Chapter 1

Introduction to The Theory of Computation

1.1 Mathematical Preliminaries and Notation

Sets

A **set** is a collection of elements, without any structure other than membership.

The usual set operations are **union** ($\cup$), **intersection** ($\cap$), **difference** ($-$) and **complementation** defined as

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

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

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

$$\overline{S} = \{\ x : x \in U \text{ and } x \notin S\ \}.$$

DeMorgan's laws

$$\overline{S_1 \cup S_2} = \overline{S_1} \cap \overline{S_2},$$

$$\overline{S_1 \cap S_2} = \overline{S_1} \cup \overline{S_2}.$$

A set S_1 is said to be a **subset** of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S.$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a **proper subset** of S; we write this as

$$S_1 \subset S.$$

If S_1 and S_2 have no common element, then the sets are said to be **disjoint**. We write this as

$$S_1 \cap S_2 = \varnothing.$$

A set is said to be finite if it contains a **finite** number of elements; otherwise it is **infinite**. The set of all subsets of a set S is called the **powerset** of S and is denoted by 2^S. If S is finite, then

$$|2^S| = 2^{|S|}.$$

The sets whose elements are ordered sequences of elements from other sets are said to be the **Cartesian product** of other sets. For the Cartesian product of n sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 \times \cdots \times S_n = \{\ (x_1,\ x_2,\ \cdots,\ x_n) : x_i \in S_i\ \}.$$

Suppose that $S_1,\ S_2,\ \cdots,\ S_n$ are subsets of a given set S and that the following holds:

1. The subsets $S_1,\ S_2,\ \cdots,\ S_n$ are mutually disjoint;
2. $S_1 \cup S_2 \cup \cdots \cup S_n = S$;
3. none of the S_i is empty.

Then $S_1,\ S_2,\ \cdots,\ S_n$ is called a **partition** of S.

Functions and Relations

A **function** is a rule that assigns to elements of one set a unique element of another set. If f denotes a function, then the first set is called the **domain** of f, and the second set is its **range**. We write

$$f : S_1 \to S_2$$

to indicate that the domain of f is a subset of S_1 and that the range of f is a subset of S_2. If the domain of f is all of S_1, we say that f is a **total function** on S_1; otherwise f is said to be a **partial function**.

Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. We say that

1. f has **order at most** g if there exists a positive constant c such that for all sufficiently large n

$$f(n) \leqslant c|g(n)| \qquad \xrightarrow{\text{expressed as}} \qquad f(n) = O(g(n)).$$

2. f has **order at least** g if there exists a positive constant c such that for all sufficiently large n

$$f(n) \geqslant c|g(n)| \qquad \xrightarrow{\text{expressed as}} \qquad f(n) = \Omega(g(n)).$$

3. f and g have the **same order of magnitude** if there exist constant c_1 and c_2 such that for all sufficiently large n

$$c_1|g(n)| \leqslant |f(n)| \leqslant c_2|g(n)| \qquad \xrightarrow{\text{expressed as}} \qquad f(n) = \Theta(g(n)).$$

Some functions can be represented by a set of pairs

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

where x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, the set is called a **relation**.

Equivalence is a generalization of the concept of equality (identity). A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules:

1. The reflexivity rule
$$x \equiv x \text{ for all } x;$$
2. The symmetry rule
$$\text{if } x \equiv y, \text{ then } y \equiv x;$$
3. The transitivity rule
$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

If S is a set on which we have a defined equivalence relation, then we can use this equivalence to partition the set into **equivalence classes**.

Graphs and Trees

A graph is a construct consisting of two finite sets, the set $V = \{ v_1, v_2, \cdots, v_n \}$ of **vertices** and the set $E = \{ e_1, e_2, \cdots, e_m \}$ of **edges**. Each edge is a pair of vertices from V, for instance

$$e_i = (v_j, v_k)$$

is an edge from v_j to v_k. We say that the edge e_i is an outgoing edge for v_j and an incoming edge for v_k.

---

1. A sequence of edges $(v_i, v_j), (v_j, v_k), \cdots, (v_m, v_n)$ is said to be a **walk** from v_i to v_n;
2. The length of a walk is the total number of edges traversed in going from the initial vertex to the final one;
3. A walk in which no edge is repeated is said to be a **path**;
4. A path is **simple** if no vertex is repeated;
5. A walk from v_i to itself with no repeated edges is called a **cycle** with **base** v_i;
6. An edge from a vertex to itself is called a **loop**.

A tree is a directed graph that has no cycles and that has one distinct vertex, called the **root**, such that there is exactly one path from the root to every other vertex.

---

1. The vertices which have no outgoing edges are called the **leaves** of the tree;
2. If there is an edge from v_i to v_j, then v_i is said to be the **parent** of v_j, and v_j the **child** of v_i;
3. The **level** associated with each vertex is the number of edges in the path from the root to the vertex;

4. The **height** of the tree is the largest level number of any vertex;

5. In **ordered trees**, an ordering with the nodes is associated with the nodes at each level.

Proof Techniques

Proof by induction

Induction is a technique by which the truth of a number of statements can be infered from the truth of a few specific instances. Suppose we have a sequence of statements $P_1,\ P_2,\ \cdots$ we want to prove to be true. Furthermore, suppose also that the following holds:

1. For some $k \geqslant 1$, we know that $P_1,\ P_2,\ \cdots,\ P_k$ are true.
2. The problem is such that for any $n \geqslant k$, the truths of $P_1,\ P_2,\ \cdots,\ P_n$ imply the truth of P_{n+1}.

We can then use induction to show that every statement in this sequence is true.

1. The starting statements $P_1,\ P_2,\ \cdots,\ P_k$ are called the **basis** of the induction.
2. The step connecting P_n with P_{n+1} is called the **inductive step**.
3. The inductive step is generally made easier by the **inductive assumption** that $P_1,\ P_2,\ \cdots,\ P_n$ are true, then argue that the truth of these statements guarantees the truth of P_{n+1}.

Proof by contradiction

Suppose we want to prove that some statement P is true. We then assume, for the moment, that P is false and see where that assumption leads us. If we arrive at a conclusion that we know is incorrect, we can lay the blame on the starting assumption and conclude that P must be true.

1.2 Three Basic Concepts

Languages

An **alphabet** Σ is a finite, nonempty set of symbols. A **string** w is a finite sequence of symbols from the alphabet Σ. The **length** of a string w, denoted by $|w|$, is the number of symbols in the string.

1. The **concatenation** of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

   $$w = a_1 a_2 \cdots a_n, \qquad v = b_1 b_2 \cdots b_m,$$

   where $a_1, a_2, \cdots a_n, b_1, b_2, \cdots, b_m \in \Sigma$ and $n, m \in \mathbb{N}^+$. Then the concatenation of w and v, denoted by wv, is

   $$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

   It is obvious that

   $$|wv| = |w| + |v|.$$

   In addition, w^n stands for the string obtained by repeating w n times.

2. The **reverse** of a string w is obtained by writing the symbols in reverse order, that is, if

   $$w = a_1 a_2 \cdots a_n,$$

   where $a_1, a_2, \cdots a_n \in \Sigma$ and $n \in \mathbb{N}^+$. Then the reverse of w, denoted by w^R, is

   $$w^R = a_n \cdots a_2 a_1.$$

The **empty string**, denoted by λ, is the string with no symbols at all. The following simple relations

$$|\lambda| = 0, \qquad \lambda w = w\lambda = w, \qquad w^0 = \lambda$$

holds for all w.

If w is a string, any string of consecutive symbols in some w is said to be a **substring** of w. If

$$w = vu,$$

then the substrings v and u are said to be a **prefix** and **suffix** of w, respectively.
If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. To exclude the empty string, we define

$$\Sigma^+ = \Sigma^* - \{\lambda\}.$$

A **language**, denoted by L, is defined as a subset of Σ^*. A string in a language L will be called a **sentence** of L. A finite language is a language with finite number of sentences; an infinite language is a language with infinite number of sentences.

1. The **complement** of a language L is defined with respect to Σ^*, that is,

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

2. The **reverse** of a language L is the set of all string reversals, that is,

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

3. The **concatenation** of two languages L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2, that is,

$$L_1L_2 = \{xy : x \in L_1, y \in L_2\}.$$

   We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}, \qquad L^1 = L,$$

   for every language.

4. The **star-closure** of a language as

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

   and the **positive closure** as

$$L^+ = L^1 \cup L^2 \cdots$$

Grammars

Definition 1.1 A grammar G is defined as a quadruple

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

where

V is a finite set of objects called **variables**,

T is a finite set of objects called **terminal symbols**,

$S \in V$ is a special symbol called the **start** variable,

P is a finite set of **productions**.

It will be assumed that the sets V and T are non-empty and disjoint. All productions rules are of the form

$$x \to y,$$

where

$$x \in (V \cup T)^+, \qquad y \in (V \cup T)^*.$$

The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

where $u, v \in (V \cup T)^*$, we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

This is written as

$$w \Rightarrow z.$$

We say that w **derives** z or that z is derived from w. If

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

we say that w_1 derives w_n and write

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

The $*$ indicates that an unspecified of steps (including zero) can be taken to derive w_n from w_1.

Definition 1.2 Let $G = (V, T, S, P)$ be a grammar. Then the set

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

is the language generated by G.

If $w \in L(G)$, then the sequence

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

is a **derivation** of the sentence w. The strings $S, w_1, w_2, \cdots, w_n$, which contain variables as well as terminals, are called **sentential forms** of the derivation.

To show that a given language is indeed generated by a certain grammar G, we must be able to show

(a) **that every $w \in L$ can be derived from S using G;**

(b) **that every string so derived is in L.**

We say that two grammars G_1 and G_2 are **equivalent** if they generate the same language, that is, if

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

Automata

An automaton is an abstract model of a digital computer.

1. Composition

   (a) **Input File**

   An automaton has a mechanism for reading input. The input is a string over a given alphabet, written on an input file, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol. The input mechanism can also detect the end of the input string (by sensing an end-of-file condition).

   (b) **Storage Device**

   The automaton can produce output of some form. It may have a temporary storage device, consisting of an unlimited number of cells, each capable of holding a single symbol from an alphabet (not necessarily the same one as the input alphabet). The automaton can read and change the contents of the storage cells.

   (c) **Control Unit**

   The automaton has a control unit, which can be in any one of a finite number of internal states, and which can change state in some defined manner.

2. Mechanism

   (a) Present state

   An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the next-state or transition function.

   (b) **Next-state and Transition Function**

   The transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one time interval to the next, output may be produced or the information in the temporary storage changed.

(c) **Configuration and Move**

The term configuration will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a move.

3. Examples

(a) **Deterministic Automata**

A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly.

(b) **Nondeterministic Automata**

At each point, a nondeterministic automata may have several possible moves, so we can only predict a set of possible actions.

(c) **Accepter**

An automaton whose output response is limited to a simple “yes” or “no” is called an accepter. Presented with an input string, an accepter either accpets the string or rejects it.

(d) **Transducer**

A more general automaton, capable of producing strings or symbols as output, is called a transducer.