Theory of Computation

Office hours: Monday 14:00-16:00
Office: 216

Curriculum

- Finite Automata
- Regular expressions
  - Regular languages
  - Regular grammars
  - Closure
  - Pigeonhole principle
  - Pumping lemma
- Context Free Languages
  - Parsing and ambiguity
  - Parse Trees
  - Pushdown automata
  - Pumping lemma for CFGs
- Turing Machines
- Curch-Turing Thesis, Halting Problem, Unsolvable Problems
- Computational Complexity: P-class, NP-class, Cooks Theorem

Things you should not care about but I know you do

- Participation
  - Obligatory (75%)
  - Counts in your evaluation
- Grading

  40% Midterm

  45% Final

  15% Exercises and Project

Things you should care about

- No video
- No photos
- The slides will be available in moodle.
- Take *handwritten* notes
- Plenty of online resources on the subject
- Pseudocode means Python.
  - It is the easiest pseudocode language I know of.
  - It can be executed.

Goal

Understanding Computation

Goal

Understanding Computation

- Fundamental capabilities of computers.
- Fundamental limitations of computers.

Goal

Understanding Computation

- Fundamental capabilities of computers.
- Fundamental limitations of computers.

There are different ways to approach the question:

- automata
- computability
- complexity

But why? I am a practitioner!

- Design a new programming language
  - Grammars
- Pattern matching
  - Finite automata
- Real-time/fast computation
  - Complexity theory

But why? I am a practitioner!

- Design a new programming language
  - Grammars
- Pattern matching
  - Finite automata
- Real-time/fast computation
  - Complexity theory

Technology is quickly outdated, but theory remains the same.

What makes some problems computationally hard and others easy?

Which problems are hard and which ones are easy

- easy: Sort a list of integers in ascending order
- hard: Schedule the classes of the university satisfying reasonable constraints

What makes some problems computationally hard and others easy?

- Fast answer
  - We don't really know!
- Real answer
  - We have a lot of insight! We classify problems
    - P
    - NP
    - NP-complete
    - ...

Again, why do I care?

Many applications rely on complexity theory. For example, most cryptosystems are based on the assumption that a problem is hard to solve, but easy to confirm a solution.

Again, why do I care?

Many applications rely on complexity theory. For example, most cryptosystems are based on the assumption that a problem is hard to solve, but easy to confirm a solution.
Integer Factorization:

- Factorize: 62615533
- Multiply : $7907 \cdot 7919$

What if my problem is actually hard?

- Understanding the root of difficulty
  - Simplify the problem
- If the exact solution is hard, but not really needed
  - Find an approximate solution
- Is it always hard, or just in worst case?
  - It may be easy almost always, so it is practical.
- Other models of computation
  - For example randomized algorithms

Computability Theory

Is it even possible to solve it?

- Some problems are not solvable by computers
  - Kurt Gödel, Alan Turing, Alonzo Church
- Complexity Theory
  - Easy vs Hard
- Computability Theory
  - Solvable vs Unsolvable

Theory of Automata

Mathematical models of computation

Applications of automata:

- Finite Automaton
  - text processing, compilers, hardware design
- Context-Free Grammars
  - programming languages, artificial intelligence

Automata

Strings and Languages

Definition (Alphabet)

An alphabet is a non-empty finite set.

Definition (String)

A string over an alphabet is a finite sequence of symbols from the alphabet.

Example

- $\Sigma_1 = \{0, 1\}$
  - 0101101
  - 01111110101
  - 01112110101
- $\Sigma_2 = \{a, b, c, d, f, t, o, m, n, u, x, z\}$
  - *automaton*
  - *automata*
  - *aabaabcaduzfo*
  - *merhaba*

Strings and Languages

Let Σ be an alphabet.

Definition (Length)

Given a string w over Σ, the **length** of w, denoted by $|w|$, is the number of symbols it contains.

Definition (Empty String)

The string of length zero, denoted by ε, is called the **empty string**.

Definition (Reverse)

Given a string $w = w_1 w_2 \dots w_n$ over Σ, the **reverse** of w, denoted by w^R, is the string $w_n w_{n-1} \dots w_2 w_1$.

Definition (Substring)

Given two strings $w = w_1 w_2 \dots w_n$ and $z = z_1 z_2 \dots z_m$ over Σ, z is a **substring** of w iff $w = w_1 \dots w_i z_1 z_2 \dots z_m w_{i+m+1} \dots w_n$.

Strings and Languages

Let $\Sigma = \{0, 1, 2, 3\}$ be an alphabet.

Example

- 0101101

  length 7

  reverse 1011010

  - Substrings
    - 110
    - 1011
    - 01010
- 12213

  length 5

  reverse 31221

  - Substrings
    - 1
    - 213
    - 23

Operations on Strings

Definition (Concatenation)

Given two strings $w = w_1 w_2 \ldots w_n$ and $z = z_1 z_2 \ldots z_m$ over Σ, the concatenation of w and z, denoted by wz, is the string

$$wz = w_1 \ldots w_n z_1 z_2 \ldots z_m.$$

Concatenation of w with itself k times, is denoted by w^k.

Example

- $\Sigma_1 = \{0, 1, 2\}$, $w = 010$, $z = 111$
  - $wz = 010111$
  - $z^2 = 111111$
  - $w^3 z^2 w = 010010010111111010$

Definition (Prefix)

Given two strings w and z over Σ, we say that w is a prefix of z if there exists a string x such that $wx = z$. Proper prefix if $w \neq z$.

Lexicographic ordering

Definition (lex order)

The lexicographic order of strings is the same as the familiar dictionary order.

Definition (shortlex order)

The lexicographic order of strings is the same as the familiar dictionary order.

- shorter strings precede longer strings
- strings of equal length are sorted with lex order.

Example

- $\Sigma_1 = \{0, 1\}$
  - $\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$

Language

Definition (Language)

A language is a set of strings.

Definition (Prefix-free Language)

A language is prefix-free if no member is a proper prefix of another member.

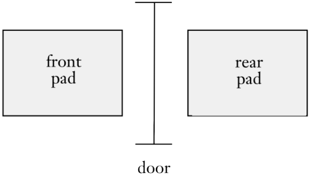
Finite State Machine

a.k.a finite automaton

Finite Automata

- Good model for computers with limited amount of memory
- Simple but useful
- In the core of electromechanical devices

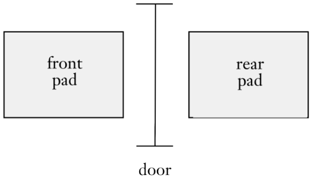
Example



- Two states: OPEN and CLOSED
- Four Input Conditions: FRONT, REAR, BOTH, NEITHER

Finite Automata

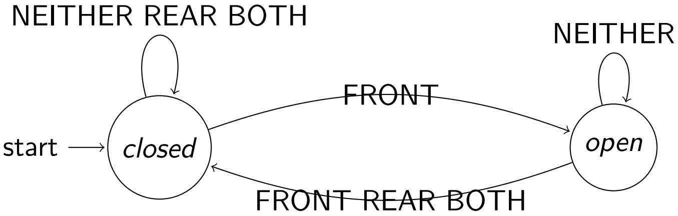
Example



- When CLOSED:
  - NEITHER or REAR or BOTH: CLOSED
  - FRONT: OPEN
- When OPEN:
  - FRONT or REAR or BOTH: OPEN
  - NEITHER: CLOSED

Finite Automata

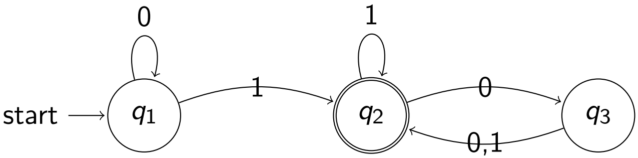
Example



- When CLOSED:
  - NEITHER or REAR or BOTH: CLOSED
  - FRONT: OPEN
- When OPEN:
  - FRONT or REAR or BOTH: OPEN
  - NEITHER: CLOSED

Finite Automata

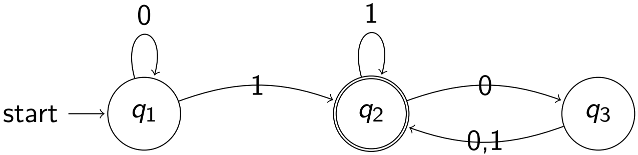
Example



- Three states $q_1.q_2.q_3$.
- q_1 is the **start state**
- q_2 is the accept state
- The arrows are called **transitions**
- The output is either **accept** or **reject**.

Finite Automata

Example



- Start with the start sate
- Receive symbols from input string
  - Traverse the link labeled with the received symbol
  - Transition to another state
- When the input string is exhausted
  - accept: if the final state is an accept state
  - reject: if the final state is not an accept state

Finite Automata

Definition

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

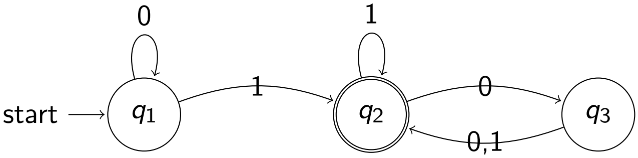
- Q is a finite set called the **states**
- Σ is a finite set called the **alphabet**
- $\delta : Q \times \Sigma \to Q$ is the **transition function**
- q_0 is the **start state**
- $F \subseteq Q$ is the **set of accept states**

Finite Automata

Definition

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

- Q is a finite set called the **states**
- Σ is a finite set called the **alphabet**
- $\delta : Q \times \Sigma \to Q$ is the **transition function**
- q_0 is the **start state**
- $F \subseteq Q$ is the **set of accept states**



Finite Automata

Definition

Given a Finite State Machine M, let A be the set of all strings that M accepts. Then we say that:

- A is the **language of machine** M, denoted by $L(M) = A$.
- M **recognizes** A.

Notes:

- An FSM may accept several strings, but always recognizes only one language.
- If an FSM does not accept any string, it recognizes the empty language $\emptyset$.

Finite Automata

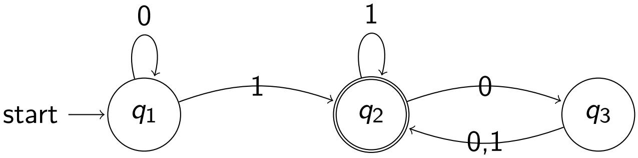
Definition

Given a Finite State Machine M, let A be the set of all strings that M accepts. Then we say that:

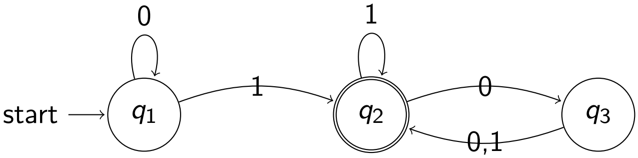
- A is the **language of machine** M, denoted by $L(M) = A$.
- M **recognizes** A.

Notes:

- An FSM may accept several strings, but always recognizes only one language.
- If an FSM does not accept any string, it recognizes the empty language $\emptyset$.



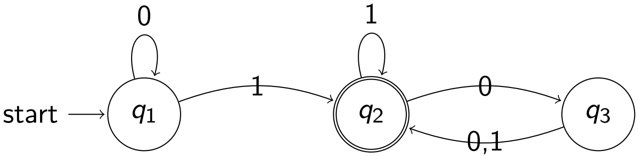
Finite Automata



Example

What is the language recognized by this machine?

Finite Automata



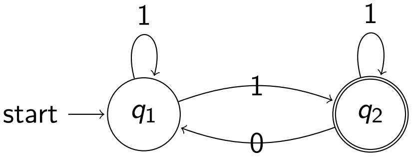
Example

What is the language recognized by this machine?

$$A = \{w : w \text{ contains at least one } 1$$

$$\text{and an even number of } 0 \text{ following the last } 1\}$$

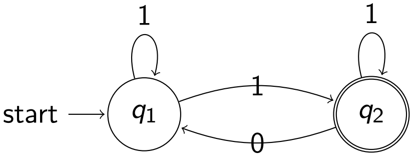
Finite Automata



Example

What is the language recognized by this machine?

Finite Automata

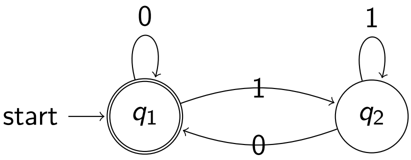


Example

What is the language recognized by this machine?

$$A = \{w : w \text{ ends with } 1\}$$

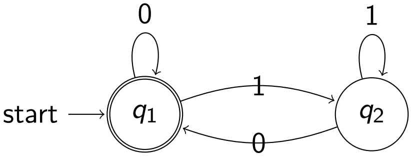
Finite Automata



Example

What is the language recognized by this machine?

Finite Automata



Example

What is the language recognized by this machine?

$$A = \{w : w \text{ ends with } 0 \text{ or } w = \varepsilon\}$$

Regular Language

Definition

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