CS303T Theory of Computation

R. Kabaleeshwaran

August 9, 2022

Outline

Course Logistics

- Minors - 15 Marks (each)
- Final Exam - 50 Marks
- Assignments/Presentation - 10 Marks
- Quiz/Scribe/Interaction/Surprise Test - 10 Marks
- Total - 100 Marks

Introduction

- Study of abstract computing devices
- 1930 *(Alan Turing)* An abstract machine (Turing machines) that had all the capabilities of today's computers
  - Boundary of a computing machine
  - What a computing machine could do and what it could not do
- 1940-1950 Finite Automata - originally proposed to model brain functions
- 1950 *(Noam Chomsky)* - study of formal grammars
- 1969 *(Stephen Cook)* extended Turing's study of what could and what could not be computed
  - Tractable - problems that can be solved efficiently by a computer
  - Intractable or NP-hard

  "**The theoretical developments bear directly on what computer scientists do today**"

Why to Study Automata Theory

- Finite automata are a useful model for many important kinds of hardware and software
  - Software for designing and checking the behavior of digital circuits
  - The first phase of a typical compiler is “lexical analyzer” - which breaks the input text into logical units (such as keywords, identifiers, punctuation)
  - Test processing software
  - (Secure) Communications protocols
- Automata are essential for the study of the limits of computation. Two important issues
  - What can a computer do at all?
    - This study is called decidability.
    - The problem that can be solved by computer are called Decidable.
  - What can a computer do efficiently?
    - This study is called intractability
    - The problem that can be solved by computer using ‘slowly growing function’ are called tractable.

Course Overview

- Deterministing Finite Automata (DFA)
- Non-deterministing Finite Automata (NFA)
- Regular Expression (RE)
- Regular Grammer (RG)
- Context free Grammer (CFG)
- Pushdown Automata (PDA)
- Turing Machine (TM)
- (Un)Decidable Problems

Introduction to Automata Theory

- An alphabet is a finite, nonempty set of symbols. Conventional symbol Σ
  - Ex: $\Sigma = \{0, 1\}$, $\Sigma = \{a, b, \ldots, z\}$
- A string (or sometimes word) is a finite sequence of symbols chosen from some alphabet
  - Ex: 0101100 is a string from Σ^*, where $\Sigma = \{0, 1\}$
- Length of a string is the number of symbols in the string
  - $|010101| = 6$
- Empty string - ϵ (the string with zero occurrences of symbols)
- Σ^k - set of strings of length k, each symbols is from Σ
  - $\Sigma = \{0, 1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$, and so on
  - $\Sigma^0 = \{\epsilon\}$ and $|\epsilon| = 0$
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots$

Concatenation

- Concatenation of x and y is denoted as xy, where x, y are strings
  - Ex: $x = 011$ and $y = 010$, then $xy = 011010$ and $yx = 010011$
- ϵ is the identity for concatenation
  - For any string w, then $\epsilon w = w\epsilon = w$ hold

Language

- A set of strings all of which are chosen from some Σ^* is called a Language
- If Σ is an alphabet and $L \subseteq \Sigma^*$, then L is a language over Σ
- Observation:
  - Notice that a language over Σ need not include strings with all the symbols of Σ
  - Languages can be viewed as sets of strings
- Σ^* is a language for any alphabet Σ
- $\emptyset$ - Empty Language over any alphabet Σ (No string)
- $\{\epsilon\}$ - a language containing an empty string over any alphabet Σ
- Observation: $\emptyset \neq \{\epsilon\}$

Examples

- $L_1 = \{w | w \text{ consists of an equal number of 0's and 1's}\}$
  - $0110 \in L_1$, $01101 \notin L_1$
- $L_2 = \{w | w \text{ is a binary integer that is prime}\}$
  - $011 \in L_2$ (as $011_2 = 3$ a prime),
  - $1100 \notin L_2$ (as $1100_2 = 12$ not a prime)
- $L_3 = \{w | w \text{ is a syntactically correct C program}\}$
- $L_4 = \{0^n 1^n | n \geq 0\}$
  - $0011 \in L_4$, $011 \notin L_4$
- $L_5 = \{0^m 1^n 0^m | n \geq 0, m \geq 1\}$
  - $00100 \in L_5$, $0100 \notin L_5$