happen everywhere: On your laptop, on your cell phone, in nature, …

Why do study Automata Theory

Finite automata are a useful model for hardware and software:

- Software for designing and checking digital circuits.
- Lexical analyzer of compilers.
- Finding words and patterns in large bodies of text, e.g. in web pages.
- Verification of systems with finite number of states, e.g. communication protocols, software testing etc...

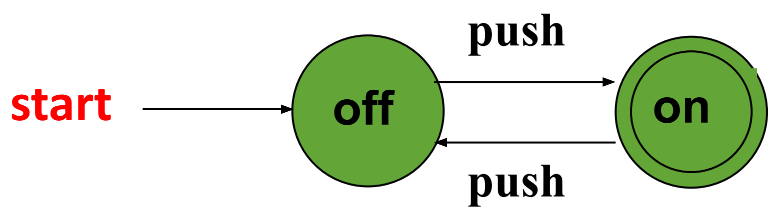
Applications



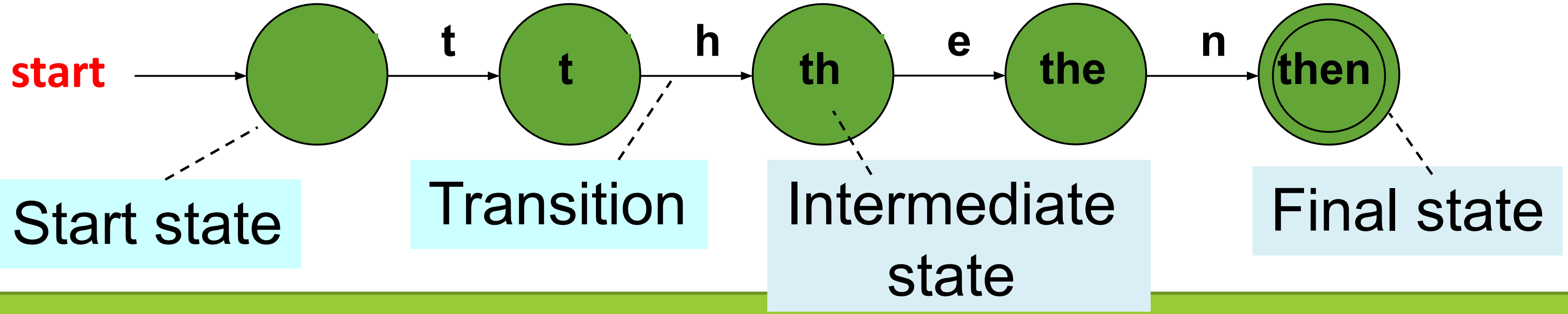
Finite Automata

Is the simplest kind of machine

--- an FA modeling an on/off switch



an FA modeling recognition of the keyword "then" in a lexical analyzer



Formal Proof

Deductive Proofs

From the given statement(s) to a conclusion statement (what we want to prove)

- Logical progression by direct implications

Example for parsing a statement:

- “If $y \geq 4$, then $2^y \geq y^2$.”

given *conclusion*

(there are other ways of writing this).

Example: Deductive proof

Let <u>Claim 1:</u> If $y \geq 4$, then $2^y \geq y^2$.

Let x be any number which is obtained by adding the squares of 4 positive integers.

<u>Claim 2:</u>

Given x and assuming that Claim 1 is true, prove that $2^x \geq x^2$

- <u>Proof:</u>
  1) Given: $x = a^2 + b^2 + c^2 + d^2$
  2) Given: $a \geq 1, b \geq 1, c \geq 1, d \geq 1$
  3) $\Rightarrow a^2 \geq 1, b^2 \geq 1, c^2 \geq 1, d^2 \geq 1$ (by 2)
  4) $\Rightarrow x \geq 4$ (by 1 & 3)
  5) $\Rightarrow 2^x \geq x^2$ (by 4 and Claim 1)

"implies" or *"follows"*

Quantifiers

"For all" or "For every"

- Universal proofs
- Notation*=?

"There exists"

- Used in existential proofs
- Notation*=?

Implication is denoted by =>

- E.g., "IF A THEN B" can also be written as "A=>B"

Proving techniques

- By contradiction
  - Start with the statement contradictory to the given statement
  - E.g., To prove (A => B), we start with:
    - (A and ~B)
    - … and then show that could never happen

    What if you want to prove that "(A and B => C or D)"?

- By induction
  - (3 steps) Basis, inductive hypothesis, inductive step
- By contrapositive statement
  - If *A* then *B* ≡ If ~*B* then ~*A*

Proving techniques…

- By counter-example
  - Show an example that disproves the claim

- Note: There is no such thing called a "proof by example"!
  - So when asked to prove a claim, an example that satisfied that claim is *not* a proof

“If-and-Only-If” statements

- “A if and only if B” (A <==> B)
  - *(if part)* if B then A (<=)
  - *(only if part)* A only if B (=>)

    (same as “if A then B”)
- “If and only if” is abbreviated as “iff”
  - i.e., “A iff B”
- Example:
  - Theorem: *Let x be a real number. Then floor of x = ceiling of x if and only if x is an integer.*
- Proofs for iff have two parts
  - One for the “if part” & another for the “only if part”

Preliminaries

Central Concepts of Automata Theory

- Symbols ---- a, b, 0, 1,2 etc
- Alphabet --- a set of symbols
- Strings --- a sequence of symbols from an alphabet
- Language --- a set of strings from the same alphabet

Alphabets

An alphabet is a finite, nonempty set of symbols.

- Conventional notation --- Σ

- The term "symbol" is usually undefined.

- Examples ---
  - Binary alphabet $\Sigma = \{0, 1\}$.
  - $\Sigma = \{a, b, \ldots, z\} \ldots$

Strings

A string (or word) is a finite sequence of symbols from an alphabet.

- Example ---
  - 1011 is a string from alphabet $\Sigma = \{0, 1\}$
- Empty string ε --- a string with zero occurrences of symbols
- Length $|w|$ of string w --- the number of positions for symbols in w
  - Examples --- $|0111|=4$, $|\varepsilon|=0$, …

String..

- Power of an alphabet Σ^k ---

  a set of all strings of length k

  - Examples ---
    - given $\Sigma = \{0, 1\}$, we have
  - $\Sigma^0 = \{\varepsilon\}$, $\Sigma^2 = \{00, 01, 10, 11\}$
  - Supplemental --- $1^0 = \varepsilon$, $(01)^0 = \varepsilon$, …
- Set of all strings over Σ --- denoted as Σ^*
- It is not difficult to know that

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$

Strings..

- Σ^+ = set of nonempty strings from Σ

  $= \Sigma^* - \{\varepsilon\}$

- Therefore, we have

- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$
- $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$

- Concatenation of two strings *x* and y --- *xy*
  - Example ---
    - if $x = 01101$, $y = 110$, then

      $xy = 01101110$, $xx = x^2 = 0110101101$, …

- ε is the identity for concatenation

  since $\varepsilon w = w\varepsilon = w$.

Strings..

- Power of a string ---
  - Defined by concatenation ---
    - $x^i = xx\ldots x$ (x concatenated i times)
  - Defined by recursion ---
    - $x^0 = \varepsilon$ (by definition)
    - $x^i = xx^{i-1}$

Languages

a language is a set of strings all chosen from some Σ^*

- If Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a language over Σ.
  - *Examples ---*
    - The set of all legal English words is a language. Why? What is the alphabet here?

      Answer: the set of all letters

    - A legal program of C is a language. Why? What is the alphabet here?

      Answer: a subset of the ASCII characters.

Languages

- More examples of languages ---
  - The set of all strings of n 0's followed by n 1's for $n \geq 0$:

    $\{\varepsilon, 01, 0011, 000111, \ldots\}$
  - Σ^* is an infinite language for any alphabet Σ.
  - φ = the empty language (not the empty string ε) is a language over any alphabet.
  - $\{\varepsilon\}$ is a language over any alphabet (consisting of only one string, the empty string ε).

Finite Automata

Finite Automata
(or Finite State Machines)

Automatons are abstract models of machines that perform computations on an input by moving through a series of states or configurations. At each state of the computation, a transition function determines the next configuration on the basis of a finite portion of the present configuration. As a result, once the computation reaches an accepting configuration, it accepts that input.

This is the simplest kind of machine.

We will study 3 types of Finite Automata:

- Deterministic Finite Automata (DFA)
- Non-deterministic Finite Automata (NFA)
- Finite Automata with ε-transitions (ε-NFA)

Deterministic Finite Automata (DFA)

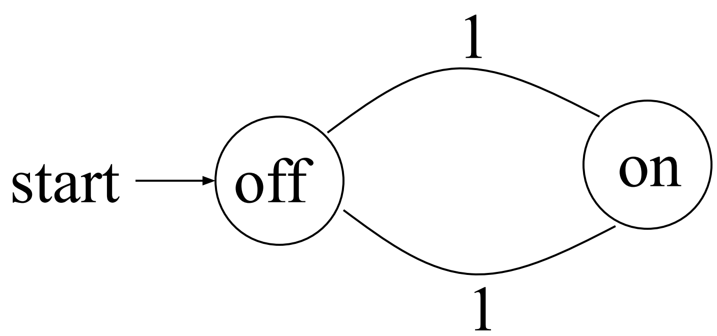
- Refers to the fact that on each input there is one and only state to which the automaton has a transition from its current state.

Non-deterministic Finite Automata (NFA)

- Has a transition in multiple states at the same time for one input symbol

Deterministic Finite Automata (DFA)

We have seen a simple example before:



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

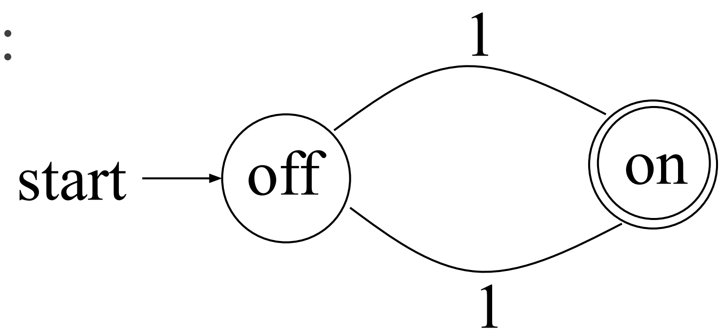
A DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

Q is a finite set of <u>states</u>
Σ is a finite input <u>alphabet</u>
δ is the <u>transition function</u> mapping $Q \times \Sigma$ to Q
q_0 in Q is the <u>initial state</u> (only one)
$F \subseteq Q$ is a set of <u>final states</u> (zero or more)

- "Final" and "accepting" are synonyms.

Definition of DFA

For example:



Q is the set of states: {on, off}

Σ is the set of input symbols: {1}

δ is the transitions: off $\times$ 1 $\rightarrow$ on; on $\times$ 1 $\rightarrow$ off

q_0 is the initial state: off

F is the set of final states (double circle): {on}

The Transition Function

- Takes two arguments: a state and an input symbol.
- δ(q, a) = the state that the DFA goes to when it is in state *q* and input *a* is received.
- Nodes = states.
- Arcs represent transition function.
  - Arc from state p to state q labeled by all those input symbols that have transitions from p to q.
- Arrow labeled “Start” to the start state.
- Final states indicated by double circles.

Example #1

A DFA A accepts string *w* if there is a path from q_0 to an accepting (or final) state that is labeled by *w*

Build a DFA for the following language:

- L = {w | w is a binary string that contains 01 as a substring}

Steps for building a DFA to recognize L:

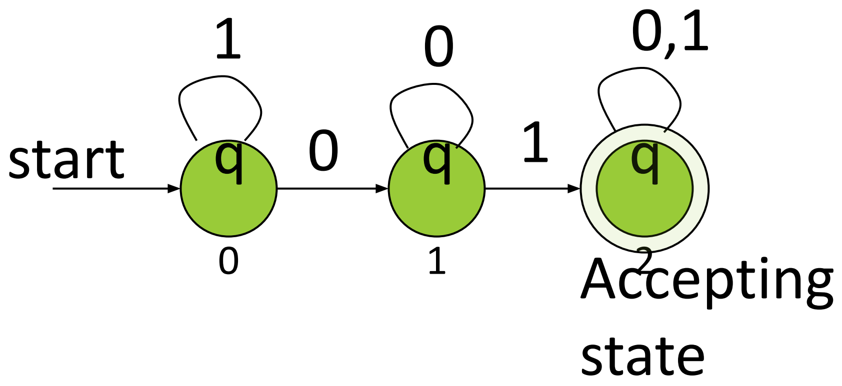
- ∑ = {0,1}
- Decide on the states: Q
- Designate start state and final state(s)
- δ: Decide on the transitions:

"Final" states == same as "accepting states"

Other states == same as "non-accepting states"

DFA for strings containing 01

- What makes this DFA deterministic?



- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0,1\}$
- start state $= q_0$
- $F = \{q_2\}$
- Transition table

symbols

δ	**0**	**1**
→ **q_0**	q_1	q_0
q_1	q_1	q_2
***q_2**	q_2	q_2

states

Language of a DFA

Automata of all kinds define languages.

If A is an automaton, L(A) is its language.

For a DFA A, L(A) is the set of strings labeling paths from the start state to a final state.

Formally: L(A) = the set of strings w such that $\delta(q_0, w)$ is in F.

- The language of our example DFA is:

{w | w is in {0,1}* and w does not have two consecutive 0's}

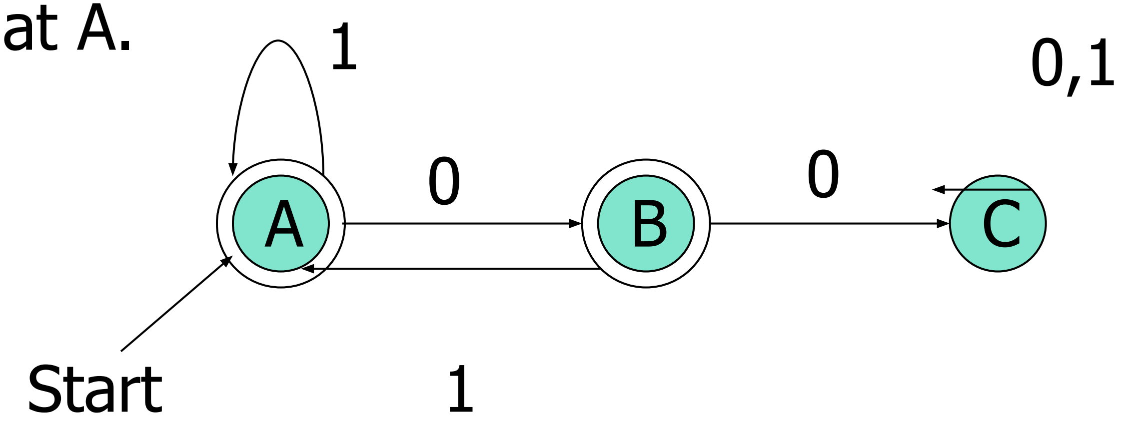
These conditions about w are true.

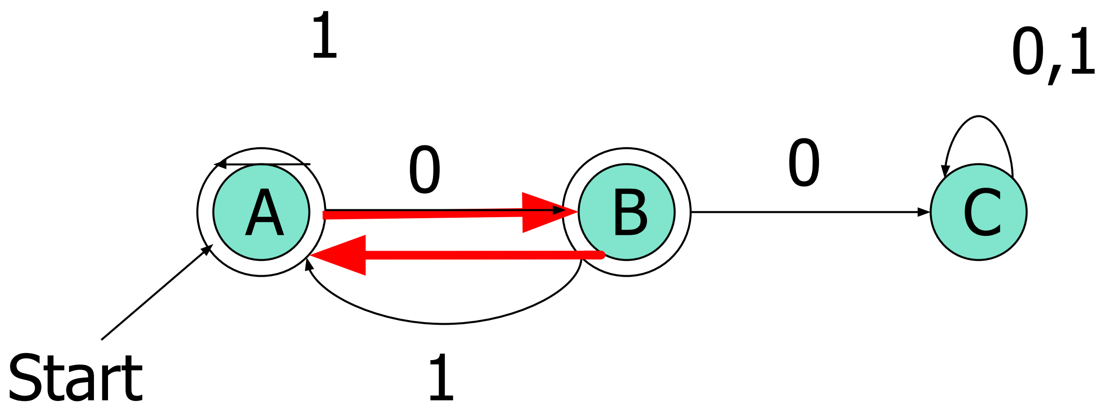
Read a *set former* as "The set of strings w…

Example: String in a Language

String 010 is in the language of the DFA below.

Start at A.

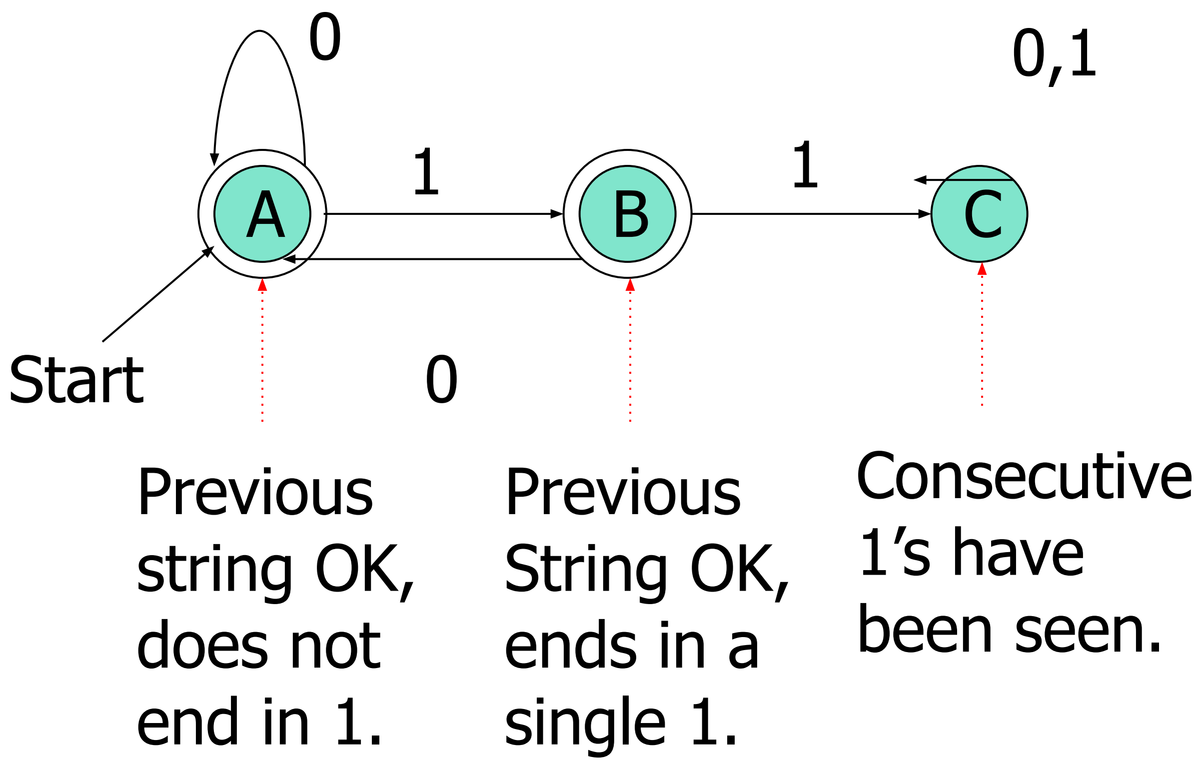




{w | w is in {0,1}* and w does not have two consecutive 0's}

Example #2

Accepts all strings without two consecutive 1's.



Transition Table

Final states starred

Columns = input symbols

Arrow for start state

		0	1
→	* A	A	B
	* B	A	C
	C	C	C

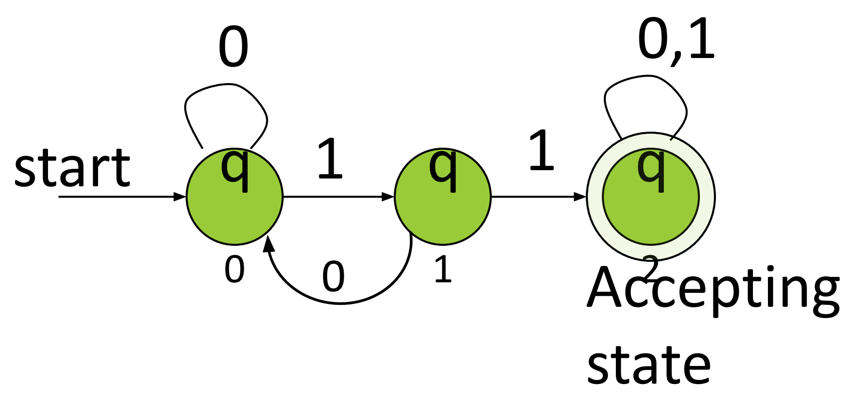
Rows = states

Example #3

Clamping Logic:

- A clamping circuit waits for a ”1” input, and turns on forever. However, to avoid clamping on spurious noise, we’ll design a DFA that waits for *two consecutive 1s* in a row before clamping on.

- Build a DFA for the following language:
  L = { w | w is a bit string which contains the substring 11}

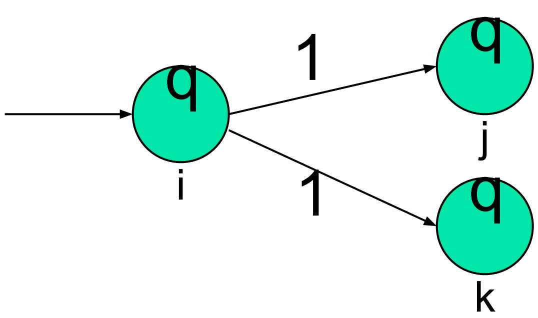


Non-deterministic finite automaton (NFA)

A *non-deterministic finite automaton* has the ability to be in several states at once.

Transitions from a state on an input symbol can be to any set of states.

- Implying that the machine can exist in more than one state at the same time
- Transitions could be non-deterministic



- Each transition function therefore maps to a set of states

NFA

A Non-deterministic Finite Automaton (NFA) consists of:

- Q ==> a finite set of states
- ∑ ==> a finite set of input symbols (alphabet)
- q_0 ==> a start state
- F ==> set of accepting states
- Δ ==> a transition function, which is a mapping between Q x ∑ ==> subset of Q

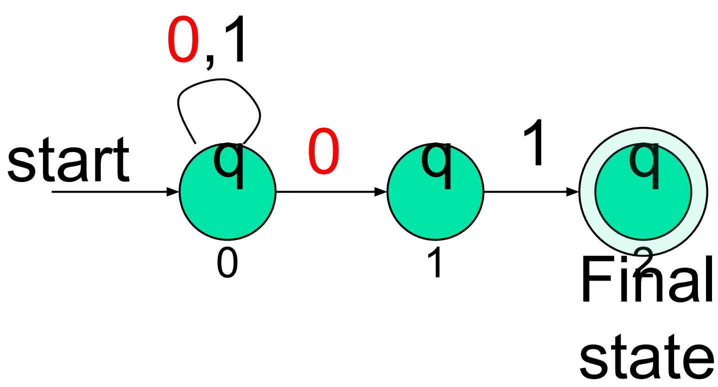
An NFA is also defined by the 5-tuple:

- {Q, ∑ , q_0,F, Δ }

Example # 4

An NFA accepting all strings that ends in 01.

Why is this non-deterministic?



- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0,1\}$
- start state = q_0
- $F = \{q_2\}$
- Transition table

symbols

states

δ	**0**	**1**
→ **q_0**	$\{q_0, q_1\}$	$\{q_0\}$
q_1	Φ	$\{q_2\}$
***q_2**	Φ	Φ

Thank you 😊