Introduction and Overview

Yo-Sub Han
CS, Yonsei University

Overview of Unit

① Starters

② Motivation for course

③ Overview of course

④ Warming up for the main course

Starters

- Instructor: Dr. Yo-Sub Han, EngA 533
  - Email: emmous@yonsei.ac.kr
  - Course URL: http://yscec.yonsei.ac.kr
  - Office hours: Tue. 2pm–4pm or appointed by email
- Teaching assistants:
  - Marco Cognetta, Yu-Jin Nam (남유진)
  - Email: {cognetta.marco , yjntrg}@gmail.com
  - Office hours: Tue. 5pm–6pm
  - Office: EngA 532

Class Info.

- This course provides formal language and automata theory. We study the fundamental knowledge on computation and computability. In particular, we examine finite-state automata (regular languages), pushdown automata (context-free languages) and Turing machines (unrestricted languages)
- Time: Tue. 11:00am–11:50am and Thu. 10:00pm–11:50am

Texts

- One required text: *Introduction to Automata Theory, Languages, and Computation* (3rd edition) by John E. Hopcroft, Rajeev Motwani and Jeffery D. Ullman
- I am using this text because it tries to give an overview of the material that we cover. I will also provide lecture notes (not 100%) based on the text
- You can get useful information from one of the authors' homepage at `http://infolab.stanford.edu/~llman/ialc.html`

Philosophy

- Lectures provide a first pass at course material. They provide preliminary understanding and knowledge
- Readings extend and reinforce lecture material
- Homeworks provide training in course material. You learn by doing
- Examinations and quizes test your personal knowledge of the covered material

Evaluation

- There will be two examinations:
  - 1st exam: 20%
  - 2nd exam: 40% (or 60%)
- Pop quiz, Homework: 30%, top 6 (out of 8) $\times$ 5%
- Class participation including discussion and attendence: 10%
- If you miss more than 8 times, then your grade is F
  - The number of missing classes directly affects your grade
- If you cheat on quiz or exam, then both you and the person who helped you will be given at most zero

Classroom Etiquette (Professionalism)

- No mobile phones—switch off or set vibration mode in classroom. SMS is NOT allowed!
- No notebook/smartpad; They are distractions.
- You should not talk during lectures: After one request, you will be asked to leave the lecture
- But, you should ask questions—alone or in pairs
- You should interact—alone or in pairs

Course Schedule

- 1st: Introduction and background
- 2nd – 5th: Regular languages
- 6th – 10th: Context-free languages
- 11th – 14th: Turing machines and related topics
- 15th: Self-study
- 16th: Final exam

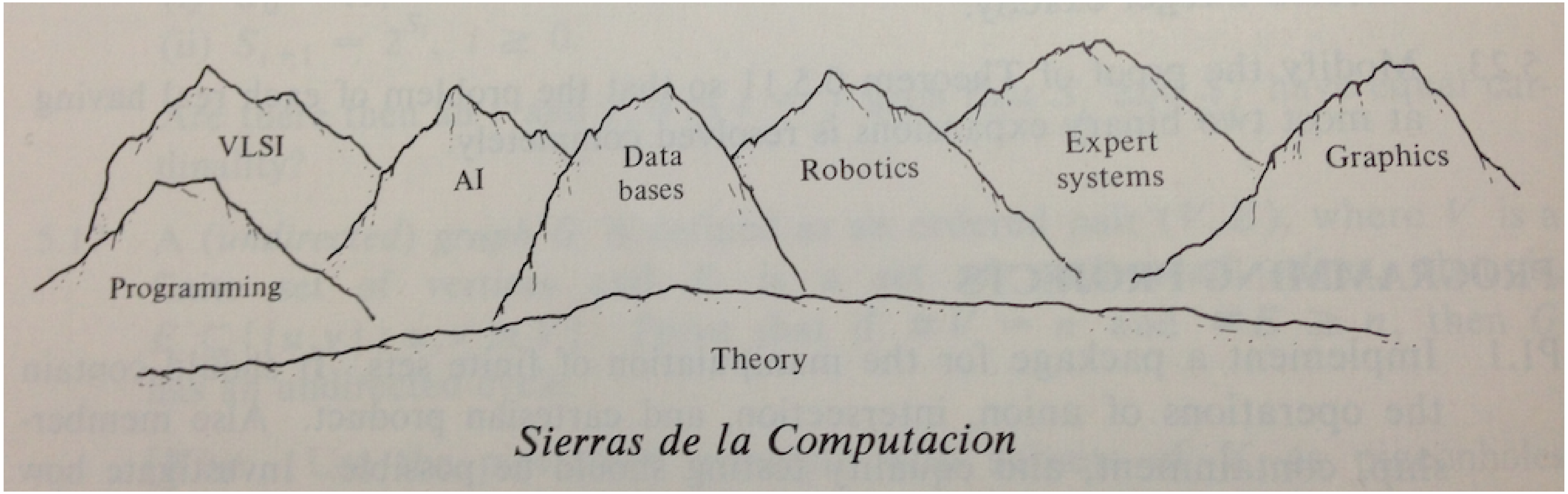
Motivation and Overview

What do we learn from this course, automata and formal languages: theory of computation.

> *Mathematical study of computing machines, their fundamental capabilities and their limitations*

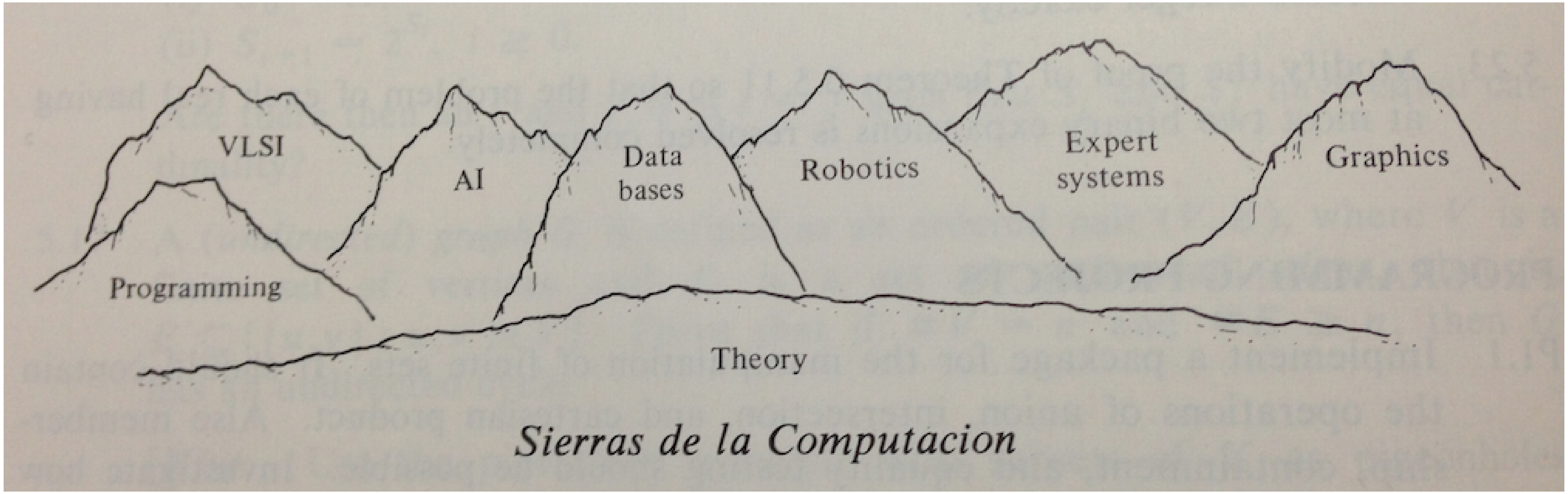
- **Q1:** What problems are solvable, in PRINCIPLE, by computer and what problems are not? Note that we assume there are unsolvable problems! Is this assumption valid? Decidability and computability theory answer these questions; We will focus on them in CSI3109
- **Q2:** What problems are solvable, in PRACTICE, by computer and what problems are not? Complexity theory (NP-hardness) answers these questions; See *Algorithm Design* course

Who Cares for Theory?



- Softwares: Dangerous areas
  - Power plants
  - Banking systems
  - Cars
  - and so on...

Who Cares for Theory?



We need a guide or guidebook on our journey throguh the Sierras de la Computation.

- Early warning system of dangerous areas

  1. Recognize new dangerous problems
  2. Derive the known knowledge beyond the partial knowledge
  3. Develop a feel for what kind of problems are dangerous

This course

- gives you feel and fluency with some of the fundamental abstract models
- provides their properties
- develops your ability with proof techniques

Theory, action as an early warning system, provides a science of the impossible — what shouldn't be attempted becasue it cannot be done!

- Warning
- Guidelines

Theory of Computation

- the impossible problems
- the possible-with-unlimited-resources-but-impossible-with-limited-resources problems
- the possible-with-limited-resources problems

When the resource is *time*,

- the undecidable
- the intractable
- the tractable

Theory of Computation: Examples

- Compilers are standard system software that translate programs written in high-level languages (C, JAVA, C#) into programs written in assembly language or machine code.

  A compiler can detect syntax errors in the programs that we write. Can we write a compiler that will detect *"infinite loops"*?

  No! The general problem of whether or not a program terminates under all inputs is UNSOLVABLE. We will see why in this course

- The programs that we usually write compute functions.

  Given two programs, determine whether or not they compute the same function. This problem is also UNSOLVABLE

Alphabets, Strings and Languages

An alphabet is a finite, nonempty set of symbols denoted by Σ. Common alphabets include:

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

Alphabets, Strings and Languages

- A string (or sometimes *word*) is a finite sequence of symbols chosen from some alphabet. e.g., 01101, 111 over $\Sigma = \{0, 1\}$
- The empty string is the string with zero occurrences of symbols. This string, denoted by λ (in text ϵ), is a string that may be chosen from any alphabet
- The length of a string is the number of symbol occurrences. e.g., 01101 has lengh 5. The standard notion for the length of a string w is $|w|$. e.g., $|011| = 3$ and $|\lambda| = 0$
- The occurrence $|w|_\sigma$ of σ is the number of σ occurrences. e.g., $|01101|_0 = 2$
- The power Σ^k of an alphabet is the set of strings of length k, each of whose symbols is chosen from Σ. $\Sigma^0 = \{\lambda\}$ for any Σ. If $\Sigma = \{0, 1\}$,

  $\Sigma^1 = \{0, 1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$, $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
- The set of all strings over Σ is denoted by Σ^* (Kleene star or Kleene/star closure).

  e.g., $\{0, 1\}^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, \ldots\}$.

  In other words, $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$

Alphabets, Strings and Languages

- The set of nonempty strings from Σ is denoted by Σ^+. Thus,
  - $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$.
  - $\Sigma^* = \Sigma^+ \cup \{\lambda\}$.
- Given two strings x and y, $x \cdot y$ denotes the (con)catenation of x and y; the string formed by making a copy of x followed by y. (We often omit the catenation operation symbol $\cdot$.)

  e.g., if $x = 01101$ and $y = 110$, then $xy = 01101110$ and $yx = 11001101$
- λ is identity for catenation since for any string w, $\lambda w = w\lambda = w$

Alphabets, Strings and Languages

- A language is a set of strings all of which are chosen from some Σ^*. If Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a *language over* Σ.

  e.g., C (the set of compilable C programs), Korean or English

  L may be *infinite* but there is some finite set of symbols of which all its strings are composed.

- Language examples
  - The set of all binary strings consisting of some number of 0's followed by an equal number of 1's; that is, $\{\lambda, 01, 0011, 000111, \cdots\}$
  - The set of all binary strings with an equal number of 0's and 1's: $\{\lambda, 01, 10, 0011, 0101, 1001, \ldots\}$
  - The set of binary numbers whose valuse is a prime: $\{10, 11, 101, 111, 1011, \ldots\}$
  - Σ^* is a langauge for any alphabet Σ
  - $\emptyset$ is the empty language over any alphabet
  - $\{\lambda\}$, the language consisting of only the empty string, is also a language over any alphabet. Note that $\emptyset \neq \{\lambda\}$; the former has no strings but the latter has one string.

Sets (warming-up)

- A set is a collection of elements.
  $L = \{a, b, c, d\}$ is a set of four elements. Note that the four elements are distinct
- Let L be a set. $z \in L$ denotes that z is in L; and $z \notin L$ denotes that z is not in L
- The empty set $\emptyset$ contains zero elements. All other sets are nonempty
- A set is finite if it has a finite number of elements. Otherwise, the set is infinite
- A is a subset of B (written as $A \subseteq B$), if every element of A is an element of B
- Two sets A and B are equal (written as $A = B$) if and only if $A \subseteq B$ and $B \subseteq A$
- If $A \subseteq B$ and $A \neq B$, then A is a proper subset of B, written as $A \subset B$. (By this definition, $\emptyset$ is a proper subset of every nonempty set.)

Sets Operations (warming-up)

Let A and B be two sets:

- Intersection $A \cap B$: It is the set $\{x \mid x \in A \text{ and } x \in B\}$ of all elements that are both in A and B. If $A \cap B = \emptyset$, then A and B are disjoint
- Union $A \cup B$: It is the set $\{x \mid x \in A \text{ or } x \in B\}$ of all elements that are in A or in B. Note that "or" is inclusive
- Difference $A - B$ or $A \setminus B$: It is the set $\{x \mid x \in A \text{ and } x \notin B\}$ of all elements that are in A but are not in B
- Power set 2^A: The set of all subsets of a set A is the power set of A. e.g.: $2^{\{a,b\}} = \{\emptyset, \{a\}, \{b\}, \{a,b\}\}$.
- Partition: A partition of A is any set $\{A_1, A_2, \ldots\}$ of nonempty subsets of A such that
  1. $A = A_1 \cup A_2 \cup \cdots$ and
  2. $A_i \cap A_j = \emptyset$ for all $i \neq j$ (mutual disjointness)

Cartesian Product and Functions (warming-up)

- The Cartesian product $A \times B$ of two sets A and B is the set of all possible ordered pairs (a, b) with $a \in A$ and $b \in B$. e.g.: $\{1, 2\} \times \{3, 4\} = \{(1, 3), (1, 4), (2, 3), (2, 4)\}$.

- A function from A to B (a binary function), written as $f : A \rightarrow B$, is a binary relation R on A and B such that, for each $a \in A$, if $(a, b) \in R$ and $(a, c) \in R$, then $b = c$ and, for each $a \in A$, there is either exactly one $b \in B$ such that $f(a) = b$ or there is no $b \in B$ such that $f(a) = b$. Such a fuction is said to be a partial function.

  A function f is total if, for every $a \in A$, there is a $b \in B$ such that $f(a) = b$. Thus, every total function is partial but the converse does not hold.

Cartesian Product and Functions (warming-up)

For example, letting $f = R = \{(1,3),(2,4)\}$. Then, $f(1) = 3$ and $f(2) = 4$.

Bijections:

- $f : A \to B$ is one-to-one if, for any distinct $a, a' \in A$, $f(a) \neq f(a')$.
- $f : A \to B$ is onto if, for all $b \in B$, there is some $a \in A$ such that $f(a) = b$.
- A total function f is bijectivie if it is both one-to-one and onto.
- Example: Let $A = \{1,2\}$ and $B = \{3,4\}$. $f_1 = \{(1,3),(2,4)\}$ is a bijection and $f_2 = \{(1,4),(2,4)\}$ is neither one-to-one nor onto.

Inductive Proofs (warming-up)

Prove a statement $S(X)$ about a family X of objects (e.g., integers, trees) in two parts:

1. *Basis:* Prove for one or several small values of X directly
2. *Inductive step:* Assume $S(Y)$ for Y "smaller than" X; prove $S(X)$ using the assumption

Statement: A binary tree with n leaves has $2n-1$ nodes.

- Formally, $S(T)$: if T is a binary tree with n leaves, then, T has $2n-1$ nodes
- Induction is on the size $=$ number of nodes of T

Basis: If T has 1 leaf, it is a one-node tree. $1 = 2 \times 1 - 1$.
Induction: Assume that $S(U)$ for trees with fewer nodes than T.
T must be a root plus two subtres U and V. If U and V have u and v leaves, respectively, and T has t leaves, then $u + v = t$. By the inductive hypothesis, U and V has $2u-1$ and $2v-1$ nodes, respectively. Then T has $1 + (2u-1) + (2v-1) = 2(u+v) - 1 = 2t - 1$ nodes.

If-And-Only-If Proofs (warming-up)

A statement is often written as "X if and only if Y". We are then required to do two things:

1. Prove the if-part: Assume Y and prove X
2. Prove the only-if-part: Assume X and prove Y

Note that

- The if and only-if parts are converses of each other
- One part, say "if X, then Y", says nothing about whether Y is true when X is false
- An equivalent form to "if X, then Y" is "if not Y, then not X"; the latter is the contrapositive of the former.