Theory of Computing

Introduction and Preliminaries
(Based on [Sipser 2006, 2013])

Yu-Fang Chen

Department of Information Management
National Taiwan University

Overview

What It Is

IM NTU

- The central question:

  What are the fundamental capabilities and limitations of computers?

- Three main areas:
  - *Automata Theory*
  - *Computability Theory*
  - *Complexity Theory*

Complexity Theory

- Some problems are easy and some hard.
  For example, sorting is easy and scheduling is hard.
- The central question of complexity theory:
  What makes some problems computationally hard and others easy?
- We don't have the answer to it.
- However, researchers have found a scheme for classifying problems according to their computational difficulty.
- One practical application: cryptography/security.

Computability Theory

- Alan Turing, among other mathematicians, discovered in the 1930s that certain basic problems cannot be solved by computers.
- One example is the problem of determining whether a mathematical statement is true or false.
- Theoretical models of computers developed at that time eventually lead to the construction of actual computers.
- The theories of computability and complexity are closely related.
- *Complexity theory* seeks to classify problems as easy ones and hard ones, while in *computability theory* the classification is by whether the problem is solvable or not.

Unsolvable Problem

Example

Write a program $T(P)$ which accepts program text P as input and returns 1 if P will terminate, 0 if not.

Solution.

It cannot be done. Suppose there is such a program T. Let us consider the following program M:

1. **if** $T(M) = 1$ **then**
2.   **while true do od**
3. **else** $\{T(M) = 0\}$
4.   **exit**

What is $T(M)$? Suppose $T(M) = 1$, M terminates. Therefore $T(M) = 0$, or it would end in an infinite loop. On the other hand, suppose $T(M) = 0$, M does not terminate. Hence $T(M) = 1$ because this is the only case where M does not terminate. Both cases are contradiction. T does not exist. □

Automata Theory

- The theories of computability and complexity require a precise, formal definition of a *computer*.
- *Automata theory* deals with the definitions and properties of mathematical models of computation.
- Two basic and practically useful models:
  - *Finite-state*, or simply *finite*, *automaton*
  - *Context-free grammar* (pushdown automaton)

Mathematical Notions and Terminology

Sets

- A *set* is a group of (possibly infinite) objects; its objects are called *elements* or *members*.
- The set without any element is called the *empty* set (written $\emptyset$).
- Let A, B be sets.
  - $A \cup B$ denotes the *union* of A and B.
  - $A \cap B$ denotes the *intersection* of A and B.
  - $\overline{A}$ denotes the *complement* of A (with respect to some *universe* U).
  - $A \subseteq B$ denotes that A is a *subset* of B.
  - $A \subsetneq B$ denotes that A is a *proper subset* of B.
- The *power set* of a set A (written 2^A) is the set consisting of all subsets of A.
- If the number of occurrences matters, we use *multiset* instead.

Sets (cont.)

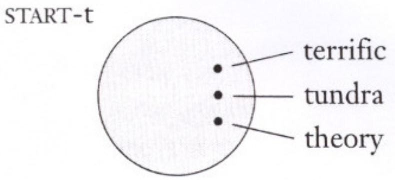


FIGURE 0.1
Venn diagram for the set of English words starting with “t”

Source: [Sipser 2006]

Sets (cont.)

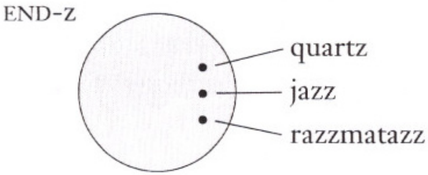


FIGURE 0.2
Venn diagram for the set of English words ending with "z"

Source: [Sipser 2006]

Sets (cont.)

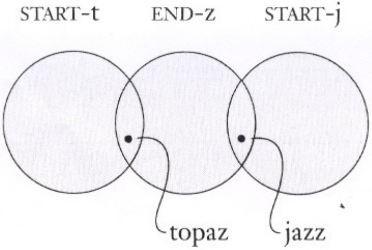


FIGURE 0.3
Overlapping circles indicate common elements

Source: [Sipser 2006]

Sets (cont.)

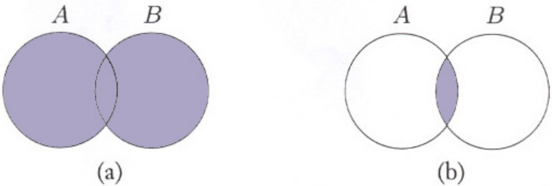


FIGURE 0.4
Diagrams for (a) $A \cup B$ and (b) $A \cap B$

Source: [Sipser 2006]

Russell's Paradox

- "The Serbian barber only shaves those who do not shave themselves."
- Consider the following set

$$A = \{x : x \notin A\}.$$

- Is $A \in A$?

Sequences and Tuples

- A *sequence* is a (possibly infinite) list of ordered objects.
- A finite sequence of k elements is also called k-*tuple*; a 2-tuple is also called a *pair*.
- The *Cartesian product* of sets A and B (written $A \times B$) is defined by

$$A \times B = \{(a, b) : a \in A \text{ and } b \in B\}.$$

- We can take Cartesian products of k sets $A_1, A_2, \ldots, A_k$

$$A_1 \times A_2 \times \cdots \times A_k = \{(a_1, a_2, \ldots, a_k) : a_i \in A_i \text{ for every } 1 \leq i \leq k\}.$$

- Define

$$A^k = \overbrace{A \times A \times \cdots \times A}^{k}.$$

Functions

- A *function* sets up an *input-output* relationship, where the same input always produces the same output.
- If f is a function whose output is b when the input is a, we write $f(a) = b$.
- A function is also called a *mapping*; if $f(a) = b$, we say that f maps a to b.

Functions (cont.)

- A *function* $f : D \to R$ maps an element in the *domain* D to an element in the *range* R.
- Write $f(a) = b$ if f maps $a \in D$ to $b \in R$.
- When $f : A_1 \times A_2 \times \cdots \times A_k \to B$, we say f is a k*-ary function* and k is the *arity* of f.
  - When $k = 1$, f is a *unary function*.
  - When $k = 2$, f is a *binary function*.

Relations

- A *predicate*, or property, is a function whose range is {TRUE,FALSE}.
- A predicate whose domain is a set of k-tuples $A \times \ldots \times A$ is called a (k-ary) *relation* on A.
- A 2-ary relation is also called a *binary relation*.

Equivalence Relations

- An *equivalence relation* is a special type of binary relation that captures the notion of two objects being *equal* in some sense.
- A binary relation R on A is an equivalence relation if
  1. R is *reflexive* (for every x in A, xRx),
  2. R is *symmetric* (for every x and y in A, xRy if and only if yRx), and
  3. R is *transitive* (for every x, y, and z in A, xRy and yRz implies xRz).

Graphs

- An *undirected graph* (or a *graph*) consists of a set of *nodes* (or *vertices*) and a set of *edges*.
- The number of edges attached to a node is the *degree* of the node.
- A graph G is a *subgraph* of a graph H if the nodes of G are a subset of nodes of H, and the edges of G are those of H on the corresponding nodes.
- A *path* is a sequence of nodes connected by edges.
- A *simple path* is a path without repetitive nodes.
- A graph is *connected* if there is a path between any two nodes.
- A path is a *cycle* if it starts and ends in the same node.
- A *simple cycle* is a cycle with at least three nodes and repeating only the first and last nodes.
- A graph is a *tree* if it is connected and has no simple cycle.
- A tree has a special designated node called its *root*.
- The nodes with degree 1 in a tree are called *leaves*.

Graphs

IM NTU

- If edges in a graph are arrows, the graph is a *directed graph*.
- The number of arrows from a node is the *outdegree* of the node; the number of arrows to a node is the *indegree* of the node.
- A path whose arrows point in the same direction is a *directed path*.
- A directed graph is *strongly connected* if a directed path connects every two nodes.

Strings and Languages

- An *alphabet* is any finite set of *symbols*.
- A *string* over an alphabet is a finite sequence of symbols from that alphabet.
- The *length* of a string w, written as $|w|$, is the number of symbols that w contains.
- The string of length 0 is called the *empty string*, written as ε.
- The *concatenation* of x and y, written as xy, is the string obtained from appending y to the end of x.
- A *language* is a set of strings.
- More notions and terms: *reverse*, *substring*, *lexicographic ordering*.

Boolean Logic

- *Boolean logic* is a mathematical system built around the two *Boolean values* TRUE (1) and FALSE (0).
- Boolean values can be manipulated with *Boolean operations*: *negation* or NOT ($\neg$), *conjunction* or AND ($\wedge$), *disjunction* or OR ($\vee$).

$$
\begin{array}{lll}
0 \wedge 0 \triangleq 0 & 0 \vee 0 \triangleq 0 & \neg 0 \triangleq 1 \\
0 \wedge 1 \triangleq 0 & 0 \vee 1 \triangleq 1 & \neg 1 \triangleq 0 \\
1 \wedge 0 \triangleq 0 & 1 \vee 0 \triangleq 1 & \\
1 \wedge 1 \triangleq 1 & 1 \vee 1 \triangleq 1 &
\end{array}
$$

- Unknown Boolean values are represented symbolically by *Boolean variables* or *propositions*, e.g., P, Q, etc.

Boolean Logic (cont.)

- Additional Boolean operations: *exclusive or* or XOR ($\oplus$), *equality/equivalence* ($\leftrightarrow$ or $\equiv$), *implication* ($\rightarrow$).

$$
\begin{array}{ccc}
0 \oplus 0 \triangleq 0 & 0 \leftrightarrow 0 \triangleq 1 & 0 \rightarrow 0 \triangleq 1 \\
0 \oplus 1 \triangleq 1 & 0 \leftrightarrow 1 \triangleq 0 & 0 \rightarrow 1 \triangleq 1 \\
1 \oplus 0 \triangleq 1 & 1 \leftrightarrow 0 \triangleq 0 & 1 \rightarrow 0 \triangleq 0 \\
1 \oplus 1 \triangleq 0 & 1 \leftrightarrow 1 \triangleq 1 & 1 \rightarrow 1 \triangleq 1
\end{array}
$$

- All in terms of conjunction and negation:

$$
\begin{array}{lcl}
P \vee Q & \equiv & \neg(\neg P \wedge \neg Q) \\
P \rightarrow Q & \equiv & \neg P \vee Q \\
P \leftrightarrow Q & \equiv & (P \rightarrow Q) \wedge (Q \rightarrow P) \\
P \oplus Q & \equiv & \neg(P \leftrightarrow Q)
\end{array}
$$

Logical Equivalences and Laws

- Two logical expressions/formulae are *equivalent* if each of them implies the other, i.e., they have the same truth value.
- Equivalence plays a role analogous to equality in algebra.
- Some laws of Boolean logic:
  - (Distributive) $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$
  - (Distributive) $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
  - (De Morgan's) $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
  - (De Morgan's) $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$

Definitions, Theorems, and Proofs

Definitions, Theorems, and Proofs

- *Definitions* describe the objects and notions that we use. Precision is essential to any definition.
- After we have defined various objects and notions, we usually make *mathematical statements* about them. Again, the statements must be precise.
- A *proof* is a convincing logical argument that a statement is true. The only way to determine the truth or falsity of a mathematical statement is with a mathematical proof.
- A *theorem* is a mathematical statement proven true. *Lemmas* are proven statements for assisting the proof of another more significant statement.
- *Corollaries* are statements seen to follow easily from other proven ones.

Finding Proofs

- Find proofs isn't always easy; no one has a recipe for it.
- Below are some helpful general strategies:
  1. Carefully read the statement you want to prove.
  2. Rewrite the statement in your own words.
  3. Break it down and consider each part separately. For example, $P \iff Q$ consists of two parts: $P \to Q$ (the forward direction) and $Q \to P$ (the reverse direction).
  4. Try to get an intuitive feeling of why it should be true.

Tips for Producing a Proof

- A well-written proof is a sequence of statements, wherein each one follows by simple reasoning from previous statements in the sequence.
- Tips for producing a proof:
  - *Be patient*. Finding proofs takes time.
  - *Come back to it*. Look over the statement, think about it, leave it, and then return some time later.
  - *Be neat*. Use simple, clear text and/or pictures; make it easy for others to understand.
  - *Be concise*. Emphasize high-level ideas, but be sure to include enough details of reasoning.

An Example Proof

IM NTU

Theorem

For any two sets A and B, $\overline{A \cup B} = \overline{A} \cap \overline{B}$.

Proof. We show that every element of $\overline{A \cup B}$ is also an element of $\overline{A} \cap \overline{B}$ and vice versa.

Forward ($x \in \overline{A \cup B} \rightarrow x \in \overline{A} \cap \overline{B}$):

$$
\begin{array}{lll}
 & x \in \overline{A \cup B} & \\
\rightarrow & x \notin A \cup B & \text{, def. of complement} \\
\rightarrow & x \notin A \text{ and } x \notin B & \text{, def. of union} \\
\rightarrow & x \in \overline{A} \text{ and } x \in \overline{B} & \text{, def. of complement} \\
\rightarrow & x \in \overline{A} \cap \overline{B} & \text{, def. of intersection}
\end{array}
$$

Reverse ($x \in \overline{A} \cap \overline{B} \rightarrow x \in \overline{A \cup B}$): ...

Another Example Proof

IM NTU

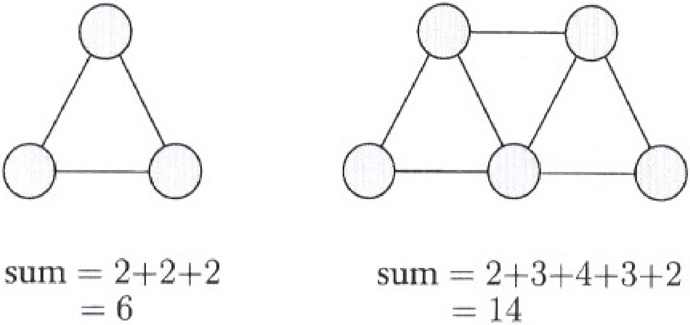
Theorem

In any graph G, the sum of the degrees of the nodes of G is an even number.

Proof.

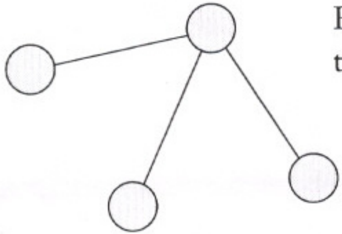
- Every edge in G connects two nodes, contributing 1 to the degree of each.
- Therefore, each edge contributes 2 to the sum of the degrees of all the nodes.
- If G has e edges, then the sum of the degrees of the nodes of G is $2e$, which is even.

Another Example Proof (cont.)



Source: [Sipser 2006]

Another Example Proof (cont.)



Every time an edge is added,
the sum increases by 2.

Source: [Sipser 2006]

Types of Proof

Types of Proof

- *Proof by construction*:
  prove that a particular type of object exists, by showing how to construct the object.
- *Proof by contradiction*:
  prove a statement by first assuming that the statement is false and then showing that the assumption leads to an obviously false consequence, called a contradiction.
- *Proof by induction*:
  prove that all elements of an infinite set have a specified property, by exploiting the inductive structure of the set.

Proof by Construction

Theorem

For each even number n greater than 2, there exists a 3-regular graph with n nodes.

Proof. Construct a graph $G = (V, E)$ with n $(= 2k \geq 2)$ nodes as follows.

Let V be $\{0, 1, \ldots, n-1\}$ and E be defined as

$$\begin{aligned} E \;=\; & \{\{i, i+1\} \mid \text{for } 0 \leq i \leq n-2\} \cup \\ & \{\{n-1, 0\}\} \cup \\ & \{\{i, i+n/2\} \mid \text{for } 0 \leq i \leq n/2-1\}. \end{aligned}$$

Proof by Contradiction

Theorem

$\sqrt{2}$ *is irrational.*

Proof. Assume toward a contradiction that $\sqrt{2}$ is rational, i.e., $\sqrt{2} = \frac{m}{n}$ for some integers m and n, which *cannot both be even*.

$\sqrt{2} = \frac{m}{n}$	, from the assumption
$n\sqrt{2} = m$	, multipl. both sides by n
$2n^2 = m^2$	, square both sides
m is even	, m^2 is even
$2n^2 = (2k)^2 = 4k^2$	, from the above two
$n^2 = 2k^2$	, divide both sides by 2
n is even	, n^2 is even

Now both m and n are even, a contradiction.

Fallacious Arguments

Example

Show $1 = 2$.

Fallacious Argument.

Let a and b be two equal positive numbers. Hence $a = b$. We multiply both sides by a and have $a^2 = ab$. Subtract b^2 from both sides, we have $a^2 - b^2 = ab - b^2$. Thus $(a + b)(a - b) = b(a - b)$. Therefore $a + b = b$. Since $a = b$, we have $2b = b$ and $2 = 1$. □

Example

Show symmetry and transitivity imply reflexivity?

Fallacious Argument.

By symmetry, we have $x \sim y$ and thus $y \sim x$. By transitivity, $x \sim y$ and $y \sim x$ implies $x \sim x$. □