Lecture Notes for CSCI C241/H241

Induction, Recursion and Programming*

Steven D. Johnson
Computer Science Department
Indiana University School of Informatics

*Induction, Recursion, and Programming is a Working title. Some content derives from the book Induction, Recursion, and Programming by Mitchell Wand (North-Holland, 1976). All rights to original content are reserved by Mitchell Wand.
## Contents

1 **Sets** 1
   1.1 Set Builder Notation .................................. 3
   1.2 Set Operations ........................................ 4
   1.3 Languages ............................................ 8
   1.4 A Simple Algorithmic Language ....................... 11

2 **Relations, Functions** 15
   2.1 Functions ............................................. 16
      2.1.1 Function (and Relation) Composition ............ 19
      2.1.2 Infix Notation .................................. 21
   2.2 Relations on a Single Set ............................ 23
      2.2.1 Attaching Information to Graphs ............... 26
   2.3 Trees .................................................. 27
   2.4 DAGs .................................................. 33
   2.5 Equivalence Relations ................................ 35
   2.6 Partial Orders* ....................................... 38

3 **Propositional Logic and Boolean Algebra** 41
   3.1 Propositions and Truth Tables ....................... 41
      3.1.1 Implication* ..................................... 41
   3.2 Truth Tables .......................................... 44
      3.2.1 Logical Equivalence ............................ 45
   3.3 Boolean Algebra ....................................... 47
      3.3.1 Duality ........................................... 49
   3.4 Normal Forms .......................................... 51
      3.4.1 Decision Diagrams* .............................. 52
   3.5 Application of Boolean Algebra to Hardware Synthesis* .. 55

4 **Counting** 61
   4.1 Extended Operations .................................. 61
   4.2 Cardinality ........................................... 63
   4.3 Permutations and Combinations ....................... 64
Chapter 1

Sets

The concepts of set and set membership are fundamental. A set is determined by its elements (or members) and to say that \( x \) is an element of the set \( S \), we write:

\[ x \in S \]

To say that \( y \) is not a member of \( S \) we write:

\[ y \notin S \]

Simple sets are specified by listing all of their elements between braces. A set \( A \) of five numbers is specified:

\[ A = \{1, 2, 3, 4, 5\} \]

A set \( B \) of three colors is specified:

\[ B = \{\text{red, blue, green}\} \]

A set \( C \) of three symbolic color names is specified

\[ C = \{\text{red, blue, green}\} \]

This book uses teletype font for concrete syntax, the purely typographical form of a symbol.

Example 1.1 List the set \( S \) of whole numbers between 1 and 15 which are evenly divisible by either 2 or 3.

Solution: We might begin by listing the numbers divisible by 2:

\[ \{2, 4, 6, 8, 10, 12, 14, \ldots \} \]

and then the numbers divisible by 3:

\[ \ldots, 3, 6, 9, 12, 15\} \]
So the set we are looking for could be written
\[ S = \{2, 4, 6, 8, 10, 12, 14, 3, 6, 9, 12, 15\} \]

This listing contains duplications and the numbers are listed in a strange order. Neither of these problems makes the description incorrect, but it is less confusing to list each element just once. A better description for \( S \) is
\[ \{2, 3, 4, 6, 8, 9, 10, 12, 14, 15\} \]

One way to abbreviate a long list of elements is to use ellipses to indicate a large, possibly infinite, range of values. For example, the set of lower-case letters from \( a \) to \( z \) could be expressed as follows:
\[ \{a, b, \ldots, z\} \]

The set of numbers ranging from 1 to 10,000 could be specified:
\[ \{1, 2, 3, \ldots, 10000\} \]

The following definition uses ellipses to describe some infinite sets that are used throughout this book.

**Definition 1.1**

(a) The set of whole numbers is \( \mathbb{W} = \{1, 2, 3, \ldots\} \).

(b) The set of natural numbers is \( \mathbb{N} = \{0, 1, 2, 3, \ldots\} \).

(c) The set of integers is \( \mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, 3, \ldots\} \).

Ellipses are useful when it is natural to list a set’s elements in some consecutive order, but care is needed in their use. Consider the set specification:
\[ B = \{2, 4, \ldots, 64\} \]

It is not completely obvious from this set definition whether the elements of \( B \) are:

- the even numbers from 2 to 64:
  \[ B \equiv \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64\}, \]

- or the powers-of-2 from 2 to 64:
  \[ B \equiv \{2, 4, 8, 16, 32, 64\}, \]
or something else.

The rule-of-thumb is to take the simplest sequence that clearly describes the set exactly, but “simplest” is a matter of judgment involving assumptions about the knowledge of the Reader. One way to avoid confusion is to include a formula representing a typical element of the list; for example:

\[
B = \{2, \ldots, 2i, \ldots, 64\}
\]

This description says that the elements of \(B\) have the form \(2i\); they are even numbers. Although even this definition relies on the reader to understand that \(i\) refers to a whole number, the specification is better determined.

### 1.1 Set Builder Notation

A more general way to specify the elements of a set is to write down a property that is satisfied by the elements of the set—and only by those elements. The notation for doing this is called *set builder notation*:

\[
\{x \in U \mid P[x]\}
\]

\(U\) is the *universe* or *domain* from which prospective elements \(x\) originate, and \(P[x]\) stands for some property that the members—and only the members—must satisfy. The property \(P\) must “make sense” with respect to \(U\); that is, is, \(P[u]\) must be either *true* or *false* for every element of \(U\). The set specified contains all of the elements for which \(P[x]\) is *true*. Mention of \(U\) may be omitted if either the surrounding context or \(P\) make clear what \(U\) is.

For example, the second version of \(B\) above might be specified

\[
B = \{x \in \mathbb{W} \mid x = 2^i \text{ for } i \in \mathbb{W} \text{ such that } 1 \leq i \leq 6\}
\]

The property \(P[x]\) in this case is

\[
P[x] \equiv \text{“}x = 2^i \text{ for some } i \in \mathbb{W} \text{ such that } 1 \leq i \leq 6\text{”}
\]

Even though \(P\) contains two variables, \(x\) and \(i\), it is nevertheless a statement about \(x\) only; \(P\) quantifies \(i\), stating \(i \in \{1, 2, 3, 4, 5, 6\}\).

In this instance, the set-builder description is not much of an improvement over simply listing the elements, although if \(i\) ranged from 1 to 100 it would be. We can do a little better by writing a formula in place of the simple variable \(x\):

\[
B = \{2^i \mid i \in \mathbb{W} \text{ and } 1 \leq i \leq 6\}
\]

we have omitted the declaration “\(2^i \in \mathbb{W}\)” because it is clear from the context of the example that \(B\) is a set of whole numbers.

**Remark.** You may have noticed that property \(P\), above, is defined using ‘[’, ‘]’ and ‘≡’, rather than ‘(’, ‘)’ and ‘=’. There is no difference in meaning, but throughout this book the \(P[x] \equiv “\ldots”\) notation is used for the purpose of linguistic identification, that is, naming formulas rather than the values they denote.
Example 1.2 Specify the infinite set $E$ of nonnegative even numbers using ellipses and then again using set-builder notation

Solution: Using ellipses, one could write $E = \{0, 2, 4, 6, \ldots \}$ It would be clearer to include a representative element $E = \{0, 2, \ldots , 2i, \ldots \}$. Using set-builder notation, we would write $\{2n \mid n \in \mathbb{N}\}$. □

Here are some other numerical set names used in this book.

Definition 1.2

(a) The set of integers modulo $n$ is $\mathbb{Z}_n = \{0, 1, \ldots , n-1\}$.

(b) The set of rational numbers, $\mathbb{Q}$, consists of all fractions: $\mathbb{Q} = \{ \frac{n}{m} \mid n \in \mathbb{N} \text{ and } m \in \mathbb{W} \}$

(c) The real numbers,† $\mathbb{R}$.

1.2 Set Operations

There is that set which contains no elements. One way to express this set is to place nothing between braces: $\{\}$. We also use the symbol $\emptyset$ for this set.

Definition 1.3 The empty set, denoted by $\emptyset$, has no elements: $\emptyset = \{\}$.

A common mistake is writing “{$\emptyset$}” for the empty set. However, $\{\emptyset\}$ is not the “really empty” set but rather a set with a single element, namely, $\emptyset$.

Sets are compared by asking what elements they have in common.

Definition 1.4 Let $A$ and $B$ be two sets.

(a) $A$ equals $B$, written $A = B$, if $A$ and $B$ contain exactly the same elements.

(b) $A$ contains $B$, written $B \subseteq A$, if every element of $B$ is also an element of $A$. $A$ is said to properly contain $B$ when $B \subseteq A$ and $B \neq A$. This is sometimes written as $B \subset A$.

(c) $A$ and $B$ are disjoint when they have no elements in common, that is, $A \cap B = \emptyset$.

†Despite the fact we have learned about and worked with real numbers most of our lives, it is hard to state a concise property explaining just what a real number is, $\mathbb{R} = \{x \mid R(x)\}$. Two dictionary definitions are

1. Every real number can be expressed with an infinite decimal expansion, for instance, $\frac{1}{3} = 0.333\ldots$

2. Every real number is the limit of an infinite sequence of rational numbers.

While true, both of these properties raise more questions than they answer.
1.2. SET OPERATIONS

To prove that two sets, \( A \) and \( B \) are equal, one often shows that each contains the other. The following proposition follows immediately from Definition 1.4.

**Fact 1.1** \( A = B \) iff \( A \subseteq B \) and \( B \subseteq A \).

There are numerous ways to build new sets from sets which are given. The most common of these have names and special symbols associated with them as defined below.

**Definition 1.5** Let \( A \) and \( B \) be two sets.

(a) The intersection of \( A \) and \( B \), written \( A \cap B \), is the collection of elements that \( A \) and \( B \) have in common. That is,

\[
A \cap B = \{ x \mid x \in A \text{ and } x \in B \}
\]

(b) The union of \( A \) and \( B \), written \( A \cup B \), is the collection of all those elements in either set or both. That is,

\[
A \cup B = \{ x \mid x \in A \text{ or } x \in B \}
\]

c) The (set) difference of \( A \) and \( B \), written \( A \setminus B \) is the collection of those elements of \( A \) which are not in \( B \). That is,

\[
A \setminus B = \{ x \mid x \in A \text{ and } x \notin B \}
\]

d) The power set of \( A \) written \( \mathcal{P}(A) \), is the collection of \( A \)'s subsets. That is,

\[
\mathcal{P}(A) = \{ S \mid S \subseteq A \}
\]

e) The product of \( A \) and \( B \), written \( A \times B \) is the collection of all ordered pairs whose first elements come from \( A \) and whose second elements come from \( B \). That is,

\[
A \times B = \{ (a, b) \mid a \in A \text{ and } b \in B \}
\]

**Example 1.3** Let \( A \) be the set \( \{1, 5\} \); and let \( B \) be the set \( \{1, 2, 3\} \). Describe the sets \( A \cap B, A \cup B, A \setminus B, \mathcal{P}(A), A \times B, B \times A, \) and \( A \times \mathbb{N} \).

**Solution:** The sets are

\[
A \cap B = \{1\}
\]

\[
A \cup B = \{1, 2, 3, 5\}
\]

\[
A \setminus B = \{5\}
\]

\[
\mathcal{P}(A) = \{\emptyset, \{1\}, \{5\}, \{1, 5\}\}
\]

\[
A \times B = \{(1, 1), (1, 2), (1, 3), (5, 1), (5, 2), (5, 3)\}
\]

\[
B \times A = \{(1, 1), (1, 5), (2, 1), (2, 5), (3, 1), (3, 5)\}
\]

\[
A \times \mathbb{N} = \{(1, n) \mid n \in \mathbb{N}\} \cup \{(5, n) \mid n \in \mathbb{N}\}
\]

\[
= \{(1, 0), (5, 0), (1, 1), (5, 1), \ldots (1, i), (5, i), \ldots\}
\]
In listing the elements of $A \cup B$, each distinct element is written just once. Consult Definition 1.4 to verify that each element of $\mathcal{P}(A)$ is, in fact, a subset of $A$.

Ordered pairs $(1, 3)$ and $(3, 1)$ are unequal, for while they contain the same numbers, these numbers are in a different order. Thus, $A \times B$ and $B \times A$ are distinct sets because, for instance, $(1, 3) \in A \times B$ but $(1, 3) \notin B \times A$. However $A \times B$ and $B \times A$ are not disjoint; they share the element $(1, 1)$.

**Example 1.4** Let $A = \{a, b\}$; let $B = \{0, 1\}$; and let $C = \{1, 3\}$. Compare the sets $(A \times B) \times C$ and $A \times (B \times C)$

**Solution:** First,

$$A \times B = \{(a, 0), (a, 1), (b, 0), (b, 1)\}$$

Now, the set $(A \times B) \times C$ is a set of ordered pairs, each of which has an ordered pair in its first position:

$$(A \times B) \times C = \{( (a, 0), 1), ( (a, 1), 1), ( (b, 0), 1), ( (b, 1), 1),
( (a, 0), 3), ( (a, 1), 3), ( (b, 0), 3), ( (b, 1), 3) \}$$

Elements of the set $A \times (B \times C)$ have the same “information content” but a different structure:

$$A \times (B \times C) = \{(a, (0, 1)), (a, (1, 1)), (b, (0, 1)), (b, (1, 1)),
(a, (0, 3)), (a, (1, 3)), (b, (0, 3)), (b, (1, 3)) \}$$

Each of the ordered pairs in $A \times (B \times C)$ has its first element coming from $A$ and its second element coming from $B \times C$.

As Example 1.4 illustrates, compound set products may introduce structure that is not wanted. The next definition extends the notion of “product” to an arbitrary number of sets, as well as other variations of the “set-$\times$” operation.

**Definition 1.6** The product of $n$ sets, $A_1, A_2, \ldots, A_n$, is

$$A_1 \times A_2 \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) \mid a_i \in A_i, \ 1 \leq i \leq n\}$$

The elements of $A_1 \times A_2 \times A_3$ are called ordered $n$-tuples. The $n$-fold product $A^n$ is

$$A^n = \underbrace{A \times \cdots \times A}_{n \text{ times}}$$

By convention, $A^0$ is the empty set.
1.2. SET OPERATIONS

Exercises 1.2

1. List the following sets:

(a) \( \{ 2^i \mid i \in \mathbb{N} \text{ and } 0 \leq i \leq 8 \} \)
(b) \( \{ i^2 \mid i \in \mathbb{N} \text{ and } 0 \leq i \leq 8 \} \)
(c) \( \{ 2k + 1 \mid k \in \mathbb{N} \} \)
(d) \( \{ m \mid 23 < m < 29 \text{ and } m \text{ is a prime number} \} \)

2. Let \( A = \{a, b\} \); let \( B = \{1, 2, 3\} \); let \( C = \emptyset \); and let \( D = \{a, b, c, d\} \). List the following sets:

(a) \( A \cup B \)
(b) \( A \cap B \)
(c) \( A \times B \)
(d) \( \mathcal{P}(A) \)
(e) \( B \times \emptyset \)
(f) \( A \cup C \)
(g) \( A \cap D \)
(h) \( A^2 \)
(i) \( \mathcal{P}(\emptyset) \)
(j) \( (D \cap A) \times B \)

3. Let \( A = \{a, b\} \); let \( B = \{1, 2, 3\} \); and let \( E = A \times B \). List the following sets:

(a) \( \{(x, y, y) \mid (x, y) \in E\} \)
(b) \( \{(x, x) \mid x \in E\} \)
(c) \( \{(y, z) \mid (x, y) \in E \text{ and } z \in B\} \)

4. In this book, the set \( S = \{1, 2, 2, 3, 3\} \) denotes a three-element set in which 2 and 3 have both been listed twice. So we understand that this another way to describe the set \( \{1, 2, 3\} \). Some other books interpret \( \{1, 2, 2, 3, 3\} \) as a five element multiset allowing multiple occurrences of equal elements. Write a version of Definition 1.5 for multisets.

5. Define a set \( P_n \) representing the prime divisors of a number \( n \).

COMMENT: A simple set of numbers does not work because the convention is to allow listing redundant elements. So if one were to define \( P_{72} \) to be \( \{2, 2, 2, 3, 3\} \), the specified set is actually just \( \{2, 3\} \).

The question is asking you to devise a way, using just sets and set operations, to “represent” a number’s prime decomposition. Unless you know more about how you are going to use this representation, there is no best way to do it. Nevertheless, you should take “elegance,” (economy of expression, utility of notation) into consideration.
1.3 Languages

Any set can be thought of as an alphabet of symbols from which we can build the words, phrases, and sentences of a language.

**Definition 1.7** Let \( V \) be any set. The set of words over \( V \), denoted by \( V^+ \), consists of all finite sequences of symbols from \( V \). \( V \) is called the alphabet for \( V^+ \).

**Example 1.5** Let \( V = \{ a, b, c \} \) and list \( V^+ \).

**Solution:** The set of words over \( V \) is

\[
V^+ = \{ a, b, c, \\
aa, ab, ac, ba, bb, bc, ca, cb, cc, \\
aaa, aab, aac, aba, abb, abc, aca, acb, acc, \\
bba, bab, bac, bba, bbb, bbc, bca, bcb, bcc, \\
caa, cab, cac, cba, cbb, cbc, cca, ccb, ccc, \\
AAAA, AAAB, AAAC, AABA, ... , CCCC, ...
\]

This listing shows all the one-letter words in the first line, all the two-letter words in the second line, and so forth. Within each line, the words listed systematically, in “alphabetic” order.

**Example 1.6** Let \( W = \{ 5, 17 \} \). List \( W^+ \).

**Solution:** In the description below, an explicit concatenation mark is used to set individual symbols apart.

\[
W^+ = \{ 5, 17, 5'5, 17'17, 17'17, 5'5'5, 5'5'17, 5'17'17, ... \}
\]

For instance the word \( 17'5'5 \) is composed of the three symbols \( 17, 5, \) and \( 5 \) from \( W \).

The concatenation symbol is omitted—unless to do so causes confusion—just as the multiplication symbol is omitted in arithmetic formulas. It should be clear that the concatenation of two words is a word.

**Definition 1.8** The concatenation of words \( u \) and \( v \), is the word formed by juxtaposing \( u \) and \( v \), that is, first spelling \( u \) and then spelling \( v \); This new word is denoted by \( uv \), or where possible, simply by \( uv \). We write \( u^n \) for \( \underbrace{u u \cdots u}_{n \text{ times}} \).
Example 1.7 Let $V = \{a, b, c\}$ and consider $V^+$, as defined in Example 1.5.
Let $u = aab$, $v = cc$, and $w = abc$.

\[
\begin{align*}
cia \hat{a} \ b\ b\ b\ b\ a &= \ a\ b\ b\ b\ a \\
(a \hat{c})u &= acaab \\
v\hat{b}v &= ccbcc \\
u\hat{w} &= aabccabc \\
uuu &= u^3 = aabaabaab
\end{align*}
\]

If $u$, $v$, and $w$ are words, then

\[
u(v\hat{w}) = (u\hat{v})w
\]

In other words, when three or more words are concatenated, it does not matter whether the concatenation is done from left to right or from right to left (or in any other way) so long as the order is preserved.

It is sometimes useful to include a “word” containing no letters. The following definition provides a symbol for this word.

**Definition 1.9** The empty word, over any alphabet, is denoted by $\varepsilon$. Given alphabet $V$, for any word $w \in V^+$,

\[
\varepsilon \hat{w} = w \quad \text{and} \quad w\hat{\varepsilon} = w
\]

The language $V^*$ includes all words in $V^+$ together with $\varepsilon$,

\[
V^* = V^+ \cup \{\varepsilon\}
\]

Usually, we are interested in some particular subset of words over an alphabet.

**Definition 1.10** A language over alphabet $V$ is any subset of $V^+$.

**Example 1.8** Describe the language of decimal numerals as might be accepted in a programming language.

**Solution:** Let

\[
D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}
\]

represent the set of digits. We will also need punctuation symbols from the set

\[
P = \{., +, -\}
\]

The components of a numeral are
(a) A sign, which may be omitted for a positive number. Hence the sign comes from the set
\[ S = \{ +, -, \varepsilon \} \]
(b) the integer part, a string of one or more digits, \( D^+ \).
(c) the fractional part, if present, is a period followed by a string of zero or more digits,
\[ E = \{ \varepsilon \} \cup \{ . \, f \mid f \in D^* \} \]
Putting these together, the language of decimal numerals is described by
\[ \text{Numerals} = \{ s \, m \, f \mid s \in S, m \in D^+, \text{ and } f \in E \} \]
Here are some word instances in \text{Numeral}:
(a) The word \(+2.731\) is a valid numeral. It breaks down into a sign, integral part, and factional part according to the specification expression
\[ \begin{array}{ccc}
S & 2 & .735 \\
D^+ & E
\end{array} \]
(b) The word \(42\) is also valid, having an empty sign and fractional part
\[ \begin{array}{ccc}
\varepsilon & 42 & \varepsilon
\end{array} \]
(c) The words \(55.126.99\), \(++5\) and \(\varepsilon\) do not satisfy the description and hence are not in the language intended.

Exercises 1.3

1. Let \( V = \{ a, b, \$ \} \). For each of the following languages \( L_i \subseteq V^+ \), list enough elements to make it clear what each contains.
   (a) In language \( L_1 \) each word has exactly one \( \$ \) and equally many \( a \) s as \( b \) s.
   (b) In each word of language \( L_2 \), \( a \) s and \( b \) s alternate with any number of \( \$ \) s mixed in.
   (c) In each word of language \( L_3 \), no \( a \) occurs next to a \( b \).
   (d) \( L_4 = \{ u \, \$ \, v \mid u \in \{ a \}^+ \text{ and } v \in \{ \$, b \}^+ \} \)
   (e) \( L_5 = \{ a^{k \cdot \$} b^{k \cdot \$} \mid k \in \mathbb{N} \} \)

2. In Exercise 1.5 the listing of \( \{ a, b, c \}^+ \) shows that there are 3 one-letter words and 9 two-letter words. The third and fourth rows of the listing do not show all the possible words. How many words would there be in the third row; that is, how many three-letter words are there in \( \{ a, b, c \}^+ \)? How many four-letter words? How many \( n \)-letter words?
1.4 A Simple Algorithmic Language

A very simple programming language is used later in this book. We introduce the language informally in this section; in Chapter ??, once we have the needed mathematical foundations, we examine this language in more detail. It is a structured, sequential language of statements, similar in form to many languages that exist today like C and Java. It is assumed that you have some experience with, and intuition about, programming in this kind of language.

There are just four kinds of statements.

1. The assignment statement, has the form

   \[ v := E \]

   The object to the left of the assignment symbol is called an identifier, or sometimes program variable (but never just “variable”). To the right is an expression, \( E \), whose value is calculated and then associated with the program variable from that point on. Program variables can be simple names, such as \( x \) and answer, or array references, such as \( a[i] \) and \( b[5, j] \).

2. A conditional statement has the form

   \[ \text{if } T \text{ then } S_1 \text{ else } S_2 \]

   Where \( S_1 \) and \( S_2 \) are, themselves, statements. If the test \( T \) holds then statement \( S_1 \) is executed; otherwise, statement \( S_2 \) is executed;

3. A repetition statement has the form

   \[ \text{while } T \text{ do } S \]

   Statement \( S \) is repeatedly executed so long as the test \( T \) remains true.

4. Finally, a compound statement has the form

   \[ \text{begin } S_1 ; S_2 ; \ldots ; S_n \text{ end} \]

   Statements \( S_1, S_2, \ldots, S_n \) are executed in order.

Figure 1.1 shows an equivalent specification of the Statement language. It uses Backus-Naur notation, or “BNF,” a form often seen in programming manuals. Unlike the description above, Figure 1.1 says nothing about what statements mean, only what they look like. BNF only describes what sentences are syntactically correct.

Both descriptions are self referencing, meaning that they refer to the language in the process of defining it. Conditional, repetition, and compound statements contain statements. Although self reference used in this way may seem natural and intuitive, not all self-referencing definitions are meaningful.
Figure 1.1: Partial description of the Statement programming language STMT, expressed in Backus-Naur form (Sec. ??)

For example, consider the language described by items 1–3 above, leaving out the assignment statement. Can you give an example of a program in that language?

The statement language has no input/output operations. We can talk about the result of a program in terms of the final values of its identifiers. Here is an example of a program in the language of statements:

\[
\{ A, B \in \mathbb{N} \} \\
P: \begin{array}{l}
    \text{begin} \\
    x := A; \\
    y := B; \\
    z := 0; \\
    \ell_1: \text{while } x \neq 0 \text{ do } \{ \text{This is the loop } \ell_1 \} \\
    \quad \text{begin} \\
    \quad x := x - 1; \\
    \quad \ell_2: \text{end} \\
    \quad z := z + y \\
    \quad \text{end} \\
    \{ z = AB \}
\end{array}
\]

The program labels, \( P, \ell_1 \) and \( \ell_2 \) are not part of the language—there is no \texttt{goto} statement so labels aren’t needed—but are used in discussions to refer to points of the program.

Comments written between braces, \( \{ \cdots \} \), are called \textit{assertions}. For now, comments are optional, but in Chapter ?? they become a formal part of the language, used to reason logically about a program’s data state. After some staring perhaps, it should be clear that program \( P \) computes the product of natural numbers \( A \) and \( B \), as the comments assert.

\textbf{Exercises 1.4}
1. In the programming language just described, write programs for each of the following specifications.

   (a) Assume that program variables $x$ and $y$ have been initialized with values in $\mathbb{N}$. Compute the sum of these values, leaving the result in program variable $z$, using only the operations of adding or subtracting 1 to (from) a program variable.

   (b) Assume that program variables $x$ and $y$ have been initialized with values in $\mathbb{N}$. Compute the product of $x$ and $y$ using only the operations of addition and subtraction.

   (c) Assume that program variables $x$ and $y$ have been initialized with values, $A$ and $B$ respectively, in $\mathbb{N}$. Compute the value $A^B$ using only addition and multiplication.

   (d) Write a program to compute the quotient, $q$, and remainder, $r$ of two values initially held in variables $x$ (the dividend) and $y$, the divisor. Assume that you have only addition and subtraction.

   (e) Write a program to compute the greatest common divisor, gcd($x, y$), of $A, B \in \mathbb{N}$ held in program variables $x$ and $y$ respectively, using only addition and subtraction.

   (f) Assume that in addition to ‘+’ and ‘−’ you also have an operation, half($v$) that divides its operand, $v$ by two. Use this operation to improve the performance of the gcd program of the previous exercise.
Chapter 2

Relations, Functions

After sets, the most fundamental concept we will use is that of a relation.

Definition 2.1 If $A$ and $B$ are sets, then any subset of $A \times B$ may be called a relation from domain $A$ to range $B$.

We often simply simply $R \subseteq A \times B$ to say, “$R$ is a relation from $A$ to $B$.” This chapter is devoted to introducing the extensive vocabulary used in classifying and describing relations. We shall begin with the very important class of functions, and then discuss a number of other classes.

It is often helpful to draw pictures of relations, or graphs. In analytic geometry, a common form of graph is the Cartesian product, after the philosopher/mathematician René Descartes who first conceived it. The domain and range are laid out on horizontal and vertical axes. Elements of the relation are shown according to their coordinates on the resulting plane. For example, let the domain $A$ be the set $\{a, b, c, d\}$; let the range $B$ be the set $\{a, b, c, d\}$; and consider the relation from $A$ to $B$,

$$R = \{(a, a), (b, b), (b, d), (d, c)\}$$

A Cartesian graph of $R$, looks like this:

```
  d
 /|
/ |
  c
/  |
/   |
  b
/    |
/     |
  a

 a b c d
```
Another way to represent \( R \), called a bipartite graph, is drawn as follows. First, write down all the elements of the domain \( A \) in a column (or row). Next, write down all the elements of the range \( B \) in another column. Whenever there is an ordered pair, \((x, y) \in R\), draw an arrow from \( x \) to \( y \). A bipartite graph of \( R \) is shown below:

When the domain and range of a relation are the same set, as is the case here for \( R \), one can draw a directed graph representing the relation. In a directed graph, the elements are written down just once. Here is a directed graph of our example \( R \):

The points in a directed graph are called nodes. The arrows are called edges.

In computer science we often deal with finite, discrete sets. The bipartite and directed graphs of relations on such sets sometimes convey information more clearly than the corresponding Cartesian graphs, and we shall see them often in this book.

### 2.1 Functions

A common way to think about a function is as “a rule.” Sometimes, the rule is specified by a formula; for instance, we might write

\[
f(x) = x^2 + 5x + 6
\]

to specify a parabolic function. There is the idea of a function as a ”box” that computes a result based on its inputs:

\[
x \rightarrow f \rightarrow f(x)
\]
This notion of a function is close to our notion of a computer program. Too close. The concepts of “function” and “program” differ in some fundamental ways and it is important not to associate them too closely.

Instead, we can think of a function as a particular kind of relation. The property that distinguishes functions from other kinds of relations is that functions associate just one range element for a given domain element.

**Definition 2.2** A function from $X$ to $Y$ is a relation $f \subseteq X \times Y$ in which, for every $x \in X$, there is exactly one $(x, y) \in f$.

We write

$$f : X \rightarrow Y$$

to indicate that $f$ is a function from $X$ to $Y$. If $(x, y) \in f$, we say that $y$ is the value of $f$ at $x$, and write $f(x) = y$.

**Example 2.1** Let $X = \{a, b, c, d\}$; let $Y = \{1, 2, 3, 4\}$; and let

$$f = \{(a, 1), (b, 1), (c, 3), (d, 2)\}$$

The relation $f \subseteq X \times Y$ is a function because each of the possible inputs from domain $X$ is associated with exactly one output. In the bipartite graph of $f$, we can see that there is just one arrow from each element of the domain.

![Bipartite graph example](image)

**Example 2.2** Let sets $X$ and $Y$ be as given in the previous example, and let

$$g = \{(a, 3), (c, 2), (b, 4), (d, 1), (a, 4)\}$$

This relation is not a function because more than one value is associated with $a$:

![Bipartite graph example](image)
It does not make sense to use the notation \( g(x) = y \) when \( g \) is not a function, for then we would have \( 3 \neq g(a) = 4 \).

\[ \text{chosen} \]

**Example 2.3** Let sets \( X \) and \( Y \) be as given in Example 2.1 and let

\[ h = \{(a, 3), (c, 4), (d, 1)\} \]

This relation is not a function because \( h \) has no value for \( b \):

```
 a 1
 b 2
 c 3
 d 4
```

This too is a violation of the conditions of Definition 2.2.

**Example 2.4** Considered as a relation from \( \{a, c, d\} \) to \( \{1, 2, 3, 4\} \) the relation

\[ h = \{(a, 3), (c, 2), (d, 1)\} \]

is a function:

```
 a 1
 c 2
 d 4
```

In this book, the term “function” always refers to a *well defined* relation, that is, a relation with an ordered pair for every element of its domain. We use the term *partial function* to describe relations such as \( h \) in Example 2.3.

**Definition 2.3** A relation \( f \subseteq X \times Y \) is called a partial function from \( X \) to \( Y \) if for every \( x \in X \) there is at most one \( y \in Y \) such that \( (x, y) \in f \).

A well defined function is analogous to a “well behaved” program that produces an output for every possible input. A partial function is like a program that diverges—gets “stuck in a loop”—on some of its inputs, producing no output.

We can make a function from any partial function by restricting its domain to the subset of elements for which the function is defined. The following definition give a means of doing that.
Definition 2.4 Let $f \subseteq X \times Y$ be a partial function; and let $A \subseteq X$ and $B \subseteq Y$. The image of $A$ under $f$ is the set

$$\{ f(a) \mid a \in A \}$$

, and the preimage of $B$ under $f$ is the set

$$\{ a \mid f(a) = b \text{ for some } b \in B \}$$

For notation, we use $fA$ to denote the image of $A$ under $f$ and $f^{-1}B$ to denote the preimage of $B$ under $f$.

Example 2.5 Let $X = \{1, 2, 3, 4, 5, 6\}$ and $Y = \{1, 2, 3, 4, 5\}$, and consider the partial function $g$ described by the following bipartite graph:

The following are examples of image and preimage sets:

- $gX = \{1, 3, 5\}$
- $g^{-1}Y = \{1, 3, 4, 5, 6\}$
- $g\{4, 5, 6\} = \{1, 5\}$
- $g^{-1}\{2, 4\} = \emptyset$
- $g^{-1}\{4, 5\} = \{3, 5, 6\}$

Example 2.6 Let $X = \{a, b, c, d\}$, $Y = \{1, 2, 3, 4\}$, and partial function $h = \{(a, 3), (c, 4), (d, 1)\}$ as in Examples 2.1–2.3. Then $h$ is a function considered as a relation with domain $h^{-1}(Y)$ and range $Y$.

2.1.1 Function (and Relation) Composition

Definition 2.5 Given two relations, $R \subseteq X \times Y$ and $S \subseteq Y \times Z$, the composition of $R$ and $S$, written $S \circ R$, is a relation from $X$ to $Z$ defined by:

$$S \circ R = \{(x, z) \mid \text{for some } y \in Y, \ (x, y) \in R \text{ and } (y, z) \in S\}$$
The picture below illustrates the idea of composition. It defines an edge between elements of \( X \) and \( Z \) whenever they are both related to a common element in \( Y \):

The next proposition establishes that composing two functions yields a function.

**Proposition 2.1** Let \( A, B, \) and \( C \) be sets and suppose there are two functions, \( f : A \to B \) and \( g : B \to C \). Then \( g \circ f \) is also a function.

**Proof:** Let \( u \in A \). Since \( f \) is a function, there is a unique \( f(u) \in B \), such that \((u, f(u)) \in f\). Since \( g \) is a function, there is a unique \( g(f(u)) \in C \), such that \((f(u), g(f(u))) \in g\). By definition, we have \((u, g(f(u))) \in g \circ f\), and we have shown that this ordered pair is unique with respect to \( u \). Since \( u \) was arbitrary, \( g \circ f \) is a function. \( \square \)

Function composition is usually defined without reference to the more general notion of composition of relations. The typical wording of the definition is

\[
\text{Let } f : A \to B \text{ and } g : B \to C. \text{ The composition of } g \text{ and } f \text{ is defined by } (g \circ f)(x) = g(f(x)).
\]

Our proof of Proposition 2.1 establishes that \( g \circ f \) is “well defined”; that is, \( g \circ f \) is actually a function relating exactly on range element to every domain element. Verifying well-definedness is an essential part of the defining process, but it is often left to the reader.

The following definition describes three properties that a function might have.

**Definition 2.6** A function \( f : X \to Y \) is:

(a) surjective, or “onto”, iff for each \( y \in Y \) there is an \( x \in X \) such that \( f(x) = y \).

(b) injective or “one-to-one”, if for every \( x \) and \( x' \in X \), \( f(x) = f(x') \) only if \( x = x' \).

(c) bijective if \( f \) is surjective and injective.
2.1. FUNCTIONS

2.1.2 Infix Notation

If \( f : X \times Y \to Z \) we say \( f \) is a two-place function taking its first argument from \( X \) and its second argument from \( Y \). Such an \( f \) is a relation from \( X \times Y \) to \( Z \), and a typical element of \( f \) is \((x, y, z)\). Following the notation introduced after Definition 2.2 we should write

\[ f((a, b)) = c \]

but it is conventional in mathematics to write

\[ f(a, b) = c \]

instead. Similarly, we write \( g(a, b, c) \) for the result of a three-place function, \( g : A \times B \times C \to D \), and so on.

Often, the expressions for common two-place functions are even more concise. For example, addition is a two-place function, \( + : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \). So, using our “general purpose” notation for functions, we would write sums as, for example,

\[ +(3, 5) = 8 \]

Of course we don’t write sums that way; instead we write

\[ 3 + 5 = 8 \]

“\( +(3, 5) \)” is called prefix notation and “\( 3 + 5 \)” is called infix notation. Infix is the usual notation for two-place arithmetic operators, and for many other two-place operators as well, such as set operations, concatenation and the logical connectives to be discussed in Chapter 3. We use infix in the next definition, which gives names to some special properties of two-place functions.

**Definition 2.7** A two-place function \( \odot : A \times A \to A \) is said to be

(a) commutative iff for all \( x, y \in A \), \( x \odot y = y \odot x \);

(b) associative iff for all \( x, y, z \in A \), \( x \odot (y \odot z) = (x \odot y) \odot z \).

(c) Finally, \( e \in A \) is an identity for \( \odot \) iff for all \( x \in A \), \( x \odot e = e \odot x = x \).

**Example 2.7** Integer addition is associative and commutative with zero as an identity. Integer multiplication is associative and commutative with 1 as an identity. Integer subtraction is neither commutative nor associative. Subtraction does not have an identity as defined; \( x - 0 = x \) but \( 0 - x \neq x \). \( \square \)

**Example 2.8** Concatenation of words is associative and has an identity, \( \varepsilon \), but is not commutative. \( \text{abc} \cdot \text{def} \neq \text{def} \cdot \text{abc} \). \( \square \)
CHAPTER 2. RELATIONS, FUNCTIONS

Exercises 2.1

1. Let $A = \{1, 2\}$ and $B = \{2, 3, 4\}$. Which of the following relations from $A$ to $B$ are functions?

   (a) $\{(1, 3), (2, 4)\}$
   (b) $\{(1, 3), (1, 4)\}$
   (c) $\{(1, 3), (1, 3)\}$
   (d) $\{(1, 3), (2, 5)\}$
   (e) $\{(2, 2), (1, 4)\}$

2. Is $\{(1, 2), (2, 3)\}$ a function

   (a) from $\{(1, 2)\}$ to $\{(2, 3)\}$?
   (b) from $\mathbb{N}$ to $\mathbb{N}$?
   (c) from $\{1, 2\}$ to $\mathbb{N}$?

3. Let $A = \{1, 2\}$ and $B = \{2, 3, 4\}$.

   (a) List a relation that is an injective function from $A$ to $B$.
   (b) List a relation that is a surjective function from $B$ to $A$.
   (c) List two bijections from $B$ to $B$.

4. Let $f : A^2 \to A$ be given by the following table:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$f(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

   Show that $f$ is commutative and associative. What is the identity of $f$?

5. Let $f : A \to B$ and suppose $S$ and $T$ are both subsets of $A$.

   (a) Prove that $f(S \cap T) \subseteq fS \cap fT$.
   (b) Take both $A$ and $B$ to be $\{a, b, c\}$. Define a function $f$ and two subsets $S$ and $T$ for which $f(S \cap T) \neq fS \cap fT$.
   (c) State a condition under which, in general, $f(S \cap T) = fS \cap fT$.

6. Let $f : A \to B$ and $S \subseteq A$ and $T \subseteq B$ prove or disprove the following:

   (a) $f^{-1}(fS) = S$.
   (b) $f(f^{-1}T) = T$.

7. Prove: If $f : X \times X \to X$, and $e$ and $e'$ are both identities of $f$, then $e = e'$.

8. Prove: If $f : A \to B$ and $g : B \to C$ are onto functions, then $g \circ f$ is onto.

9. Prove: If $f : A \to B$ and $g : B \to C$ are one-to-one functions, then $g \circ f$ is one-to-one.

10. Prove: if $f : A \to B$ is a bijection, then there exists a bijection from $B$ to $A$. 

Copyright © 2012 Steven D. Johnson  August 20, 2012
2.2. RELATIONS ON A SINGLE SET

2.2 Relations on a Single Set

Let us now consider the important class of relations $R \subseteq A \times A$, in which the domain and range are the same set. The next definition establishes basic terminology for describing relations of this kind.

**Definition 2.8** A relation $R \subseteq A \times A$ is

(a) reflexive iff for every $a \in A$ $(a, a) \in R$, and irreflexive iff for every $a \in A$ $(a, a) \notin R$.

(b) symmetric iff $(x, y) \in R$ implies $(y, x) \in R$ and antisymmetric iff $(x, y) \in R$ and $(y, x) \in R$ imply $x = y$.

(c) transitive iff whenever $(x, y) \in R$ and $(y, z) \in R$ it is also the case that $(x, z) \in R$.

**Example 2.9** Let $A = \{a, b, c, d, e, f\}$. The relation $R$, whose graph is shown below, is reflexive.

![Graph](image)

It is not symmetric because, for example, $(c, f) \in R$ but $(f, c) \notin R$.

**Example 2.10** Let $A = \{a, b, c, d, e, f\}$, as in the previous example. The relation $T$ whose graph is shown below is transitive, but is neither reflexive, nor symmetric, nor irreflexive, nor antisymmetric.

![Graph](image)
Saying that a relation is “not symmetric” (or non-symmetric) is not the same as saying it is antisymmetric. Here for example, we have \((a, f)\) and \((f, a)\) in \(T\) (so \(T\) isn’t antisymmetric), but we also have \((e, c)\) \(\in T\) while \((c, e)\) \(\not\in T\) (so \(T\) isn’t symmetric, either). □

The notion of transitivity suggests that whenever there are two arrows “in sequence,” then there is a single arrow from the tail of the first to the head of the second. In the relation \(T\) of Example 2.10, for example, we have:

\[(e, b) \in T \text{ and } (b, c) \in T \] but also \((e, c) \in T\);
\[(b, c) \in T \text{ and } (c, d) \in T \] but also \((e, d) \in T\);
and so on.

In general, let \(R\) be transitive, and \((a_1, a_2), (a_2, a_3), \text{ and } (a_3, a_4)\) all be arrows in \(R\). By transitivity, \((a_1, a_3) \in R\), so, applying transitivity to \((a_1, a_3)\) and \((a_3, a_4)\), the arrow \((a_1, a_4) \in R\). This argument can be extended to any chain of arrows. We begin by formalizing the notion of a “chain of arrows.”

**Definition 2.9** Let \(R \subseteq A \times A\) be a relation. A path from \(a\) to \(b\) in \(R\) is a sequence

\[\langle x_1, x_2, \ldots, x_n \rangle\]

such that

\[(a) \quad n \geq 1\]
\[(b) \quad \text{for each } i, x_i \in A\]
\[(c) \quad x_1 = a \text{ and } x_n = b\]
\[(d) \quad \text{for each } i \text{ such that } 1 \leq i < n, (x_i, x_{i+1}) \in R\]

If \(a_0 = a_n\), we call the path a cycle. We say \(n\) is the length. A graph which has no cycles is said to be acyclic.

**Proposition 2.2** \(R\) is transitive iff whenever there is a path from \(a\) to \(b\) in \(R\), then there is an edge \((a, b) \in R\).

**Proof:** \((\Leftarrow)\) Assume that if there is a path from \(a\) to \(b\) in \(R\), then there is an arrow from \(a\) to \(b\) in \(R\). We would like to show that \(R\) is transitive, that is, if \((a_1, a_2) \in R\) and \((a_2, a_3) \in R\), then \((a_1, a_3) \in R\). If \((a_1, a_2) \in R\) and \((a_2, a_3) \in R\), then \((a_1, a_2, a_3)\) is a path from \(a_1\) to \(a_3\) in \(R\). By the assumption, since there is a path from \(a_1\) to \(a_3\) in \(R\), there is an arrow from \(a_1\) to \(a_3\) in \(R\), which is just what we need to show that \(R\) is transitive.

\((\Rightarrow)\) Assume that \((a_1, \ldots, a_n)\) is a path in \(R\), and that \(R\) is transitive. We shall show that for each \(i\) such that \(1 \leq i \leq n\), there is an arrow \((a_0, a_i) \in R\). We shall do this by giving an algorithm that, starting with the arrow \((a_0, a_1)\), builds up an arrow \((a_0, a_n)\) by successive applications of transitivity: Imagine
we have already built the arrow \((a_0, a_i) \in R\).

Since \((a_0, \ldots, a_i, a_{i+1}, \ldots, a_n)\) is a path in \(R\), we know that there is an arrow \((a_i, a_{i+1}) \in R\). Now transitivity requires that there be an arrow \((a_0, a_{i+1}) \in R\). We repeat this "extension" until we reach \(a_n\).

Knowing that all paths are of finite length (Definition 2.9 says this by specifying a length, \(n\)) the algorithm certainly demonstrates that the desired edge, \((a_1, a_n)\), exists and it even shows how to determine it. Constructive arguments of this form are often used to prove something exists. When a proof is based on an algorithm, one should first ask two questions:

(a) **Is it a definite procedure?** At every step, it must be clear and unambiguous what to do next.

(b) **Does it terminate?** It must be clearly evident that the procedure makes progress toward completion and does not go on forever.

Of course, the algorithm must also achieve its intended purpose—in this case, finding an edge from \(a_0\) to \(a_n\). Proof narratives often focus on the "correctness" aspect, leaving it to the Reader to confirm definiteness and termination.

In computing, we are often more interested in the **structure** of a graph than in details such as the domain of its nodes or in how it is depicted on paper. The two graphs shown below have the same structure, even though their nodes are labeled differently and they are laid out differently:

To have the same structure, an exact correspondence must exist between nodes, and this correspondence must "preserve edges." These qualities are formalized in the next definition.

**Definition 2.10** Two graphs \(R \subseteq A \times A\) and \(S \subseteq B \times B\) are said to be **isomorphic** iff there exists a bijection \(f: A \rightarrow B\) such that \((a, a') \in R\) iff \((f(a), f(a')) \in S\). The bijection \(f\) is called an isomorphism between \(R\) and \(S\).
Example 2.11 The bijection represents the correspondence between nodes. In the diagram above, we have \( A = \{a, b, c, d, e\} \) and \( B = \{v, w, x, y, z\} \). One possible bijection is

\[
f = \{(a, x), (b, z), (c, w), (d, y), (e, v)\}
\]

Check that there is a resulting correspondence between the edges of the two relations.

\[\square\]

2.2.1 Attaching Information to Graphs

We have already seen graph diagrams that contain information other than the just the graph structure. For instance, the element a node represents is shown next to the node. These instances are annotations that make it easier to interpret the drawing. In many applications, it is desirable to affix information as part of the mathematical representation—as opposed to just depicting it in a drawing. Defining this association is called labeling.

Definition 2.11 Given a relation \( R \subset A \times A \), a labeling of \( R \) is either:

(a) an edge labeling, \( \ell : R \rightarrow L \), mapping the edges to some set \( L \), or
(b) a node labeling, \( \ell' : A \rightarrow L' \), mapping nodes to some set \( L' \).

Labelings may be depicted in any way that makes clear what the labeling is. The graph below has both node and edge labels, depicted as circles and pentangles, respectively.

\[
\begin{align*}
A = \{w, x, y, z\}, & \quad L = \{a, b, c, d\}, \\
& \quad L' = \{1, 2, 3, 4, 5, 6\} \text{ and} \\
\ell : w & \mapsto a \quad \ell' : (w, w) \mapsto 1 \\
x & \mapsto b \quad (w, x) \mapsto 2 \\
y & \mapsto c \quad (x, y) \mapsto 3 \\
z & \mapsto d \quad (y, z) \mapsto 4 \\
& \quad (z, y) \mapsto 5 \\
& \quad (z, w) \mapsto 6
\end{align*}
\]

In this picture the node’s names, \( w, x, y \) and \( z \), do not appear, but their labels do. Nevertheless, we will often refer to \( w \) as “node \( a\),” instead of the more technically correct “the node labeled by \( a\).”
Exercises 2.2

1. Let

\[ A = \{a, b, c, d, e\} \]
\[ R = \{(a, b), (a, c), (b, a), (c, a), (c, d), (c, e), (d, c), (e, c)\} \]

(a) Draw the bipartite graph representation of \( R \).
(b) Draw the directed graph representation of \( R \).
(c) Is \( R \) symmetric? reflexive? transitive?

2. Let \( R = \{(a, a)\}, A = \{a\}, \) and \( B = \{a, b\} \). Is \( R \subseteq A \times A \) reflexive? Is \( R \subseteq B \times B \) reflexive? Draw both relations as bipartite graphs and directed graphs.

3. Let \( A = \{a\}, B = \{a, b\} \).

(a) List all the relations \( R \subseteq A \times A \).
(b) List all the relations \( R \subseteq B \times B \).
(c) Of the relations in (a) and (b), which are reflexive? Symmetric? Transitive?

4. A relation that is not symmetric is said to be asymmetric. Draw a directed graph that is asymmetric but not antisymmetric.

5. Draw a directed graph that is symmetric and transitive, but not reflexive.

6. Draw all the directed graphs on a set with two elements. Indicate which of these graphs are isomorphic to one another.

2.3 Trees

Trees are a fundamental structure seen in many areas of computer science. As data structures, trees have the desirable properties for searching and traversal. A number of results about trees are proved in this section. The proofs illustrate a kind of argument, “proof by construction,” that is common in computer science.

Definition 2.12 A tree is a finite acyclic directed graph \( R \subseteq A \times A \) in which there is one node (called the root) with in-degree 0, and every other node has in-degree 1. A node in a tree with out-degree 0 is called a leaf.

Just as one must include the domain and range in the declaration of relations, and particularly functions, it is necessary to include \( A \) in the declaration “\( R \subseteq A \times A \) is a tree” (See Exercise 3). One may say, for short, “\( R \) is a tree over \( A \).”
Example 2.12 There are two distinct for a set of three nodes. Here they are:

### Theorem 2.3

If $R \subseteq A \times A$ is a tree and $x \in A$ is not the root of $R$, then there is exactly one path from the root to $x$ in $R$.

**Proof:** We shall first construct one path from the root to $x$, and then show that it is unique. For the first part, see Figure 2.1. Since $x$ is not the root, $x$ has in-degree 1, so there must be a path $\langle a_1, x \rangle$. Assume we have constructed a path $\langle a_n, a_{n-1}, \ldots, x \rangle$. If $a_n$ is the root, we are done. If $a_n$ is not the root, then the in-degree of $a_n$ is 1, so there must be some node $a_{n+1}$ and arrow $(a_{n+1}, a_n) \in R$. Hence, $\langle a_{n+1}, a_n, \ldots, x \rangle$ is a path in $R$. Now, the length of this path can be no greater than the number of elements in $A$. For if some node were repeated, then $R$ would have a cycle, contradicting the assumption that $R$ is a tree (Defn. 2.12). Therefore, our path-building

The next result formalizes a property of trees mentioned at the beginning of this section.
procedure must eventually halt; but it can only halt by finding an \(a_{n+1}\) which is the root. In other words, the procedure must halt with a path from the root to \(x\).

Now assume that there are two paths

\[
\langle a_n, a_{n-1}, \ldots, x \rangle \quad \text{and} \quad \langle b_m, b_{m-1}, \ldots, x \rangle
\]

with \(a_n = b_m = \text{the root}\). Then at some point the paths must converge: There is an integer \(j\) such that \(a_{j+1} \neq b_{j+1}\), but \(a_j = b_j, \ a_{j-1} = b_{j-1}, \ldots, a_1 = b_1\):

\[
\begin{array}{c}
\begin{array}{c}
\text{So } (a_{j+1}, a_j) \in R \text{ and } (b_{j+1}, a_j) \in R. \text{ Therefore, } a_j \text{ must have in-degree of at least } 2, \text{ contradicting the assumption that } R \text{ is a tree (Defn. 2.12). Thus, there cannot be two paths and, by the previous argument, there must be at least one. This concludes the proof.}
\end{array}
\end{array}
\]

Like the proof of Theorem 2.2, this proof is constructive, describing two algorithms that iterate an unknown number of times before the argument is complete. Recall the discussion after Theorem 2.2 and decide whether the procedures described are definite and terminating. In both cases, what to do next
is well determined and iteration is limited either by the finite size of the node set or the finite length of the path.

The proof narrative can be criticized as being “redundant” in the sense that the two parts are very similar and can rather easily be condensed into a single argument combining both existence and uniqueness. (See Exercise 6.) However, the argument follows a pattern commonly seen in uniqueness proofs:

existence: Show that there is at least one object with the desired property.
uniqueness: Show that there is no more than one object with the desired property. This is often done by assuming more than one exist and showing that this assumption leads to a contradiction.

This proof strategy is reflected in the phrase, “There is one and only one \( x \) with property \( P(x) \),” suggesting that there are two things to prove.

Because of this unique-path property, it is easy to write programs that visit every node of a tree exactly once. One could use the same algorithms on a general directed graph if one could identify a tree “hidden” in the directed graph. Such a hidden tree is called a spanning tree of the graph.

**Definition 2.13** If \( G \subseteq A \times A \) is a directed graph and if \( R \subseteq A \times A \) is a tree and \( R \subseteq G \), then we say \( R \) is a spanning tree of \( G \).

As Figure 2.2 illustrates, a single graph may have many different spanning trees. If \( r \) is the root of a spanning tree \( R \) of \( G \), there must be a path in \( R \) from \( r \) to \( x \) for every node in the tree. Since \( R \subseteq G \), there must be a path in \( R \) from \( r \) to \( x \) for every node in \( G \). Theorem 2.4 states that this property is all that is needed for \( G \) to have a spanning tree.

**Definition 2.14** If \( G \subseteq A \times A \) is a rooted graph iff there is a node \( r \in A \) (the root) such that for every \( x \in A \) there is a path from \( r \) to \( x \) in \( G \).

**Theorem 2.4** Let \( G \subseteq A \times A \) by a finite rooted graph with root \( r \). Then \( G \) has a spanning tree with root \( r \).

**Proof:** We shall construct a sequence of trees

\[
R_1 \subseteq A_1 \times A_1 \\
R_2 \subseteq A_2 \times A_2 \\
\vdots \\
R_k \subseteq A_k \times A_k \\
\vdots 
\]

each with root \( r \) and with \( R_k \subseteq G \). Each \( A_k \) will contain \( k \) nodes, so if \( G \) has \( N \) nodes, \( A_N = A \) and \( R_N \) will be a spanning tree for \( G \).
First, let $A_1 = \{r\}$ and $R_1 = \emptyset$. (You can check that $R_1 \subseteq A_1 \times A_1$ is a tree.) Now imagine we have built $R_k \subseteq A_k \times A_k$, with $k < N$, and let us construct $R_{k+1} \subseteq A_{k+1} \times A_{k+1}$. Since $A_k$ has $k$ elements, and $k < N$, there must be some $z \in A$ such that $z \not\in A_k$. Since $G$ is rooted, let $\langle a_0, a_1, \ldots, a_p \rangle$ be a path from the root $r = a_0$ to $z = a_p$. Since $R_k \subseteq A_k \times A_k$, $r \in A_k$ and $z \not\in A_k$, there must be some $j$ such that $a_0, a_1, \ldots, a_j$ all belong to $A_k$, but $a_{j+1} \not\in A_k$.

Set $A_{k+1} = A \cup \{a_{j+1}\}$ and $R_{k+1} = R_k \cup \{(a_j, a_{j+1})\}$.

Now $a_{j+1} \not\in A_k$, so $(a_j, a_{j+1})$ is the only arrow to to $a_{j+1}$. So $a_{j+1}$ has in-degree 1, and we could not have created a cycle by adding $(a_j, a_{j+1})$. Furthermore, $(a_j, a_{j+1}) \in G$, so $R_{k+1} \subseteq G$. Last, there is still no arrow ending at $r$, so $r$ is the root of $R_{k+1}$. Hence, $R_{k+1}$ has the required properties. Perform the construction $N$ times, and $R_N$ will be the desired spanning tree. □

Figure 2.3 shows the construction of a spanning tree as described in the theorem. Try the construction yourself, using different $z$'s and different paths, to construct a different spanning tree.

Exercises 2.3

1. Draw all the nonisomorphic trees with five vertices.

2. A binary tree is a tree in which every nonleaf has an out-degree of two.

   (a) Draw all the distinct (nonisomorphic) binary trees with five nodes.
Figure 2.3: Constructing a spanning tree
2.4. DAGs

(b) Draw all the distinct (nonisomorphic) binary trees with six nodes.

(c) Based on your answers to (a) and (b), state a property about binary trees.

3. Let \( R = \{(a, b)\}, \) \( A = \{a, b\}, \) \( B = \{a, b, c\}. \) Is \( R \subseteq A \times A \) a tree? Is \( R \subseteq B \times B \) a tree?

4. Draw all the spanning trees of the following directed graph:

5. Prove: If \( A = \{a\}, \) there exists exactly one \( R \subseteq A \times A \) that is a tree.

6. Rewrite the proof of Theorem 2.3 combining the two parts of the proof into a single argument. This argument might begin, “Since \( x \) is not the root, it has in-degree 1, so \( \langle a_1, x \rangle \) is the only edge leading to \( x \) in \( R; \) and \( \langle a_1, x \rangle \) is the only path of length one ending at \( x. \ldots. \)"

2.4 DAGs

Computer data structures often take advantage of the fact that every datum has an address in the computer’s memory. If two data structures contain exactly the same information, that information coalesced into a single object whose address may be shared by different access points to the same address. This kind of sharing suggests a kind of graph structure similar to that of a tree but more compact.

Definition 2.15 A directed acyclic graph, or DAG, is a rooted graph containing no cyclic paths

Consider the tree \( G_1 \) on the left below. It has isomorphic subtrees rooted at \( e \) and \( g. \) DAG \( G_2 \) on the right is obtained by adding the edge \( (c, e) \) and
removing notes \( g, k, l \) and \( n \) as well as the edges among them.

The two graphs can be thought of as being “structurally” similar in the sense that whenever there is a path from \( a \) to \( x \) in \( G_1 \), there is a corresponding path in \( G_2 \) from \( a \) to a node corresponding to \( x \) in \( G_2 \). For instance, the path

\[ \langle a, c, g, l, n \rangle \]

in \( G_1 \) corresponds to

\[ \langle a, c, e, j, m \rangle \]

in \( G_1 \)

The two paths contain different nodes, so this notion of “similarity” must take into account a correspondence between nodes, as was the case with graph isomorphism defined earlier (Defn. 2.10). This is a weaker correspondence capturing the idea that one graph structure can be embedded in another.

**Definition 2.16** Two directed graphs \( R \subseteq A \times A \) and \( S \subseteq B \times B \) are said to be homomorphic iff there exists a function \( h: A \to B \) such that if \( (a, a') \in R \) then \( (h(a), h(a')) \in S \). Such a function \( h \) is called a homomorphism from \( A \) to \( B \).

**Example 2.14** Graphs \( G_1 \) and \( G_2 \), shown earlier are homomorphic under the mapping \( h \) given by

\[
\begin{align*}
h &: a \mapsto a & h &: e \mapsto e & h &: i \mapsto i & h &: m \mapsto m \\
h &: b \mapsto b & h &: f \mapsto f & h &: j \mapsto j & h &: n \mapsto m \\
h &: c \mapsto c & h &: g \mapsto e & h &: k \mapsto i \\
h &: d \mapsto d & h &: h \mapsto h & h &: l \mapsto j
\end{align*}
\]
Exercises 2.4

1. Let $R \subseteq A^2$ $S \subseteq B^2$ $T \subseteq C^2$ and suppose that $f: A \to B$ and $g: B \to C$ are homomorphisms. Prove that the composition $g \circ f$ is a homomorphism.

2. Some textbooks define a graph homomorphism to be a surjective function:

   Two directed graphs $R \subseteq A \times A$ and $S \subseteq B \times B$ are said to be homomorphic iff there exists a surjection $h: A \to B$ such that if $(a, a') \in R$ then $(h(a), h(a')) \in S$.

   Let us call this kind of homomorphism a strong homomorphism. Prove that the composition of two strong homomorphisms is a strong homomorphism.

2.5 Equivalence Relations*

Notions of equivalence are used throughout mathematics. A fundamental kind of equivalence is equality, between numbers or sets for example. But there are also many ways that we might regard two distinct objects to be equivalent. For example, suppose we have a sack of marbles. We might regard two marbles as equivalent if they are the same size and color. Similarly, we might say two programs are equivalent if they produce the same output for a given input, ignoring other features such as speed, clarity, and so forth.

What qualities must the notion of equivalence have? In the first place, equivalence is a relation on some set of objects. The following three properties capture the sense of what equivalence means:

- Every object is equivalent to itself.
- Whenever $x$ is equivalent to $y$, it is also the case that $y$ is equivalent to $x$.
- Whenever $x$ is equivalent to $y$ and $y$ is equivalent to $z$, it is also the case that $x$ is equivalent to $z$.

In other words,

Definition 2.17 A relation that is reflexive, symmetric, and transitive is called an equivalence relation.

Here is an example of an equivalence relation, depicted as a directed graph:
Definition 2.18 Let \( R \subseteq A \times A \) be an equivalence relation, and let \( a \in A \). The equivalence class of \( a \) under \( R \), written \([a]_R\), is defined as

\[
[a]_R = \{ a' \in A \mid (a, a') \in R \}
\]

When we can determine \( R \) from the context, we omit the “under \( R \)” and write just \([a] \). The “clusters” property may be expressed as follows.

Theorem 2.5 If \( R \) is an equivalence relation on \( A \) and \( a, b \in A \), then

(a) if \((a, b) \in R\), then \([a] = [b]\).

(b) if \((a, b) \notin R\), then \([a] \cap [b] = \emptyset\).

Proof: (a) Assume \((a, b) \in R\). We shall show \([b] \subseteq [a]\) and \([a] \subseteq [b]\). To show \([b] \subseteq [a]\), let \( x \in [b]\). Then \((b, x) \in R\). Since \((a, b) \in R\) and \((b, x) \in R\), by transitivity, \((a, x) \in R\). Hence, \(x \in [a]\). Since we have shown any member of \([b]\) is also a member of \([a]\), \([b] \subseteq [a]\).

To show \([a] \subseteq [b]\), let \( y \in [a]\). So \((a, y) \in R\) and since \( R \) is a symmetric relation, \((y, a) \in R\). We have assumed that \((a, b) \in R\) and since \( R \) is transitive, \((y, b) \in R\). By symmetry again, \((b, y) \in R\); and so \( y \in [b]\). Hence \([a] \subseteq [b]\).

(b) We shall show \([a] \cap [b] \neq \emptyset\) implies \((a, b) \in R\). If \([a] \cap [b] \neq \emptyset\), then there is some \( z \) such that \( z \in [a]\) and \( z \in [b]\). Since \( z \in [a]\), we have \((a, c) \in R\), and since \( z \in [b]\), we have \((b, c) \in R\) and by symmetry, \((c, b) \in R\). \( R \) is a transitive relation, so it follows that \((a, b) \in R\). \(\square\)

Corollary 2.6 If \( a, b \in A \), then either \([a] = [b]\) or \([a] \cap [b] = \emptyset\).

Proof: By Theorem 2.5, if \((a, b) \in R\), then \([a] = [b]\); and if \((a, b) \notin R\), then \([a] \cap [b] = \emptyset\). \(\square\)

Sometimes, it is preferable to think of a set in terms of its equivalence classes, rather than its individual elements. In programming, for example, this is the difference between a specification and an implementation. Think of a library of mathematical routines. The user of the library may want a procedure to compute the square root of a number—any number of functionally equivalent routines could be written to do that. The implementer of the library needs to provide one representative of this equivalence class.

Definition 2.19 If \( R \) is an equivalence relation on \( A \), the quotient set \( A/R \) is

\[
\{ [a]_R \mid a \in A \}
\]
Example 2.15 Let $A = \{1, 2, 3\}$ and $R = \{(1,1), (1,2), (2,1), (2,2), (3,3)\}$. Then $R$ is an equivalence class on $A$ and

\[
\begin{align*}
[1]_R &= \{1, 2\} \\
[2]_R &= \{1, 2\} \\
[3]_R &= \{3\}
\end{align*}
\]

So $A/R = \{\{1, 2\}, \{3\}\}$.

Now, $A/R$ is a subset of the power set, $\mathcal{P}(A)$, and so we might ask: which subsets of $\mathcal{P}(A)$ are also quotient sets? That is, which subsets of $\mathcal{P}(A)$ are equal to $A/R$ for some equivalence relation $R$? The following definition and theorem supply the answers and give us another way to characterize equivalence relations.

**Definition 2.20** Let $A$ be a set. A subset $\Delta$ of $\mathcal{P}(A)$ is a partition of $A$ iff

(a) each $S \in \Delta$ is nonempty, and

(b) for each $a \in A$ there is exactly one $S \in \Delta$ such that $a \in S$.

**Theorem 2.7** A subset $\Delta$ of $\mathcal{P}(A)$ is a partition iff $\Delta = A/R$ for some equivalence relation $R$ on $A$.

**Proof:** $(\Leftarrow)$ If $R$ is an equivalence relation, we claim $A/R$ is a partition. If $a \in A$, then $a \in [a]$, so there is some $S \in A/R$ such that $a \in S$. By Corollary 2.6 $[a]$ is the only equivalence class containing $a$. Furthermore, each equivalence class is nonempty. So $a/R$ is a partition.

$(\Rightarrow)$ The definition of a partition implies that we can define a function, $f : A \rightarrow \Delta$, mapping each $a \in A$ to the set $S$ of which it is an element. That is,

$f(a) = S$ iff $a \in S$

Define a relation $R \subseteq A \times A$ as follows

$R = \{(a, b) \mid f(a) = f(b)\}$

$R$ is easily seen to be an equivalence relation. We claim $A/R = \Delta$. Let $a \in A$ and $f(a) = S$. Then

$S = \{x \mid f(x) = S\}$ (since $x \in S$ iff $f(x) = S$)

$= \{x \mid f(x) = f(a)\}$

$= [a]_R$

So for each $a \in A$, $[a] = f(a) \in \Delta$; hence, $A/R \subseteq \Delta$. Conversely, let $S \in \Delta$. By the definition of partition, $S$ is nonempty, so choose $a \in S$. Hence, $f(a) = S$, and by the previous argument, $S = [a]_R$. So $\Delta \subseteq A/R$. $\Box$
Exercises 2.5

1. List $A/R$ for each of the following equivalence relations:

   (a) $A = \{x \in \mathbb{N} \mid 1 \leq x \leq 6\}$
   $R = \{(x, y) \mid (x - y) \text{ is evenly divisible by } 3\}$

   (b) $A = \{x \in \mathbb{N} \mid 1 \leq x \leq 8\}$
   $R = \{(x, y) \mid x = 2^i k \text{ and } y = 2^j k \text{ for odd } k\}$

   (c) $A = \{x \in \mathbb{N} \mid 1 \leq x \leq 24\}$
   $R = \{(x, y) \mid x = 2^i 3^j k \text{ and } y = 2^{i'} 3^{j'} k' \text{ and } i + j = i' + j'
   \text{ and } k, k' \text{ are not evenly divisible by 2 or 3}\}$

2. If $A$ and $B$ are sets and $f : A \to B$, define the kernel of $f$ [denoted $\text{Ker}(f)$] to be $\{(x, y) \mid x, y \in A \text{ and } f(x) = f(y)\}$. Prove that for any $f$, $\text{Ker}(f)$ is an equivalence relation.

3. Prove: Suppose $R$ is an equivalence relation on $A$. Define a set $B$ and a function $f : A \to B$ such that $R = \text{Ker}(f)$.

2.6 Partial Orders*

Set containment and numeric inequality, are examples of a class of relations called partial orders. These relations are of great importance in the theory of computer science.

Definition 2.21 A relation that is reflexive, antisymmetric and transitive is called a partial order. If the condition of antisymmetry is removed, the relation is called a preorder.

Example 2.16 Set containment is a partial order. The proofs are left as exercises:

- Reflexivity. If $S$ is any set, then $S \subseteq S$.
- Antisymmetry. If $S \subseteq T$ and $S \neq T$, then it is not the case that $T \subseteq S$.
- Transitivity. If $S \subseteq T$ and $T \subseteq U$ then $S \subseteq U$.

A special case of partial order is one in which all the elements are related to all others.

Definition 2.22 A total order is a partial order $R \subseteq A \times A$ with the additional property that for all $x, y \in A$, either $(x, y) \in R$ or $(y, x) \in R$. 
Example 2.17 \( \mathbb{N}, \mathbb{Q} \) and \( \mathbb{R} \) are assumed to be totally ordered by the relation \( \leq \).

\[ \square \]

Example 2.18 A tree is not a partial order because trees are neither reflective nor transitive. There are many times when one wants to “extend” the graph of a tree to make it into a partial order, incorporating concepts like “node \( m \) is a descendent of node \( n \)”.

Extending a tree \( T \) (or any graph for that matter) to a partial order is conceptually straightforward:

(a) To make \( T \subseteq A \times A \) reflexive, add self-edges to every \( a \in A \). The graph \( T' = T \cup \{(a, a) \mid a \in A\} \) is called the reflexive closure of \( T \).

(b) To make \( T \) transitive, add an edge from \((a, b)\) to \( T \) whenever there is a path \( \langle a, n_1, \ldots, n_k, b \rangle \) in \( T \). The resulting graph, call it \( T^* \), is called the transitive closure of \( T' \).

\[ \square \]

Although it is intuitively clear what it is, writing down a definition of \( T^* \) is thought provoking. \( T^* \) could be defined as

\[ T^* = T' \cup \{(a, b) \mid \text{there is a path from } a \text{ to } b \text{ in } T^*\} \]

In this definition \( T^* \) is being defined in terms of itself, and we need to consider whether this equation is meaningful. A self referencing definitions are not always valid or even meaningful. This point is discussed further in Chapter ??.

Example 2.19 Define the relation \( R \subseteq \mathbb{N}^2 \times \mathbb{N}^2 \) on ordered pairs as follows:

\[ ((n, m), (k, l)) \in R \text{ iff } \]

(a) \( n \leq k \) or

(b) \( n = k \) and \( m \leq l \).

That that \( R \) is a partial order follows from the fact that \( \leq \) is a partial order. It is reflexive because, for all \( n, m \in \mathbb{N}, ((n, m), (n, m)) \in R \); and if \( ((n, m), (k, l)) \in R \) and \( ((k, l), (u, v)) \in R \), so is \( ((n, m), (u, v)) \in R \). To prove transitivity, there are several cases to consider, specifically:

- \( n = k = u \)
- \( n = k \leq u \)
- \( n \leq k = u \)
- \( n \leq k \leq u \)

Check that in each case, transitivity holds. Because this ordering is like an “alphabetical” listing, it is called the lexicographic ordering and is often denoted by the symbol \( \leq_L \). Notice that even though \( \leq \) is a total order, \( \leq_L \) is not, and that even when the underlying ordering is not total, the lexicographic ordering is still well defined.

\[ \square \]
Example 2.20 Let $A$ be an alphabet and consider the language $A^*$ of all words built from letters in $A$, including $\varepsilon$. If $u, v \in A^*$ the $u$ is called a prefix of $v$ if there exists a word $w$ such that $u^*w = v$. Check that the prefix relation is a partial order.

Exercises 2.6

1. In the lexicographic ordering of $\mathbb{N}^2$, give an example of two elements that are not related.

2. In the prefix ordering of $A^*$, give an example of two elements that are not related.

3. Definition 2.21 states that a partial order is a reflexive, anti-symmetric, transitive relation. Add the edges needed to extend this tree to a partial order.
Chapter 3

Propositional Logic and Boolean Algebra

3.1 Propositions and Truth Tables

A proposition is a statement of fact, a sentence to which a value of true or false can be assigned. Compound propositions are built from simpler propositions using logical connectives, such as “and,” “or,” and “implies.” A propositional formula is an expression involving simple propositions and logical connectives. Suppose that \( P \) and \( Q \) stand for propositional formulas. Then the following are also propositional formulas:

\[
\neg P \quad \text{negation} \\
P \land Q \quad \text{conjunction} \\
P \lor Q \quad \text{disjunction} \\
P \Rightarrow Q \quad \text{implication} \\
P \Leftrightarrow Q \quad \text{coincidence} \\
P \Leftrightarrow | Q \quad \text{exclusive-or}
\]

Figure 3.1 shows some of the ways these connectives are expressed in English. The following definition tells what the connectives mean.

**Definition 3.1** The tables below define how the propositional connectives are interpreted.

| \( P \) | \( \neg P \) | \( P \land Q \) | \( P \lor Q \) | \( P \Rightarrow Q \) | \( P \Leftrightarrow Q \) | \( P \Leftrightarrow | Q \) |
|---|---|---|---|---|---|---|
| \( F \) | \( F \) | \( F \) | \( T \) | \( T \) | \( F \) |
| \( T \) | \( T \) | \( F \) | \( T \) | \( F \) | \( T \) |
| \( T \) | \( T \) | \( T \) | \( T \) | \( T \) | \( F \) |

3.1.1 Implication*

Implication, \( P \Rightarrow Q \), sometimes becomes confusing when considered in isolation. Since explaining it may simply compound the confusion, you may wish to defer
CHAPTER 3. PROPOSITIONAL LOGIC AND BOOLEAN ALGEBRA

\[ \neg P \quad \{ \text{It is not the case that } P.} \]
\[
\begin{align*}
P \land Q & \quad \{ P \text{ and } Q.} \\
P \text{ but } Q. &
\end{align*}
\]

\[ P \lor Q \quad \{ P \text{ or } Q.}
\]
\[
\begin{align*}
either P \text{ or } Q \text{ or both} & \\
at least one of } P \text{ and } Q &
\end{align*}
\]

\[ P \implies Q \quad \{ P \text{ implies } Q.}
\]
\[
\begin{align*}
\text{if } P \text{ then } Q. & \\
Q \text{ whenever } P. &
\end{align*}
\]

\[ P \iff Q \quad \{ P \text{ if and only if } Q.}
\]
\[
\begin{align*}
P \text{ if } Q. & \\
P \text{ exactly when } Q. & \\
P \text{ is necessary and sufficient for } Q. &
\end{align*}
\]
\[
\begin{align*}
\text{Whenever } P \text{ then } Q \text{ and conversely.}
\end{align*}
\]

\[ P \oplus Q \quad \{ \text{Either } P \text{ or } Q \text{ but not both.}
\]
\[
\begin{align*}
\text{Exactly one of } P \text{ and } Q. & \\
P \text{ exclusive-or } Q. &
\end{align*}
\]

Figure 3.1: The logical connectives and some corresponding English utterances.
3.1. PROPOSITIONS AND TRUTH TABLES

reading this section until a question like “What does $\Rightarrow$ really mean?” comes to mind and motivates you to read about it.

We are not accustomed to thinking about what “$P$ implies $Q$” should mean when the antecedent $P$ is known to be false or when the consequent $Q$ is known to be true. In most mathematical arguments, there is a connection between the two. $P$ must be used to carry the argument through. The following examples illustrate why the definition of implication is natural.

**Example 3.1 Proposition** If $A$ is any set, then $\emptyset \subseteq A$.

**Proof:** According to Definition 1.4, $\emptyset \subseteq A$ is true provided, “every element of $\emptyset$ is also an element of $A$.” This means that the statement $x \in \emptyset \Rightarrow x \in A$ must be valid, no matter what element is chosen for $x$. But no matter what $x$ is, “$x \in \emptyset$” is a false statement. In other words, “$\emptyset \subseteq A$” logically reduces to $F \Rightarrow x \in A$, and by Definition 3.1 this proposition is true whether or not $x \in A$.

Another way to put it is that no choice of $x$ exists that can be used to falsify “$x \in \emptyset \Rightarrow x \in A$.” It cannot be false, so it must be true. We say that the proposition holds vacuously, since the antecedent is logically false.

**Example 3.2 Proposition** If $A$ is any set, then $A \subseteq A$.

**Proof:** According to Definition 1.4, $A \subseteq A$ means that the statement $x \in A \Rightarrow x \in A$ must be valid no matter what $x$ is. But if “$x \in A$” is true, then the statement reduces to $T \Rightarrow T$, and if $x \not\in A$ we have $F \Rightarrow F$. In either case, the proposition is true.

We say that the proposition is tautologically valid because it reduces to a purely logical trueism. See Definition 3.2 later in this chapter.

**Example 3.3 Proposition** For all $n, m \in \mathbb{Z}$, if $a > 0$ and $b > 0$ then $(a+b)^1 \geq a^1 + b^1$.

**Proof:** By definition, raising any number to the “first power” yields the same number. In other words, for any $z \in \mathbb{Z}$, $z^1 = z$. Thus,

$$(a + b)^1 = a + b = a^1 + b^1$$

so it follows immediately that $(a+b)^1 \geq a^1 + b^1$, as desired.

The antecedent “$a > 0$ and $b > 0$” is irrelevant, the truth of the proposition does not depend on whether or not $a$ and $b$ is positive. (Had the proposition been $(a+b)^2 \geq a^2 + b^2$, the antecedent would be relevant.)

We say that this proposition holds trivially, that is, the consequent holds independently of the antecedent.
A good way to think about $P \Rightarrow Q$ is that it says, “either $P$ is false or $Q$ is true, or possibly both.” Or, “it is never the case that $Q$ is true and $P$ is false.” Or, “$Q$ (is true) only if $P$ (is also true).”

Another source of confusion lies in the multiple roles implication plays in mathematical discourse. Proposition 3.1, later in this chapter, uses implication (“if and only if”) in its statement. It is a statement about. And its proof is a chain of implications.

### 3.2 Truth Tables

Definition 3.1 gives us the means to evaluate complex propositions. We do so by first evaluating the innermost terms and then working outward. We keep track of intermediate results in a truth table. Simple examples of truth tables are used in Definition 3.1.

**Example 3.4** Evaluate the formula $(P \lor R) \Rightarrow Q$.

**Solution:** A truth table for this formula includes one row for each combination of truth values that might be assigned to its sub-formulas. In this case there are eight possibilities. To the right is the evaluation of the formula. Subterms $P \lor R$ (1) and $Q$ (2) are evaluated and then the ‘$\Rightarrow$’ (3).

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$R$</th>
<th>$(P \lor R)$</th>
<th>$\Rightarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

According to the table, there are three cases in which the proposition $(P \lor R) \Rightarrow Q$ is false.

**Example 3.5** Truth tables are sometimes used to analyze “story problems.” Consider the following example:

Dick, Jane, and Sally are working together on a programming project. Dick says, “Sally’s routine is correct; but my routine is correct only if Jane’s is.” Sally says, “If my routine has a bug, so does Dick’s; but my routine is correct.” Jane says, “Either Dick’s routine has a bug or Sally’s does; but not both.”
3.2. TRUTH TABLES

Assuming all three are telling the truth, whose routine has a bug? Whose is correct? Assuming all the routines are correct, who’s not telling the truth?

SOLUTION: Let us identify the atomic propositions. Define \( D \), \( J \) and \( S \) as follows:\footnote{Recall the remark about linguistic identification on page \( \PageIndex{3} \). A triple-equals sign is used when names are assigned to formulas.}

\[
\begin{align*}
D &\equiv \text{“Dick’s routine is correct.”} \\
J &\equiv \text{“Jane’s routine is correct.”} \\
S &\equiv \text{“Sally’s routine is correct.”}
\end{align*}
\]

The assertions made by the three programmers are

\[
\begin{align*}
\text{Dick: } & S \land (D \Rightarrow J) \\
\text{Jane: } & D \iff S \\
\text{Sally: } & (\neg S \Rightarrow \neg D) \land S
\end{align*}
\]

and we are interested in the truth of the proposition \( D \land J \land S \). Here is a truth table:

<table>
<thead>
<tr>
<th>( D )</th>
<th>( J )</th>
<th>( S )</th>
<th>( S \land (D \Rightarrow J) )</th>
<th>( D \iff S )</th>
<th>( (\neg S \Rightarrow \neg D) \land S )</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

For both cases in which Dick’s, Jane’s, and Sally’s statements are true, \( D \) is false and \( S \) is true. Thus, we can conclude that, Dick’s routine has a bug and Sally’s does not, provided all three programmers are telling the truth. We cannot draw any conclusion about Jane’s routine. The last row of the truth table is the case where all three routines are correct; and in that row, Jane’s statement is false.

3.2.1 Logical Equivalence

Propositions may be characterized and compared using truth tables. Our study of mathematical reasoning in later chapters involves truth-table analysis of the assertions made in formal proofs. The next two definitions provide a basic vocabulary for classifying propositions.

**Definition 3.2** A proposition is called a tautology when all rows of its truth table evaluate to \( T \). A proposition is called a contradiction when all rows of its truth table evaluate to \( F \). A proposition which is neither a tautology nor a contradiction is called a contingency.
Definition 3.3  Two propositions, $P$ and $Q$, are said to be logically equivalent when their truth tables are identical. Logical equivalence is written

$$P \equiv Q$$

Logical equivalence and logical coincidence are very similar concepts. The need to make a distinction between $'\equiv'$ and '$\iff$' will be made clearer in Chapter ?? on formal logic.

Proposition 3.1  $P \equiv Q$ iff $P \iff Q$ is a tautology.

How many logical connectives do we need? As the following proposition says, all the connectives we have defined can be implemented using only negation and disjunction.

Proposition 3.2  The following pairs of formulas are logically equivalent:

(a) $P \Rightarrow Q$ and $\neg(P \lor Q)$
(b) $P \land Q$ and $\neg((\neg P) \lor (\neg Q))$
(c) $(P \Leftrightarrow Q)$ and $(P \Rightarrow Q) \land (Q \Rightarrow P)$
(d) $P \leftrightarrow Q$ and $\neg(P \leftrightarrow Q)$

Proof: In each case logical equivalence is established by a comparison of truth tables, as specified in Definition 3.3. The tables for part (a) are shown below and the rest of the proof is left as an exercise.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$P \Rightarrow Q$</th>
<th>$\neg P \lor Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
</tbody>
</table>

Other sufficient sets of connectives are developed as exercises. There is much more to say about propositions. But now we have enough information about them to consider the next topic of this chapter.

Exercises 3.2

1. Let $P$ stand for the proposition “Sue says it.” Let $Q$ stand for the proposition “Sam saw it.” Let $R$ stand for the proposition “Sid did it.” Express the following sentences as formulas involving the logical connectives. If there is more than one way to translate a sentence, use truth tables to explain any differences in the meaning among these translations.

(a) Sid did it, Sam saw it, and Sue says it.
(b) If Sid did it, Sam saw it.
(c) Sid did it only if Sam saw it.
(d) Sue says it only if Sid did it, and Sam saw it.
(e) If Sue says it implies Sam saw it, Sid did it.

2. Definition 3.1 gives the meaning of five logical operations of two arguments. How many distinct logical connectives of two arguments are there?

3. Consider the logical operation defined below:

\[
\begin{array}{c|c|c|c|c}
P & Q & P \downarrow Q \\
\hline
F & F & T \\
F & T & T \\
T & F & T \\
T & T & F \\
\end{array}
\]

Show that ‘\(\downarrow\)’ can be used to implement (in the sense of Prop. 3.2) all of the operations of Definition 3.1.

4. Determine another logical operation, different than \(\downarrow\), which can be used to implement all of the operations of Definition 3.1.

3.3 Boolean Algebra

Digital computers are based on electrical systems in which there are just two voltage values. (voltage is a measure of electrical force). All the components of a digital system are carefully designed to produce and respond to just these two levels. Mathematically, the binary values of a digital system are represented as a two-element set of binary digits, or bits, \(\{0, 1\}\). The basic operations on bits are defined below.

**Definition 3.4** The bit-operations of inversion, addition, and multiplication are given by the following tables.

\[
\begin{array}{c|c|c|c|c}
\bar{x} & \cdot 0 & \cdot 1 & + 0 & + 1 \\
\hline
0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 \\
\end{array}
\]

The multiplication sign is dropped where possible, so that the expression

\[
(x \cdot \overline{y}) + (\overline{x} \cdot y)
\]

is usually written

\[
\overline{x \cdot y} + x \overline{y}
\]
There is a correspondence between logical connectives and boolean operations

<table>
<thead>
<tr>
<th>Logical Connective</th>
<th>Boolean Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘=’</td>
<td>logical equivalence</td>
</tr>
<tr>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>0</td>
<td>false</td>
</tr>
</tbody>
</table>

As we shall see in the next section, this is not the only correspondence possible, but it is nevertheless valid to manipulate propositional terms using the algebraic laws defined next.

**Proposition 3.3** Assume variables $x$, $y$, and $z$ range over bits, and let the operations of negation, addition, and multiplication be as defined in Definition 3.4. These operations obey the following algebraic identities.

<table>
<thead>
<tr>
<th>Boolean Identities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Negation</strong></td>
</tr>
<tr>
<td><strong>Identity</strong></td>
</tr>
<tr>
<td><strong>Dominance</strong></td>
</tr>
<tr>
<td><strong>Idempotence</strong></td>
</tr>
<tr>
<td><strong>Cancellation</strong></td>
</tr>
<tr>
<td><strong>Commutativity</strong></td>
</tr>
<tr>
<td><strong>Associativity</strong></td>
</tr>
<tr>
<td><strong>Distributivity</strong></td>
</tr>
<tr>
<td><strong>DeMorgan</strong></td>
</tr>
</tbody>
</table>

**Proof:** Each of the laws can be verified by comparing boolean truth tables according to Definition 3.4.

Definitions 3.3 and 3.4 form a system in which we can reason about equality. As the exercises at the end of this section illustrate, we can generalize this algebra, or system of identities, to structures other than $\{0, 1\}$.

**Definition 3.5** A set $B$ containing distinguished elements 1 and 0, and having operations ‘·’, ‘+’, and ‘¬’ which satisfy the laws of Proposition 3.3 for $x, y, z$ ranging over $B$, is called a Boolean Algebra.

**Example 3.6** Use the boolean identities to show that $a(a + b) = a$. 

Copyright © 2012 Steven D. Johnson August 20, 2012
3.3. BOOLEAN ALGEBRA

Solution: We shall show this by performing a derivation starting from the left-hand side.

\[ a(a + b) = (a + 0)(a + b) \quad \text{(identity)} \]
\[ = a + 0b \quad \text{(distributivity)} \]
\[ = a + 0 \quad \text{(dominance)} \]
\[ = a \quad \text{(identity)} \]

□

Example 3.7 Use the boolean identities to show that \( a + ab = a \).

Solution: Here is the derivation:

\[ a + ab = a1 + ab \quad \text{(identity)} \]
\[ = a(1 + b) \quad \text{(distributivity)} \]
\[ = a1 \quad \text{(dominance)} \]
\[ = a \quad \text{(identity)} \]

□

3.3.1 Duality

Compare the derivations in Examples 3.6 and 3.7 just above. The same laws are applied in the same order, the difference being that addition and multiplication are interchanged, as are 1s and 0s. Looking at the table in Proposition 3.3, it may be evident that one can always do this transformation.

The laws dealing with boolean addition and multiplication come in pairs. For every identity which holds for addition, there is a corresponding identity for multiplication, and conversely. This property is called duality.

We have already suggested a correlation between the boolean values and operations, and truth values with logical operations. For if we think of the bit 0 as meaning false and the bit 1 as meaning true, then inversion, addition, and multiplication, implement negation, disjunction, and conjunction, respectively. Compare the following tables with those in Definition 3.1:

<table>
<thead>
<tr>
<th>( \neg )x</th>
<th>( \land )</th>
<th>( \lor )</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
<td>F T T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T F T</td>
</tr>
</tbody>
</table>

But there is another way to associate truth values with bits. Let bit 0 represent true and bit 1 represent false. In this case, bit addition implements logical conjunction and bit multiplication implements disjunction:

<table>
<thead>
<tr>
<th>( \neg )x</th>
<th>( \lor )</th>
<th>( \land )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>T T T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F F F</td>
</tr>
</tbody>
</table>

Associating true with bit 1 is called the positive logic interpretation, and associating false with 1 is called the negative logic interpretation. Digital engineers use both interpretations when implementing digital “logic.”
Exercises 3.3

1. Using boolean truth tables the Distributivity and DeMorgan identities in Proposition 3.3

2. Reduce the following boolean expressions to simpler terms

   (a) \( xy + (x + y)\overline{x} + y \)
   (b) \( x + y + (\overline{x} + y + z) \)
   (c) \( yz + wx + z + [wz(xy + wz)] \)

3. Define \( x \oplus y \) to be \( x\overline{y} + \overline{x}y \). Use boolean algebra to prove

   (a) \( x \oplus y = \overline{x} \oplus \overline{y} \)
   (b) \( x(y \oplus z) = xy \oplus xz \)
   (c) \( (x \oplus y) = \overline{x} \oplus y \)

4. Let \( A = \{a, b, c\} \) and define the following correspondence for \( \mathcal{P}(A) \):

   \[
   \begin{align*}
   1 & \mapsto A \\
   0 & \mapsto \emptyset \\
   \overline{X} & \mapsto A \setminus X \\
   X \cdot Y & \mapsto X \cap Y \\
   X + Y & \mapsto X \cup Y
   \end{align*}
   \]

   Show that this correspondence forms a Boolean algebra, according do Definition 3.5.

5. Let \( D \) be the set of numbers that divide 30, \( D = \{1, 2, 3, 5, 6, 10, 15, 30\} \), and define the following correspondence.

   \[
   \begin{align*}
   1 & \mapsto 30 \\
   0 & \mapsto 1 \\
   \overline{x} & \mapsto 30 \div x \\
   x \cdot y & \mapsto \text{the greatest common divisor of } x \text{ and } y \\
   x + y & \mapsto \text{the least common multiple of } x \text{ and } y
   \end{align*}
   \]

   Show that this correspondence forms a Boolean algebra.

6. Show that the connectives \( \land \), \( \lor \), and \( \neg \) form a Boolean algebra under a notion of equality that says, \( P = Q \) iff \( P \) is logically equivalent to \( Q \).
3.4 Normal Forms

Consider the truth table, given below, for the proposition $A \iff B$ where

\[
A \equiv (P \Rightarrow Q) \land (Q \Rightarrow R) \\
B \equiv (P \Rightarrow R)
\]

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$R$</th>
<th>$(P \Rightarrow Q)$</th>
<th>$(Q \Rightarrow R)$</th>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
<td>$T^*$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
</tbody>
</table>

Comparing the truth tables for $A$ and $B$ one can see that they are not logically equivalent.

Look now at the truth table for formula $A$. It is true in four cases and false in the other four. We can construct a formula describing just the true cases by recording the truth values of $P$, $Q$ and $R$.

$A$ is true just when

- $P = F$, $Q = F$ and $R = F$, or
- $P = F$, $Q = F$ and $R = T$, or
- $P = F$, $Q = T$ and $R = T$, or
- $P = T$, $Q = T$ and $R = T$.

In each case, a formula can be written singling out exactly that case

$A$ is true just when

- $(\neg P) \land (\neg Q) \land (\neg R)$ is true, or
- $(\neg P) \land (\neg Q) \land R$ is true, or
- $(\neg P) \land Q \land R$ is true, or
- $P \land Q \land R$ is true.

Equivalently,

$A \text{ eq } (-P \land \neg Q \land \neg R) \lor (-P \land \neg Q \neg R) \lor (-P \land \neg Q \land R) \lor (P \land Q \land R)$

Similarly,

$B \text{ eq } (-P \land \neg Q \land \neg R) \lor (-P \land \neg Q \land R) \lor (-P \land Q \land \neg R) \lor (P \land Q \land R) \lor (P \land \neg Q \land R) \lor (P \land \neg Q \land R)$
One can construct such a formula from any truth table. In essence, it is just a way of describing the truth table. The result is always disjunction of and-clauses, each of which contains every variable just once in either a positive or negated instance.

The formula thus derived is called the disjunctive normal form (DNF) of the original expression, $A$ in this case.

Since every proposition has a logically equivalent DNF, we have shown that for any logical expression, there is an equivalent one expressed in terms of ‘\$\land\$', ‘\$\lor\$’ and ‘\$\neg\$’, as suggested by Proposition 3.2.

We have now seen two essentially equivalent ways to represent propositions in a way that makes them easier to analyze or compare: truth tables and DNFs. Both of them suggest computer encodings that could be used in automating the analyses.

(a) An array of 1s and 0s could be used to represent the truth table. For the proposition $A$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

(b) A list of 3-bit quantities—a bit for each variable, one for each clause in the DNF. For the proposition $A$ the list would be

$$(000 \ 001 \ 011 \ 111)$$

In both cases, the encodings contain the essential information characterizing $A$. However, to interpret the encodings, one needs additional information about how many variables are used in the formula and in what order those variables are used in developing the truth table.

### 3.4.1 Decision Diagrams*

The boolean expression $E \equiv \overline{p} \overline{r} + \overline{q} + r$ has the truth table

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$q$</td>
<td>$r$</td>
<td>$E$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Section 3.4 introduced a way to represent this truth table in disjunctive normal form written as a propositional term. Written a boolean term under a positive-logic interpretation, the analogous form is called *sum-of-products form*.

$E \equiv \overline{p} \overline{q} \overline{r} + \overline{p} q \overline{r} + p \overline{q} \overline{r} + pq \overline{r} + pq r$
Another way to represent terms is to use a tree. In the tree below, the nodes are labeled with the names of variables occurring in $E$, and the edges with boolean values indicating whether the variable is true or false along that path.

This is called a *binary decision tree* because each non-leaf has out-degree 2. We will see more general classes of decision trees in Chapter 4. Nodes are labeled with the variable whose value determines what path to follow; and edges are labeled with 1s and 0s.

A more compact representation of term $E$ can be obtained by building a DAG to which the binary decision is homomorphic (Defn. 2.16). We can do this step-wise, from the bottom up, by locating isomorphic (Defn. 2.10) subtrees and eliminating all but one of them. This is done in two steps below, first eliminating redundant leaves, and then isomorphic subtrees rooted at $r$.

So now we have a binary decision DAG in which every path from root to leaf corresponds to a path in the original tree. These paths, taken together, correspond to the clauses in a DNF.

In computer representations, additional transformations are used to make the representation still more compact. If the two branches from the node go to
the same target, that node can be eliminated.

The resulting DAG is no longer homomorphic to the original tree (Can you see why?) unless the missing variable is somehow “remembered” when traversing the graph. The DAG above is called a reduced ordered binary decision diagram or ROBDD for short. Authors often abbreviate the acronym ROBDD to just BDD.

Suppose a boolean or propositional term contains \( n \) different variables. The size of its truth table is always proportional to \( 2^n \). That is, the truth table becomes “exponentially large.” The term’s DNF may become exponentially large, depending on the number of true cases, but independent of the variable ordering. Likewise, the term’s ROBDD may become exponentially large, depending which variable contribute to the term’s value and on the variable ordering. All forms are used as computer representations, but ROBDDs are often the best choice.

Exercises 3.4

1. Construct truth tables and DNFs for the following propositional formulas

   (a) \((P \land (P \Rightarrow Q)) \Rightarrow Q\)
   (b) \(((P \Rightarrow R) \land (Q \Rightarrow R)) \Leftrightarrow ((P \land Q) \Rightarrow R)\)
   (c) \(((P \Rightarrow R) \lor (Q \Rightarrow R)) \Rightarrow ((P \lor Q) \Rightarrow R)\)
   (d) \(((P \Rightarrow R) \lor (Q \Rightarrow S)) \Rightarrow ((P \lor Q) \Rightarrow (R \lor S))\)

2. The term disjunctive normal form suggests that there might be such a thing as conjunctive normal form (CNF), and there is. What would the CNF of a formula look like? Devise a systematic way to synthesize a CNF from the truth table of a propositional formula.

3. In Section 3.4.1 it is claimed that the two graphs shown below are not
homomorphic. Consult Definition 2.16 and explain why.

3.5 Application of Boolean Algebra to Hardware Synthesis*

The Boolean algebra for \{1, 0\} has applications to the description of digital hardware. Addition of binary numbers involves the same column-by-column procedure as decimal addition, except that the arithmetic is base 2. For instance to add the numbers 1111001_2 and 101010_2,

\begin{align*}
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
+ & 1 & 1 & 1 \\
\hline
1 & 0 & 1 & 0 \\
\end{array}
\end{align*}

start at the least significant position adding the two right-most digits. If the result is 10_2 or 11_2, the “2’s place” is carried into the next column. To make the algorithm uniform, we start out with a carry-in of 0; and if the most-significant carry-out is 1, another place is given to the sum.

The procedure is implemented in hardware by connecting a series of identical single-bit full adders, one for each bit of the operands.

Copyright © 2012 Steven D. Johnson

August 20, 2012
So building an \( n \)-bit adder requires designing a full adder and replicating it \( n \) times.

Digital hardware is built using a set of devices, or *logic gates*, that operate on two distinct voltage levels, called *high* (H) and *low* (L). The actual voltage values depend on the technology used to make the devices\(^1\). The simplest of these devices realize the functions *and*, *or* and *not*, which are represented by the schematic symbols

```
  and  or  not
```

The truth table below specifies what the Full Adder does. It has two outputs, \( s \) for the sum and \( c_o \) for the carry-out, so two truth tables are needed.

\[
\begin{array}{ccc|cc}
  a & b & c_i & s & c_o \\
  0 & 0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 1 & 0 \\
  0 & 1 & 0 & 0 & 1 \\
  0 & 1 & 1 & 0 & 1 \\
  1 & 0 & 0 & 0 & 1 \\
  1 & 0 & 1 & 0 & 1 \\
  1 & 1 & 0 & 0 & 1 \\
  1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

If we express these tables in disjunctive normal form (sometimes called *sum-of-products* form in this context), a naive implementation is obtained, since we have logic gates to realize each operation.

\[
\begin{align*}
  s &= \overline{a} \overline{b} \overline{c} + \overline{a} b \overline{c} + \overline{a} \overline{b} c + a b c \\
  c_o &= \overline{a} b c + a \overline{b} c + a b \overline{c} + a b c
\end{align*}
\]

However, it would be better to reduce these formulas to something smaller, in order to use fewer gates. One way to do this is to reduce the expressions algebraically to equivalent but smaller formulas. The \( c_o \) output can be reduced

\(^1\)In integrated circuits, H is 3–5 volts and L is 0 volts relative to a reference voltage called *ground*. Of course, in physical devices the voltages are not exact, and may range a bit from their ideal values.
3.5. APPLICATION OF BOOLEAN ALGEBRA TO HARDWARE SYNTHESIS

The derivation of $c_o$ followed the typical pattern of enlarging the formula so that it could later be simplified. Such algebraic manipulations are often done with a goal in mind, and reaching that goal may involve expansion, even if the ultimate objective is reduction.

"Simplification" of $s$ is even more subtle. Suppose we have a device that realizes the exclusive-or operation,$x \oplus y \overset{\text{def}}{=} \overline{x}y + xy$
The goal now is to derive an equivalent expression that uses instances of ‘⊕’.

\[
s = \overline{a \overline{b} c} + \overline{a \overline{b} \overline{c}} + a \overline{b} \overline{c} + abc \quad \text{(truth table)}
\]

\[
= \overline{a} (\overline{b} c + b \overline{c}) + a (\overline{b} \overline{c} + bc) \quad \text{(distributivity, twice)}
\]

\[
\begin{align*}
\overline{b} \overline{c} + bc &= \\
&= \overline{b} \overline{c} + bc \quad \text{(negation)} \\
&= \overline{(b \overline{c})} \overline{(b c)} \quad \text{(DeMorgan’s Law)} \\
&= \overline{(b + c)} \overline{(b + \overline{c})} \quad \text{(DeMorgan’s Law)} \\
&= (b + c) \overline{(b + \overline{c})} \quad \text{(negation)} \\
&= (b + c) \overline{b} + (b + c) \overline{c} \quad \text{(distributivity)} \\
&= b \overline{b} + \overline{c} \overline{b} + \overline{b} \overline{c} + \overline{c} \overline{c} \quad \text{(distributivity)} \\
&= 0 + \overline{c} \overline{b} + \overline{c} \overline{b} + 0 \quad \text{(cancellation)} \\
&= \overline{bc} + \overline{bc} \quad \text{(identity, commutativity)} \\
&= \overline{a} (\overline{b} c + b \overline{c}) + a (\overline{b} \overline{c} + bc) \quad \text{(boxed derivation)}
\end{align*}
\]

A subsidiary derivation is used to refine the sub-formula \(\overline{b} \overline{c} + bc\) to \(\overline{b} c + b \overline{c}\).
Continuing with the derivation,

\[
\begin{align*}
\therefore \\
= & \quad \overline{a} (b \overline{c} + b \overline{c}) + a (b \overline{c} + b \overline{c}) \\
= & \quad \overline{a} (b \oplus c) + a (b \oplus c) \quad \text{(definition ‘⊕’)} \\
= & \quad a \oplus (b \oplus c) \quad \text{(definition ‘⊕’ (!))}
\end{align*}
\]

The implementation of \( s \) becomes

\[
\begin{tikzpicture}[node distance=1.5cm, on grid]
    \node [root] (a) {\( a \)};
    \node [not] (b) [below of=a] {\( b \)};
    \node [not] (c) [left of=b] {\( c_i \)};
    \node [and] (d) [below right of=a] {\( \text{xor} \)};
    \node [and] (e) [below right of=b] {\( \text{xor} \)};
    \node [output] (f) [right of=e] {\( s \)};
    \draw [line] (a) -- (d);
    \draw [line] (b) -- (e);
    \draw [line] (c) -- (d);
    \draw [line] (d) -- (f);
    \draw [line] (e) -- (f);
\end{tikzpicture}
\]

The second instance of ‘⊕’ involves sub-expressions rather than simple variables. So even in boolean algebra derivations can become very complex, as you already know from ordinary algebra.

It is possible but unlikely that the components of a real adder would be designed using algebraic derivations. Instead, the designer would “guess” an efficient implementation of the truth tables and verify it against the original specification. The verification process involves comparison of normal forms such as ROBDDs.

**Exercises 3.5**

1. Draw two distinct ROBDDs for the output \( s_1 \) of the 2-bit adder.

2. Draw the ROBDD for output \( s_2 \) of the 2-bit adder under the variable order \( \langle c, b, a, e, d \rangle \). [HINT. Work top-down, simplifying as you go along.

   Suppose that a node labelled \( v \) represents a term \( \Phi. \Phi = \overline{v} \cdot \Phi_0 + v \cdot \Phi_1 \), in which \( \Phi_0 \) and \( \Phi_1 \) are obtained by evaluating \( \Phi \) with \( v = 0 \) and \( v = 1 \), respectively.]

3. The derivation of sum bit \( s \) yields

\[
\overline{a} \overline{b} \overline{c} + \overline{a} b \overline{c} + \overline{a} b c + a b c = a \oplus (b \oplus c)
\]

Expanding the definition of ‘⊕’ yields

\[
\overline{a} \overline{b} \overline{c} + \overline{a} b \overline{c} + \overline{a} b c + a b c = \overline{a}(b \overline{c} + \overline{b} c) \oplus a(b \overline{c} + \overline{b} c)
\]

Choose a variable ordering, such as \( \langle a, b, c \rangle \). Construct and compare ROBDDs for both sides of the equation above.
Chapter 4

Counting

One often wants to know how many elements a set contains. In fact, knowing this number is often more important than knowing just what the elements are! Definition 4.1 introduces notation and terminology relating to a set’s size.

Section 4.3 lays a foundation for counting the elements in a set. This is usually done by posing a kind of experiment in which the question, ”How many elements are in set $S$?” is decomposed into a sequence of simpler decisions for which the number of outcomes is already known. One then tallies all the outcomes to get a final answer.

4.1 Extended Operations

Suppose we are given a set of numbers, $S = \{n_1, n_2, \ldots, n_k\}$, and wish to express their sum. One way to do it is to use ellipses:

$$n_1 + n_2 + \cdots + n_k$$

We might use a program:

$$\mathcal{P}: \begin{align*}
&\text{begin} \\
&i := 0; \\
&z := 0; \\
&\ell_1: \text{while } i < k \text{ do} \\
&\quad \text{begin} \\
&\quad i := i + 1; \\
&\quad z := z + n_k \\
&\quad \text{end} \\
&\{z = n_1 + \cdots + n_k\}
\end{align*}$$

You have probably also seen summation notation:

$$\sum_{i=1}^{k} n_i$$
The variable \( i \) in the formula above is called an index variable and is understood to range over integers between the lower bound 1 and the upper bound \( k \). That is, \( 1 \leq i \leq k \).

The set \( S \) could have been defined as

\[
S = \{ n_i \mid 1 \leq i \leq k \} = \{ n_1, n_2, \ldots, n_k \}
\]

And likewise, the summation might be expressed in several other ways, such as

\[
\sum \{ n_1, n_2, \ldots, n_k \}
\]

or

\[
\sum \{ n_i \mid 1 \leq i \leq k \}
\]

or even simply

\[
\sum S.
\]

In other words, we may express summation over any set of numbers, indexed or not. So what about \( \sum \emptyset \)? By convention (because it works out) we define summation over the empty set as

\[
\sum \emptyset = 0
\]

For example, a sum of the first 101 reciprocal powers of two could be written as

\[
\sum_{i=0}^{100} \frac{1}{2^i}
\]

or

\[
\sum \{ 2^{-i} \mid 0 \leq i \leq 100 \}
\]

or even

\[
\sum_{0 \leq i \leq 100} \frac{1}{2^i}
\]

In the rightmost formula, the summation bounds have now been incorporated in an inequality expression, but they mean the same thing. The infinite sum

\[
\sum \{ 2^{-i} \mid i \in \mathbb{N} \}
\]

could be written as

\[
\sum_{i=0}^{\infty} \frac{1}{2^i}
\]

or as

\[
\sum_{i \in \mathbb{N}} \frac{1}{2^i}
\]

Sometimes, indices range over sets that are not simple intervals of the number line. In fact, the indices need not be numbers at all. See Example 4.2.

Other operations can be extended in a manner similar to addition and multiplication. The requirements are that the operation be a commutative, associative function with an identity (Defn. 2.7) element for cases where the index set is empty.

**Example 4.1** Some of the extended operations arising in this book are shown in the following examples. In each example, \( I \) stands for some index set, and \( S = \{ x_i \mid i \in I \} \) is an indexed set. Note that saying “\( I \) is empty,” is equivalent to saying “\( S \) is empty.”
(a) Let \( S = \{ n_i \mid i \in I \} \) be an indexed set of numbers. The \textit{extended product} 
\[ \prod S \]

is the multiplicative product of all the numbers in \( S \)

For the empty case, \( \prod \emptyset = 1 \).
(b) Let \( S = \{ X_i \mid i \in I \} \) be an indexed collection of sets. The \textit{extended union} 
\[ \bigcup_{i \in I} X_i = \{ x \mid x \in A_j \text{ for some } j \in I \} \]

For the empty case, \( \bigcup \emptyset = \emptyset \).
(c) Let \( U \) be a predetermined set. let \( S = \{ A_i \mid i \in I \} \) be an indexed collection of \textit{subsets} of \( U \). The \textit{extended intersection} 
\[ \bigcap S = \{ a \mid a \in X_j \text{ for all } j \in I \} \]

For the empty case, \( \bigcap \emptyset = U \).
(d) Let \( S = \{ P_i \mid i \in I \} \) be an indexed collection of propositions. The \textit{extended conjunction} of these propositions is 
\[ \land S = \begin{cases} T & \text{if } P_i \text{ is } T \text{ for all } i \in I \\ F & \text{if for some } j \in I \text{ } P_j \text{ is } F \end{cases} \]

with \( \land \emptyset = U \).
(e) Let \( S = \{ P_i \mid i \in I \} \) be an indexed collection of propositions. The \textit{extended disjunction} of \( S \) is 
\[ \lor S = \begin{cases} T & \text{if for some } j \in I \text{ } P_j \text{ is } T \\ F & \text{if } P_j \text{ is } F \text{ for all } j \in I \end{cases} \]

with \( \lor \emptyset = F \).

\[ \square \]

4.2 Cardinality

**Definition 4.1** \( |S| \) \textit{denotes the number of elements in } \( S \). \( |S| \) \textit{is called the size or cardinality of } \( S \).

For example, \( |\{a, b, c, d, e\}| = 5, |\emptyset| = 0 \); and 
\[ |\{p \mid p \text{ is a prime number less than } 30\}| = 10 \]

**Fact 4.1** Let \( A \) and \( B \) be finite sets.

(a) \( |A \cup B| = |A| + |B| - |A \cap B| \)
(b) \(|A \cap B| = |A| + |B| - |A \cup B|\)

(c) \(|A \times B| = |A| \cdot |B|\)

(d) \(|A \backslash B| = |A| - |A \cap B|\)

Comparing this list with the set operations defined in Definition 1.5, \(|\mathcal{P}(A)|\) is missing. We will consider \(|\mathcal{P}(A)|\) later.

**Example 4.2**

The formula

\[\sum_{X \in \mathcal{P}(A)} |X|, \text{ or equivalently, } \sum_{X \subseteq A} |X|\]

expresses the sum of the sizes of all subsets of \(A\). For instance, if \(A = \{a, b, c\}\), then

\[
\sum_{X \in \mathcal{P}(A)} |X| = |\emptyset| + |\{a\}| + |\{b\}| + |\{c\}| + |\{a, b\}| + |\{a, c\}| + |\{b, c\}| + |\{a, b, c\}|
\]

\[= 0 + 1 + 1 + 1 + 2 + 2 + 2 + 3 = 12\]

\[\square\]

### 4.3 Permutations and Combinations

Let \(A = \{a, b, c\}\) an alphabet. How many three-letter words can be made from letters in \(A\)? One way to look at the problem is to draw three boxes and consider how many choices there are to fill each box with a letter:

\[
\square \square \square
\]

There are three choices, \(a\) or \(b\) or \(c\), for each box, so the number of three letter words is \(3 \times 3 \times 3 = 27\).

\[
\begin{array}{cccccccccccc}
aaa & aab &aac & aba & abb & abc &aca & acb & acc \\
baa &bab &bac & bba & bbb & bbe & bca & bcb & bcc \\
caa &cab &cac & cba & cbb & cbe & cca & ccb & ccc
\end{array}
\]

How many three-letter words are there in which no letter is repeated? Try it yourself, and compare your answer with

\[
\begin{array}{cccc}
abc & acb & bac & bca \\
& & cab & cba
\end{array}
\]
4.3. PERMUTATIONS AND COMBINATIONS

The difference between these two examples is that, in the first, the choice of any letter is independent of the other choices; and in the second, the choice of each letter depends on the other choices. Let \( B = \{a, b, c, d\} \) and consider the different four-letter words from \( B \) without repeated letters. Try it yourself, and compare your answer with

\[
\begin{array}{cccc}
abcd & bacd & cabd & dabc \\
abdc & bcad & cbad & dacb \\
acbd & bcda & cbda & dbac \\
acdb & bdac & cdab & dbca \\
adbc & bdca & cdab & dcab \\
adcb & bcda & cdca & dcba
\end{array}
\]

The listing is organized according to the choice of which letter is first. Having made that choice, all orderings for the remaining three letters are listed. This is a problem we have already solved: there are six possibilities. Figure 4.1(a) shows a tree, labeled to show how the listing is organized. Each path through the tree (i.e., from the root to a leaf) represents one letter ordering, as determined by reading the edge labels in order along the path from the root to a leaf. Trees used in this way are called decision trees.

If we are interested only in the number of solutions, and not what they are, the decision tree can be reduced in size by taking symmetries into account. In Figure 4.1(a) the subtrees at any level are isomorphic (Defn. 2.10) differing only in their labels. So instead of drawing all of them, we just keep track of how many there are. Figure 4.1(b) depicts a decision tree in which the edges are labeled with the number of isomorphic subtrees they represent. The product of the numbers along a path gives the number of solutions represented.

How many letter-orderings does a five-letter alphabet have? There are five choices for the first letter, and we have already shown that there are 24 ways to order the remaining four letters. Hence, the answer to the five-letter question is \( 5 \times 24 = 120 \).

Let us generalize this discussion.

**Fact 4.2** Let \( S \) be a set of size \( n \). There are

\[
1 \times 2 \times \cdots \times n
\]

different orders in which the elements of \( S \) can be listed. Such a listing, in which each \( s \in S \) occurs exactly once, is called permutation of \( S \).

The “product of whole numbers from 1 to \( n \)” is useful enough to be given its own notation:

**Definition 4.2** The product

\[
\prod_{k=1}^{n} k = 1 \times 2 \times \cdots \times n
\]
Figure 4.1: A decision tree (a) and counting tree (b) for ordering a 4-letter alphabet
is called $n$ factorial, written $n!$. By convention (Recall Example 4.1(a)), $0!$ is defined to be 1.

Thus, if $|S| = n$ it has $n!$ permutations.

How many ways are there to list two distinct elements from $A = \{a, b, c, d\}$?

Without actually doing it, we could reason as before that

1. There are four possibilities for the first letter.

2. Once the first letter is chosen, three possibilities remain for the second.

So the number of two-letter orderings is $4 \times 3 = 12$. Counting by taking the product of the successive outcomes is called the Rule of Products in some textbooks, and the Principle of Choice in others.

Another way to look at the problem is to start with something you already know—the number of permutations of a set of four elements—and eliminate redundant representatives. The listing below strikes out all but one of the permutations whose first two letters are the same.

$$
\begin{array}{cccc}
abcd & bacd & cabd & dabc \\
abdc & bcda & cdab & dcba \\
acbd & bdac & edba & dcab \\
acdb & bdca & edba & dcab \\
adbc & bdac & cdab & dcba \\
adcb & bdca & cdab & dcba \\
\end{array}
$$

So the number of 2-letter permutations is equal to the number of 4-letter permutations divided by the number which represent the same 2-letter outcome. Again, we are taking advantage of a symmetry in the problem, knowing that the partitioning is uniformly independent of what the first two letters actually are. Definition 4.3, below, summarizes this discussion.

Definition 4.3 Let $\forall$ be a set with $|A| = n$. For $m \leq n$, the number of distinct ways to list $m$ distinct elements from $A$ is

$$
\frac{n!}{(n - m)!}
$$

Such a listing is called an $m$-permutation.

Example 4.3 You have 26 trophies you would like to display across your fireplace mantel, but there is for only room 10 of them. How many different ways can you do this?

Solution 1: There are 10 positions at which a trophy can be placed, so the question is simply asking how many 10-permutations are there for a set with 26 elements. By Definition 4.3, this number is

$$
\frac{26!}{16!}
$$
or $26 \times 25 \times \cdots \times 17$. Unless you have a computer handy and the time, don’t bother to calculate this number; it is $19,275,223,968,000$.

Suppose you’ve placed 10 trophies on the mantel, and you decide it is better to place the largest one in the middle. Does swapping two of the selected trophies result in a “different” display? The problem statement fails to make this clear, but the solution given presumes that the order of the trophies matters.

Now suppose we are interested in selecting two letters from $A = \{a, b, c, d\}$ but don’t care about the order. In other words we are asking how many distinct subsets of size 2 are there in a set of four elements. As before, we can consider the set of $A$’s 2-permutations and strike out the redundant ones—those having the same two first letters in either order.

\[
\begin{align*}
\text{abcd} & \quad \text{bdac} & \quad \text{acbd} & \quad \text{adbc} \\
\text{acbd} & \quad \text{bcad} & \quad \text{abdc} & \quad \text{adcb} \\
\text{adbc} & \quad \text{cdab} & \quad \text{adcb} & \quad \text{dcab} \\
\text{abc} & \quad \text{bad} & \quad \text{cad} & \quad \text{db} \\
\text{abc} & \quad \text{ba} & \quad \text{ca} & \quad \text{db} \\
\text{abc} & \quad \text{b} & \quad \text{c} & \quad \text{a} \\
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{d}
\end{align*}
\]

**Definition 4.4** The chose function\(^1\) \(\binom{n}{k}\) is the number of ways to choose \(k\) unordered elements from a set of size \(n\). This number is given by the formula

\[
\binom{n}{k} = \frac{n!}{k! \times (n - k)!}
\]

\(\binom{n}{k}\) is calculated by dividing the number of permutations of an \(n\)-element set by \(k!\), the number of ways to permute the first \(k\) elements times \((n - k)!\) the number of ways to permute the remaining elements beyond the \(k^{th}\).

**Example 4.4** You have 26 trophies you would like to display across your fireplace mantel, but there is for only room 10 of them. How many different ways can you do this?

This is the same problem as Ex. 4.3 which took the ordering of the ten trophies into account. If the order doesn’t matter, we should divide out the number of ways they can be arranged on the mantel:

\[
\frac{19,275,223,968,000}{10!} = \frac{19,275,223,968,000}{3,628,800} = 5,311,735 = \frac{26!}{10! \times 16!} = \binom{26}{10}
\]

\(^1\)Also known as the binomial coefficient, and the combinational number.
Example 4.5 In how many ways can a set six elements, \( A = \{a, b, c, d, e, f\} \), be partitioned into three subsets, \( X, Y \) and \( Z \), containing three, two and one elements, respectively?

Solve this problem by counting the partitions one at a time:

(a) By Definition 4.4 there are

\[
\binom{6}{3} = \frac{6!}{3! \times 3!} = \frac{6 \cdot 5 \cdot 4}{3 \cdot 2 \cdot 1} = 20
\]

ways to choose three elements from \( A \).

(b) Once \( X \) has been determined, there are three elements left to choose for \( Y \). The number of ways to do that is

\[
\binom{3}{2} = \frac{3!}{2! \times 1!} = 3
\]

(c) Once \( X \) and \( Y \) have been determined, there is one remaining element to choose for \( Z \)

\[
\binom{1}{1} = \frac{1!}{1! \times 0!} = \frac{1}{1 \cdot 1} = 1
\]

(d) The number of partitionings is the product of the numbers of these choices, 

\[20 \cdot 3 \cdot 1 = 60.\]

It shouldn’t matter what order you choose \( X, Y \) and \( Z \). Check that

\[
\binom{6}{1} \times \binom{5}{2} \times \binom{3}{3} = \binom{6}{2} \times \binom{4}{1} \times \binom{3}{3} = \text{etc.}
\]

\[\Box\]

In Step (b) of Ex. 4.5 calculated that there were \( \binom{3}{2} \) ways to choose two elements from 3 for \( Y \). An equivalent problem is to choose one element \textit{not} to include in \( Y \), and there are \( \binom{3}{1} \) ways to do that. In general,

**Proposition 4.3**

\[
\binom{n}{k} = \binom{n}{n-k}
\]

**Proof:** By Definition 4.4 and since \( n - (n - k) = k \),

\[
\binom{n}{k} = \frac{n!}{k! \times (n-k)!} = \frac{n!}{(n-k)! \times k!} = \frac{n!}{(n-k)! \times (n-(n-k))!} = \binom{n}{n-k}
\]

\[\Box\]
Example 4.6 You’re having a dinner party for sixteen guests, including yourself, of which half are male and half are female. You have two tables each seating eight, with four places on two sides:

\[
\begin{array}{|c|c|}
\hline
? & ? \\
? & ? \\
\hline
\end{array}
\quad
\begin{array}{|c|c|}
\hline
? & ? \\
? & ? \\
\hline
\end{array}
\]

Table 1
Table 2

You want to make sure that both tables have an equal number of guys and gals. Bob is coming, but you don’t know about Jane yet. If Jane does come you need to make sure that she and Bob sit at different tables. How many ways are there to do this?

**Solution:** There are two cases to consider, according to whether or not Jane comes to the party

1. If Jane is not coming, then anyone can sit at either table. You need to choose four guys and four gals to sit at Table 1. The number of ways to do this is

\[
\binom{8}{4} \times \binom{8}{4}
\]

Once the people sitting at Table 1 are determined, all the rest will be assigned to Table 2, and there is only one way to do that.

2. If Jane comes she must sit at one table and Bob at the other.

   (a) Suppose Jane is assigned to Table 1. Then you need to choose:
      
      i. 3 other gals from the remaining 7 to sit at Table 1. There are \(\binom{7}{3}\) ways to do that.
      
      ii. Four guys from the remaining set of 7—Bob is excluded—will sit at Table 1. There are \(\binom{7}{4}\) ways to do that.
      
      iii. Once the assignments are made to Table 1, all the remaining guests sit at Table 2, so there is only just one choice.

      So there are \(\binom{7}{3} \times \binom{7}{4}\) seating assignments in this case.

   (b) If Jane is assigned to Table 2, it’s the same problem. So the number of assignments in this case is also \(\binom{7}{3} \times \binom{7}{4}\).

The answer, then, is that there are

\[
\binom{8}{4} \times \binom{8}{4} + \binom{7}{3} \times \binom{7}{4} + \binom{7}{3} \times \binom{7}{4}
\]

ways to assign people to tables. \(\square\)
You may already be thinking that there are many more ways to assign seating for this party. Once 8 people are selected for a table, there are 8! ways to arrange them. In the next example we will consider seating assignments. A counting tree for Example 4.6 might look like

The total number of solutions for a problem is the sum, over all the paths through (i.e., from the root to a leaf) its counting tree of the product of numbers along each path. More formally,

| **Fact 4.4** | let $T \subseteq A \times A$ be a counting tree with root $r$ and edge labeling $L : T \rightarrow \mathbb{N}$. Let $P = \{(r, a_1, \ldots, \ell) | \ell$ a leaf in $T\}$. The number of solutions represented by $T$ is |
|---|---|
| --- | $\sum_{p \in P} \left( \prod_{e \in p} L(e) \right)$ |

The formula above takes some liberties with notation. The indexing specifier “$e \in p$” is saying “take all the edges in (or along) path $p$.” Although in Chapter 2 paths are defined to be sequences, not sets, the meaning is should be clear.

**Example 4.7** For the same party as Example 4.6, how many ways are there to assign guests to seats in such a way that every guy is sitting next to at least one gal, and vice versa.

**Solution:** Decompose the problem by first assigning guests to tables, as in Example 4.6. Now assign seats along each side of each table. One way to do this is simply to list all the possibilities: guy-gal-guy-gal, gal-guy-gal-guy,
guy-gal-gal-guy and gal-guy-guy-gal. So the decision tree looks like

\[ (\binom{4}{1})^2 + 2 \left[ \binom{4}{1} \times \binom{4}{1} \right] \]

\[ \text{Assign Tables} \]

\[ 2 \text{ tables, 2 rows} \]

\[ 4 \]

\[ \text{Gender arrangement} \]

\[ 4 \]

\[ \text{Permute guys} \]

\[ 2! = 2 \]

\[ \text{Permute gals} \]

\[ 2! = 2 \]

and since \( \binom{7}{3} = \binom{7}{4} \), the number of solutions is

\[
64 \left[ \left( \frac{8}{4} \right)^2 + 2 \left( \frac{7}{4} \right)^2 \right]
\]

\[ \square \]

Exercises 4.3

1. Suppose you want to assign seats for a single row of 4 guys and 4 gals in such a way that each guy is sitting next to at least one gal, and vice versa. How many ways are there to do this? HINT: Use a decision tree, and practice by solving the 3-guy, 3-gal problem.

2. In Example 4.7 suppose you want to assign seats so that each guy is sitting next to or across from at least one gal, and vice versa. How many ways are there to do this? HINT: List out all the gender arrangements.

3. A standard deck of cards has 52 cards consisting of 13 cards in each of four suits: \( \spadesuit, \heartsuit, \diamondsuit, \clubsuit \). In each suit, cards have face values from \( \{1, 2, \ldots, 13\} \), each card having a different face value. A hand is a set of five cards from the deck. A hand is called a flush if all five cards are of the same suit. A hand is called a straight if the five cards are sequential in value, for instance, \( \{3\heartsuit, 4\spadesuit, 5\spadesuit, 6\diamondsuit, 7\heartsuit\} \).
(a) How many different flushes are there in a standard deck?
(b) How many different straights are there in a standard deck?

4. Prove: For all $n, k \in \mathbb{N}$, 
\[
\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}
\]

5. If $|A| = n$ guess the value of $|\mathcal{P}(A)|$ by listing a few small examples.
Index

., 7
^, 8
acyclic, 24
algebra, 48
alphabet, 8
antecedent, 43
antisymmetric, 23
assertion, 12
asymmetric, 27
Backus-Naur notation, 11
BDD, 54
bijection, 20
binomial coefficient, 68
bipartite graph, 16
bit, 47
BNF, 11

cardinality, 63
Choice
    Principle of, 67
    choose number, 68
    composition
        of relations, 19
concatenation, 8
conjunction, 63
consequent, 43
constructive argument, 25
constructive proof, 29
contains, 4
contingency, 45
contradiction, 45
cycle, 24
cyclic, 24

DAG, 33
decision DAG

    reduced ordered, 54
decision tree, 65
dependent choice, 65
derivation, 49
difference
    of sets, 5
directed acyclic graph, 33
directed graph, 16
disjoint, 4
disjunctive normal form, 52
DNF, 52
domain, 3
domain of a relation, 15

ε, 7
edge (of a directed graph), 16
element
    of a set, 1
ellipses, 2
empty word, 9
equality
    of sets, 4
equivalence class, 36
equivalence relation, 35
extended intersection, 63
extended product, 63
extended union, 63

function, 16, 17
    2-place, 21
    well defined, 20

graph
    bipartite, 16
directed, 16
    of a relation, 15

homomorphism, 34
identifier, 11
image
  of a function, 19
independent choice, 65
index variable, 62
infix notation, 21
injective function, 20
integers, 2
intersection
  of sets, 3
irreflexive, 23
isomorphism, 65
labeling a relation, 26
language, 8
  of statements, 11
leaf (of a tree), 27
length
  of a path, 24
logically equivalent, 16
member
  of a set, 1
N, 2
∅, 4
n-tuple, 6
natural numbers, 2
negative logic, 49
node (of a directed graph), 16
numeral, 9
one-to-one function, 20
onto function, 20
ordered pair, 5
P (A), 5
partial function, 18
partial order, 38
partition, 37
path, 24
positive logic, 49
power set, 6
prefix notation, 21
preimage, 19
preorder, 38
product (of sets), 5
program variable, 11
proposition, 41
propositional formula, 41
Q, 4
quantification, 3
quotient set, 36
R, 4
range of a relation, 15
rational numbers, 4
real numbers, 4
reflexive, 23
relation, 15
ROBDD, 54
root
  of a graph, 30
  of a tree, 27
rooted graph, 30
Rule of Products, 67
self reference, 11
set, 1
set builder notation, 3
size
  of a set, 63
spanning tree, 30
statement
  assignment, 11
  compound, 11
  conditional, 11
  repetition, 11
statements
  language of, 11
structure of a graph, 25
subsidary derivation, 58
sum-of-products form, 52
surjective function, 20
symbol, 8
symmetric, 23
syntax
  concrete, 1
  tautologically valid, 43
tautology, 45
total order, 38
INDEX

transitive, 23
tree, 27
trivial argument, 43
truth table, 44

union
  of sets, 5
universe of discourse, 3

vacuous argument, 43
verification, 59

Ψ, 2
well defined, 18
well defined function, 20
whole numbers, 2
word, 8

Z, 2
\( \mathbb{Z}_n \), 4