Preliminaries

Notes on Automata and Theory of Computation

Chia-Ping Chen

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan ROC

Overview

- Theory of Computation
- Mathematical Preliminaries
  - Sets
  - Functions
  - Graphs and Trees
  - Mathematical Proof
- Languages, Grammars and Automata
- Applications

Theory of Computation

- Theoretical foundation of computer science
- Provides common underlying principles
- Related directly to applications such as programming languages (compilers)
- Intellectually stimulating and fun
- Includes models of automata, formal languages, grammars, computability and complexity

Sets

- A set is a collection of objects, called elements.
- A set can be specified by enclosing the description of its elements in braces. For example,
  - $S = \{1, 2, 3\}$
  - $S = \{1, 2, 3, \ldots\}$
  - $S = \{i : i > 0, i \text{ is prime}\}$
  - $S = \{i \mid i > 0, i \text{ is prime}\}$
- The membership of x in a set S is denoted by $x \in S$.
- A finite set consists of a finite number of elements.
- A infinite set consists of an infinite number of elements. It can be either countable or uncountable.

Set Operation

- union

$$S_1 \cup S_2 = \{x : x \in S_1 \text{ or } x \in S_2\}$$

- intersection

$$S_1 \cap S_2 = \{x : x \in S_1 \text{ and } x \in S_2\}$$

- difference

$$S_1 - S_2 = \{x : x \in S_1 \text{ and } x \notin S_2\}$$

- complementation

$$\overline{S} = U - S$$

Special Sets

- The **empty set** or **null set** is the set which contains no elements. It is denoted by $\emptyset$.
- The **universal set** is the set containing all possible elements. It is denoted by U.
- The following properties are true.

$$\begin{aligned} S \cup \emptyset &= S - \emptyset = S \\ S \cap \emptyset &= \emptyset \\ \overline{\emptyset} &= U \\ \overline{\overline{S}} &= S \end{aligned}$$

Subsets

- A set S_1 is said to be a **subset** of S if every element of S_1 is an element of S. This is denoted by

$$S_1 \subseteq S.$$

- A set S_1 is said to be a **proper subset** of S if

$$S_1 \subseteq S \text{ and } S - S_1 \neq \emptyset.$$

- Two sets S_1 and S_2 are said to be **disjoint** if

$$S_1 \cap S_2 = \emptyset.$$

- A collection of sets $S_1 \ldots S_n$ is said to be a **partition** of S if they are disjoint and their union is S.

Powerset and Cartesian Product

- The **powerset** of S is the set of all subsets of S. It is denoted by 2^S. Note that 2^S is a set of sets.
- The **Cartesian product** of two sets S_1, S_2 is defined by

$$S_1 \times S_2 = \{(x, y) : x \in S_1, y \in S_2\}.$$

- We can look at a few examples.

Functions

- A function is a rule that assigns to an element of a set, called domain, a unique element in another set, called range.
- We write

$$f : S_1 \rightarrow S_2$$

  to indicate the domain of f is a subset of S_1, and the range is a subset of S_2.
  - f is called a total function if the domain of f is S_1.
  - Otherwise it is called a partial function.

Order of Magnitude

- Functions defined on the set of positive integers, $\mathcal{Z}^+$, are frequently encountered in this course.
- We are often interested in the behaviors of these functions as the arguments become large.
- Let $f(n), g(n)$ be two functions defined on $\mathcal{Z}^+$.

O, Ω, and Θ

- If $\exists\ c, n_0 > 0\ \ s.t.\ |f(n)| \leq c|g(n)|\ \forall n > n_0$, we say f has order at most g, and denote it as

$$f(n) = O(g(n)).$$

- If $\exists\ c, n_0 > 0\ \ s.t.\ |f(n)| \geq c|g(n)|\ \forall n > n_0$, we say f has order at least g, and denote it as

$$f(n) = \Omega(g(n)).$$

- If f has order at least g and at most g, we say f and g have the same order of magnitude, and denote it as

$$f(n) = \Theta(g(n)).$$

- We can look at a few examples.

Relation

- A function might be represented by a set of pairs,

$$\{(x_1, y_1), (x_2, y_2), \ldots\}.$$

  In such a representation, each x_i can appear only once in the set.

- A relation is more general than a function in the sense that an x may appear more than once in the above set.

Equivalence Class

- An equivalence relation is one which ensures that

$$x \equiv x \; \forall x$$
$$x \equiv y \;\Rightarrow y \equiv x$$
$$x \equiv y,\; y \equiv z \;\Rightarrow x \equiv z.$$

- It is a generalization of equality.
- We can use an equivalence relation to partition a set into equivalence classes. In each class, the elements are equivalent.
- We can look at a few examples.

Graphs

- A graph G consists of vertices and edges. We can define a graph by the set of vertices V and the set of edges E,
$$G = (V, E).$$
- Each edge in E is a pair of vertices from V.
- A graph can be directed or undirected. In a directed graph, an edge $e_i = (v_j, v_k)$ means that e_i starts at vertex v_j and ends at vertex v_k.
- What is V and E for Figure 1.1?

Walk, Path and Cycle

- A (directed) **walk** from v_i to v_n is a sequence of edges

$$(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n),$$

  that starts at v_i and ends at v_n.
- A **path** is a walk with no repeated edges.
- A path is **simple** if no vertex is repeated.
- A **cycle** with base v_i is a walk from v_i to itself. It is simple if no other vertex is repeated.
- An edge from a vertex to itself is called a **loop**.

Tree

- A (directed) tree is a particular kind of graph.
  - It has no cycles.
  - It has a vertex, called **root**, that there is exactly one path from the root to any other vertex.
  - There are some vertices without outgoing edges. They are called **leaves**.
  - If there is an edge (v_i, v_j), then v_i is called the **parent** of v_j and v_j is called a **child** of v_i.
- The **level** of a vertex is the number of edges from the root to it.
- The **height** of a tree is the largest level of vertices.

Formal Proof

- In order to develop a theory, statements are required to be proved, to make sure they are correct.
- It is generally insufficient to assert the correctness of a statement by supportive instances. (However, a statement can be invalidated by any counterexample.)
- Learning how to prove also helps to create a program. The implicit verification you do in your mind guides you to design your program.
- Proving something to be true could be very tricky. Fortunately, there are a number of techniques that could be useful.

Deductive Proof

- A deductive proof for ‘if H then C’ consists of a sequence of statements led by the hypothesis H (a.k.a. given statement) and ended by the conclusion statement C.
- Each statement in the sequence is established logically by the previous statements and other implicit facts.
- We can prove that if x is a sum of the squares of four positive integers (H), then $2^x \geq x^2$ (C) by deductive proof. That is,

$$H \;\Rightarrow x \geq 4 \;\Rightarrow\; C.$$

Reduction to Definition

- Sometimes it is useful to covert all terms to their basic definitions.
- For example, to show that 'If S is a finite subset of an infinite set U and T is the complement of S, then T is infinite.'
  - S is finite $\Rightarrow\ \exists\, n$ such that $|S| = n$
  - U is infinite $\Rightarrow \nexists\, p$ such that $|U| = p$
  - $T = \overline{S}\ \Rightarrow\ S \cap T = \emptyset, S \cup T = U$

  Assuming T to be finite, we can reach a contradiction. So T must be infinite.

Contrapositive

- The contrapositive of the statement

  if H then C

  is

  if not C then not H.

- A statement and its contrapositive are either both true or both false.

Converse

- The converse of the statement

$$\text{if } H \text{ then } C$$

is

$$\text{if } C \text{ then } H.$$

- A statement and its converse do not always have the same truth value.

Counterexample

- A statement cannot be proved to be true by any number of positive examples.
- A statement can be proved to be false by the existence of *one* counterexample.
- That is why one cannot simply test a program millions of times to justify its correctness.
- The statement

  all primes are odd

  is not true since 2 is prime and 2 is not odd. 2 is an counterexample.

Proof by Induction

- Proof by induction is used to prove a collection of statements indexed by integers. There are two parts to prove.
  - **basis**: For some $k \geq 1$, we prove $P_1, \ldots, P_k$ to be true.
  - **induction**: For any $n \geq k$, we prove that the truths of $P_1, \ldots, P_n$ imply the truth of P_{n+1}.
- Example

$$S_n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

Structural Induction

- Some structures are defined recursively. There are some basic cases, and the more complicated cases are defined through the application of specific operations.
- For example, a tree can be defined recursively by
  - A single node is a tree.
  - If $T_1, \ldots, T_k$ are trees, then we can form a new tree by creating a new root node and joining $T_1, \ldots, T_k$ to this node.
- Statement about structures defined recursively can be proved by structural induction.
- We can prove a tree has one more nodes than it has edges by structural induction.

Proof by Contradiction

- It works as follows. To prove some statement is true, we assume the opposite to be true and arrive at a contradiction to something known to be true.
- We have seen an example of proof by contradiction earlier.
- We can use this method to prove that $\sqrt{2}$ is irrational.

Language

- An **alphabet** is a finite, non-empty set of symbols.
- A **string** is a finite sequence of symbols from some alphabet.
- Given an alphabet Σ, we use Σ^* to denote the set of strings of zero or more symbols from Σ.
- A **language** is a subset of Σ^*.

Strings

- **concatenation**: If $u = u_1 \ldots u_n, v = v_1 \ldots v_m$, then $uv = u_1 \ldots u_n v_1 \ldots v_m$.
- **length**: $|v| = m, |u| = n$.
- **reverse**: $v^R = v_m \ldots v_1$.
- **power**: w^n is the concatenation of n copies of $w's$.
- **substring**: a string of consecutive symbols of w.
- **prefix** and **suffix**: If $w = uv$, then u is a prefix and v is a suffix of w.

String Length

- A recursive definition for string length is

$$\begin{cases} |a| = 1 \\ |ua| = |u| + 1. \end{cases}$$

- We will show that

$$|uv| = |u| + |v|, \text{ for all } u, v.$$

  - By definition, this is true for any u and $|v| = 1$.
  - Assuming it is true for any u and $|v| = 1, \ldots, k$. For $|v| = k + 1$, let $v = wa$ where $|w| = k$. Then

$$|uv| = |uwa| = |uw| + 1 = |u| + |w| + 1 = |u| + k + 1.$$

Operations on Languages

- **complement**

$$\overline{L} = \Sigma^* - L.$$

- **concatenation**

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}$$

- **reverse**

$$L^R = \{w : w^R \in L\}$$

- **power** (a recursive definition)

$$L^0 = \{\lambda\}, \ \ L^{n+1} = L^n L.$$

Closures

- **star-closure** (a.k.a. **Kleene closure**)

$$L^* = L^0 \cup L^1 \cup L^2 \dots$$

- **positive closure**

$$L^+ = L^1 \cup L^2 \cup L^3 \dots$$

- It helps to look at some examples.

Grammars

- A grammar for English tells us whether a sentence is well-formed or not.
- So it actually defines a set of (grammatical) sentences, i.e., a language.
- Formally, a grammar is a quadruple

$$G = (V, T, S, P)$$

  - V is a finite set of **variables**
  - T is a finite set of **terminals**
  - $S \in V$ is the **start symbol**
  - P is a finite set of **production rules**

Production Rule

- A production rule is of the form

$$x \to y, \quad \text{where } x \in (V \cup T)^+ \text{ and } y \in (V \cup T)^*.$$

- The application of such a rule changes a string $w = uxv$ to $z = uyv$. This is also written as

$$w \Rightarrow z.$$

- If $w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$, then we say w_1 **derives** w_n. This is also denoted by

$$w_1 \stackrel{*}{\Rightarrow} w_n.$$

The Language of a Grammar

- The language defined (or generated) by a grammar $G = (V, T, S, P)$ is the set of terminal strings derived from S,

$$L(G) = \{w \in T^* : S \stackrel{*}{\Rightarrow} w\}.$$

- If $w \in L(G)$, then there exists a sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w.$$

  This sequence is called a **derivation** of w. $S, w_1, \ldots, w_n$ are called the **sentential forms** of the derivation.

- It helps to look at some examples.

Equivalent Grammars

- A given language may have multiple grammars (or none) to generate it. These grammars are equivalent in the sense that they generate the same language.
- Formally, two grammars G_1, G_2 are equivalent if

$$L(G_1) = L(G_2).$$

Automata

- An automaton
  - can read input
  - can read and write data in some temporary space
  - has a control unit
- A **configuration** consists of the state identity, the input data and position, and the content of the storage.
- A change from one configuration to the next is called a **move**. Moves are governed by the transition function.
- The basic components of an automaton are shown in Figure 1.4.

Classes of Automata

- **finite acceptors**, **pushdown automata**, and **Turing machines**: they differ in their temporary storages.
- **deterministic** vs. **nondeterministic**: a deterministic automaton must have a unique move for each configuration, while a nondeterministic automaton has a set of possible moves (including none).
- **acceptor** vs. **transducer**: an acceptor simply determines whether an input string is accepted; a transducer outputs a string of symbols.

Applications

- A variable identifier in the c language
  - is a sequence of letters, digits and underscores
  - starts with a letter or underscore

  These rules can be implemented by a grammar, or an automaton as shown in Figure 1.6.
- A binary adder can be implemented as a transducer with two states. One is for `carry` and the other is for `no carry`.