CS372
FORMAL LANGUAGES & THE THEORY OF COMPUTATION

Dr.Nguyen Thi Thu Huong
Phone: +84 24 38696121, Mobi: +84 903253796
Email: huongnt@soict.hust.edu.vn,
huong.nguyenthithu@hust.edu.vn

Description

- Computational models and how they relate to each other.
- Contents:
  - Finite automata
  - Regular expressions and languages
  - context-free grammars and languages
  - Turing machines
  - Un-decidable and intractable problems

Course Objectives

- Introduction to the formal language theory
- Understanding computation models and their usage in the design and construction of important kinds of software
- Skills of using formal proofs

What is the course about

- *Formal Languages*
- *Automata*
- *Computability*
- *Complexity*

Textbooks

- Michael Sipser. *Introduction to the Theory of Computation* (3rd edition). Required.
- Thomas A. Sudkamp. *Languages and Machines*. (3rd edition). Optinal.
- Savage, J. E. *Models of Computation. Exploring the Power of Computing*. Optional.

Evaluation

Attendance	10%
Homework assignments	25%
Presentation	15%
Midterm	20%
Final exam	30%

Assignment of Grades

A	90 - 100
B	80 - 89
C	70 – 79
D	60 – 69
F	59 and below

Unit 1
Introduction

Introduction

- Formal Languages
- Automata
- Computability
- Complexity
- Mathematical preliminaries
- String and languages
- Types of proof

What is theory of computation

- Computation $\rightarrow$ Writing programs $\rightarrow$ Running programs
- Program: algorithm expressed by a programming language
- Algorithm: a recipe for carrying out input to output transformation
- Every algorithm computes a function

  $f: D \text{ (input)} \rightarrow R \text{ (output)}$

What is theory of computation

- Can we use computers to compute all the function? Are all functions algorithm?
- Basic goal: Identify the class of functions which admit an algorithm to compute them.
- Solve set membership problem:

  Given a set S and a. a $\in$S?

membership problem for

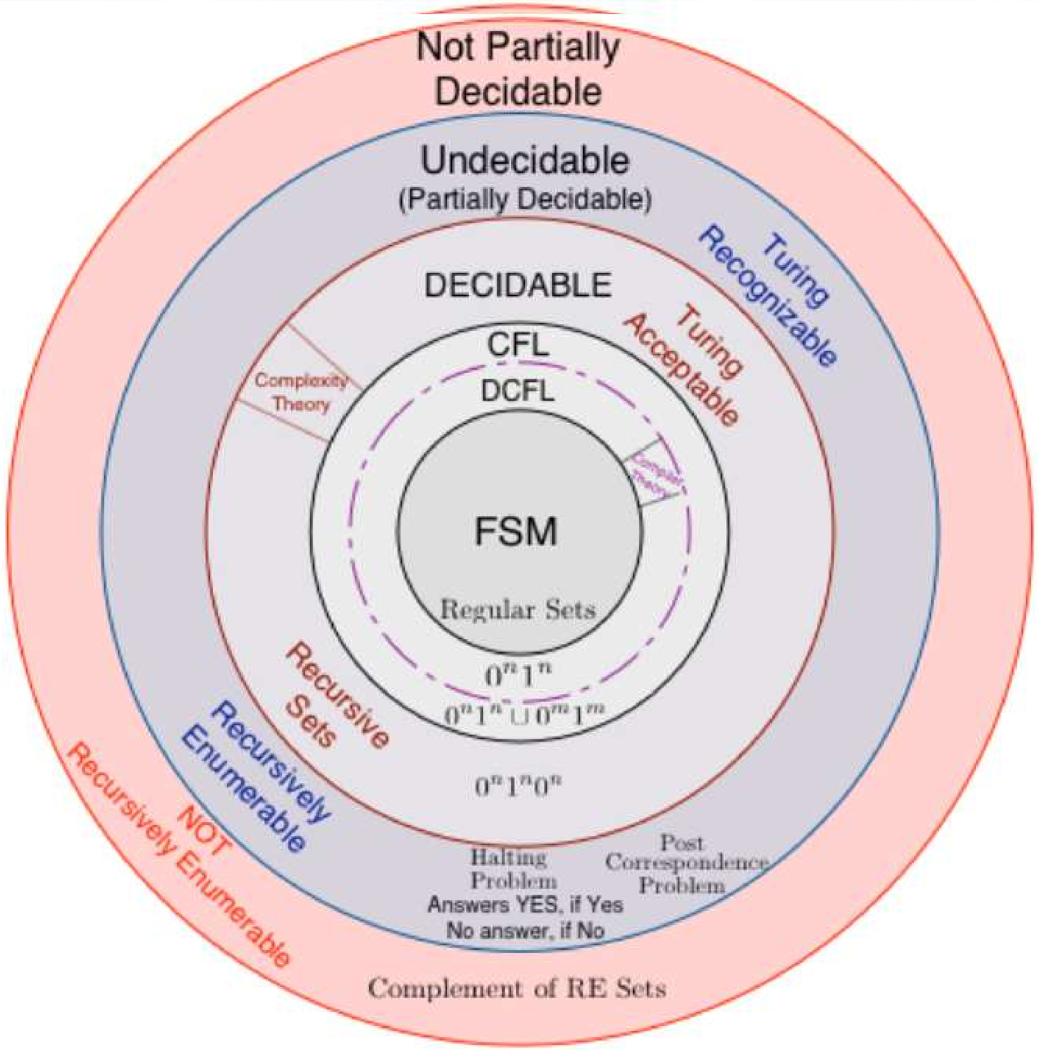
  graph(f) = {(a,b) | f(a) = b}

- No algorithm to solve the set membership problem for graph(f) $\rightarrow$ no algorithm to compute function f.

What is theory of computation

- Graph (f) : Set of finite strings
- Concepts of symbol, string, languages
- Formal Language

Containment hierarchy of classes of formal languages



Formal Languages

- An abstraction of the notion of a “problem”
- Problems are call computational processes can be reduced to one of Determining membership in a set (of strings)
- Formalize the concept of mechanical computation
- Define of the term “algorithm”
- Characterize problems that are or are not suitable for mechanical computation.
-

Formal Languages

- Finite State Automata.
- Regular Expressions and Regular Grammars.
- Context-free Grammars and Pushdown Automata.

Automata

- Automata (singular Automaton) are abstract mathematical devices that can
  - Determine membership in a 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
- Memory is crucial:
  - Finite Memory
  - Infinite Memory
    - Limited Access
    - Unlimited Access

Automata

- There are different types of automata for different classes of languages.
- They differ in
  - the amount of memory then have (finite vs infinite)
  - what kind of access to the memory they allow.
- Automata can behave non-deterministically
- A non-deterministic automaton can at any point, among possible next steps, pick one step and proceed
- This gives the conceptual illusion of (infinitely) parallel computation for some classes of automata. All branches of a computation proceed in parallel (sort of)

Computability

- What is computational power?
- What does computational power depend on?
- What does it mean for a problem to be computable ?
- Are there any uncomputable functions or unsolvable problems?

Computability

- Computation models: Turing machines (TM).
- Turing-decidable and Turing-recognizable languages.
- Enhancements of TMs: multi-tape TMs, non-deterministic TMs. Equivalence of these and the standard TM.
- Diagonalization. Acceptance problem is undecidable; Acceptance problem is recognizable; the complement of the Acceptance problem is unrecognizable.
- Reductions. Examples of other undecidable languages. Rice's theorem. Post's Correspondence Problem (PCP) is undecidable.

Complexity

- Running time of Turing Machines. The classes P, NP, NP-hard, and NP-complete.
- Cook-Levin Theorem. SAT is NP-complete. Some reductions.

Applications

- Pattern matching
- Design and Verification
- Parsing Languages
- Natural language processing
- Algorithm design and analysis

Mathematical Preliminaries

- Sets
- Sequences and Tuples
- Functions and Relations
- Strings and Languages
- Boolean Logic

Strings and Languages

- Alphabets
- Strings
- Languages
- Operations on Languages

Alphabet

Symbol

A physical entity that we shall not formally define; we shall rely on intuition.

Alphabet

A finite, non-empty set of symbols

- We often use the symbol ∑ (sigma) to denote an alphabet
- Examples of alphabet
  - Binary: ∑ = {0,1}
  - All lower case letters: ∑ = {a,b,c,..z}
  - Alphanumeric: ∑ = {a-z, A-Z, 0-9}
  - DNA molecule letters: ∑ = {a,c,g,t}(guanine, adenine, thymine, and cytosine)
  - C character set

C character set

Types	Character Set
Lowercase Letters	a –z
Uppercase Letters	A - Z
Digits	0-9
Special Characters	~! # $% ^ & *( )_ +| \' - = { } [] : ” ; < > ? , . /
White Spaces	Tab Or New line Or Space

Strings

String (sentence)

A finite sequence of symbols chosen from some alphabet

- Empty string is ε
- Length of a string *w,* denoted by |*w*|, is equal to the number of symbols in the string
  - *E.g., x = 010100* $\quad$ *|x| = 6*
  - *x = 01 ε 0 ε 1 ε 00 ε* $\quad$ *|x| = ?*

String Operations

- xy = concatenation of two strings x and y
- x^R= reversion of x

Powers of an Alphabet

If Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using the exponential notation:

$\sum^k$ = the set of all strings of length k

Examples:

- $\Sigma^0 = \{\varepsilon\}$,regardless of what alphabet Σ is. That is the only string of length 0
- If $\Sigma = \{0,1\}$, then:

  $\Sigma^1 = \{0,1\}$

  $\Sigma^2 = \{00, 01,10,11\}$

  $\Sigma^3 = \{000, 001, 010, 011, 100,101,110,111\}$

Star

- $\sum *$: The set of all strings over alphabet $\sum$
  - $\sum^* = \sum^0 \cup \sum^1 \cup \sum^2 \cup \ldots$

- The symbol $*$ is called Kleene star and is named after the mathematician and logician Stephen Cole Kleene.

- $\sum^+ = \sum^1 \cup \sum^2 \cup \sum^3 \cup \ldots$
- Thus $\sum^* = \sum^+ \cup \{\varepsilon\}$, $\sum^+ = \sum^* - \{\varepsilon\}$

Languages

L is a said to be a language over alphabet Σ, only if $L \subseteq \Sigma^*$

➔ this is because Σ^* is the set of all strings (of all possible length including 0) over the given alphabet Σ

Examples:

1. Let L_1 be *the* language of all strings consisting of *n* 0's followed by *n* 1's:

   $L_1 = \{\varepsilon, 01, 0011, 000111, \ldots\}$

2. Let L_2 be *the* language of all strings with equal number of 0's and 1's:

   $L_2 = \{\varepsilon, 01, 10, 0011, 1100, 0101, 1010, 1001, \ldots\}$

Definition: Ø denotes the Empty language

NO

- Let $L = \{\varepsilon\}$; Is $L = Ø$?

Examples of Languages

Some examples of languages:

- The set of all words over {a, b},
- The set { $a^n \mid n$ is a prime number },
- Programming language Pascal: the set of syntactically correct programs in Pascal
- The set of inputs upon which a certain Turing machine halts.

Operations on Languages

Several operations can be used to produce new languages from given ones. Suppose L_1 and L_2 are languages over some common alphabet.

- The *concatenation* L_1L_2 consists of all strings of the form *vw* where *v* is a string from L_1 and *w* is a string from L_2.
- The *intersection* of L_1 and L_2 consists of all strings which are contained in L_1 and also in L_2.
- The *union* of L_1 and L_2 consists of all strings which are contained in $L1$ or in L_2.
- The *complement* of the language L_1 consists of all strings over the alphabet which are not contained in L_1.
- The *star(Kleene star)* L_1^* consists of all strings which can be written in the form $w_1w_2...w_n$ with strings wi in L_1 and $n \geq 0$. Note that this includes the empty string ε because $n = 0$ is allowed
- The *reverse* L_1^R contains the reversed versions of all the strings in L_1.

Sets of Languages

- The power set of Σ^* , the set of all its subsets, is denoted as $P(\Sigma^*)$

Describing Languages

- Interesting languages are infinite
- Computers cannot work with infinite object.
- We need finite descriptions of infinite sets
- $L = \{a^n b^n \ : n \geq 0\}$ is fine but not terribly useful!
- We need to be able to use these descriptions in mechanizable procedures

Chomsky's Hierarchy

- Type-0 languages (recursive enumerable)

  recognized by a Turing machine

  generated by a type-0 grammar
- Type-1 languages (context-sensitive)

  recognized by a linear bounded automaton

  generated by a context sensitive grammar
- Type-2 languages (context-free)

  recognized by a non-deterministic pushdown automaton

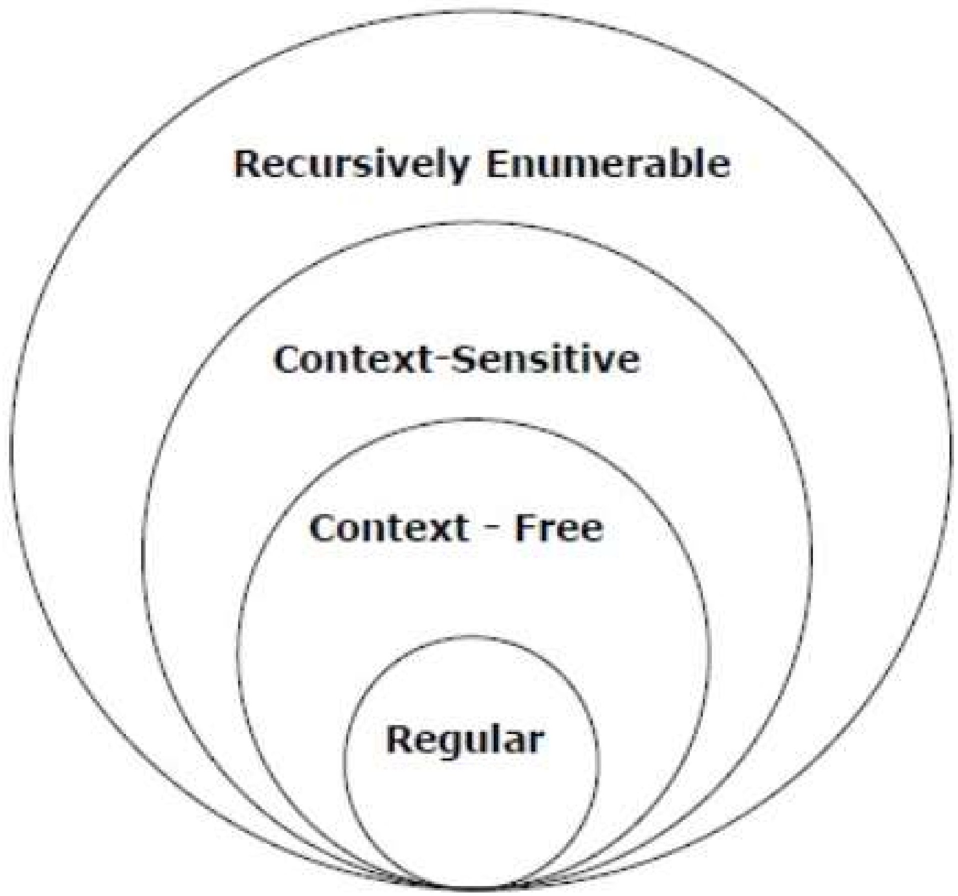
  generated by a context free grammar
- Type-3 languages (regular)

  denoted by a regular expression,

  generated by a regular grammar

  recognized by a finite state automaton

Chomsky's Hierarchy



Types of proof

- Definition, theorems and proof
- Types of proof
- Proof by induction

Definition, theorems and proof

- Definitions describe the objects and notions that we use
- A proof is a convincing logical argument that a statement is true
- A theorem is a mathematical statement proved true

Types of proof

- Proof by construction
- Proof by contradiction
- Proof by induction

Proof by induction

- Consider infinite set of the natural numbers, N = {1,2,3, ...}, and the predicate P.
- Our goal is to prove that P(n) is true for each natural number $n \geq k$.
- Proof by induction consists of two parts, *the basis* and *the induction step*
- Basis: Prove that P(k) is true.
- For each $n \geq k$, assume that P(k),...P(n) are true and use this assumption (*Induction hypothesis*) to show thatP(n + 1) is true.

Example of proof by induction

- Assume $S_1(n) = 0 + 1 + ... + n$. For all $n \geq 0$, prove that $S_1(n) = n(n + 1)/2$.

Proof

- Predicate P on n, P(n), is True if $S_1(n) = n(n + 1)/2$ and False otherwise.
- BASIS STEP: Clearly, $S_1(0) = 0$
- INDUCTION HYPOTHESIS: $S_1(k) = k(k + 1)/2$ for $k = 0,1,2,...,n$.
- INDUCTION STEP:
  - By the definition of the sum for S_1 $S_1(n+1) = S_1(n)+ n + 1$.
  - It follows that $S_1(n + 1) = n(n + 1)/2 + n + 1$.
  - Factoring out n + 1 and rewriting the expression, we have $S_1(n + 1) = (n + 1)((n + 1) + 1)/2$, exactly the desired form. Thus, the statement of the theorem follows for all values of n.