CS304: Automata and Formal Languages

Lec 4

Introduction to Finite Automata

Rachit Nimavat

Aug 5, 2025

Outline

Alphabet, Strings, and Languages

Alphabet Σ: A finite, non-empty set of symbols

- e.g. the binary alphabet $\Sigma = \{0, 1\}$

String w: A finite sequence of symbols taken from an alphabet

- e.g. $w = 001$ is a string over Σ
- ε is an empty string with $|\varepsilon| = 0$ symbols
- $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \dots, \}$ is the set of all possible strings over Σ

Language L: A set of strings over a particular alphabet

- Any language $L \subseteq \Sigma^*$
- Languages $\equiv$ Functions

  Each language L can be thought of as an indicator function

  Each boolean function f can be thought of as a language

Operations on Languages

Recall, languages are sets. All set operations apply naturally.

Union: OR operation $L_1 \cup L_2 := \{w \mid w \in L_1 \vee w \in L_2\}$

Intersection: AND operation $L_1 \cap L_2 := \{w \mid w \in L_1 \wedge w \in L_2\}$

Concatenation: followed-by/cross-product $L_1 L_2 := \{xy \mid x \in L_1 \wedge y \in L_2\}$
e.g. $L_1 = \{a, b\}$ $L_2 = \{p, q\}$. $L_1 L_2 = \{ap, aq, bp, bq\}$

Kleene Star: 0-or-more-times $L^* = \bigcup_{i=0}^{\infty} L^i$
e.g. $L = \{a, b\}$. $L^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \ldots\}$
What is the alphabet of L^*? Σ: same as the alphabet of L

These operations allow us to build complex languages from simpler building blocks

Language = Problem

Recall **Languages** $\equiv$ **Functions**:

Theorem 1.1

Every language L over Σ uniquely corresponds to a function $f : \Sigma^* \mapsto \{0,1\}$.

How can a computer recognize if a string belongs to a language?

e.g. primality testing: **Is a number n prime or not?**
Let L_{primes} be language over $\Sigma = \{0,1\}$ of binary representations of prime numbers
Primality testing equivalent to checking binary$(n) \in L_{\text{primes}}$?

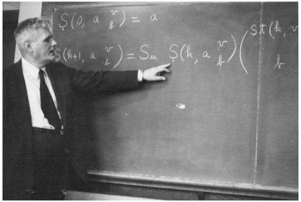
e.g. image classification: **Is a given image a picture of dog?**
An image can be represented as a finite string of pixel data.
Let Σ be the set of all possible pixel color values and let L_{dog} be language of all strings over Σ that form a recognizable image of a dog.
Image classification equivalent to checking if a given image string $w_{img} \in L_{dog}$.

The Central Question

Fix an alphabet (usually $\Sigma = \{0, 1\}$) and a language $L \subseteq \Sigma^*$
Key question in theoretical computer science: given a string $w \in \Sigma^*$, does $w \in L$ or not?

A Question Ahead of Its Time

This was the driving question for pioneers like **Alan Turing**, and **Alonzo Church** in the 1930s — years before the first computers were built



The Central Question

Fix an alphabet (usually $\Sigma = \{0, 1\}$) and a language $L \subseteq \Sigma^*$

Key question in theoretical computer science: given a string $w \in \Sigma^*$, does $w \in L$ or not?

A Question Ahead of Its Time

This was the driving question for pioneers like **Alan Turing**, and **Alonzo Church** in the 1930s — years before the first computers were built

To answer this, they

- invented formal, "idealistic" models of computation
- developed the theory and terminology for machines like the **Turing Machine** and the very **Automata** we will study, ... all before the hardware existed

A Modern Parallel: Quantum Computing

This is happening again today. We design algorithms for **Quantum Computers** even while large-scale, fault-tolerant quantum hardware remains an idealized machine of the future

Benefit of Hindsight

Question 1

Fix an alphabet Σ and a language $L \subseteq \Sigma^*$. Given a string $w \in \Sigma^*$, does $w \in L$ or not?

e.g. $\Sigma = \{0, 1\}$ $L = \{w \mid w$ is a sequence of 0s followed by a single $1\} = \{0\}^* \cdot \{1\}$.
Shorthand notation: $L = 0^*1$
$L = \{1, 01, 001, 0001, ...\}$

Question 2

How does a C program to "recognize" strings in L look like?

Benefit of Hindsight - C Code

```c
// Returns true if w is in the language 0*1
bool recognize(const char* w) {
    int state = 0; // 0: start, 1: accept, 2: fail
    int len = strlen(w);
    for (int i = 0; i < len; i++) {
        char input = w[i];
        if (state == 0) { // In start state
            if (input == '0') state = 0;
            else if (input == '1') state = 1;
            else state = 2; // Invalid char
        } else if (state == 1) { // accept state
            state = 2; // fail if unexpected char
        }
        // If state is 2 (fail), it stays 2
    }
    return (state == 1);
}
```

- The loop processes the string one character at a time
- The `state` variable is the machine's **only memory**
- The `if/else` logic represents the program's **rules** or **transitions**.

Benefit of Hindsight - Stripping the Syntax

Input - `const char* w`

Read-Only Input: The `const char* w` represents a **read-only input tape**. We can inspect the input string but not change it

One-Way Movement: The `for` loop, which processes the string from left to right, models a **one-way read head**. We read one symbol at a time and never go back.

Benefit of Hindsight - Stripping the Syntax

State variable could only be in one of three states:

- **State 0 (q_0):** "We have only seen zeros so far." $\rightarrow$ **start state**
- **State 1 (q_1):** "We have seen zero or more zeros, followed by exactly one 1." $\rightarrow$ **accepting state**
- **State 2 (q_{fail}):** "We have seen an invalid sequence (e.g., a '0' after a '1', or a second '1')." $\rightarrow$ **reject state**.

Our program is just an implementation of an abstract machine that

has a unidirectional read-only access to input

has a finite number of states

moves between these states based on predefined rules.

This abstract machine is a **Finite Automaton**!

Deterministic Finite Automata (DFA)

Formal Definition

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

‘symbol’	description	in our code
Q	finite set of states	`state` $\in \{0, 1, 2\}$
Σ	underlying finite alphabet	$\Sigma = \{$‘0’, ‘1’$\}$
δ	transition function between states	the ‘core logic’ in C code
$q_0 \in Q$	start state	`state` $= 0$
$F \subseteq Q$	accepting states	`state` $= 1$

Deterministic Finite Automata (DFA)

Visualizing DFA

State Diagram is the intuitive way we visualize and work with DFAs

Conventions:

- States Q are circles
- Start state (q_0) has an incoming arrow labeled start
- Accepting states (F) are double circles
- Transitions (δ) are arrows between states, labeled with input symbols from Σ

Our DFA for $L = 0^*1$: