

Chapter 0

Mathematical Preliminaries



Section 0.0: Introduction

While this book is about computing, it takes an abstract and formal approach in its investigations. At the heart of the content of this book is providing statements of the basic definitions with impeccable precision, and deducing the conclusions that flow from them with indisputable logic. We are mainly interested in establishing properties that entire categories of computations must possess rather than considering individual computations. This means that the ability to reason about formal systems is critical to following our development.

In this chapter we provide a succinct review of the mathematical prerequisites that the reader needs to understand in order to assimilate the ideas of this book. It is not intended that the coverage provided here will be sufficient for the reader with little or no prior exposure to this material, and this section is not a substitute for the study of appropriate background as outlined in the Preface. It is expected that the reader has previously studied these topics, and needs only to be reminded of some of these ideas. In fact, most readers should not need to do more than skim this chapter to align their previous background with the terminology and notation used in this book. Since terminology on these basic topics differs little from one writer to

another, this should require minimal attention from readers with appropriate prerequisites.

Section 0.1: Sets and tuples

The most pivotal formalism used in this book to accomplish our goals is that of sets and their properties. The essential concept of *set* is both elementary and natural. It simply consists of grouping a number of distinct elements, the *members*, into a single unit, the set. Operations and properties that are common to all members of a set can then be treated once for the entire group rather than repeatedly for each of the members. Once we have some basic formalisms established, we will virtually always take the approach of building on them to develop the next one. However, as the first formalism to be introduced, sets cannot be explained in terms of a previously studied formalism. Therefore we must take another tack, normally either selecting abstract axioms that capture the relevant properties, or placing more reliance on informality and intuition. In this section we take the latter approach.

We will always employ sets within a specific *universe of discourse*. This gives us a definite context in which statements are to be interpreted. A statement such as “all persons over the age of 18 by election day are allowed to vote” is not sufficiently precise for our purposes. We need to know in addition, for example, that these persons are citizens of, for example, the United States. In such a case our universe of discourse might be all persons who are citizens of the United States (and we may wish to incorporate other constraints as well). A basic premise we will always assume is that there is a pre-determined collection — the **universe** — of all the objects that are in the domain of discussion. The universe will be selected as context for the issues we wish to discuss. For example in computing, one frequently used universe is the collection **Nat** of all natural numbers (i.e., non-negative integers), **Nat** = {0, 1, 2, 3, ... }.

Once the universe is settled, to define a set we just need to describe how the elements of that set are determined. The key relationship for defining a set *S* is **membership** — if an object *x* from the universe is an element of *S*, we say that *x* is a **member of S** and write $x \in S$. There is no ordering between the members of a set that is implied by the fact of their membership. That is,

sets are deliberately an unordered structure for aggregating elements. Also, elements never occur in a set repeatedly — either an element is a member of a set or it's not.

Sets may be defined in a number of ways. As long as it is clearly determined which elements are members of the set (and which are not), that is all that matters. One standard set notation is to write the elements of the set enclosed in "curly braces", sometimes called the *roster notation*. For instance, $\{2, 4, 6\}$ denotes the set with the three members 2, 4, and 6 (say from the universe \mathbf{Nat}). While we must pick an order to write down the elements of a set, this should not be interpreted as establishing an order between the elements. Thus $\{2, 4, 6\}$ and $\{4, 2, 6\}$ are two notations for the same set. Most of the sets of interest will be infinite, and for infinite sets we cannot simply list all the elements. The set notation is extended to permit the description of infinite sets with the following *set comprehension*, or *set abstraction* notation: $\{\text{form of item} \mid \text{property of item}\}$.

The "item form" portion of a set comprehension describes the form of potential members of the set, and the "property" portion describes a condition that must be satisfied in order for items of the indicated form to be members of the set — the set defined consists of *all* items of the indicated form having the specified property. For example, $\{n \in \mathbf{Nat} \mid n > 5\}$ describes the infinite set of all natural numbers greater than 5, $\{6, 7, 8, \dots\}$. Of course, with this notation the set that is described may be either finite or infinite. The last example is an infinite set, while $\{n \in \mathbf{Nat} \mid 0 \leq n \leq 9\}$ is finite. Often the universe is implicitly clear from context and we may simply write $\{n \mid n > 5\}$. Either the "item" description or the "property" description may be more involved. For instance, $\{2 * n \mid n \in \mathbf{Nat}\}$ describes the set of all even natural numbers.

Unrestricted use of informal definitions of sets can lead to paradoxes — logically impossible situations. No doubt the best known of these is Russell's paradox that is stated as follows: let \mathfrak{R} denote the set of all those sets which are not members of themselves, $\mathfrak{R} = \{S \text{ is a set} \mid S \notin S\}$. At first glance this appears to be a well-defined, but somewhat curious collection (since sets we normally conceive of are never members of themselves). However, if we ask the question of whether \mathfrak{R} is a member of \mathfrak{R} we find that this collection is much more than just "curious". Suppose that $\mathfrak{R} \in \mathfrak{R}$.

Then by the definition of \mathfrak{R} it must be that $\mathfrak{R} \notin \mathfrak{R}$, a contradiction. On the other hand, if $\mathfrak{R} \notin \mathfrak{R}$, then by the definition of \mathfrak{R} , $\mathfrak{R} \in \mathfrak{R}$, again a contradiction! Since by the basic nature of sets, it must be that either $\mathfrak{R} \in \mathfrak{R}$ or $\mathfrak{R} \notin \mathfrak{R}$, and each assumption leads to a contradiction, we have a logically impossible situation. It is very disconcerting that a paradox is obtained from an apparently simple use of our basic notation for defining sets. The discovery of such paradoxical situations has led mathematicians to carefully examine the cause of this trouble, and it was found that the abstraction approach, where a set is described by defining a property of its members, must be restricted. For a precise axiomatic development that resolves these situations, the reader can consult e.g. (Sup 72). We will continue to treat sets informally in this book, and by adhering to our insistence that we always begin with a pre-determined universe, we will avoid any paradox.

There is one seemingly peculiar set, the **empty set**, written \emptyset , which has *no* elements. The empty set is perfectly well defined since we know exactly what elements are its members (i.e., none). However since we wish to use sets to group elements together, it might appear that the empty set would be of virtually no use (it groups nothing). In fact, useful applications of the empty set occur with surprising frequency.

There are a variety of relations and operations on sets that are of use to us later. We have already touched on one of them above when we indicated when two sets are “the same”. We formalize this now. For sets S and T in universe u , we say that S is a **subset** of T , written $S \subseteq T$, provided that every member of S is a member of T . S and T are **equal** (as sets), written $S = T$, provided that they contain exactly the same elements — that is, if $S \subseteq T$ and $T \subseteq S$. Hence proofs of set equality will normally consist of these two subset arguments. Also, S is a **proper subset** of T , written $S \subset T$, provided that $S \subseteq T$ and $S \neq T$. Note that for *every* set S , $\emptyset \subseteq S$. Finally, two sets are **disjoint** if they have no elements in common.

There are a number of operations on sets that yield other sets. The most basic are union, intersection, and complement. If S and T are sets in universe U , then

- the **union** of S and T , written $S \cup T$, is $S \cup T = \{x \in U \mid x \in S \text{ or } x \in T \text{ (or both)}\}$,

- the **intersection** of S and T , written $S \cap T$, is $S \cap T = \{x \in U \mid x \in S \text{ and } x \in T\}$,
- the **complement** of S , written $\neg S$, is $\neg S = \{x \in U \mid x \notin S\}$, and
- the **set difference** of S and T , written $S - T$, is $S - T = \{x \in S \mid x \notin T\} = S \cap \neg T$.

The three basic operations satisfy numerous identities. We will have frequent occasions where it will be valuable to use basic set identities. For instance, for all sets S , T , and V , the following set equalities are all easily verified:

- (i) $S \cup S = S \cap S = S$,
- (ii) $S \cup T = T \cup S$,
- (iii) $S \cap T = T \cap S$,
- (iv) $S \cup (T \cap V) = (S \cup T) \cap V$,
- (v) $S \cap (T \cup V) = (S \cap T) \cup (S \cap V)$,
- (vi) $S \cap (T \cup V) = (S \cap T) \cup (S \cap V)$,
- (vii) $S \cup \emptyset = S$,
- (viii) $S \cap \emptyset = \emptyset$,
- (ix) $S \cup U = U$,
- (x) $S \cap U = S$,
- (xi) $S \cup \neg S = U$,
- (xii) $S \cap \neg S = \emptyset$,
- (xiii) $\neg(\neg S) = S$.

Especially useful results show how union can be expressed in terms of intersection and complement, and similarly how intersection can be expressed in terms of union and complement. We state this as our first formal theorem of this chapter.

Theorem 0.1.1 (DeMorgan's laws): For any subsets S and T

- (a) $\neg(S \cup T) = \neg S \cap \neg T$, and
- (b) $\neg(S \cap T) = \neg S \cup \neg T$.

Proof:

Part (a).

Since this is a set equality, it must be shown that each of the sets is a subset of the other.

Step 1: show that $\neg(S \cup T) \subseteq \neg S \cap \neg T$.

Let $x \in (\neg(S \cup T))$. Then $x \notin S \cup T$ so $x \notin S$ and $x \notin T$. But then $x \in (\neg S)$ and $x \in (\neg T)$, so $x \in (\neg S \cap \neg T)$.

Step 2: show that $\neg S \cap \neg T \subseteq \neg(S \cup T)$.

This argument is similar to that in step 1 and is left to the reader.

Part (b) follows simply by applying part (a) re-expressed as $S \cup T = \neg(\neg S \cap \neg T)$. Then expand the union on the right-hand side of (b) using this version of (a) and we have, $\neg S \cup \neg T = \neg(\neg(\neg S) \cap \neg(\neg T)) = \neg(S \cap T)$.

□

The proof of part (a) of Theorem 0.1.1 provides a simple but typical example of how to construct a set equality argument. The proof of part (b) illustrates building on known identities to justify others.

Another operation on a set S that we will encounter is the **power set** operation, written $\wp(S)$, that is the collection of all subsets of S , $\wp(S) = \{T \mid T \subseteq S\}$. For example, $\wp(\{2, 4, 6\}) = \{\emptyset, \{2\}, \{4\}, \{6\}, \{2, 4\}, \{2, 6\}, \{4, 6\}, \{2, 4, 6\}\}$. The power set of S is often written as 2^S since a set S with n elements yields a power set $\wp(S)$ with 2^n elements. This is seen by noticing that for each of the n elements, there is a binary value determined by whether or not the element belongs to the set — hence 2^n . Note however that there is no intention to restrict the application of the power set operation to finite sets.

One final set operation is of particular interest, the Cartesian product of sets. Given two sets S and T , their **Cartesian product**, written $S \times T$, is the collection of all ordered pairs consisting of a first element from S and a second element from T . Namely, $S \times T = \{\langle s, t \rangle \mid s \in S \text{ and } t \in T\}$. In fact, this operation can be applied to any number of sets. If S_1, S_2, \dots, S_k are sets, then their **k-fold Cartesian product** is $S_1 \times S_2 \times \dots \times S_k = \{\langle a_1, a_2, \dots, a_k \rangle \mid a_i \in S_i, 1 \leq i \leq k\}$; the elements of this Cartesian product are called **k-tuples** (so the term “pair” is synonymous with 2-tuple). In contrast with ordinary sets, the order of items in a k -tuple is significant, and one item may appear repeatedly in a k -tuple. A k -tuple may also be thought of as a **sequence of length k** . We will discuss sequences more fully in the next section.

One particularly important distinction for most of the topics of this book is whether a set is finite or infinite. A set can be characterized as **infinite** if it can be placed in one-to-one correspondence with one of its proper subsets, and otherwise it is **finite**. For example, $\{2, 4, 6\}$ cannot be placed in one-to-one correspondence with any of its proper subsets. However, \mathbf{Nat} can be placed in one-to-one correspondence with the proper subset of all even natural numbers by the correspondence $n \leftrightarrow 2*n$, and this reflects the fact that \mathbf{Nat} is infinite. The concept of infinite sets will be reconsidered in the next section after functions are treated more formally.

Section 0.2: Functions and relations

The next step on the mathematical ladder is the formalization of the idea of a “relationship” between objects. This idea can be precisely described using set concepts, and we elaborate these developments now.

Definition 0.2.1: a (binary) **relation** ρ between sets S and T is just a subset of the Cartesian product, $\rho \subseteq S \times T$. We often write $s \rho t$ in place of $\langle s, t \rangle \in \rho$ and say that the relation ρ “holds” between s and t .

The relationships that we customarily work with are all modeled by this formalization. For instance, the usual magnitude order ($<$) on the Natural numbers can be thought of as the (infinite) set of all ordered pairs $\langle m, n \rangle$ where $m < n$. Actually a relation need not be binary — it may be k -ary for any $k > 0$. The **arity** of a relation (or function) is generally thought of as the number of “arguments” it requires. In the spirit of the approach here, we may have a k -fold Cartesian product, and the relation consists of a subset of k -tuples. For example, we could define the 3-ary “Pythagorean” relation between Natural numbers x , y , and z provided that $x^2 + y^2 = z^2$ so that $\text{Pythagorean}(3, 4, 5)$ holds while $\text{Pythagorean}(1, 1, 1)$ does not.

There are a number of properties of relations that will be of significance in this book. There are also some standard ways of combining relations to create other relations that we will rely on.

Definition 0.2.2: let $\rho \subseteq S \times S$ be a binary relation. Then

- ρ is **reflexive** if $s \rho s$ for all $s \in S$,

- ρ is **symmetric** if $s \rho t$ implies $t \rho s$ for all $s, t \in S$,
- ρ is **asymmetric** if $s \rho t$ and $t \rho s$ implies $s=t$ for all $s, t \in S$,
- ρ is **transitive** if $s \rho t$ and $t \rho u$ implies $s \rho u$ for all $s, t, u \in S$.

We sometimes have occasion to utilize, for example, the **transitive closure** of a binary relation ρ . This refers to the smallest relation ρ' (in the subset sense) that contains ρ and is transitive. We may make use of the other properties in a similar way. Relations that combine certain of these basic properties play a key role in many investigations. We next consider one of the most fundamental types of relation.

Definition 0.2.3: a binary relation $\rho \subseteq S \times S$ is an **equivalence relation** if it is reflexive, symmetric, and transitive. Elements of S for which ρ holds are said to be **equivalent under ρ** ; if ρ is understood from context we may just say the elements are **equivalent**. The **equivalence class** of an element $s \in S$ is the set of all elements of S that are equivalent to it and we use the notation $[s]_\rho = \{t \in S \mid s \rho t\}$; if ρ is understood from context we may just write $[s]$ for its equivalence class

In subsequent chapters of this book, there turn out to be diverse circumstances where different elements are regarded as "essentially the same" for certain purposes — that is, equivalent with respect to some equivalence relation or other. The next result about equivalence relations offers an indispensable characterization for conceptualizing such similarities.

Theorem 0.2.1: the equivalence classes of an equivalence relation $\rho \subseteq S \times S$ partition the set S into disjoint non-empty subsets whose union is S , and any such partition determines a unique equivalence relation.

Proof:

First suppose that ρ is an equivalence relation and for two equivalence classes $[s]$ and $[t]$ we have that $[s] \cap [t] \neq \emptyset$. Then there exists $r \in S$ so that $s \rho r$, and $t \rho r$. But then by symmetry, $r \rho t$ and then by transitivity $s \rho t$. But if $s \rho t$, then for any element r , $s \rho r$ if and only if $t \rho r$ so $[s] = [t]$. So two classes are either the same or they are disjoint. Also since $s \in [s]$ for each $s \in S$, the union of all the classes must yield S . Therefore the equivalence classes partition S .

Conversely, suppose there are a collection of non empty subsets Π that partition S — that is, the subsets of Π are pair-wise disjoint and their union is S . Then for $s, t \in S$ define the relation ρ by $s \rho t$ if and only if s and t are in the same set of Π . Then clearly ρ is an equivalence relation (reflexive, symmetric, and transitive) and its equivalence classes are the subsets of Π .
 \square

Definition 0.2.4: a binary relation $\rho \subseteq S \times S$ is a **partial order** if it is reflexive and transitive. If for every pair $s, t \in S$ either $s \rho t$ or $t \rho s$, then ρ is called a **total order**.

The usual magnitude comparisons of numbers, $x \leq y$, is an example of a total order since the magnitudes of any two numbers can be compared. The subset relation on sets is an excellent example of a partial order — it is reflexive and transitive, but some sets are incomparable with this relation, e.g., $\{1, 2\} \not\subseteq \{2, 3\}$, and $\{2, 3\} \not\subseteq \{1, 2\}$.

Definition 0.2.5: a **partial function** f from set S to set T , written $f: S \dashrightarrow T$, is a relation $f \subseteq S \times T$ so that if $\langle s, t_1 \rangle \in f$ and $\langle s, t_2 \rangle \in f$, then $t_1 = t_2$; S is called the **domain** of f , and T is its **range**. The **defined set** of partial function $f: S \dashrightarrow T$ is $\text{defined}(f) = \{s \in S \mid \text{there exists } t \in T \text{ so that } \langle s, t \rangle \in f\}$. If $f: S \dashrightarrow T$ is a partial function so that $\text{defined}(f) = S$, then f is called a **total function** and written $f: S \rightarrow T$. Elements of the set S are often referred to as **arguments** of the function, and those in set T as **results** of the function.

When the term *function* is used without qualification, it is normally understood to refer to a total function. Of course, a total function is just a partial function that is defined for its entire domain. Later in this book we will encounter many situations where definedness is a critical issue, and the functions involved are partial. Therefore we will need to be quite careful about the distinction. For functions, including partial functions, rather than writing $\langle s, t \rangle \in f$, we write the familiar functional notation, $f(s) = t$. Also the domain of a partial function may be a set of k -tuples, so we do not restrict ourselves to functions of a single argument. Two other properties of functions are important to note.

A (partial) function is just a special kind of relation — one where arguments uniquely determine a result (if any). Any analysis we could carry out on a general relation could also be performed on a function. However,

the property of being functional makes relations so special that most of the time we take a different perspective on their analysis.

Definition 0.2.6: a **finite sequence σ of length n** ($n \geq 0$) of elements from set S is a function $\sigma: \{1, 2, \dots, n\} \rightarrow S$. For $1 \leq i \leq n$, the i^{th} element of σ is $\sigma(i)$; usually a sequence is written as $\sigma = s_1 s_2 \dots s_n$, where $\sigma(i) = s_i \in S$; for the special case when $n=0$ we have a function with an empty domain.

This is an alternative formalization to the sequence concept. It is not unusual to formalize the same idea in several different ways. Often one alternative simplifies an aspect that is awkward when viewed other ways. For instance, one of the key basic operations performed on two finite sequences is to concatenate (or append) them to form another sequence. With the function formalization it is easy to provide a rigorous definition of the operation.

Definition 0.2.7: If σ is a sequence of length n and σ' is a sequence of length m , then their **concatenation** is the sequence of length $n+m$ written by simply juxtaposing the two sequences, $\sigma\sigma'$, and defined (as a function) by $\sigma\sigma'(i) = \sigma(i)$ if $1 \leq i \leq n$, and $\sigma\sigma'(i) = \sigma'(i-n)$ if $n+1 \leq i \leq n+m$.

While we will not take the space to do so, it is easy to formalize *infinite sequences* over set S this way too. One simply defines them to be functions from \mathbf{Nat} to S . Of course sequence properties change drastically for infinite sequences — e.g., one loses the concept of length in this case, and the basic operation of concatenation cannot be applied to infinite sequences.

Definition 0.2.8: a partial function $f: S \rightarrow T$ is an **injection** (injective, one-to-one, or 1-1) if $f(s_1) = f(s_2)$ implies that $s_1 = s_2$; also f is a **surjection** (surjective, or onto) if $T = \{t \in T \mid \text{there exists } s \in S \text{ so that } f(s) = t\}$. A total function from a set into itself that is both an injection and a surjection is called a **permutation**.

One topic where these ideas are particularly helpful is in formalizing the definition of infinite sets. For the finite sets, we can list all the members and count them. This count is referred to as the **cardinality** of a set. For infinite sets, we would never finish counting, so another approach must be used to extend the idea of cardinality to such sets. Since we cannot count their

members, the accepted method to determine if two infinite sets are equally numerous is to see if a one-to-one correspondence exists between them. For example, the set of natural numbers **Nat** is equally numerous with the set of all positive and negative integers, **Int**. This can be seen by the correspondence

Nat		Int
0	\leftrightarrow	0
1	\leftrightarrow	1
2	\leftrightarrow	-1
3	\leftrightarrow	2
4	\leftrightarrow	-2
5	\leftrightarrow	3
6	\leftrightarrow	-3
etc.		

where in general the correspondence is for an odd natural number $2n+1 \leftrightarrow$ positive integer $n+1$, and positive even natural number $2n \leftrightarrow$ negative integer $-n$ (and, of course, $0 \leftrightarrow 0$).

Two sets are said to have the **same cardinality** if they can be placed in one-to-one correspondence, and the **cardinal number** of a set is regarded as the class of all sets that can be placed in one-to-one correspondence with the set. For finite sets this is identical to saying the two sets have the same number of members. But for infinite sets, there is no “number” (i.e., member of **Nat**) of members. For this reason, names are assigned to some infinite cardinalities. The cardinality of **Nat** is called \aleph_0 (pronounced alaph null) — that is, any set which can be placed in one-to-one correspondence with **Nat** is said to have cardinality \aleph_0 and is called a **countably infinite set**. So, for instance, the even Natural numbers are countably infinite (via correspondence $2n \leftrightarrow n$).

The set of all real numbers has cardinality **c** (for continuum). G. Cantor was the first to convincingly demonstrate that not all infinite sets are equally numerous. He proved that the natural numbers are not as numerous as the set of positive real numbers (i.e., $\aleph_0 < c$) by a technique known as *diagonalization*. This is a proof strategy that we will find essential in another context later in the book, so we present his proof here.

Theorem 0.2.2: $\aleph_0 < \mathfrak{c}$.

Proof (by contradiction):

Clearly **Nat** is a subset of the real numbers and so there is a 1-1 correspondence to a subset of the reals, and so $\aleph_0 \leq \mathfrak{c}$. We prove that **Nat** cannot even be put in one-to-one correspondence with all the real numbers x where $0 \leq x \leq 1$, much less than all real numbers (the real numbers between 0 and 1 can be placed in 1-1 correspondence with all real numbers, a fact we do not prove here). Suppose that there is a one-to-one correspondence, say

$$\begin{array}{lcl} 0 & \leftrightarrow & x_1 = .d_{11}d_{12}d_{13} \dots \\ 1 & \leftrightarrow & x_2 = .d_{21}d_{22}d_{23} \dots \\ \dots & & \dots \\ n-1 & \leftrightarrow & x_n = .d_{n1}d_{n2}d_{n3} \dots \\ \dots & & \dots \end{array}$$

where for each Natural number $n-1$, we indicate the (infinite) decimal expansion $.d_{n1}d_{n2}d_{n3} \dots$ of its corresponding real number (so the real number 1 is expressed as $.999 \dots$). Then consider the real number r whose decimal expansion $.e_1e_2e_3 \dots$ is defined as follows: $e_k = 5$ if $d_{kk} \neq 5$, and $e_k = 6$ if $d_{kk} = 5$ for $k = 1, 2, 3, \dots$. Now every real number, $0 \leq x \leq 1$, must appear among x_1, x_2, x_3, \dots . However, r is a real number which differs from all these since it is constructed so that the k^{th} digit of its decimal expansion, e_k , differs from the k^{th} digit of the decimal expansion of x_k , d_{kk} , for every k . But this contradicts the assumption that we have a one-to-one correspondence with all the real numbers, $0 \leq x \leq 1$. Hence no such correspondence can exist.

□

Ordinary arithmetic of Natural numbers can be regarded as cardinal operations on finite sets. For instance, if we take the union of a set with two elements and a disjoint set with three elements, we obtain a set with five elements. However, the rules of arithmetic for infinite cardinals are rather different. For instance, if we can take the union of two disjoint, countably infinite sets the result is another countably infinite set — e.g., the union of the even Naturals and the odd Naturals. That is, $\aleph_0 + \aleph_0 = \aleph_0$. The set operation corresponding to multiplication in arithmetic is Cartesian product — for finite sets the cardinality of the Cartesian product is the product of the

cardinalities of the component sets, e.g., the Cartesian product of a set with two elements and a set with three elements results in a set with six elements. However, again for infinite sets a different behavior is obtained as $\aleph_0 \times \aleph_0 = \aleph_0$. There is also a set counterpart to exponentiation in arithmetic. For finite sets forming the set of all (total) functions from a set S with n elements to a set T with m elements, yields a set $\{f \mid f:S \rightarrow T\}$ with n^m elements. This is verified by noting that for each of the n possible arguments there are m choices for the functional image, and each choice leads to a distinct function. Here when we pass to the infinite case however, $\aleph_0^{\aleph_0} \neq \aleph_0$ as we next prove (in fact $\aleph_0^{\aleph_0} = \mathfrak{c}$).

Theorem 0.2.3: the set of all functions from Natural numbers to Natural numbers is uncountable .

Proof:

Consider the set $\text{fun}(\mathbf{Nat}) = \{f \mid f: \mathbf{Nat} \rightarrow \mathbf{Nat}\}$. Now suppose that $\text{fun}(\mathbf{Nat})$ is countable and let an enumeration of all these functions be f_1, f_2, f_3, \dots . Then define a function $g : \mathbf{Nat} \rightarrow \mathbf{Nat}$ where for each $n \in \mathbf{Nat}$

$$g(n) = \begin{cases} 2 & \text{if } f_n(n) = 1 \\ 1 & \text{if } f_n(n) \neq 1 \end{cases}$$

Now clearly g is a well defined element of $\text{fun}(\mathbf{Nat})$, however, g is missing from the enumeration as for each $n \in \mathbf{Nat}$, $g \neq f_n$ since $g(n) = 2$ if $f_n(n) = 1$, and $g(n) = 1$ if $f_n(n) \neq 1$. Hence if we assume that $\text{fun}(\mathbf{Nat})$ is countable we reach a contradiction, and so $\text{fun}(\mathbf{Nat})$ must be uncountable. Notice that this is another diagonalization argument.

□

We include one last result about cardinality in this section.

Theorem 0.2.4: the power set of the Natural numbers, $\wp(\mathbf{Nat})$, is uncountable.

Proof:

We use the diagonalization technique once more. Suppose there is a 1-1 correspondence between \mathbf{Nat} and $\wp(\mathbf{Nat})$, say $n \leftrightarrow S_n \subseteq \mathbf{Nat}$, so S_0, S_1, S_2, \dots is an enumeration of *all* subsets of \mathbf{Nat} . Then define $S \subseteq \mathbf{Nat}$ by $S =$

$\{k \in \mathbf{Nat} \mid k \notin S_k\}$. But then for every k , $S \neq S_k$ since if $k \in S$, then $k \notin S_k$, and if $k \notin S$, then $k \in S_k$. Hence there is a contradiction and so there can be no 1-1 correspondence between \mathbf{Nat} and $\wp(\mathbf{Nat})$.

□

As we saw in Theorem 0.2.2, the set of real numbers is not countably infinite. Theorem 0.2.3 tells us that the same is true of the set of all functions from Natural numbers to Natural numbers, and Theorem 0.2.4 shows the same is the case for the power set of the Natural numbers. We will see later that some of the limitations of computational mechanisms can be inferred from a straight-forward “counting” analysis. Just basic cardinality comparisons will provide an indication of limitations that must exist.

Section 0.3: Graphs and Trees

The graph topic involves an assemblage of a number of subtly different ideas. Graphs provide us with a more visual perspective of the relation concept. From one point of view there is no difference at all between a binary relation and a graph. However, the way in which we use the graph formalism is much different than the relation formalism.

Definition 0.3.1: a **directed graph** is a pair $G = (V, E)$ where V is a set called the **vertices** (or **nodes**) of the graph, and $E \subseteq V \times V$ is a set of ordered pairs from V called the **edges**. If V is finite, G is called a **finite directed graph**.

Usually we consider finite graphs, but there are situations where infinite graphs are of interest. A directed graph has edges connecting its vertices that have an orientation since an edge is an *ordered* pair of vertices. The edges can be regarded as a binary relation on the set of vertices. Generally, questions about graphs revolve around how to determine “path properties” from the edge relation.

Definition 0.3.2: a (finite) **path of length k** from vertex u to vertex v , $\pi(u, v)$, in a directed graph G is a (finite) sequence of $k \geq 0$ edges, $\langle u_1, v_1 \rangle, \langle u_2, v_2 \rangle, \dots, \langle u_k, v_k \rangle$ so that $u = u_1$, $v = v_k$, and $v_i = u_{i+1}$ for all $1 \leq i < k$. A path provides a

multi-step connection between two nodes. If $\pi(u,v)$ has no repeated edges, it is called a **simple** path. If $u=v$, π is called a **cycle**.

Definition 0.3.3: a (non directed) **graph** $G = (V, E)$ is a directed graph such that E is a symmetric relation.

In a graph, whenever there is an edge $\langle u, v \rangle$, there is *always* an edge $\langle v, u \rangle$. For this reason these two directed edges are usually thought of as a single unoriented edge.

In a graph, the relation between two nodes of being connected by a path is an equivalence relation, and the equivalence classes are called **connected components**.

Definition 0.3.4: a graph G is called **connected** if there is a path between every pair of nodes (i.e., there is one connected component).

In a directed graph there may be a path $\pi(u,v)$ but no path $\pi(v,u)$ in some cases. If the edge relation is not symmetric, the path relation is not an equivalence relation, and the idea of connected components is lost. Instead we consider the **reachability set** of a node — $\text{reach}(u) = \{v \mid \exists \text{ path } \pi(u,v)\}$.

Definition 0.3.5: a non directed graph that is connected and has no simple cycles is called a **tree**. Often trees are **rooted** — this just means that one of the nodes is distinguished as the **root**.

The hierarchical relationship inherent in a tree structure makes it of central importance. Trees are quite special structures as reflected in the following result.

Theorem 0.3.1: Let $G = (V, E)$ be a finite (non directed) graph with e edges and $v \geq 1$ vertices. Then the following conditions are logically equivalent:

- (1) G is a tree,
- (2) G is connected and $e = v - 1$,
- (3) G has no simple cycles, and if an edge is added between any two vertices not joined by an edge, then a simple cycle is formed,
- (4) G is connected and the deletion of any edge leaves G disconnected,
- (5) every pair of vertices is connected by a unique simple path.

For some applications this graph concept of a tree captures just the right sense of things. For instance in ancestral relationships, each person has two parents and there is no compelling reason to distinguish one as coming before the other. On the other hand, in an arithmetic expression such as $A-B*C$ depicted as the tree in Figure 0.3.1, the order between the two child nodes of the root is a critical aspect of the description. Reversing that order yields the tree in Figure 0.3.2 which is another expression altogether, namely $B*C-A$.

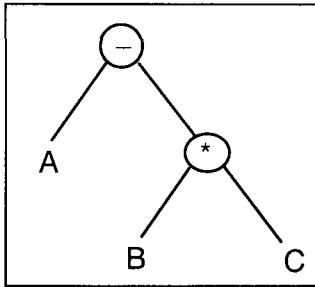


Figure 0.3.1.

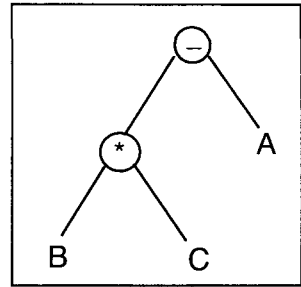


Figure 0.3.2.

When trees are used for such an application (and numerous others), there is an order to the child nodes that is essential to recognize. Such trees are called **oriented trees**. The formalism for oriented trees must incorporate a means to describe a total order between the child nodes of each node of the graph. This may be accomplished by adding this information to the definition of an ordinary tree. Another way of doing this is given in the next definition.

Definition 0.3.6: an **oriented tree** T is a prefix closed subset of finite sequences from \mathbf{Nat} , $T \subseteq \mathbf{Nat}^*$; the set T is **prefix closed** if for each $x, y \in \mathbf{Nat}^*$ so that $xy \in T$, $x \in T$. The elements of T constitute the vertices, and for each $kx \in T$ where $k \in \mathbf{Nat}$ there is an edge to x which is necessarily also a member of T (and ϵ is the root). Vertices are oriented by the lexicographical ordering of sequences.

Section 0.4: Proof Techniques

In this section we briefly review the important proof methods which are used in this book. We recall the predominant methods and show an example of each of these proof strategies. After illustrating various proof techniques, we conclude with some comments intended to clarify the role we adopt for proofs in this book. Since this book develops the deductive aspects of the discipline of computer science, it is vital that the reader be acquainted with the basic tools of formal reasoning. If the presentation in this section cannot be followed with relative ease, it is an indication that the reader does not have the background assumed in the body of the book.

Logical connectives and inference rules

Logical connectives are the basic means of combining assertions into other compound forms. The notations we use for the usual logical connectives are as follows:

- \wedge -- and,
- \vee -- or,
- \sim -- not,
- \supset -- implication.

These logical operations have the familiar truth-table definitions given below, using T and F as the symbols for true and false, respectively:

		\wedge	\vee	\supset
T	T	T	T	T
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

	\sim
T	F
F	T

Proving the truth of logical implications is a task we will face over and over again. It is especially important to fully absorb the definition of implication, $\alpha \supset \beta$ (if α then β , or α implies β). When the hypothesis α is false, the implication is regarded as having no force and it therefore cannot be wrong. We sometimes say the implication is *vacuously true* in these cases. Therefore the proof of the truth of an implication can ignore the circumstance when the hypothesis is false since in these cases the implication is true “vacuously” and no argument is required. To establish the truth of an implication, we need only show that whenever the hypothesis is true, the conclusion must also be true.

Rules of inference are unfailing methods for ascertaining the truth of new assertions from known truths. Rules of inference are sometimes presented in the schematic form

$$\frac{\text{hypothesis}}{\text{conclusion,}}$$

where “hypothesis” denotes the known truth, and “conclusion” is the assertion whose truth is deduced from it.

For instance, to formalize reasoning about implication we have the familiar rule of *modus ponens* that is depicted as

$$\frac{\tau \wedge (\tau \supset \rho)}{\rho} \quad (\text{modus ponens}).$$

An assertion that is either true or false is usually referred to as a **proposition**. The logical connectives indicated here, together with the rules governing them, are referred to as the **propositional calculus**.

Quantification.

Commonly when we make a logical assertion, the elements referred to by the assertion have “intentions” associated with them. The goal of precision in our statements makes it essential to express these intentions explicitly. There are two basic intentions that recur and provide the basis for the expression of all others. A specific notation is commonly accepted for writing them succinctly. For instance, when we write an equation such as $x^2 - 2x + 1 = 0$, we do not intend to assert that *all* numbers x have this property. Rather we intend to single out certain numbers with this property, if they exist. Parameterized assertions such as this that may be true for some arguments and false for others are called **predicates**. A predicate may have one or more parameters (a proposition can be regarded as a predicate with no parameters). For instance, $x > y$ is a predicate with two parameters. We expect that by assigning arguments to each of its parameters, an assertion becomes either true or false — $2 > 1$ is true and $1 > 2$ is false.

The first of the standard intentions is referred to as **existential quantification**, and is written with the symbol \exists . It is used to assert that

values of a parameter exist that result in a true proposition. For instance, we write $\exists x(x^2 - 2x + 1 = 0)$ [read: there exists x such that ...] is the formal logical formula that asserts that this equation has a solution, a true proposition (over the real numbers) in this case. When a parameterized predicate is quantified, we obtain a proposition and this may be true or false — e.g., $\exists x(x^2 + 1 = 0)$ is false (assuming the universe of real numbers).

The existential sense does not apply when we write an assertion such as $x^2 \geq 0$. In a case such as this we intend that this claim is true for *all* values. This is the second of the standard intentions — **universal quantification**, written with the symbol \forall . It is used to assert that a true proposition results for all values of the parameter. For instance, we write $\forall x(x^2 \geq 0)$ [read: for all x ...]. Of course, this may also result in a true or false proposition.

When making assertions concerning predicates, we must always have a clear understanding of the domain from which the parameter values are drawn. The rules governing the quantifications discussed here are referred to as the **predicate calculus**. Since applying quantification to a predicate may result in a proposition, the propositional calculus is included as a basic portion of the predicate calculus. In addition, there are rules that pertain to quantification. For a predicate $p(x)$, the two primary instances of predicate calculus rules are the logical equivalencies of $\forall x(p(x))$ and $\sim(\exists x(\sim p(x)))$, and $\exists x(p(x))$ and $\sim(\forall x(\sim p(x)))$.

Proof by induction.

Induction is the most commonly used proof strategy in this book. It has been referred to as the mathematical version of domino theory. Induction applies when there is a parameterized family of assertions $\tau(n)$, one for each $n \in \mathbf{Nat}$. To show that the assertion $\tau(n)$ is true for each $n \in \mathbf{Nat}$, we are faced with demonstrating infinitely many proofs! Instead of doing that we prove *two* things: (i) $\tau(0)$ is true — this is called the *basis*, or *anchor*, and (ii) assuming the truth of τ for each argument (the *induction hypothesis*), implies its truth for the next, $\tau(n) \supset \tau(n+1)$ — this is called the *induction step*. Informally, we have $\tau(0)$ true and $[\tau(0) \supset \tau(1)]$ true, so we can conclude $\tau(1)$ true; but then $\tau(1)$ true and $[\tau(1) \supset \tau(2)]$ true allows us to conclude $\tau(2)$ is true, etc. So by this rule of deduction, from the truth of $\tau(0)$ and $\forall n [\tau(n) \supset \tau(n+1)]$ we conclude the truth of $\forall n [\tau(n)]$. In scheme form

$$\frac{\tau(0) \wedge \forall n [\tau(n) \supset \tau(n+1)]}{\forall n [\tau(n)]} \quad (\text{simple induction}).$$

We illustrate this proof strategy for the

Theorem 0.4.1: The sum of the first n odd natural numbers is n^2 .

Proof — by induction:

Formally, the parameterized assertion is: for all $n \geq 0$, $n^2 = \sum_{i=1}^n (2i - 1)$.

Anchor case — $n=0$:

The sum of zero odd integers is 0 and $0^2 = 0$.

Induction step: assume that $n^2 = \sum_{i=1}^n (2i - 1)$

Then by the induction hypothesis and basic algebra $\sum_{i=1}^{n+1} (2i - 1) =$
 $(\sum_{i=1}^n (2i - 1)) + 2(n+1) - 1 = n^2 + 2n + 1 = (n+1)^2$, and so the induction is

extended. Hence by the principle of induction, our result is true for all n .

□

Simple induction has several variants, and we will eventually use them all. For instance, sometimes a stronger inductive hypothesis is necessary, and there are several basis cases as in

$$\frac{\tau(0) \wedge \tau(1) \wedge \forall n [\tau(n) \wedge \tau(n+1) \supset \tau(n+2)]}{\forall n [\tau(n)]} \quad (\text{extended induction}).$$

The proof of an assertion $\forall n [\tau(n)]$ by this extended induction can be regarded as a proof of $\forall n [\tau'(n)]$ by simple induction, where $\tau'(n)$ is defined to be the assertion $\tau(n) \wedge \tau(n+1)$ since then $\tau'(0)$ is synonymous with $\tau(0) \wedge \tau(1)$ and $\forall n [\tau'(n) \supset \tau'(n+1)]$ is synonymous with $\forall n [[\tau(n) \wedge \tau(n+1)] \supset [\tau(n+1) \wedge \tau(n+2)]]$ which is logically equivalent to $\forall n [[\tau(n) \wedge \tau(n+1)] \supset \tau(n+2)]$.

Proof by contradiction.

This is also sometimes referred to as indirect proof. With this proof technique, the idea is to start from some assumption and deduce a known falsehood. If each step is an indisputable consequence of the previous, and we are led to a false conclusion, it must be that the original assumption cannot be true. So with this strategy we start by assuming the negation of the assertion we wish to prove. When we are then able to deduce a known falsehood, it must be that the negation we assumed is not true, and therefore the assertion itself is. Note that we make no intuitive commitment to the actual truth of the starting assumption (quite the contrary) — rather we seek to determine what implications its truth might have. Schematically,

$$\begin{array}{l} \sim\tau \supset \text{false} \\ \text{-----} \\ \tau \end{array} \quad (\text{contradiction}).$$

As a sample we offer the proof of the

Theorem 0.4.2: There are no whole positive numbers m, n ($m, n \in \mathbf{Nat}$ with $m, n > 0$) so that $m^2 = 2n^2$ (i.e., doubling a perfect square never produces another perfect square)

Proof -- by contradiction:

Assume that such pairs of numbers exist, and let m be the smallest value so that for some number n , $m^2 = 2n^2$. Then since m^2 is even, m must be even — note that this small fact is also proven by contradiction since an odd number multiplied by an odd number is odd, hence if m is odd, m^2 is too. Therefore take $m = 2p$. Then $2n^2 = m^2 = (2p)^2 = 4p^2$ and so $n^2 = 2p^2$. Again since n^2 is even, n must be even, say $n = 2q$. But then $2p^2 = n^2 = (2q)^2 = 4q^2$, so $p^2 = 2q^2$. However, $p < m$ (since $n^2 = 2p^2$ implies that $p < n$, and $m^2 = 2n^2$ implies $n < m$) and this contradicts m being chosen to be the smallest number with this property. Hence it must be that no such pairs of numbers exist.

□

Proof by construction.

The last proof strategy we mention is a bit different than the others. This is a strategy that we employ when the assertion we seek to prove is an existentially quantified statement. In this case we can prove the assertion by identifying one object that satisfies the statement. The determination of an

object that satisfies the statement (together with the justification of this satisfaction) is usually called a construction. Schematically,

$$\frac{\tau(a)}{\exists x [\tau(x)]} \quad (\text{construction}).$$

As a sample of this approach, consider the

Theorem 0.4.3: for every natural number n , there exists a prime number $p > n$ (i.e., there are infinitely many prime numbers).

Proof — by construction:

Given n , let p be the smallest prime factor of $1+n!$. Now $1+n! = 1+(2*3*4* \dots *n)$ must yield the remainder of 1 when divided by 2, 3, 4, ..., n . Hence $1+n!$ has no factors that are as small as n , and so $p > n$.

□

What is a proof?

In this book we use a careful but informal style of proof (if you don't find them “informal”, formal logic proofs would persuade you — see the suggested readings at the end of the chapter). Our goal is to convince another (perhaps even oneself) that there is an indisputable chain of inference from known truths to the point in question. Overlooking possible cases, or leaving gaps in a chain of reason is not acceptable. If a claim is made that one thing follows “clearly” from another, but the reader requires an extended effort to verify this, then something has gone wrong. This is a situation we seek to avoid. One may “know” that one assertion surely follows from another, but this is intuition speaking; it is important to be able to confirm that intuition with step by step justification.

However, we usually do not prefer to see every last detail of a proof either. By analogy, we might visualize a Pascal version and a machine language version of the same program. We normally prefer to judge the correctness of the behavior of the program from the Pascal version even though many issues are more thoroughly elaborated in the machine language version. While strictly speaking they may be required, too many details will cloud the logic of an argument rather than clarifying it. Omitting aspects that the reader can routinely supply, frequently *improves* the presentation. For instance, if one wants to sort the values in an array to facilitate the remaining considerations, it is reasonable to state just that much — “from now on

assume the array is sorted.” Of course, there are many ways to do sorting, and the best way in a given situation may require careful analysis. But including all of the analysis about how to do the sort correctly will distract the reader from the main line of the argument.

So what is the answer? It is that we do not have an unequivocal characterization of what we wish to call a proof! The level of detail is a matter of judgment, convention, and experience. Remember that the basic goal is to convince an informed reader of the incontestable truth of the assertion (so the reader's assumed background enters as a variable). If we are appealing to a known or previously proven result, we should make this clear. It is good form to always help the reader with comments about the general organization of an argument right at the beginning — e.g., induction proof, proof by contradiction, etc. This helps to align the reader with the kind of detail the writer may find routine and choose to omit. The ultimate aim of rigor and proof is greater precision in understanding. Understanding is associated with good intuition about the area under consideration. Proof involves being able to verify intuition with technical details *as appropriate*. The body of this book provides numerous proof examples, and the reader should have had prior experience as well. When called on to create a proof, consider all of these sources as well as your own sense of what is irresistibly convincing. One of the consequences of studying this book should be a significant improvement in the reader's ability to understand and construct proofs.

Formulating proofs.

One of the principle activities in this book is the construction of proofs. This is the process of convincing ourselves of the inescapable consequences of certain assumptions. This process not only verifies the stated assertions, it yields insight as well — we not only see *what* must be true, we see *why* it must be true. Reading the proofs provided in the body of this book is a first step to gaining insight into the material treated. But even greater insight can be obtained by progressing to the level where one constructs the proof rather than reading someone else's. This is rather analogous to the process of learning to program computers. One can learn to read the programs written by others, even quite complicated programs. But this does not prepare one to write programs, even fairly simple ones. It is essential to engage in the program writing activity in order to obtain a deeper understanding of programming. Similarly with proving, it is impossible to get a solid grasp

without yourself constructing proofs. To this end, many exercises are provided at the end of each chapter and the reader is urged to give some of them a try. Sample solutions to selected exercises are provided in the back of the book.

The analogy between proving and programming has more substance than just the motivation mentioned in the previous paragraph. There is a good case supporting the similarity. In fact there is a research area of artificial intelligence called “automatic programming” (or “automatic program synthesis”) that exploits this connection. The goal of this research area is to replace the human activity of program writing by having an automated computer system that writes the programs! The approach that is taken is to provide an input/output specification that describes the desired relationship between inputs and outputs — a “pre-condition” describes requirements placed on the inputs (e.g., $X \geq 0$) and a “post-condition” describes the relationship with the resulting outputs (e.g., $Y^2 = X$ for a program computing the square root Y of input X). An automatic theorem prover is used to show that the existence of outputs satisfying the desired post-condition is a logical consequence of the pre-condition. Then from this existence proof, a program to effect the transformation that creates such outputs is mechanically extracted. The interested reader can consult [M-W 92] for an elaboration. The point for us is that writing proofs is in many ways similar to writing programs, so programming skills can be helpful in the proving activity and vice-versa.

Section 0.5: Suggestions for Further Reading.

Many texts on discrete structures provide an extended treatment of the topics reviewed in this chapter, for instance, [Epp 90, Hei 95, KBR 96]. Moreover each of these topics has been individually studied in depth and written about extensively, so many additional sources can be easily located.