COM3064 Automata Theory

Week 1: Introduction

Lecturer: Dr. Sevgi YİĞİT SERT
Spring 2023

Resources: Introduction to The Theory of Computation, M. Sipser,
Introduction to Automata Theory, Languages, and Computation, J.E. Hopcroft, R. Motwani, and J.D. Ullman
BBM401 Automata Theory and Formal Languages, İlyas Çiçekli

What is Automata Theory?

- **Automata theory** is the study of abstract ***computing devices or machines***.

- In 1930s, **Turing** studied an abstract machine (***Turing machine***) that had all the capabilities of today's computers.
  - Turing's goal was to describe precisely the boundary between what ***a computing machine could do and what it could not do.***

- In 1940s and 1950s, simpler kinds of machines (**finite automata**) were studied.
  - Chomsky began the study of formal grammars that have close relationships to abstract automata and serve today as the basis of some important software components (i.e., compilers).
  - In 1969, Cook extended Turing's study of what could and what could not be computed. and separate those problems that can be solved efficiently by computer or solved but takes so much time

Why Study Automata?

- **Automata theory** is the ***core of computer science***.
- Automata theory presents ***many useful models for software and hardware***.
  - In compilers, we use finite automata for lexical analyzers, and push down automatons for parsers.
  - In search engines, we use finite automata to determine tokens in web pages.
  - Finite automata model protocols, electronic circuits.
  - Context-free grammars are used to describe the syntax of essentially every programming language.
  - Automata theory offers many useful models for natural language processing.
- It gives us a framework in which we can ask ***what computation is***, ***what we can compute***, ***what is necessary for computation***, and ***how computations can be carried out***.
  - It provides insights how we can ***design and build*** both ***real and abstract machines***—and even programming languages!

Automata, Computability and Complexity

- **Automata**, **Computability** and **Complexity** are linked by the question:
  - ***"What are the fundamental capabilities and limitations of computers?"***
- In **complexity theory**, the objective is to classify problems *as* ***easy problems*** and ***hard problems***.
- In **computability theory**, the objective is to classify problems as **solvable problems** and non-solvable problems.
- **Automata theory** deals with the definitions and properties of mathematical models of computation.
  - Finite automata are used in text processing, compilers, and hardware design.
  - Context–free grammars are used in programming languages and artificial intelligence.
  - Turing machines represent computable functions.

Central Concepts of Automata Theory - Alphabets

- An **alphabet** is a finite, non empty set of symbols.
- We use the symbol Σ for an alphabet.

- $\Sigma = \{0,1\}$, the binary alphabet
- $\Sigma = \{a,b,c,\ldots,z\}$ the set of all lower-case letters
- The set of ASCII characters is an alphabet.

Central Concepts of Automata Theory - Strings

- A **string** is a finite sequence of symbols chosen from some alphabet.
- 01101 is a string from the alphabet $\Sigma = \{0,1\}$.

  Some other strings: 11, 010, 1,0

- The **empty string**, denoted as ε, is a string of zero occurrences of symbols.
- **Length of string:** number of symbols in the string

  $|ab| = 2 \quad |b| = 1 \quad |\varepsilon| = 0$

- String z is a **substring** of w if z appears consecutively within w.

  cad is a substring of abracadabra

Central Concepts of Automata Theory - Strings

Powers of an alphabet:

- If $\sum$ is an alphabet, the set of all strings of a certain length from that alphabet by using an exponential notation.
- $\sum^k$ is the set of strings of length k from $\sum$.
- Let $\sum = \{0,1\}$. $\sum^0 = \{\varepsilon\}$ $\sum^1 = \{0,1\}$ $\sum^2 = \{00,01,10,11\}$
- The set of all strings over an alphabet is denoted by $\sum^*$.

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

- The set of nonempty strings from the alphabet $\sum$ is $\sum^+$.

$$\sum^+ = \sum^1 \cup \sum^2 \cup \ldots$$

Concatenation of strings

- If **x** = abc and **y** = de then **xy** = abcde

Central Concepts of Automata Theory - Language

- A set of strings that are chosen from Σ^* is called as a **language**.
- If Σ is an alphabet, and **L** $\subseteq \Sigma^*$, then L is a **language** over Σ.
- A language over Σ may not include strings with all symbols of Σ.
- Some Languages:
  - The language of all strings consisting of *n* 0's followed by *n* 1's for some $n \geq 0$:
    $\{\varepsilon, 01, 0011, 000111, \ldots\}$
  - Σ^* is a language
  - Empty set is a language. The empty language is denoted by $\emptyset$.
  - The set $\{\varepsilon\}$ is a language, $\{\varepsilon\}$ is not equal to the empty language.
  - The set of all syntactically correct C programs is a language.
  - Turkish, English are languages.

Set-Formers to Define Languages

- A **set-former** is a common way to define a language

  {w | something about w}

  {w | w consists of equal number of 0's and 1's}

  {w | w is a binary integer that is prime}

- Sometimes we replace w with an expression

  $\{0^n1^n \mid n \geq 1\}$: the set of 0 to the n, 1 to the n such that n is greater than or equal to 1, this language consists of the strings {01, 0011, 000111,...}

Decision Problem

- In automata theory, a **decision problem** is the question of deciding whether a given string is a member of a particular language.
- **If Σ is an alphabet, and L is a language over Σ , then the decision problem is: Given a string w in Σ^* , decide whether or not w is in L.**
- In order to make decision requires some computational resources.
  - Deciding whether a given string is a correct C identifier
  - Deciding whether a given string is a syntactically correct C program.
- Some decision problems are simple, some others are harder.

Automata

- **Automata** are abstract mathematical devices that can
  - Determine membership in a language (set of strings)
  - Transduce strings from one set to another
- They have all the aspects of a computer
  - input and output
  - memory
  - ability to make decisions
  - transform input to output
- Automata differ in
  - the amount of memory they have (***finite*** vs ***infinite***)
  - what kind of access to the memory they allow

Automata

- Automata can behave ***deterministically*** or ***non-deterministically***
  - For a ***deterministic automaton***, there is only one possible alternative at any point, and it can only pick that one and proceed.
  - A non-deterministic automaton have multiple possible next steps at any point, and it picks one step and proceeds.
- We have different types of automata for different classes of languages.
  - **Finite State Automata** (for ***regular languages***)
  - **Pushdown Automata** (for ***context-free languages***)
  - **Turing Machines** (for ***Turing recognizable languages - recursively enumerable languages***)
    - Decision problem for Turing recognizable languages are solvable.
    - There are languages that are not Turing recognizable, and the decision problem for them is unsolvable.

Formal Proofs - Terminology

- When we study automata theory, we encounter theorems that we have to prove.
- ***Definition***: precise description of the objects and notions that we use
- ***Mathematical statem***ent: precise statement about the objects and notions that we use, typically that it has some property
  - Is either true or false
- ***Proof***: convincing argument shows that a statement is true
- ***Theorem***: statement that has been proved to be true
- ***Lemma***: true statement used to prove a theorem
- ***Corollary***: true statement that is deduced easily from a theorem

Formal Proofs

- There are different forms of proofs:
  - Proof by construction
  - Proof by Induction
  - Proof by Contradiction
- Finding proofs isn't always easy.
- A few tips for producing a proof:
  - *Be patient*: finding proofs takes time
  - *Come back to it*: Think about it a bit, and then return a few minutes or hours later.
  - *Be neat and concise*: Write your statements using simple and clear picture/text and be sure that your reasoning is understood easily.

Proof by Construction

- This proof technique consists of a sequence of statement starting from some *initial statement* (hypothesis or given statements) proceeding directly to a *conclusion statement* .

- Each step of the proof follows from a given fact or previous statements (or their combinations) by an accepted **logical principle** (known facts, laws and formulas).

- **Example**: For every graph G, the sum of the degrees of all the nodes in G is an even number.

  Proof:

  Every edge in G is connected to two nodes.
  Each edge contributes 1 to the degree of each node to which it is connected.
  Therefore, each edge contributes 2 to the sum of the degrees of all the nodes.
  If G contains *e* edges, then the sum of the degrees of all the nodes of G is *2e*, which is an even number.

Proof by Induction

- An **inductive proof** has three parts:
  - Basis : Prove that *P(1)* is true.
  - Induction Hypothesis : Assume that *P(i)* is true
  - Induction Step : For each $i \geq 1$, assume that *P(i)* is true and use this assumption to show that *P(i + 1)* is true

Proof by Induction

Theorem: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **for all n≥1**

Proof : (by induction on n)

Basis: n = 1 $\quad \sum_{i=1}^{1} i = \frac{1(1+1)}{2} \qquad 1=1$

Induction Hypothesis: Suppose that $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$ for some k ≥ 1.

Induction Step : We have to show that $\sum_{i=1}^{k+1} i = \frac{(k+1)(k+2)}{2}$

$$\sum_{i=1}^{k+1} i = \sum_{i=1}^{k} i + (k+1)$$

$$= \frac{k(k+1)}{2} + (k+1) \quad \text{by the inductive hypothesis}$$

$$= \frac{k(k+1) + 2(k+1)}{2} \quad = \frac{(k+1)(k+2)}{2}$$

It follows that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ for all n ≥ 1. □

Proof by Contradiction

- We assume that the theorem is false and then show that this assumption leads to an obviously false consequence, called a contradiction.

- **Example:** √2 is irrational.

Proof: A rational number is the *ratio* of integers m and n, e.g. 2/3

First, we assume that √2 is rational.

$$\sqrt{2} = \frac{m}{n}, \text{ m and n are integers.}$$

We multiply both sides of the equation by n and obtain

$$n\sqrt{2} = m$$

We square both sides and obtain

$$2n^2 = m^2$$

$$m = 2k$$

Proof by Contradiction

- We assume that the theorem is false and then show that this assumption leads to an obviously false consequence, called a contradiction.

- **Example:** √2 is irrational.

Proof:

$$2n^2 = (2k)^2$$
$$2n^2 = 4k^2$$

Dividing both sides by 2, we obtain

$$n^2 = 2k^2$$

This result shows that n^2 is even and hence that n is even.
Thus we have established that both m and n are even.
But we had earlier reduced m and n so that they were *not* both even—a contradiction.