Chapter 1 Introduction to the Theory of Computation

- Computer science is a practical discipline. Those who work in it often have a marked preference for useful and tangible problems over theoretical speculation. Theory provides concepts and principles that help us understand the general nature of the discipline. The field of computer science includes a wide range of special topics, from machine design to programming. The use of computers in the real world involves a wealth of specific detail that must be learned for a successful application. This makes computer science a very diverse and broad discipline.

- In this course, we will look at models that represent features at the core of all computers and their applications. To model the hardware of a computer, we introduce the notion of an automaton (plural, automata).

- An automaton is a construct that possesses all the indispensable features of a digital computer. It accepts input, produces output, may have some temporary storage, and can make decisions in transforming the input into the output. A formal language is an abstraction of the general characteristics of programming languages. A formal language consists of a set of symbols and some rules of formation by which these symbols can be combined into entities called sentences. We can learn a great deal about programming languages from formal languages. In Section 1.1, we review the main ideas from mathematics that will be required.

1.1 Mathematical Preliminaries and Notation

Sets

- A **set** is a collection of elements, without any structure other than membership. To indicate that x is an element of the set S, we write $x \epsilon S$. The statement that x is not in S is written $x \notin S$. A set is specified by enclosing some description of its elements in curly braces; for example, the set of integers 0,1,2 is shown as

$$S = \{0,1,2\}$$

- The usual set operations are **union** (∪),**intersection** (∩),and **difference**(-), defined as;

$$S_1 \cup S_2 = \{x: x \epsilon S_1 \ or \ x \epsilon S_2\}$$
$$S_1 \cap S_2 = \{x: x \epsilon S_1 \ and \ x \epsilon S_2\}$$
$$S_1 - S_2 = \{x: \ x \epsilon S_1 \ and \ x \notin S_2\}$$

- Another basic operation is **complementation**. The complement of a set S, denoted by S, consists of all elements not in S. To make this, meaningful, we need to know what the **universal set** U of all possible elements is. If U is specified, then

$$\overline{S} = \{x: \ x \epsilon U \ and \ x \notin S\}$$

- The set with no elements, called the **empty set** or the **null set** is denoted by ϕ. From the definition of a set, it is obvious that

$$S \cup \phi = S - \phi = S,$$

$$S \cap \phi = \phi,$$

$$\overline{\phi} = U,$$

$$\overline{\overline{S}} = S$$

- The following useful identities, known as the **DeMorgan's laws,**

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

are needed on several occasions.

- 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, that is, $S_1 \cap S_2 = \phi$,then the sets are said to be **disjoint.**

- A set is said to be **finite** if it contains a finite number of elements; otherwise it is **infinite**. The size of a finite set is the number of elements in it; this is denoted by $|S|$.

- A given set normally has many subsets. The set of all subsets of a set S is called the **powerset** of S and is denoted by 2^S. Observe that 2^S is a set of sets.

Example 1.1

If S is the set $\{a, b, c\}$, then its powerset is

$$2^S = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Here $|S| = 3$ and $|2^S| = 8$. This is an instance of a general result; if S is finite, then

$$\left|2^S\right| = 2^{|S|}.$$

- In many of our examples, the elements of a set are ordered sequences of elements from other sets. Such sets are said to be the **Cartesian product** of other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 X S_2 = \{(x, y) : x \epsilon S_1, y \epsilon S_2\}$$

Example 1.2

Let $S_1 = \{2, 4\}$ and $S_2 = \{2, 3, 5, 6\}$. Then

$$S_1 \times S_2 = \{(2, 2), (2, 3), (2, 5), (2, 6), (4, 2), (4, 3), (4, 5), (4, 6)\}.$$

Note that the order in which the elements of a pair are written matters. The pair (4, 2) is in $S_1 \times S_2$, but (2, 4) is not.

The notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

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

A set can be divided by separating it into a number of subsets. Suppose that S_1, S_2, S_n are subsets of a given set S and that the following holds:

1. The subsets $S_1, S_2, \ldots S_n$ are mutually disjoint;

2. $S_1 \cup S_2 \cup \ldots \cup S_n = S$;

3. none of the S_i is empty.

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

Functions and Relation

- 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

$$f: S_1 \longrightarrow S_2$$

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**.

In many applications, the domain and range of the functions involved are in the set of positive integers. Furthermore, we are often interested only in the behavior of these functions as their arguments become very large. In such cases an understanding of the growth rates may suffice and a common order of magnitude notation can be used. Let $f(n)$ and $g(n)$ be functions whose domain is a subset of the positive integers. If there exists a positive constant c such that for all sufficiently large n

$$f(n) \leq c|g(n)|,$$

we say that f has **order at most** g. We write this as

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

If

$$|f(n)| \geq c|g(n)|,$$

then f has **order at least** g, for which we use

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

Finally, if there exist constants c_1 and c_2 such that

$$c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)|,$$

f and g have the **same order of magnitude**, expressed as

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

Example 1.3

Let

$$f(n) = 2n^2 + 3n,$$
$$g(n) = n^3,$$
$$h(n) = 10n^2 + 100.$$

Then

$$f(n) = O(g(n)),$$
$$g(n) = \Omega(h(n)),$$
$$f(n) = \Theta(h(n)).$$

In order-of-magnitude notation, the symbol = should not be interpreted as equality and order-of-magnitude expressions cannot be treated like ordinary expressions. Manipulations such as

$$O(n) + O(n) = 2O(n)$$

are not sensible and can lead to incorrect conclusions. Still, if used properly, the order-of-magnitude arguments can be effective, as we will see in later chapters.

One kind of relation is that of **equivalence**, a generalization of the concept of equality (identity). To indicate that a pair (x, y) is in an equivalence relation, we write

$$x \equiv y.$$

A relation denoted by $\equiv$ is considered an equivalence if it satisfies three rules: the reflexivity rule

$$x \equiv x \text{ for all } x;$$

the symmetry rule

$$\text{if } x \equiv y, \text{ then } y \equiv x;$$

and the transitivity rule

$$\text{if } x \equiv y \text{ and } y \equiv z, \text{ then } x \equiv z.$$

Example 1.4

On the set of nonnegative integers, we can define a relation

$$x \equiv y$$

if and only if

$$x \bmod 3 = y \bmod 3.$$

Then $2 \equiv 5$, $12 \equiv 0$, and $0 \equiv 36$. Clearly this is an equivalence relation, as it satisfies reflexivity, symmetry, and transitivity.

Graphs and Trees

- A graph is a construct consisting of two finite sets, the set $V = \{v_1, v_2, \dots, v_n\}$ of vertices and the set $E = \{e_1, e_2, \dots, e_m\}$ 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. Such a construct is actually a directed graph (digraph), since we associate a direction (from $v_j$to v_k) with each edge.

- Graphs are conveniently visualized by diagrams in which the vertices represented as circles and the edges as lines with arrows connecting the vertices. The graph with vertices $\{v_1, v_2, v_3\}$ and edges $\{(v_1, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_3)\}$ is depicted in Figure 1.1.

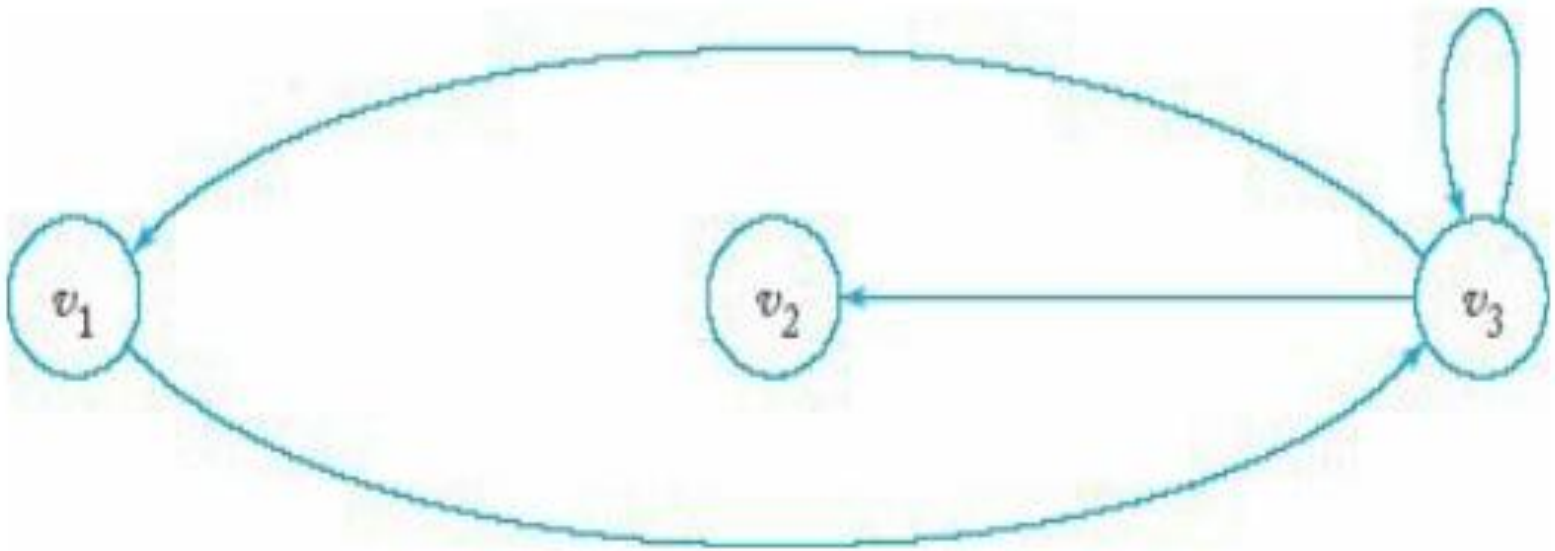

Figure 1.1

- A sequence of edges $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ is said to be a walk from v_i to v_n. A walk in which no edge is repeated is said to be a path; a path is simple if no vertex is repeated. A walk from v_i, to itself with no repeated edges is called a cycle with base v_i.An edge from a vertex to itself is called a loop. In Figure 1.1 there is a loop on vertex v_3.

- Trees are a particular type of graph. 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. This definition implies that the root has no incoming edges and that there are some vertices without outgoing edges. These are called the **leaves** of the tree. If there is an edge from v_i, to v_j then v_i is said to be the **parent** of v_j, and the v_j **child** of v_i.

- The **level** associated with each vertex is the number of edges in the path from the root to the vertex. The **height** of the tree is the largestlevel number of any vertex. These terms are illustrated in Figure 1.2.

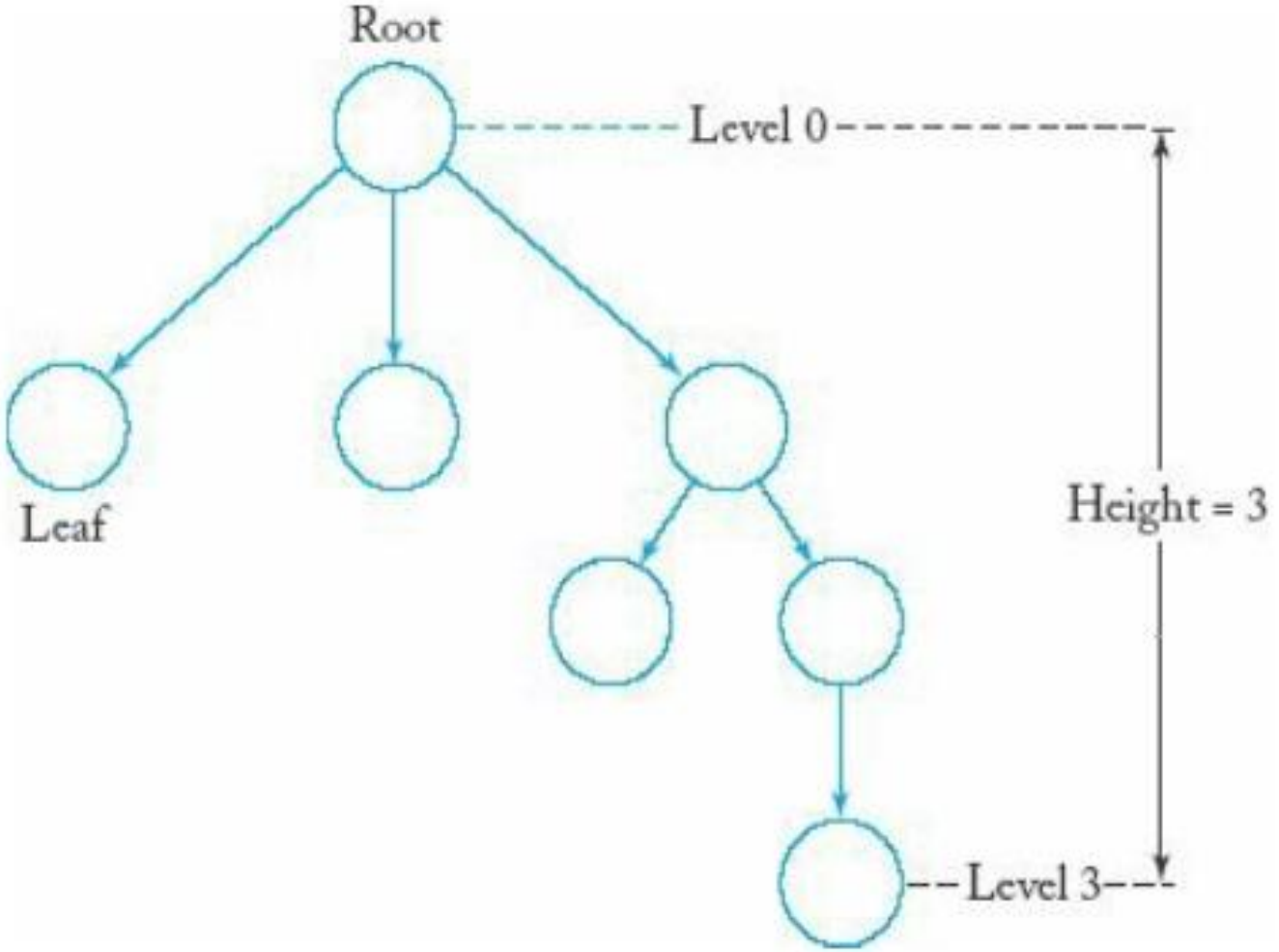

Figure 1.2

1.2 Three Basic Concepts

- Three fundamental ideas are the major themes of this course: languages, grammars and automata. In the course of our study we will explore many results about these concepts and about their relationship to each other.

Languages

- We are all familiar with the notion of natural languages, such as English and French. Still, most of us would probably find it difficult to say exactly what the word “language” means. Dictionaries define the term informally as a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for their manipulation. While this gives us an intuitive idea of what a language is, it is not sufficient as a definition for the study of formal languages. We need a precise definition for the term.

- We start with a finite, nonempty set Σ of symbols, called the alphabet. From the individual symbols we construct strings, which are finite sequences of symbols from the alphabet. For example, if the alphabet Σ = {a, b}, then abab and aaabbba are strings on Σ. With few exceptions, we will use lowercase letters a, b, c,…for elements of Σ and u, υ, ω,…for string names. We will write, for example, to indicate

$$w = abaaa$$

to indicate that the string named w has the specific value abaaa.

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

$$w = a_1 a_2 \cdots a_n$$

and

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

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

The **reverse** of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

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

The **length** of a string w, denoted by $|w|$, is the number of symbols in the string. We will frequently need to refer to the **empty string**, which is a string with no symbols at all. It will be denoted by λ. The following simple relations

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

hold for all w.

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 a **suffix** of w, respectively. For example, if $w =$ *abbab*, then $\{\lambda, a, ab, abb, abba, abbab\}$ is the set of all prefixes of w, while *bab*, *ab*, *b* are some of its suffixes.

Simple properties of strings, such as their length, are very intuitive and probably need little elaboration. For example, if u and v are strings, then the length of their concatenation is the sum of the individual lengths, that is,

$$|uv| = |u| + |v| .$$

If $\sum$ is an alphabet, then we use $\sum^*$ to denote the set of strings obtained by concatenating zero or more symbols from $\sum$. The set $\sum^*$ always contains λ. To exclude the empty string, we define

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

While $\sum$ is finite by assumption, $\sum^*$ and $\sum^+$ are always infinite since there is no limit on the length of the strings in these sets. A language is defined very generally as a subset of $\sum^*$. A string in a language L will be called a **sentence** of L. This definition is quite broad; any set of strings on an alphabet $\sum$ can be considered a language. Later we will study methods by which specific languages can be defined and described; this will enable us to give some structure to this rather broad concept. For the moment, though, we will just look at a few specific examples.

Example 1.3

Let $\sum = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, ...\}\,.$$

The set

$$\{a, aa, aab\}$$

is a language on $\sum$. Because it has a finite number of sentences, we call it a finite language. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on $\sum$. The strings *aabb* and *aaaabbbb* are in the language L, but the string *abb* is not in L. This language is infinite. Most interesting languages are infinite.

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

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

for every language L.

Finally, we define the **star-closure** of a language as

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

and the **positive closure** as

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

Example 1.4

If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that *n* and *m* in the above are unrelated; the string *aabbaaabbb* is in L^2.

The reverse of L is easily described in set notation as

$$L^R = \{b^n a^n : n \geq 0\},$$

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.