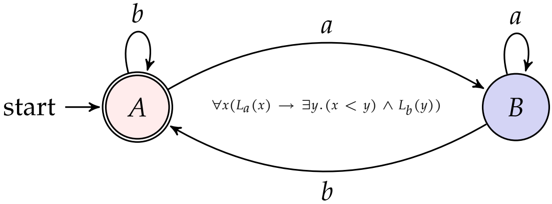
CS 208: Automata Theory and Logic

Lecture 2: Finite State Automata

Ashutosh Trivedi



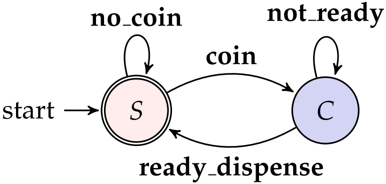
Department of Computer Science and Engineering,
Indian Institute of Technology Bombay.

Computation With Finitely Many States

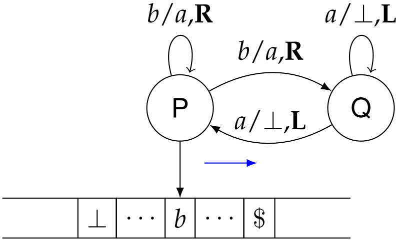
Non-determinism

Machines and their Mathematical Abstractions

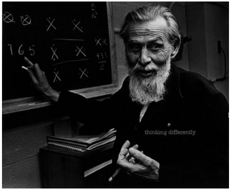
Finite instruction machine with finite memory (Finite State Automata)

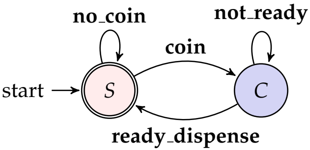


Finite instruction machine with unbounded memory (Turing machine)



Finite State Automata





- Introduced first by two neuro-psychologists Warren S. McCullough and Walter Pitts in 1943 as a model for human brain!
- Finite automata can naturally model microprocessors and even software programs working on variables with bounded domain
- Capture so-called regular sets of sequences that occur in many different fields (logic, algebra, regEx)
- Nice theoretical properties
- Applications in digital circuit/protocol verification, compilers, pattern recognition, etc.

Calculemus! — Gottfried Wilhelm von Leibniz

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.
- Recognize a binary (decimal) string that is a multiple of 2.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.
- Recognize a binary (decimal) string that is a multiple of 2.
- Recognize a binary (decimal) string that is a multiple of 3.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.
- Recognize a binary (decimal) string that is a multiple of 2.
- Recognize a binary (decimal) string that is a multiple of 3.
- Recognize a string with well-matched parenthesis.

Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.
- Recognize a binary (decimal) string that is a multiple of 2.
- Recognize a binary (decimal) string that is a multiple of 3.
- Recognize a string with well-matched parenthesis.
- Recognize a # separated string of the form $w\#\overline{w}$.

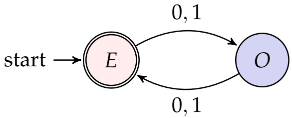
Calculemus! — Gottfried Wilhelm von Leibniz

Let us observe our mental process while we compute the following:

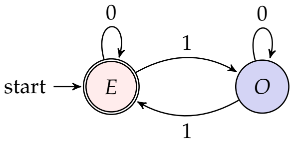
- Recognize a string of an even length.
- Recognize a binary string of an even number of 0's.
- Recognize a binary string of an odd number of 0's.
- Recognize a string that contains your roll number.
- Recognize a binary (decimal) string that is a multiple of 2.
- Recognize a binary (decimal) string that is a multiple of 3.
- Recognize a string with well-matched parenthesis.
- Recognize a # separated string of the form $w\#\overline{w}$.
- Recognize a string with a prime number of 1's

Finite State Automata

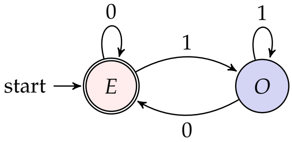
Automaton accepting strings of even length:



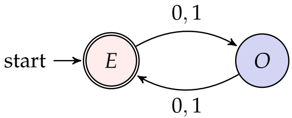
Automaton accepting strings with an even number of 1's:



Automaton accepting even strings (multiple of 2):



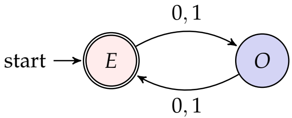
Finite State Automata



A finite state automaton is a tuple $(S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times \Sigma \to S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

Finite State Automata



A finite state automaton is a tuple $(S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times \Sigma \to S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

Example: The automaton in the figure above can be represented as $(S, \Sigma, \delta, s_0, F)$ where $S = \{E, O\}$, $\Sigma = \{0, 1\}$, $s_0 = E$, $F = \{E\}$, and transition function δ is such that

- $\delta(E, 0) = O$, $\delta(E, 1) = 0$, and $\delta(O, 0) = E$, $\delta(O, 1) = E$.

State Diagram

Let's draw the state diagram of the following automaton $(S, \Sigma, \delta, s_1, F)$:

- $S = \{s_1, s_2, s_3\}$
- $\Sigma = \{0, 1\}$,
- δ is given in a tabular form below:

S	0	1
s_1	s_1	s_2
s_2	s_3	s_2
s_3	s_2	s_2

- s_1 is the initial state, and
- $F = \{s_2\}$.

State Diagram

Let's draw the state diagram of the following automaton $(S, \Sigma, \delta, s_1, F)$:

- $S = \{s_1, s_2, s_3\}$
- $\Sigma = \{0, 1\}$,
- δ is given in a tabular form below:

S	0	1
s_1	s_1	s_2
s_2	s_3	s_2
s_3	s_2	s_2

- s_1 is the initial state, and
- $F = \{s_2\}$.

What does it accept?

Semantics of the finite state automata

A finite state automaton (DFA) is a tuple $(S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times \Sigma \to S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.
- A computation or a run of a DFA on a string $w = a_0 a_1 \ldots a_{n-1}$ is the finite sequence

  $$s_0, a_1 s_1, a_2, \ldots, a_{n-1}, s_n$$

  where s_0 is the starting state, and $\delta(s_{i-1}, a_i) = s_{i+1}$.
- A run is accepting if the last state of the unique computation is an accept state, i.e. $s_n \in F$.
- Language of a DFA A

  $$L(A) = \{w \ : \ \text{the unique run of } A \text{ on } w \text{ is accepting}\}.$$

Semantics of the finite state automata

A finite state automaton (DFA) is a tuple $(S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times \Sigma \to S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.
- A computation or a run of a DFA on a string $w = a_0 a_1 \ldots a_{n-1}$ is the finite sequence

  $$s_0, a_1 s_1, a_2, \ldots, a_{n-1}, s_n$$

  where s_0 is the starting state, and $\delta(s_{i-1}, a_i) = s_{i+1}$.
- A run is accepting if the last state of the unique computation is an accept state, i.e. $s_n \in F$.
- Language of a DFA A

  $$L(A) = \{w \ : \ \text{the unique run of } A \text{ on } w \text{ is accepting}\}.$$

Definition (Regular Languages)

A language is called regular if it is accepted by a finite state automaton.

Properties of Regular Languages

Let A and B be languages (remember they are sets). We define the following operations on them:

- Union: $A \cup B = \{w \ : \ w \in A \text{ or } w \in B\}$
- Concatenation: $AB = \{wv \ : \ w \in A \text{ and } v \in B\}$
- Closure (Kleene Closure, or Star):
  $A^* = \{w_1 w_2 \ldots w_k \ : \ k \geq 0 \text{ and } w_i \in A\}$. In other words:

  $$A^* = \cup_{i \geq 0} A^i$$

  where $A^0 = \emptyset$, $A^1 = A$, $A^2 = AA$, and so on.

Define the notion of a set being closed under an operation (say, $\mathbb{N}$ and $\times$).

Properties of Regular Languages

Let A and B be languages (remember they are sets). We define the following operations on them:

- Union: $A \cup B = \{w : w \in A \text{ or } w \in B\}$
- Concatenation: $AB = \{wv : w \in A \text{ and } v \in B\}$
- Closure (Kleene Closure, or Star):
  $A^* = \{w_1 w_2 \ldots w_k : k \geq 0 \text{ and } w_i \in A\}$. In other words:

$$A^* = \cup_{i \geq 0} A^i$$

where $A^0 = \emptyset$, $A^1 = A$, $A^2 = AA$, and so on.

Define the notion of a set being closed under an operation (say, $\mathbb{N}$ and $\times$).

Theorem

The class of regular languages is closed under union, concatenation, and Kleene closure.

Closure under Union

Lemma

The class of regular languages is closed under union.

Proof.

- Let A_1 and A_1 be regular languages.

Closure under Union

Lemma

The class of regular languages is closed under union.

Proof.

- Let A_1 and A_1 be regular languages.
- Let $M_1 = (S_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (S_2, \Sigma, \delta_2, s_2, F_2)$ be finite automata accepting these languages.

Closure under Union

Lemma

The class of regular languages is closed under union.

Proof.

- Let A_1 and A_1 be regular languages.
- Let $M_1 = (S_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (S_2, \Sigma, \delta_2, s_2, F_2)$ be finite automata accepting these languages.
- Simulate both automata together!
- The language $A \cup B$ is accept by the resulting finite state automaton, and hence is regular.

$\square$

Closure under Union

Lemma

The class of regular languages is closed under union.

Proof.

- Let A_1 and A_1 be regular languages.
- Let $M_1 = (S_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (S_2, \Sigma, \delta_2, s_2, F_2)$ be finite automata accepting these languages.
- Simulate both automata together!
- The language $A \cup B$ is accept by the resulting finite state automaton, and hence is regular.

$\square$

Class Exercise: Extend this construction for intersection.

Closure under Concatenation

Lemma

The class of regular languages is closed under concatenation.

Proof.

(Attempt).

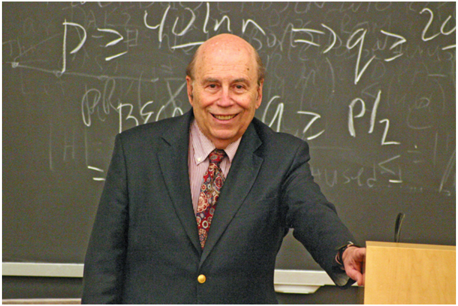
- Let A_1 and A_1 be regular languages.
- Let $M_1 = (S_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (S_2, \Sigma, \delta_2, s_2, F_2)$ be finite automata accepting these languages.
- How can we find an automaton that accepts the concatenation?
- Does this automaton fit our definition of a finite state automaton?
- Determinism vs Non-determinism

□

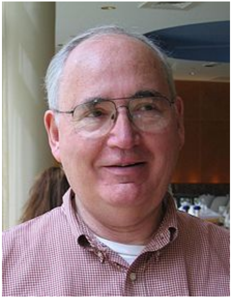
Computation With Finitely Many States

Non-determinism

Nondeterministic Finite State Automata

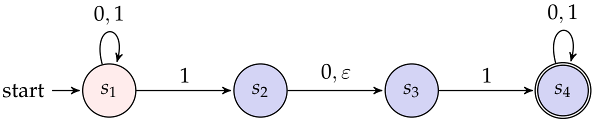


Michael O. Rabin

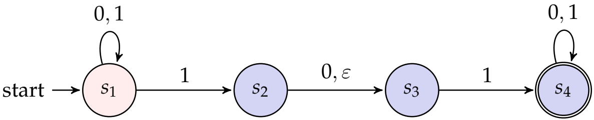


Dana Scott

Non-deterministic Finite State Automata



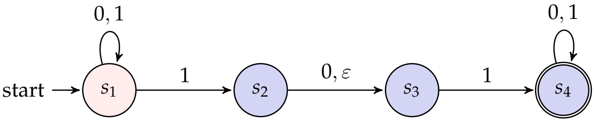
Non-deterministic Finite State Automata



A non-deterministic finite state automaton (NFA) is a tuple $(S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

Non-deterministic Finite State Automata



A non-deterministic finite state automaton (NFA) is a tuple $(S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

Example: Difference between a deterministic vs a non-deterministic computation (above NFA on a string 010110).

Non-deterministic Finite Automata: Semantics

A non-deterministic finite state automaton (NFA) is a tuple $(S, \Sigma, \delta, s_0, F)$, where:

- S is a finite set called the states;
- Σ is a finite set called the alphabet;
- $\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ is the transition function;
- $s_0 \in S$ is the start state; and
- $F \subseteq S$ is the set of accept states.

- A computation or a run of a NFA on a string $w = a_0 a_1 \ldots a_{n-1}$ is a finite sequence

$$s_0, r_1 s_1, r_2, \ldots, r_{k-1}, s_n$$

  where s_0 is the starting state, and $s_{i+1} \in \delta(s_{i-1}, r_i)$ and $r_0 r_1 \ldots r_{k-1} = a_0 a_1 \ldots a_{n-1}$.
- Unlike DFA, there can be multiple runs of an NFA on a string.
- A run is accepting if the last state of some computation is an accepting state $s_n \in F$.
- Language of a NFA A $L(A) = \{w \;:\; \text{some run of } A \text{ on } w \text{ is accepting}\}$.

NFA vs DFA

NFA are often more convenient to design than DFA, e.g.:

- $\{w \ : \ w$ contains 1 in the third last position$\}$.
- $\{w : : \ w$ is a multiple of 2 or a multiple of 3$\}$.
- Union and intersection of two DFAs as an NFA
- Some other examples

Equivalence of NFA and DFA

Theorem

Every non-deterministic finite automaton has an equivalent (accepting the same language) deterministic finite automaton.

Proof.

- For the sake of simplicity assume NFA is ε-free.
- Design a DFA that simulates a given NFA.
- Note that NFA can be in a number of states at any given time
- How are the states of the corresponding DFA?
- Define initial state and accepting states
- Define the transition function

$\square$

Equivalence of NFA and DFA

Theorem

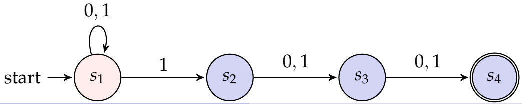
Every non-deterministic finite automaton has an equivalent (accepting the same language) deterministic finite automaton.

Proof.

- For the sake of simplicity assume NFA is ε-free.
- Design a DFA that simulates a given NFA.
- Note that NFA can be in a number of states at any given time
- How are the states of the corresponding DFA?
- Define initial state and accepting states
- Define the transition function

□

Determinize the following automaton:



Extension

Exercise: Extend the previous construction in the presence of ε-transitions.
Hint: ε-closure of a set of states.

Closure under Regular Operations

Theorem

The class of regular languages is closed under union, concatenation, and Kleene closure.

Proof.

- We have already seen the closure under union as a DFA and as an NFA.
- Concatenation and Kleene closure can easily be defined as an NFA using ε-transitions.
- The theorem follows from the equivalence of NFA and DFA.

□