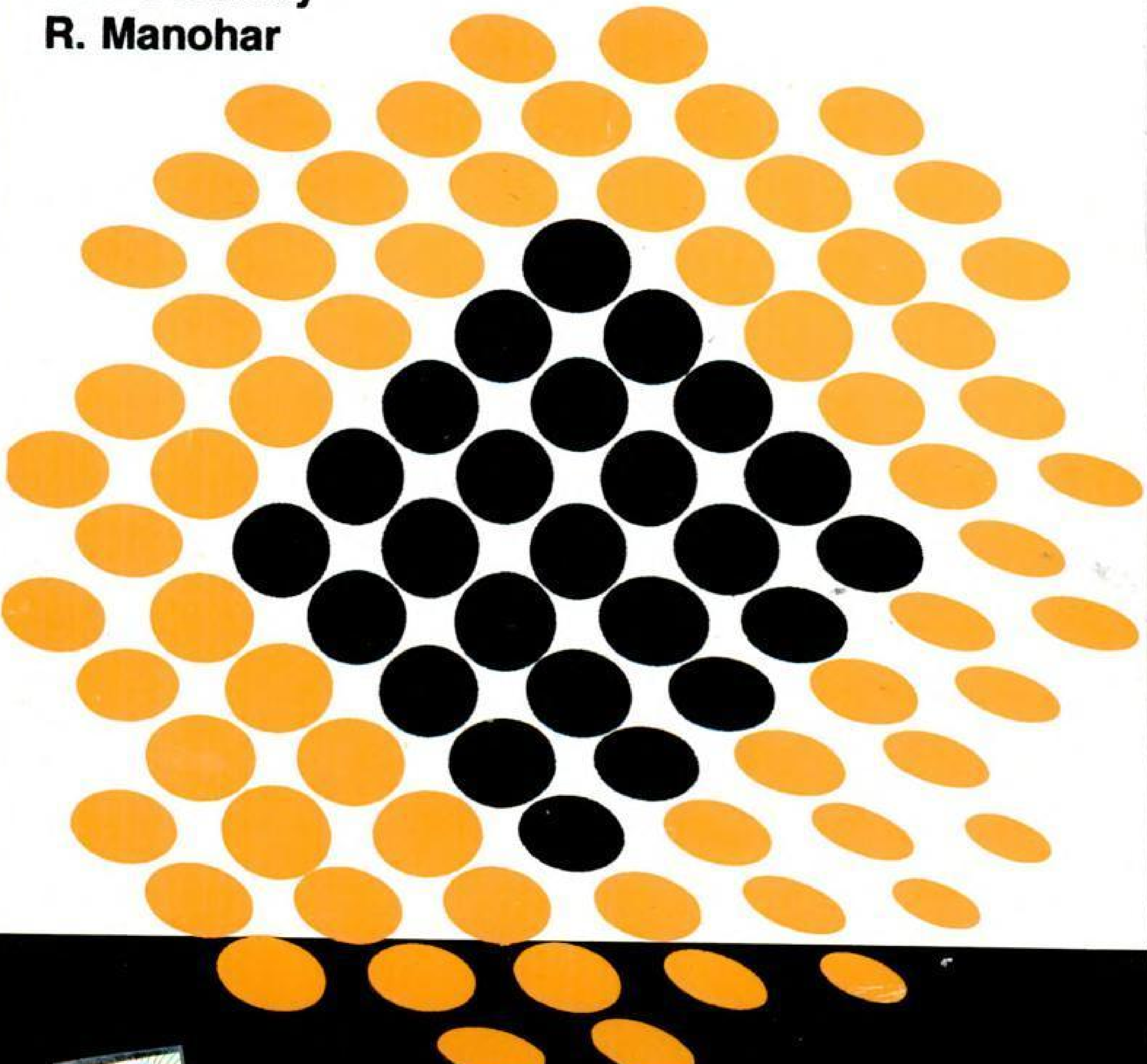
Discrete Mathematical Structures with Applications to Computer Science

J.P. Tremblay
R. Manohar



TATA McGRAW-HILL EDITION
FOR SALE IN INDIA ONLY

Tata McGraw-Hill

Discrete Mathematical Structures with Applications to Computer Science

Tata McGraw-Hill Edition 1997

35th reprint 2008
RYLYYDLXRZQXX

Reprinted in India by arrangement with The McGraw-Hill Companies, Inc., New York

For.Sale in India Only

Library of Congress Cataloging-in-Publication Data

Tremblay, Jean-Paul, date
Discrete mathematical structures with applications to computer science.
(McGraw-Hill computer science series)
Includes bibliographies
1. Mathematics.—1961 2. Electronic data processing. 3. Machine theory. I.
Manohar, R., date joint author. II. Title.
QA39.2.T72 510'.2'40901 74-23954
ISBN 0-07-065142-6

ISBN-13: 978-0-07-463113-3
ISBN-10: 0-07-463113-6

Published by Tata McGraw-Hill Publishing Company Limited,
7 West Patel Nagar, New Delhi 110 008, and printed at
Sai Printo Pack Pvt. Ltd., New Delhi 110 020

The McGraw-Hill Companies

CONTENTS

PREFACE

Several advanced books in computer science begin with a chapter consisting of a selection of mathematical topics that the reader is assumed to know. The exposition of such topics is usually brief, and the principal results that are summarized become prerequisites for the remainder of the text. It is not possible to learn these topics from such a brief treatment. Nor is it possible for undergraduate students in computer science to study all the topics they are required to know by attending courses dealing with each individual topic as given by mathematics departments. In general, the trend is to select several topics in mathematics that are essential to the study of many computer science areas and to expose the students to the mathematical prerequisites in some other way. A similar development has occurred in most engineering curricula. In the same spirit, this book discusses certain selected topics in mathematics which can be referred to as "discrete mathematics." No prerequisites except the mathematical maturity of a high school student are assumed. Although many students taking a course in discrete mathematics may have had a freshman course in calculus, such a course is by no means a prerequisite to the study of this book. However, any additional mathematical courses taken by students will aid in their development of mathematical maturity.

It is not our intention to cover all topics in discrete mathematics. The omission of counting techniques, permutations, and probability will be felt by some readers. We have assumed that many high school students will have had some exposure to these topics.

The selection of the topics was governed by our desire to introduce most of the basic terminology used in as many advanced courses in computer science as possible. In order to motivate the students properly, we feel that it is important to consider certain applications as the terminology is introduced. There are several advantages in using this approach. Students begin to see the relevance of abstract ideas and are therefore better motivated. Moreover, they gain confidence in applying these ideas to solve practical problems.

We wish to emphasize that concepts and terminology should be introduced well before they are used. Otherwise, students must invariably struggle both with the basic tools and with the subject matter to which the tools are applied. Most of the material in this book is properly a prerequisite to so many computer science courses that it should be taught no later than at the sophomore level. The book has been written with this objective in mind.

The mathematical topics to be discussed are logic, set theory, algebraic structures, Boolean algebra, graph theory, and basic computability theory. Although well-known and excellent books exist in these areas, we introduce these topics still keeping in mind that the reader will eventually use them in certain practical applications particularly related to computer science. We have strived to introduce the theoretical material in a reasonably mathematically precise manner whenever possible, while avoiding long philosophical discussions, questions of paradoxes, and any axiomatic approach to certain theories. The topics selected will also support the more advanced courses in computer science programs such as in the areas of automata, computability, artificial intelligence, formal languages and syntactical analysis, information organization and retrieval, switching theory, computer representation of discrete structures, and programming languages. It is hoped that a grasp of the theoretical material in this book will permit a student to understand most of the mathematical preliminaries which are briefly discussed at the beginning of many articles and books in the areas of computer science just mentioned.

Because the relation between the mathematics and how or where it could be applied may not be clear to the reader, the computer representation of certain mathematical structures is discussed. The need for discrete structures in computer science is motivated by the selection of certain applications from various areas in the field. Algorithms are developed for most applications, and computer programs are given for some of them. The computer representation and manipulation of discrete structures such as strings, trees, groups, and plexes are not discussed in great detail, but only to the extent which permits the formulation of a solution to a particular application.

Chapter 1 discusses mathematical logic. An elementary introduction to certain topics in logic is given to students in education, commerce, economics, and social sciences in courses usually entitled "Finite Mathematics." However, such discussions usually end with the construction of truth tables, and in certain instances a brief introduction to the inference theory of the statement calculus is included. In order for students to be able to read technical articles and books in computer science, it is necessary for them to know something about predicate calculus. Therefore, we have gone further in our discussion of logic than is usually done in books on finite mathematics. Yet we have avoided the philosophical discussions and intricate details that are found in the books on mathematical logic meant for mathematicians and philosophers. The chapter contains a brief introduction to the application of logic to two-state devices.

Chapter 2 deals with set theory. Some mathematical rigor is maintained in the discussions and proofs are sometimes given, but we do not raise the question of paradoxes and the axiomatic approach to set theory. Sets, relations, orderings, and recursive functions are discussed. The computer representation and manipulation of certain structures are introduced in this chapter. An example of the interrelationship of set theory and logic is given. The topic of recursion (and its implementation) is dealt with in some detail since many programming languages permit its use. Furthermore, the concept of recursion is important in its own right because computer scientists will encounter, throughout their careers, problems where recursion is unavoidable. The chapter concludes with an algorithm for proving theorems in the propositional calculus.

Chapter 3 discusses algebraic structures. Most books in modern algebra devote almost all their attention to group theory while little is said about semigroups and monoids. The latter are also emphasized in this chapter since it is semigroup and monoid theory which is very important in certain areas of computer science such as formal language theory, syntactic analysis, and automata. This chapter contains a number of applications dealing with topics such as the compilation of Polish expressions, languages and grammars, the theory of fast-adders, and error detecting and correcting codes.

Chapter 4 is concerned with Boolean algebra and its application to switching theory and sequential machines. An introduction to the minimization of Boolean functions and to its use in the logical design of digital computer systems is given. Sequential machines and their equivalence are also discussed.

Chapter 5 gives a brief introduction to graph theory. Elements of graph theory are indispensable in almost all computer science areas. Examples are given of its use in such areas as syntactic analysis, fault detection and diagnosis in computers, and minimal-path problems. The computer representation and manipulation of graphs are also discussed so that certain important algorithms can be included.

Finally, Chapter 6 gives a very brief introduction to computability theory. The equivalence of finite-state acceptors and regular grammars is shown. Finally, the concept of an effective procedure is introduced. It is shown that a Turing machine can evaluate any partial recursive function.

The exercises are of both a theoretical and a programming nature and are meant to further the understanding of the application of the concepts to various areas of computer science. The material in this book incorporates, in addition to logic, most of what the ACM Curriculum Committee on Computer Science recommends for the course "Introduction to Discrete Structures."[1]

[1] Course B3 in CACM 11, pp. 172–173, 1968.

We hope that this book will be of use to computer scientists, engineers, nonmathematics students who desire an intermediate coverage of topics in discrete mathematics, and mathematicians who want to familiarize themselves with the application of the theory to computer science. Students who have some previous background in modern logic and algebra will be able to master the material in one semester. For other students who have no previous knowledge of logic and algebra, this book can be used in a two-semester course. Certain topics can be selected to form a one-semester course. The omission of the applications discussed in the text will not result in any loss of continuity in the material. This book is based on the experience gained in teaching a course on discrete structures at the University of Saskatchewan at Saskatoon during the past four years.

A basic familiarity with either standard FORTRAN or PL/I is assumed. PL/I is useful in applications that involve recursion or list structures.

We owe a great deal to John A. Copeck and Richard F. Deutscher, who made many valuable criticisms and suggestions throughout the entire preparation and proofreading of the book. They also helped us in formulating and testing most of the algorithms. In particular, John Copeck assisted in the preparation of Chapter 1 and Sections 2-7, 4-4, 4-5, 5-2, 5-5, and 6-2. Also, Richard Deutscher assisted in the preparation of Chapter 2, Sections 4-4 and 5-1, and many of the figures.

We also thank Peter Hardie for his assistance in working out the details on fault diagnosis in Section 5-4 and Andrew Carson for his suggestions in Chapter 3. Robert Probert proofread Sections 5-1 and 6-2, Don McKillican and Allan Listol worked out the exercises, and Gail Galgan helped in proofreading and constructing the index. We owe a very special thanks to Alice Mae MacDonald who did such an excellent job of typing the manuscript, and to Helen Henderson and Dorothy Peake for providing typing support. This work would not have been possible without the support given by the University of Saskatchewan

J. P. TREMBLAY
R. MANOHAR

DISCRETE MATHEMATICAL STRUCTURES WITH APPLICATIONS TO COMPUTER SCIENCE

1 MATHEMATICAL LOGIC

INTRODUCTION

One of the main aims of logic is to provide rules by which one can determine whether any particular argument or reasoning is valid (correct).

Logic is concerned with all kinds of reasonings, whether they be legal arguments or mathematical proofs or conclusions in a scientific theory based upon a set of hypotheses. Because of the diversity of their application, these rules, called rules of inference, must be stated in general terms and must be independent of any particular argument or discipline involved. These rules should also be independent of any particular language used in the arguments. More precisely, in logic we are concerned with the forms of the arguments rather than with the arguments themselves. Like any other theory in science, the theory of inference is formulated in such a way that we should be able to decide about the validity of an argument by following the rules mechanically and independently of our own feelings about the argument. Of course, to proceed in this manner requires that the rules be stated unambiguously.

Any collection of rules or any theory needs a language in which these rules or theory can be stated. Natural languages are not always precise enough. They are also ambiguous and, as such, are not suitable for this purpose. It is therefore necessary first to develop a formal language called the *object language*. A formal language is one in which the syntax is well defined. In fact, every scientific discipline develops its own object language which consists of certain well-defined terms and well-specified uses of these terms. The only difference between logic and other disciplines is that in other disciplines we are concerned with the use of the object language while in logic we are as interested in analyzing our object language as we are in using it. In fact, in the first half of this chapter we shall be concerned with the development and analysis of an object language without considering its use in the theory of inference. This study has important applications in the design of computers and several other two-state devices, as is shown in Sec. 1-2.15. We emphasize this part of logic because the study of formal languages constitutes an important part in the development of means of communication with computing machines. This study is followed by the study of inference theory in Sec. 1-4. It soon becomes apparent that the object language developed thus far is very limited, and we cannot include some very simple argument forms in our inference theory. Therefore, in Sec. 1-5 we expand our object language to include predicates, and then in Sec. 1-6 we discuss the inference theory of predicate logic.

In order to avoid ambiguity, we use symbols which have been clearly defined in the object languages. An additional reason to use symbols is that they are easy to write and manipulate. Because of this use of symbols, the logic that we shall study is also called *symbolic logic*. Our study of the object language requires the use of another language. For this purpose we can choose any of the natural languages. In this case our choice is English, and so the statements about the object language will be made in English. This natural language (English) will then be called our *metalanguage*. Certain inherent difficulties in this procedure could be anticipated, because we wish to study a precise language while using another language which is not so precise.

1-1 STATEMENTS AND NOTATION

In this section we introduce certain basic units of our object language called primary (primitive, atomic) statements. We begin by assuming that the object language contains a set of declarative sentences which cannot be further broken down or analyzed into simpler sentences. These are the primary statements. Only those declarative sentences will be admitted in the object language which have one and only one of two possible values called "truth values." The two truth values are *true* and *false* and are denoted by the symbols T and F respectively. Occasionally they are also denoted by the symbols 1 and 0. The truth values have nothing to do with our feelings of the truth or falsity of these admissible sentences because these feelings are subjective and depend upon context. For our purpose, it is enough to assume that it is *possible* to assign one and only one of the two possible values to a declarative sentence. We are concerned in our study with the *effect* of assigning any particular truth value to declarative sentences rather than with the *actual* truth value of these sentences. Since we admit only two possible truth values, our logic is sometimes called a *two-valued logic*.

We develop a mechanism by which we can construct in our object language other declarative sentences having one of the two possible truth values. Note that we do not admit any other types of sentence, such as exclamatory, interrogative, etc., in the object language.

Declarative sentences in the object language are of two types. The first type includes those sentences which are considered to be primitive in the object language. These will be denoted by distinct symbols selected from the capital letters $A, B, C, \ldots, P, Q, \ldots$, while declarative sentences of the second type are obtained from the primitive ones by using certain symbols, called connectives, and certain punctuation marks, such as parentheses, to join primitive sentences. In any case, all the declarative sentences to which it is possible to assign one and only one of the two possible truth values are called *statements*. These statements which do not contain any of the connectives are called *atomic* (*primary, primitive*) *statements*.

We shall now give examples of sentences and show why some of them are not admissible in the object language and, hence, will not be symbolized.

1 Canada is a country.
2 Moscow is the capital of Spain.
3 This statement is false.
4 $1 + 101 = 110$.
5 Close the door.
6 Toronto is an old city.
7 Man will reach Mars by 1980.

Obviously Statements (1) and (2) have truth values *true* and *false* respectively. Statement (3) is not a statement according to our definition, because we cannot properly assign to it a definite truth value. If we assign the value *true*, then Sentence (3) says that Statement (3) is false. On the other hand, if we assign it the value *false*, then Sentence (3) implies that Statement (3) is true. This example illustrates a semantic paradox. In (4) we have a statement whose truth value depends upon the context; viz., if we are talking about numbers in the decimal system, then it is a false statement. On the other hand, for numbers in binary, it is a true statement. The truth value of a statement often depends upon its context, which is generally unstated but nonetheless understood. We shall soon see that we are not going to be preoccupied with the actual truth value of a statement. We shall be interested only in the fact that it *has* a truth value. In this sense (4), (6), and (7) are all statements. Note that Statement (6) is considered true in some parts of the world and false in certain other parts. The truth value of (7) could be determined only in the year 1980, or earlier if a man reaches Mars before that date. But this aspect is not of interest to us. Note that (5) is not a statement; it is a command.

Once we know those atomic statements which are admissible in the object language, we can use symbols to denote them. Methods of constructing and analyzing statements constructed from one or more atomic statements are discussed in Sec. 1-2, while the method of symbolizing atomic statements will be described here after we discuss some conventions regarding the use and mention of names in statements.

It is customary to use the *name* of an object, not the object itself, when making a statement about the object. As an example, consider the statement

8 This table is big.

The expression "this table" is used as a name of the object. The actual object, namely a particular table, is not used in the statement. It would be inconvenient to put the actual table in place of the expression "this table." Even for the case of small objects, where it may be possible to insert the actual object in place of its name, this practice would not permit us to make two simultaneous statements about the same object without using its name at one place or the other. For this reason it may be agreed that a statement about an object would contain never the object itself but only its name. In fact, we are so familiar with this convention that we take it for granted.

Consider, now, a situation in which we wish to discuss something about a *name*, so that the name is the object about which a statement is to be made. According to the rule just stated, we should use not the name itself in the statement but some name of that name. How does one give a name to a name? A usual method is to enclose the name in quotation marks and to treat it as a name for the name. For example, let us look at the following two statements.

9 Clara is smart.
10 "Clara" contains five letters.

In (9) something is said about a person whose name is Clara. But Statement (10) is not about a person but about a name. Thus "Clara" is used as a name of this name. By enclosing the name of a person in quotation marks it is made clear that the statement made in (10) is about a name and not about a person.

This convention can be explained alternatively by saying that we *use* a certain word in a sentence when that word serves as the name of an object under consideration. On the other hand, we *mention* a word in a sentence when that word is acting not as the name of an object but as the name of the word itself. To "mention" a word means that the word itself has been converted into an object of our consideration.

Throughout the text we shall be making statements not only about what we normally consider objects but also about other statements. Thus it would be necessary to name the statements under consideration. The same device used for naming names could also be used for naming statements. A statement enclosed in quotation marks will be used as the *name* of the statement. More generally, any expression enclosed in quotation marks will be used as the name of that expression. In other words, any expression that is mentioned is placed in quotation marks. The following statement illustrates the above discussion.

11 "Clara is smart" contains "Clara."

Statement (11) is a statement about Statement (9) and the word "Clara." Here Statement (9) was named first by enclosing it in quotation marks and then by using this name in (11) along with the name "Clara"!

In this discussion we have used certain other devices to name statements. One such device is to display a statement on a line separated from the main text. This method of display is assumed to have the same effect as that obtained by using quotation marks to delimit a statement within the text. Further, we have sometimes numbered these statements by inserting a number to the left of the statement. In a later reference this number is used as a name of the statement. This number is written within the text without quotation marks. Such a display and the numbering of statements permit some reduction in the number of quotation marks. Combinations of these different devices will be used throughout the text in naming statements. Thus the statement

12 "Clara is smart" is true.

could be written as "(9) is true," or equivalently,

12a (9) is true.

A particular person or an object may have more than one name. It is an accepted principle that one may substitute for the name of an object in a given statement any other name of the same object without altering the meaning of the statement. This principle was used in Statements (12) and (12*a*).

We shall be using the name-forming devices just discussed to form the names of statements. Very often such distinctions are not made in mathematical writings, and generally the difference between the name and the object is assumed to be clear from the context. However, this practice sometimes leads to confusion.

A situation analogous to the name-object concept just discussed exists in many programming languages. In particular, the distinction between the name of a variable and its value is frequently required when a procedure (function or subroutine) is invoked (called). The arguments (also called actual parameters) in the statement which invokes the procedure are associated with the (formal) parameters of the procedure either by name or by value. If the association is made by value, then only the value of an argument is passed to its corresponding parameter. This procedure implies that we cannot change the value of the argument from within the function since it is not known where this argument is stored in the computer memory. On the other hand, a call-by-name association makes the name or address of the argument available to the procedure. Such an association allows the value of an argument to be changed by instructions in the procedure. We shall now discuss how call-by-name and call-by-value associations are made in a number of programming languages.

In certain versions of FORTRAN compilers (such as IBM's FORTRAN H and G) the name of an argument, not its value, is passed to a function or subroutine. This convention also applies to the case of an argument's being a constant. The address of a constant (stored in some symbol table of the compiler) is passed to the corresponding parameter of the function. This process could lead to catastrophic results. For example, consider the simple function FUN described by the following sequence of statements:

```
INTEGER FUNCTION FUN(I)
I=5
FUN=I
RETURN
END
```

Suppose that the main program, which invokes FUN, consists of the trivial statements

```
      K = 3
      J = FUN(K) * 3
      L = FUN(3) * 3
      PRINT 10, J, L
   10 FORMAT(1H , I3, I3)
      STOP
      END
```

This program yields values of 15 and 25 for variables J and L respectively. In the evaluation of J, the address of K is known within the function. K is changed in the function to a value of 5 by the statement I = 5. The functional value returned by the function is 5, and a value of 15 for J results. The computation of L, however, is quite different. The address of 3 is passed to the function. Since the corresponding parameter I is changed to 5, the value of 3 in the symbol table in the main program will also be changed to 5. Note that since all references to the symbol table entry for constant 3 were made at compile time, all such future references in the remainder of the main program still refer to that entry or location, but the value will now be 5, not 3. More specifically, the name 3 in the right operand of the multiplication of L has a value of 5.

In other versions of FORTRAN compilers, such results are prevented by creating a dummy variable for each argument that is a constant. These internal (dummy) variables are not accessible to the programmer. A change in parameter corresponding to a dummy variable changes the value of that variable, but it does not change the value of the original argument from which it was constructed.

WATFIV permits the passing of arguments by value by merely enclosing such arguments in slashes. For example, in the function call

```
TEST(I,/K/,5)
```

the value of K is passed to the function TEST.

In PL/I arguments can be passed by value or by name. An argument is passed by value if it is enclosed within parentheses; otherwise it is passed by name. In the function call

```
TEST(I,(K),5)
```

the arguments I and K are passed by name and by value respectively.

As mentioned earlier, we shall use the capital letters $A, B, \ldots, P, Q, \ldots$ (with the exception of T and F) as well as subscripted capital letters to represent statements in symbolic logic. As an illustration, we write

13 P: It is raining today.

In Statement (13) we are including the information that "P" is a statement in symbolic logic which corresponds to the statement in English, "It is raining today." This situation is similar to the translation of the same statement into French as "Aujourd'hui il pleut." Thus "P" in (13)—"It is raining today"—and "Aujourd'hui il pleut" are the names of the same statement. Note that "P" and not P is used as the name of a statement.

1-2 CONNECTIVES

The notions of a statement and of its truth value have already been introduced. In the case of simple statements, their truth values are fairly obvious. However, it is possible to construct rather complicated statements from simpler statements by using certain connecting words or expressions known as "sentential connectives." Several such connectives are used in the English language. Because they are used with a variety of meanings, it is necessary to define a set of connectives with definite meanings. It is convenient to denote these new connectives by means of symbols. We define these connectives in this section and then develop methods to determine the truth values of statements that are formed by using them. Various properties of these statements and some relationships between them are also discussed. In addition, we show that the statements along with the connectives define an algebra that satisfies a set of properties. These properties enable us to do some calculations by using statements as objects. The algebra developed here has interesting and important applications in the field of switching theory and logical design of computers, as is shown in Sec. 1-2.15. Some of these results are also used in the theory of inference discussed in Sec. 1-4.

The statements that we consider initially are simple statements, called *atomic* or *primary statements*. As already indicated, new statements can be formed from atomic statements through the use of sentential connectives. The resulting statements are called *molecular* or *compound statements*. Thus the atomic statements are those which do not have any connectives.

In our everyday language we use connectives such as "and," "but," "or," etc., to combine two or more statements to form other statements. However, their use is not always precise and unambiguous. Therefore, we will not symbolize these connectives in our object language; however, we will define connectives which have some resemblance to the connectives in the English language.

The idea of using the capital letters P, Q, . . ., P_1, P_2, . . . to denote statements was already introduced in Sec. 1-1. Now the same symbols, namely, the capital letters with or without subscripts, will also be used to denote arbitrary statements. In this sense, a statement "P" either denotes a particular statement or serves as a placeholder for any statement whatsoever. This dual use of the same symbol to denote either a definite statement, called a *constant*, or an arbitrary statement, called a *variable*, does not cause any confusion as its use will be clear from the context. The truth value of "P" is the truth value of the actual statement which it represents. It should be emphasized that when "P" is used as a statement variable, it has no truth value and as such does not represent a statement in symbolic logic. We understand that if it is to be replaced, then its replacement must be a statement. Then the truth value of P could be determined. It is convenient to call "P" in this case a "statement formula." We discuss the notion of "statement formula" in Sec. 1-2.4. However, in the sections that follow, we often abbreviate the term "statement formula" simply by "statement." This abbreviation keeps our discussion simple and emphasizes the meaning of the connectives introduced.

As an illustration, let

P: It is raining today.

Q: It is snowing.

and let R be a statement variable whose possible replacements are P and Q. If no replacement for R is specified, it remains a statement variable and has no truth value. On the other hand, the truth values of P and Q can be determined because they are statements.

1-2.1 Negation

The *negation* of a statement is generally formed by introducing the word "not" at a proper place in the statement or by prefixing the statement with the phrase "It is not the case that." If "P" denotes a statement, then the negation of "P" is written as "$\neg P$" and read as "not P." If the truth value of "P" is T, then the truth value of "$\neg P$" is F. Also if the truth value of "P" is F, then the truth value of "$\neg P$" is T. This definition of the negation is summarized by Table 1-2.1.

Notice that we have not used the quotation marks to denote the names of the statements in the table. This practice is in keeping with the one adopted earlier, when a statement was separated from the main text and written on a separate line. From now on we shall drop the quotation marks even within the text when we use symbolic names for the statements, except in the case where this practice may lead to confusion. We now illustrate the formation of the negation of a statement.

Consider the statement

P: London is a city.

Then $\neg P$ is the statement

$\neg P$: It is not the case that London is a city.

Normally $\neg P$ can be written as

$\neg P$: London is not a city.

Although the two statements "It is not the case that London is a city" and "London is not a city" are not identical, we have translated both of them by $\neg P$. The reason is that both these statements have the same meaning in English. A given statement in the object language is denoted by a symbol, and it may correspond to several statements in English. This multiplicity happens because in a natural language one can express oneself in a variety of ways.

As an illustration, if a statement is

P: I went to my class yesterday.

then $\neg P$ is any one of the following

$\neg P$: I did not go to my class yesterday.

Table 1-2.1 TRUTH TABLE FOR NEGATION

P	$\neg P$
T	F
F	T

2 I was absent from my class yesterday.
3 It is not the case that I went to my class yesterday.

The symbol "$\neg$" has been used here to denote the negation. Alternate symbols used in the literature are "$\sim$," a bar, or "*NOT*," so that $\neg P$ is written as $\sim P$, $\bar{P}$, or *NOT P*. Note that a negation is called a connective although it only modifies a statement. In this sense, negation is a unary operation which operates on a single statement or a variable. The word "operation" will be explained in Chap. 2. For the present it is sufficient to note that an operation on statements generates other statements. We have chosen "$\neg$" to denote negation because this symbol is commonly used in the textbooks on logic and also in several programming languages, one of which will be used here.

1-2.2 Conjunction

The *conjunction* of two statements P and Q is the statement $P \wedge Q$ which is read as "P and Q." The statement $P \wedge Q$ has the truth value T whenever both P and Q have the truth value T; otherwise it has the truth value F. The conjunction is defined by Table 1-2.2.

EXAMPLE 1 Form the conjunction of

P: It is raining today.

Q: There are 20 tables in this room.

SOLUTION It is raining today and there are 20 tables in this room. ////

Normally, in our everyday language the conjunction "and" is used between two statements which have some kind of relation. Thus a statement "It is raining today and $2 + 2 = 4$" sounds odd, but in logic it is a perfectly acceptable statement formed from the statements "It is raining today" and "$2 + 2 = 4$."

EXAMPLE 2 Translate into symbolic form the statement

Jack and Jill went up the hill.

SOLUTION In order to write it as a conjunction of two statements, it is necessary first to paraphrase the statement as

Jack went up the hill and Jill went up the hill.

Table 1-2.2 TRUTH TABLE FOR CONJUNCTION

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

If we now write

P: Jack went up the hill.

Q: Jill went up the hill.

then the given statement can be written in symbolic form as $P \wedge Q$. ////

So far we have seen that the symbol $\wedge$ is used as a translation of the connective "and" appearing in English. However, the connective "and" is sometimes used in a different sense, and in such cases it cannot be translated by the symbol $\wedge$ defined above. In order to see this difference, consider the statements:

1 Roses are red and violets are blue.
2 He opened the book and started to read.
3 Jack and Jill are cousins.

In Statement (1) the conjunction "and" is used in the same sense as the symbol $\wedge$. In (2) the word "and" is used in the sense of "and then," because the action described in "he started to read" occurs after the action described in "he opened the book." In (3) the word "and" is not a conjunction. Note that our definition of conjunction is symmetric as far as P and Q are concerned; that is to say, the truth values of $P \wedge Q$ and of $Q \wedge P$ are the same for specific values of P and Q. Obviously the truth value of (1) will not change if we write it as

Violets are blue and roses are red.

On the other hand, we cannot write (2) as

He started to read and opened the book.

These examples show that the symbol $\wedge$ has a specific meaning which corresponds to the connective "and" in general, although "and" may also be used with some other meanings. Some authors use the symbol &, or a dot, or "*AND*" to denote the conjunction. Note that the conjunction is a binary operation in the sense that it connects two statements to form a new statement.

1-2.3 Disjunction

The *disjunction* of two statements P and Q is the statement $P \vee Q$ which is read as "P or Q." The statement $P \vee Q$ has the truth value F only when both P and Q have the truth value F; otherwise it is *true*. The disjunction is defined by Table 1-2.3.

Table 1-2.3 TRUTH TABLE FOR DISJUNCTION

P	Q	$P \vee Q$
T	T	T
T	F	T
F	T	T
F	F	F

The connectives $\neg$ and $\wedge$ defined earlier have the same meaning as the words "not" and "and" in general. However, the connective $\vee$ is not always the same as the word "or" because of the fact that the word "or" in English is commonly used both as an "exclusive OR" and as an "inclusive OR." For example, consider the following statements:

1 I shall watch the game on television or go to the game.
2 There is something wrong with the bulb or with the wiring.
3 Twenty or thirty animals were killed in the fire today.

In Statement (1), the connective "or" is used in the exclusive sense; that is to say, one or the other possibility exists but not both. In (2) the intended meaning is clearly one or the other or both. The connective "or" used in (2) is the "inclusive OR." In (3) the "or" is used for indicating an approximate number of animals, and it is not used as a connective.

From the definition of disjunction it is clear that $\vee$ is "inclusive OR." The symbol $\vee$ comes from the Latin word "vel" which is the "inclusive OR." It is not necessary to introduce a new symbol for "exclusive OR," since there are other ways to express it in terms of the symbols already defined. We demonstrate this point in Sec. 1-2.14.

Normally in our everyday language, the disjunction "or" is used between two statements which have some kind of relationship between them. It is not necessary in logic that there be any relationship between them according to the definition of disjunction. The truth value of $P \vee Q$ depends only upon the truth values of P and Q. As before, it may be necessary to paraphrase given statements in English before they can be translated into symbolic form. Similarly, translations of statements from symbolic logic into statements in English may require paraphrasing in order to make them grammatically acceptable.

1-2.4 Statement Formulas and Truth Tables

We have defined the connectives $\neg$, $\wedge$, and $\vee$ so far. Other connectives will be defined subsequently. We shall occasionally distinguish between two types of statements in our symbolic language. Those statements which do not contain any connectives are called *atomic* or *primary* or *simple statements.* On the other hand, those statements which contain one or more primary statements and some connectives are called *molecular* or *composite* or *compound statements.* As an example, let P and Q be any two statements. Some of the compound statements formed by using P and Q are

$$\neg P \qquad P \vee Q \qquad (P \wedge Q) \vee (\neg P) \qquad P \wedge (\neg Q) \tag{1}$$

The compound statements given above are statement formulas derived from the statement variables P and Q. Therefore, P and Q may be called the components of the statement formulas. Observe that in addition to the connectives we have also used parentheses in some cases in order to make the formula unambiguous. We discuss the rules of constructing statement formulas in Sec. 1-2.7.

Recall that a statement formula has no truth value. It is only when the statement variables in a formula are replaced by definite statements that we get a statement. This statement has a truth value which depends upon the truth values of the statements used in replacing the variables.

In the construction of formulas, the parentheses will be used in the same sense in which they are used in elementary arithmetic or algebra or sometimes in a computer programming language. This usage means that the expressions in the innermost parentheses are simplified first. With this convention in mind, $\neg(P \wedge Q)$ means the negation of $P \wedge Q$. Similarly $(P \wedge Q) \vee (Q \wedge R)$ means the disjunction of $P \wedge Q$ and $Q \wedge R$. $((P \wedge Q) \vee R) \wedge (\neg P)$ means the conjunction of $\neg P$ and $(P \wedge Q) \vee R$, while $(P \wedge Q) \vee R$ means the disjunction of $P \wedge Q$ and R.

In order to reduce the number of parentheses, we will assume that the negation affects as little as possible of what follows. Thus $\neg P \vee Q$ is written for $(\neg P) \vee Q$, and the negation means the negation of the statement immediately following the symbol $\neg$. On the other hand, according to our convention, $\neg(P \wedge Q) \vee R$ stands for the disjunction of $\neg(P \wedge Q)$ and R. The negation affects $P \wedge Q$ but not R.

Truth tables have already been introduced in the definitions of the connectives. Our basic concern is to determine the truth value of a statement formula for each possible combination of the truth values of the component statements. A table showing all such truth values is called the *truth table* of the formula. In Table 1-2.1 we constructed the truth table for $\neg P$. There is only one component or atomic statement, namely P, and so there are only two possible truth values to be considered. Thus Table 1-2.1 has only two rows. In Tables 1-2.2 and 1-2.3 we constructed truth tables for $P \wedge Q$ and $P \vee Q$ respectively. These statement formulas have two component statements, namely P and Q, and there are 2^2 possible combinations of truth values that must be considered. Thus each of the two tables has 2^2 rows. In general, if there are n distinct components in a statement formula, we need to consider 2^n possible combinations of truth values in order to obtain the truth table.

Two methods of constructing truth tables are shown in the following examples.

EXAMPLE 1 Construct the truth table for the statement formula $P \vee \neg Q$.

SOLUTION It is necessary to consider all possible truth values of P and Q. These values are entered in the first two columns of Table 1-2.4 for both methods. In the table which is arrived at by method 1, the truth values of $\neg Q$ are entered

Table 1-2.4a

P	Q	$\neg Q$	$P \vee \neg Q$
T	T	F	T
T	F	T	T
F	T	F	F
F	F	T	T

Method 1

Table 1-2.4b

P	Q	P	$\vee$	$\neg$	Q
T	T	T	T	F	T
T	F	T	T	T	F
F	T	F	F	F	T
F	F	F	T	T	F
Step Number		1	3	2	1

Method 2

in the third column, and the truth values of $P \vee \neg Q$ are entered in the fourth column. In method 2, as given in Table 1-2.4*b*, a column is drawn for each statement as well as for the connectives that appear. The truth values are entered step by step. The step numbers at the bottom of the table show the sequence followed in arriving at the final step. ////

EXAMPLE 2 Construct the truth table for $P \wedge \neg P$.

SOLUTION See Table 1-2.5. Note that the truth value is F for every possible truth value of P. In this special case, the truth value of $P \wedge \neg P$ is independent of the truth value of P. ////

EXAMPLE 3 Construct the truth table for $(P \vee Q) \vee \neg P$.

SOLUTION See Table 1-2.6. In this case the truth value of the formula $(P \vee Q) \vee \neg P$ is independent of the truth values of P and Q. This independence is due to the special construction of the formula, as we shall see in Sec. 1-2.8. ////

Table 1-2.5

P	$\neg P$	$P \wedge \neg P$
T	F	F
F	T	F

Method 1

P	P	$\wedge$	$\neg$	P
T	T	F	F	T
F	F	F	T	F
Step Number	1	3	2	1

Method 2

Table 1-2.6

P	Q	$P \vee Q$	$\neg P$	$(P \vee Q) \vee \neg P$
T	T	T	F	T
T	F	T	F	T
F	T	T	T	T
F	F	F	T	T

Method 1

P	Q	$(P$	$\vee$	$Q)$	$\vee$	$\neg$	P
T	T	T	T	T	T	F	T
T	F	T	T	F	T	F	T
F	T	F	T	T	T	T	F
F	F	F	F	F	T	T	F
Step Number		1	2	1	3	2	1

Method 2

Observe that if the truth values of the component statements are known, then the truth value of the resulting statement can be readily determined from the truth table by reading along the row which corresponds to the correct truth values of the component statements.

EXERCISES 1-2.4

1 Using the statements

R: Mark is rich.

H: Mark is happy.

write the following statements in symbolic form:

(*a*) Mark is poor but happy.
(*b*) Mark is rich or unhappy.
(*c*) Mark is neither rich nor happy.
(*d*) Mark is poor or he is both rich and unhappy.

2 Construct the truth tables for the following formulas.

(*a*) $\neg(\neg P \vee \neg Q)$
(*b*) $\neg(\neg P \wedge \neg Q)$
(*c*) $P \wedge (P \vee Q)$
(*d*) $P \wedge (Q \wedge P)$
(*e*) $(\neg P \wedge (\neg Q \wedge R)) \vee (Q \wedge R) \vee (P \wedge R)$
(*f*) $(P \wedge Q) \vee (\neg P \wedge Q) \vee (P \wedge \neg Q) \vee (\neg P \wedge \neg Q)$

3 For what truth values will the following statement be true? "It is not the case that houses are cold or haunted and it is false that cottages are warm or houses ugly." (*Hint*: There are four atomic statements.)

4 Given the truth values of P and Q as T and those of R and S as F, find the truth values of the following:

(*a*) $P \vee (Q \wedge R)$
(*b*) $(P \wedge (Q \wedge R)) \vee \neg((P \vee Q) \wedge (R \vee S))$
(*c*) $(\neg(P \wedge Q) \vee \neg R) \vee (((\neg P \wedge Q) \vee \neg R) \wedge S)$

1-2.5 Logical Capabilities of Programming Languages

In this section we discuss the logical connectives available in certain programming languages and how these connectives can be used to generate a truth table for a statement formula. The logical connectives discussed thus far are available in most programming languages. In PL/I, the connectives $\wedge$, $\vee$, and $\neg$ are written as &, |, and $\neg$ respectively. The truth values T and F are written as '1'B and '0'B respectively. In ALGOL the connectives are represented as we have written them, while T and F are written as **true** and **false** respectively. FORTRAN also permits the use of logical variables and expressions, and it is these facilities which are to be discussed in this section.

In FORTRAN, the truth values T and F are denoted by the logical constants .TRUE. and .FALSE. respectively. Logical variables and expressions in the language assume only one of the logical values at any given time. All logical variables must be explicitly declared as in the statement

```
LOGICAL P, Q, R
```

which declares the three variables P, Q, and R.

The statement that a relation exists between arithmetic expressions is itself an expression that has a truth value; in FORTRAN, these expressions are formulated from the following *relational operators*:

.LT.($<$) .LE.($\leq$) .EQ.($=$) .GE.($\geq$) .GT.($>$) .NE.($\neq$)

For example, if P has been declared LOGICAL, the statement

```
P = 5 * 2 .LT. 17
```

assigns a value of .TRUE. to P. Similarly, if Q has been appropriately declared, the statement

```
Q = A + 5 .GE. C + D
```

assigns the value .TRUE. to Q if A + 5 is greater than or equal to C + D when the statement is executed, and the value .FALSE. otherwise.

From the truth values arising, for example, from relations, more complex logical expressions can be obtained in FORTRAN by using one or more of the three logical connectives previously discussed. The logical operators .AND., .OR., and .NOT. correspond to the symbolic logical operators $\wedge$, $\vee$, and $\neg$ respectively. The statement

$$P \vee (\neg(Q \wedge R))$$

is equivalent to the FORTRAN statement

```
P .OR. (.NOT. (Q .AND. R))
```

Unnecessary parentheses are avoided in FORTRAN by using the following precedence scheme. The arithmetic operators, with their usual order of precedence, are the highest in rank and are consequently evaluated first. All relational operators have the same rank and are evaluated after the arithmetic operators. The logical operators are the last to be evaluated, and .NOT., .AND., and .OR. is their decreasing order of precedence. Of two or more binary operators having the same precedence value in an expression, the leftmost is evaluated first; for unary operators, it is the rightmost which is evaluated first. Thus, .NOT. P .AND. Q means (.NOT. P) .AND. Q; and A + B + 5.0 .LT. C + D means ((A + B) + 5.0) .LT. (C + D).

FORTRAN has a logical "IF statement" whose form is

```
IF (logical expression) statement
```

If the logical expression in the "IF statement" is true, then the statement following the expression is executed; otherwise, it is skipped. For example, when the statement

```
IF (.NOT. P .OR. Q) GO TO 100
```

is executed, it will not transfer control to statement 100 if P and Q have the values .TRUE. and .FALSE. respectively.

Arrays of logical variables can also be used in FORTRAN. The statement

```
LOGICAL CASE(10)
```

declares a one-dimensional array of type LOGICAL consisting of 10 elements. Elements of logical arrays are referenced in the same manner as any other subscripted variable.

Consider the problem of generating all possible assignments of truth values to the logical variables P, Q, and R, as shown in Table 1-2.7. There are $2^3 = 8$ possible assignments. Notice that the truth value of the variable P remains at the same value of T or F for each of four consecutive assignments of logical values. The values of variables Q and R remain at T or F for two assignments and one assignment of logical values respectively. The value of variable R changes more frequently than the value of variable Q, and that of Q more frequently than that of P. The number of times the kth logical variable remains at a constant truth value can be easily computed and is denoted by $BASE[k]$. In the case under discussion, we have three variables, and the values can be computed as

$$BASE[k] = 2^{(3-k)} \qquad k = 1, 2, 3$$

where we have associated $BASE[1]$, $BASE[2]$, and $BASE[3]$ with variables P, Q, and R respectively.

In addition to computing the $BASE$ elements, we also need to know the number of assignments which remain to be generated with a particular logical variable remaining at the same value. For example, if we had already generated the assignments TTT and TTF, then variable P would remain at its present value of T throughout the generation of the next two assignments. This information is stored in an element denoted by $LENGTH[k]$. For variable P, $LENGTH[1]$ would have a value of 2 after generation of TTT and TTF. The $LENGTH$ values associated with variables P, Q, and R are initially the same as their corresponding $BASE$ values. Therefore, initially

$$LENGTH[k] = BASE[k] \qquad k = 1, 2, 3$$

Every time an assignment is generated, each element of $LENGTH$ is decremented by 1. When the $LENGTH$ value associated with a variable becomes zero, then the truth value of that variable is negated, and the $LENGTH$ value is reset to the $BASE$ value. The algorithm for the generation of such assignments can now be precisely formulated.

Table 1-2.7

	P	Q	R	
$BASE[1]$	T	T	T	$BASE[3]$
	T	T	F	
	T	F	T	
	T	F	F	
$BASE[2]$	F	T	T	
	F	T	F	
	F	F	T	
	F	F	F	

Algorithm *NEXT* Given n logical variables having values stored in $CASE[1]$, $CASE[2]$, ..., $CASE[n]$ and two vectors $BASE$ and $LENGTH$ each having n elements, it is required to generate the next assignment of truth values for these variables.

```
1 [Initialize counter] Set $k \leftarrow 1$.
2 [Decrement $LENGTH[k]$] Set $LENGTH[k] \leftarrow LENGTH[k] - 1$.
3 [Negate variable and reset $LENGTH[k]$?] If $LENGTH[k] = 0$ then
  set $CASE[k] \leftarrow \neg CASE[k]$ and $LENGTH[k] \leftarrow BASE[k]$.
4 [Increment counter] Set $k \leftarrow k + 1$. If $k \leq n$ then go to step 2; other-
  wise Exit.                                                              ////
```

A program for algorithm *NEXT* is given in Fig. 1-2.1. The subroutine has the four parameters CASE, N, BASE, and LENGTH. All parameters except N are arrays. The logical array CASE contains an assignment of truth values for the logical variables from which the subroutine is to generate a new assignment of values. For example, for the case of three logical variables, CASE(1), CASE(2), and CASE(3) could be associated with the variable names P, Q, and R respectively. The new assignment of truth values is returned to the main program via the logical array CASE.

Let us now consider the problem of constructing a truth table for a statement formula. The following straightforward algorithm uses the various logical arrays such as *CASE*, *BASE*, and *LENGTH* which were discussed in algorithm *NEXT*.

Algorithm *TRUTH* Given a statement formula in n variables and subalgorithm *NEXT* which generates a new assignment of truth values, it is required to construct a truth table for the given statement formula.

```
1 [Initialize] Repeat for $k = 1, 2, \ldots, n$: Set $BASE[k] \leftarrow 2^{(n-k)}$,
  $LENGTH[k] \leftarrow BASE[k]$, and $CASE[k] \leftarrow F$. Set $i \leftarrow 1$ and print
  headings for the truth table.
2 [Evaluate statement] Substitute the logical values in array $CASE$
  into the statement formula. Print the values in array $CASE$ and the
  value of the statement.
3 [Obtain next assignment for variables] Invoke subalgorithm $NEXT$.
```

```
      SUBROUTINE NEXT(CASE,N,BASE,LENGTH)
C     GENERATE THE NEXT ASSIGNMENT OF LOGICAL VALUES.
      LOGICAL CASE(N)
      INTEGER BASE(N),LENGTH(N)
      DO 1 K = 1,N
      LENGTH(K) = LENGTH(K) - 1
      IF(LENGTH(K).NE.0) GO TO 1
      CASE(K) = .NOT.CASE(K)
      LENGTH(K) = BASE(K)
    1 CONTINUE
      END
```

FIGURE 1-2.1 **Program for algorithm *NEXT*.**

4 [Increment counter] Set $i \leftarrow i + 1$. If $i \leq 2^n$ then go to step 2; otherwise Exit. ////

The FORTRAN program for the algorithm is given in Fig. 1-2.2. As an example, the formula

$$\neg(P \wedge Q) \vee (R \vee P)$$

was used in the program. The program consists of a main program, a subroutine, and a function. The subroutine NEXT, given in Fig. 1-2.1, generates an assignment each time it is invoked. The function LOGIC is very simple, and its purpose is to generate a single truth value for the statement formula each time the function is invoked. The number of logical variables in the given statement formula and their associated values are passed to the function LOGIC by using the integer variable N and the logical vector CASE respectively.

For our example, the variables P, Q, and R are denoted in the program by the subscripted variables CASE(1), CASE(2), and CASE(3) respectively.

Each time the main program needs a new assignment of truth values for the variables, it calls on procedure NEXT after which the function LOGIC is invoked to evaluate the statement formula for this new assignment of values. The main program computes the BASE and LENGTH vectors for subroutine NEXT.

Initially, all logical variables are set to false, which enables subroutine NEXT to obtain the next assignment. Note that all variables could have been set to true instead. This assignment, of course, would have produced a truth table with the same information as shown in the sample output but in a different order.

1-2.6 Conditional and Biconditional

If P and Q are any two statements, then the statement $P \rightarrow Q$ which is read as "If P, then Q" is called a *conditional* statement. The statement $P \rightarrow Q$ has a truth value F when Q has the truth value F and P the truth value T; otherwise it has the truth value T. The conditional is defined by Table 1-2.8.

The statement P is called the *antecedent* and Q the *consequent* in $P \rightarrow Q$. Again, according to the definition, it is not necessary that there be any kind of relation between P and Q in order to form $P \rightarrow Q$.

Table 1-2.8 TRUTH TABLE FOR CONDITIONAL

P	Q	$P \rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

```
C   MAINLINE
C   THIS PROGRAM EVALUATES A STATEMENT FORMULA
C   AND GENERATES ITS TRUTH TABLE.
C
C   VARIABLES
C   TITLE:    TITLE FOR THE STATEMENT FORMULA
C   NAME:     VARIABLE NAMES
C   VALUE:    LOGICAL VALUE OF STATEMENT FORMULA
C   NUMBER:   NUMBER OF ROWS IN THE TRUTH TABLE
C
C   DECLARATIONS AND TITLES
        INTEGER*2 NAME(3)/'P','Q','R'/
        REAL*8 TITLE(4)/'.NOT.(P.','AND.Q).O','R.(R.OR.','P)
        LOGICAL CASE(10),LOGIC,VALUE
        INTEGER BASE(10),LENGTH(10)
C   INITIALIZE BASE, LENGTH, CASE, N, AND NUMBER.
        N = 3
        NUMBER = 2 ** N
        DO 1 K = 1,N
        BASE(K) = 2 ** (N - K)
        LENGTH(K) = BASE(K)
        CASE(K) = .FALSE.
      1 CONTINUE
C   OUTPUT HEADINGS
        WRITE(6,10) TITLE
    10 FORMAT('1',13X,'VARIABLES',13X,4A8)
        WRITE(6,20) NAME
    20 FORMAT(' ',8X,'CASE 1   2   3',/,' ',13X,3(A2,2X),21X,'VALUE',/)
C   FIND VALUE OF THE STATEMENT FORMULA, OUTPUT TRUTH VALUES,
C   AND GENERATE NEW TRUTH VALUES FOR THE LOGICAL VARIABLES.
        DO 2 I = 1,NUMBER
        VALUE = LOGIC(CASE,N)
        WRITE(6,30) (CASE(K),K = 1,N),VALUE
    30 FORMAT(' ',13X,3(L1,3X),23X,L1)
        CALL NEXT(CASE,N,BASE,LENGTH)
      2 CONTINUE
        STOP
        END

        LOGICAL FUNCTION LOGIC(CASE,N)
C   THIS FUNCTION DEFINES THE STATEMENT FORMULA TO BE EVALUATED.
        LOGICAL CASE(N)
        LOGIC = .NOT.(CASE(1).AND.CASE(2)).OR.(CASE(3).OR.CASE(1))
        RETURN
        END
```

```
          VARIABLES                 .NOT.(P.AND.Q).OR.(R.OR.P)
    CASE  1     2     3
          P     Q     R                      VALUE

          F     F     F                        T
          F     F     T                        T
          F     T     F                        T
          F     T     T                        T
          T     F     F                        T
          T     F     T                        T
          T     T     F                        T
          T     T     T                        T
```

FIGURE 1-2.2 Program for generating truth tables—mainline and function LOGIC.

EXAMPLE 1 Express in English the statement $P \rightarrow Q$ where

P: The sun is shining today.

Q: $2 + 7 > 4$.

SOLUTION If the sun is shining today, then $2 + 7 > 4$. ////

The conditional often appears very confusing to a beginner, particularly when one tries to translate a conditional in English into symbolic form. A variety of expressions are used in English which can be appropriately translated by the symbol $\rightarrow$. It is customary to represent any one of the following expressions by $P \rightarrow Q$:

1 Q is necessary for P.
2 P is sufficient for Q.
3 Q if P.
4 P only if Q.
5 P implies Q.

We shall avoid the translation "implies." Although, in mathematics, the statements "If P, then Q" and "P implies Q" are used interchangeably, we want to use the word "implies" in a different way.

In our everyday language, we use the conditional statements in a more restricted sense. It is customary to assume some kind of relationship or implication or feeling of cause and effect between the antecedent and the consequent in using the conditional. For example, the statement "If I get the book, then I shall read it tonight" sounds reasonable because the second statement "I shall read it (the book) tonight" refers to the book mentioned in the first part of the statement. On the other hand, a statement such as "If I get the book, then this room is red" does not make sense to us in our conventional language. However, according to our definition of the conditional, the last statement is perfectly acceptable and has a truth value which depends on the truth values of the two statements being connected.

The first two entries in Table 1-2.8 are similar to what we would expect in our everyday language. Thus, if P is true and Q is true, then $P \rightarrow Q$ is true. Similarly, if P is true and Q is false, then "If P, then Q" appears to be false. Consider, for example, the statement "If I get the money, then I shall buy the car." If I actually get the money and buy the car, then the statement appears to be correct or true. On the other hand, if I do not buy the car even though I get the money, then the statement is false. Normally, when a conditional statement is made, we assume that the antecedent is true. Because of this convention in English, the first two entries in the truth table do not appear strange. Referring to the above statement again, if I do not get the money and I still buy the car, it is not so clear whether the statement made earlier is true or false. Also, if I do not buy the car and I do not get the money, then it is not intuitively clear whether the statement made is true or false. It may be possible to justify entries in the last two rows of the truth table by considering special examples or even by emphasizing certain aspects of the statements given in the above examples. However, it is best to consider Table 1-2.8 as the definition of the conditional in which the entries in the last two rows are arbitrarily assigned in order to avoid any am-

biguity. Any other choice for the last two entries would correspond to some other connective which has either been defined or will be defined. In general, the use of "If ..., then ..." in English has only partial resemblance to the use of the conditional → as defined here.

EXAMPLE 2 Write the following statement in symbolic form.

If either Jerry takes Calculus or Ken takes Sociology, then Larry will take English.

SOLUTION Denoting the statements as

J: Jerry takes Calculus.

K: Ken takes Sociology.

L: Larry takes English.

the above statement can be symbolized as

$$(J \vee K) \rightarrow L \qquad ////$$

EXAMPLE 3 Write in symbolic form the statement

The crop will be destroyed if there is a flood.

SOLUTION Let the statements be denoted as

C: The crop will be destroyed.

F: There is a flood.

Note that the given statement uses "if" in the sense of "If ..., then ...." It is better to rewrite the given statement as "If there is a flood, then the crop will be destroyed." Now it is easy to symbolize it as

$$F \rightarrow C \qquad ////$$

EXAMPLE 4 Construct the truth table for $(P \rightarrow Q) \wedge (Q \rightarrow P)$.

SOLUTION See Table 1-2.9. Note that the given formula has the truth value T whenever both P and Q have identical truth values. ////

If P and Q are any two statements, then the statement $P \rightleftarrows Q$, which is read as "P if and only if Q" and abbreviated as "P iff Q," is called a *biconditional statement*. The statement $P \rightleftarrows Q$ has the truth value T whenever both P and

Table 1-2.9

P	Q	$P \rightarrow Q$	$Q \rightarrow P$	$(P \rightarrow Q) \wedge (Q \rightarrow P)$
T	T	T	T	T
T	F	F	T	F
F	T	T	F	F
F	F	T	T	T

Table 1-2.10 TRUTH TABLE FOR BICONDITIONAL

P	Q	$P \rightleftarrows Q$
T	T	T
T	F	F
F	T	F
F	F	T

Table 1-2.11

P	Q	$P \wedge Q$	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$	$\neg P \vee \neg Q$	$\neg(P \wedge Q) \rightleftarrows (\neg P \vee \neg Q)$
T	T	T	F	F	F	F	T
T	F	F	T	F	T	T	T
F	T	F	T	T	F	T	T
F	F	F	T	T	T	T	T

Q have identical truth values. Table 1-2.10 defines the biconditional. The statement $P \rightleftarrows Q$ is also translated as "P is necessary and sufficient for Q." Note that the truth values of $(P \to Q) \wedge (Q \to P)$ given in Table 1-2.9 are identical to the truth values of $P \rightleftarrows Q$ defined here.

EXAMPLE 5 Construct the truth table for the formula

$$\neg(P \wedge Q) \rightleftarrows (\neg P \vee \neg Q)$$

SOLUTION See Table 1-2.11. Note that the truth values of the given formula are T for all possible truth values of P and Q. ////

EXERCISES 1-2.6

1 Show that the truth values of the following formulas are independent of their components.
(*a*) $(P \wedge (P \to Q)) \to Q$
(*b*) $(P \to Q) \rightleftarrows (\neg P \vee Q)$
(*c*) $((P \to Q) \wedge (Q \to R)) \to (P \to R)$
(*d*) $(P \rightleftarrows Q) \rightleftarrows ((P \wedge Q) \vee (\neg P \wedge \neg Q))$

2 Construct the truth tables of the following formulas.
(*a*) $(Q \wedge (P \to Q)) \to P$
(*b*) $\neg(P \vee (Q \wedge R)) \rightleftarrows ((P \vee Q) \wedge (P \vee R))$

3 A connective denoted by $\overline{\vee}$ is defined by Table 1-2.12. Find a formula using P, Q, and the connectives $\wedge$, $\vee$, and $\neg$ whose truth values are identical to the truth values of $P \overline{\vee} Q$.

4 Given the truth values of P and Q as T and those of R and S as F, find the truth values of the following:
(*a*) $(\neg(P \wedge Q) \vee \neg R) \vee ((Q \rightleftarrows \neg P) \to (R \vee \neg S))$
(*b*) $(P \rightleftarrows R) \wedge (\neg Q \to S)$
(*c*) $(P \vee (Q \to (R \wedge \neg P))) \rightleftarrows (Q \vee \neg S)$

Table 1-2.12

P	Q	$P \bar{\vee} Q$
T	T	F
T	F	T
F	T	T
F	F	F

1-2.7 Well-formed Formulas

The notion of a statement formula has already been introduced. A statement formula is not a statement (although, for the sake of brevity, we have often called it a statement); however, a statement can be obtained from it by replacing the variables by statements. A *statement formula* is an expression which is a string consisting of variables (capital letters with or without subscripts), parentheses, and connective symbols. Not every string of these symbols is a formula. We shall now give a recursive definition of a statement formula, often called a well-formed formula (wff). A *well-formed formula* can be generated by the following rules:

1 A statement variable standing alone is a well-formed formula.

2 If A is a well-formed formula, then $\urcorner A$ is a well-formed formula.

3 If A and B are well-formed formulas, then $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, and $(A \rightleftarrows B)$ are well-formed formulas.

4 A string of symbols containing the statement variables, connectives, and parentheses is a well-formed formula, iff it can be obtained by finitely many applications of the rules 1, 2, and 3.

According to this definition, the following are well-formed formulas:

$$\urcorner(P \wedge Q) \qquad \urcorner(P \vee Q) \qquad (P \rightarrow (P \vee Q)) \qquad (P \rightarrow (Q \rightarrow R))$$

$$(((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightleftarrows (P \rightarrow R))$$

The following are not well-formed formulas.

1 $\urcorner P \wedge Q$. Obviously P and Q are well-formed formulas. A wff would be either $(\urcorner P \wedge Q)$ or $\urcorner(P \wedge Q)$.

2 $(P \rightarrow Q) \rightarrow (\wedge Q)$. This is not a wff because $\wedge Q$ is not.

3 $(P \rightarrow Q$. Note that $(P \rightarrow Q)$ is a wff.

4 $(P \wedge Q) \rightarrow Q)$. The reason for this not being a wff is that one of the parentheses in the beginning is missing. $((P \wedge Q) \rightarrow Q)$ is a wff, while $(P \wedge Q) \rightarrow Q$ is still not a wff.

It is possible to introduce some conventions so that the number of parentheses used can be reduced. In fact, there are conventions which, when followed, allow one to dispense with all the parentheses. We shall not discuss these conventions here. For the sake of convenience we shall omit the outer parentheses. Thus we write $P \wedge Q$ in place of $(P \wedge Q)$, $(P \wedge Q) \rightarrow Q$ in place of $((P \wedge Q) \rightarrow Q)$, and $((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightleftarrows (P \rightarrow R)$ instead of $(((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightleftarrows (P \rightarrow R))$. Since the only formulas we will encounter are well-formed formulas, we will refer to well-formed formulas as formulas.

1-2.8 Tautologies

Well-formed formulas have been defined. We also know how to construct the truth table of a given formula. Let us consider what a truth table represents. If definite statements are substituted for the variables in a formula, there results a statement. The truth value of this resulting statement depends upon the truth values of the statements substituted for the variables. Such a truth value appears as one of the entries in the final column of the truth table. Observe that this entry will not change even if any of the definite statements that replace particular variables are themselves replaced by other statements, as long as the truth values associated with all variables are unchanged. In other words, an entry in the final column depends only on the truth values of the statements assigned to the variables rather than on the statements themselves. Different rows correspond to different sets of truth value assignments. A truth table is therefore a summary of the truth values of the resulting statements for all possible assignments of values to the variables appearing in a formula. It must be emphasized that a statemen' formula does not have a truth value. In our discussion which follows we shall, for the sake of simplicity, use the expression "the truth value of a statement formula" to mean the entries in the final column of the truth table of the formula.

In general, the final column of a truth table of a given formula contains both T and F. There are some formulas whose truth values are always T or always F regardless of the truth value assignments to the variables. This situation occurs because of the special construction of these formulas. We have already seen some examples of such formulas.

Consider, for example, the statement formulas $P \vee \neg P$ and $P \wedge \neg P$ in Table 1-2.13. The truth values of $P \vee \neg P$ and $P \wedge \neg P$, which are T and F respectively, are independent of the statement by which the variable P may be replaced.

A statement formula which is true regardless of the truth values of the statements which replace the variables in it is called a *universally valid formula* or a *tautology* or a *logical truth*. A statement formula which is false regardless of the truth values of the statements which replace the variables in it is called a *contradiction*. Obviously, the negation of a contradiction is a tautology. We may say that a statement formula which is a tautology is *identically true* and a formula which is a contradiction is *identically false*.

A straightforward method to determine whether a given formula is a tautology is to construct its truth table. This process can always be used but often becomes tedious, particularly when the number of distinct variables is large or when the formula is complicated. Recall that the numbers of rows in a truth table is 2^n, where n is the number of distinct variables in the formula. Later,

Table 1-2.13

P	$\neg P$	$P \vee \neg P$	$P \wedge \neg P$
T	F	T	F
F	T	T	F

alternative methods will be developed that will be able to determine whether a statement formula is a tautology without having to construct its truth table.

A simple fact about tautologies is that the conjunction of two tautologies is also a tautology. Let us denote by A and B two statement formulas which are tautologies. If we assign any truth values to the variables of A and B, then the truth values of both A and B will be T. Thus the truth value of $A \wedge B$ will be T, so that $A \wedge B$ will be a tautology.

A formula A is called a *substitution instance* of another formula B if A can be obtained from B by substituting formulas for some variables of B, with the condition that the same formula is substituted for the same variable each time it occurs. We now illustrate this concept. Let

$$B\colon P \to (J \wedge P)$$

Substitute $R \rightleftarrows S$ for P in B, and we get

$$A\colon\ (R \rightleftarrows S) \to (J \wedge (R \rightleftarrows S))$$

Then A is a substitution instance of B. Note that

$$(R \rightleftarrows S) \to (J \wedge P)$$

is not a substitution instance of B because the variable P in $J \wedge P$ was not replaced by $R \rightleftarrows S$. It is possible to substitute more than one variable by other formulas, provided that all substitutions are considered to occur simultaneously. For example, substitution instances of $P \to \neg Q$ are

1 $(R \wedge \neg S) \to \neg(J \vee M)$
2 $(R \wedge \neg S) \to \neg(R \wedge \neg S)$
3 $(R \wedge \neg S) \to \neg P$
4 $Q \to \neg(P \wedge \neg Q)$

In (2) both P and Q have been replaced by $R \wedge \neg S$. In (4), P is replaced by Q and Q by $P \wedge \neg Q$.

Next, consider the following formulas which result from $P \to \neg Q$.

1 Substitute $P \vee Q$ for P and R for Q to get the substitution instance $(P \vee Q) \to \neg R$.

2 First substitute $P \vee Q$ for P to obtain the substitution instance $(P \vee Q) \to \neg Q$. Next, substitute R for Q in $(P \vee Q) \to \neg Q$, and we get $(P \vee R) \to \neg R$. This formula is a substitution instance of $(P \vee Q) \to \neg Q$, but it is not a substitution instance of $P \to \neg Q$ under the substitution $(P \vee Q)$ for P and R for Q. This statement is true because we did not substitute simultaneously as we did in (1).

It may be noted that in constructing substitution instances of a formula, substitutions are made for the atomic formula and never for the molecular formula. Thus $P \to Q$ is not a substitution instance of $P \to \neg R$, because it is R which must be replaced and not $\neg R$.

The importance of the above concept lies in the fact that any substitution instance of a tautology is a tautology. Consider the tautology $P \vee \neg P$. Regardless of what is substituted for P, the truth value of $P \vee \neg P$ is always T. Therefore, it we substitute any statement formula for P, the resulting formula will be a tautology. Hence the following substitution instances of $P \vee \neg P$ are tautologies.

$$(R \to S) \vee \neg(R \to S)$$
$$((P \vee S) \wedge R) \vee \neg((P \vee S) \wedge R)$$
$$(((P \vee \neg Q) \to R) \rightleftarrows S) \vee \neg(((P \vee \neg Q) \to R) \rightleftarrows S)$$

Thus, if it is possible to detect whether a given formula is a substitution instance of a tautology, then it is immediately known that the given formula is also a tautology. Similarly, one can start with a tautology and write a large number of formulas which are substitution instances of this tautology and hence are themselves tautologies.

EXERCISES 1-2.8

1 From the formulas given below select those which are well-formed according to the definition in Sec. 1-2.7, and indicate which ones are tautologies or contradictions.
(*a*) $(P \to (P \vee Q))$
(*b*) $((P \to (\neg P)) \to \neg P)$
(*c*) $((\neg Q \wedge P) \wedge Q)$
(*d*) $((P \to (Q \to R)) \to ((P \to Q) \to (P \to R)))$
(*e*) $((\neg P \to Q) \to (Q \to P)))$
(*f*) $((P \wedge Q) \rightleftarrows P)$

2 Produce the substitution instances of the following formulas for the given substitutions.
(*a*) $(((P \to Q) \to P) \to P)$; substitute $(P \to Q)$ for P and $((P \wedge Q) \to R)$ for Q.
(*b*) $((P \to Q) \to (Q \to P))$; substitute Q for P and $(P \wedge \neg P)$ for Q.

3 Determine the formulas which are substitution instances of other formulas in the list and give the substitutions.
(*a*) $(P \to (Q \to P))$
(*b*) $((((P \to Q) \wedge (R \to S)) \wedge (P \vee R)) \to (Q \vee S))$
(*c*) $(Q \to ((P \to P) \to Q))$
(*d*) $(P \to ((P \to (Q \to P)) \to P))$
(*e*) $((((R \to S) \wedge (Q \to P)) \wedge (R \vee Q)) \to (S \vee P))$

1-2.9 Equivalence of Formulas

Let A and B be two statement formulas and let $P_1, P_2, \ldots, P_n$ denote all the variables occurring in both A and B. Consider an assignment of truth values to $P_1, P_2, \ldots, P_n$ and the resulting truth values of A and B. If the truth value of A is equal to the truth value of B for every one of the 2^n possible sets of truth values assigned to $P_1, P_2, \ldots, P_n$, then A and B are said to be *equivalent*. Assuming that the variables and the assignment of truth values to the variables appear in the same order in the truth tables of A and B, then the final columns in the truth tables for A and B are identical if A and B are equivalent.

Here are some examples of formulas which are equivalent. Verify their equivalence by truth tables.

1 $\neg\neg P$ is equivalent to P.
2 $P \vee P$ is equivalent to P.

3 $(P \wedge \neg P) \vee Q$ is equivalent to Q.
4 $P \vee \neg P$ is equivalent to $Q \vee \neg Q$.

In the definition of equivalence of two formulas, it is not necessary to assume that they both contain the same variables. This point is illustrated in the examples given in (3) and (4) above. It may, however, be noted that if two formulas are equivalent and a particular variable occurs in only one of them, then the truth value of this formula is independent of this variable. For example, in (3) the truth value of $(P \wedge \neg P) \vee Q$ is independent of the truth value of P. Similarly in (4), the truth values of $P \vee \neg P$ and $Q \vee \neg Q$ are each independent of P and Q.

Recalling the truth table (Table 1-2.10) in the definition of the biconditional, it is clear that $P \rightleftarrows Q$ is true whenever both P and Q have the same truth values. Therefore the statement formulas A and B are equivalent provided $A \rightleftarrows B$ is a tautology; and, conversely, if $A \rightleftarrows B$ is a tautology, then A and B are equivalent. We shall represent the equivalence of two formulas, say A and B, by writing "$A \Leftrightarrow B$," which is read as "A is equivalent to B." Note that the expression "$A \Leftrightarrow B$" which can also be displayed as

$$A \Leftrightarrow B$$

should be written as

$$``A" \Leftrightarrow ``B"$$

according to the rules given earlier (in Sec. 1-1) regarding the use and mention of expressions. Observe that "$A \Leftrightarrow B$" is a statement in English (the metalanguage) and not in the object language. Also the symbol "$\Leftrightarrow$" is not a connective but a symbol in the metalanguage. Having noted this, we shall often drop the quotation marks because this will not lead to any ambiguity.

Equivalence is a symmetric relation; that is, "A is equivalent to B" is the same as "B is equivalent to A." Also if $A \Leftrightarrow B$ and $B \Leftrightarrow C$, then $A \Leftrightarrow C$. This relationship may also be expressed by saying that the equivalence of statement formulas is transitive.

As in the case of tautologies, one method to determine whether any two statement formulas are equivalent is to construct their truth tables. All combinations of truth values associated with the variables appearing in both formulas are presented in the table, and the final columns (for the two formulas) are compared.

EXAMPLE 1 Prove $(P \to Q) \Leftrightarrow (\neg P \vee Q)$.

SOLUTION See Table 1-2.14. Note that the truth values in the columns for $P \to Q$ and $\neg P \vee Q$ are identical, and so the biconditional will have the truth value T. To compare columns, it is not necessary to form the biconditional; thus the last column could have been avoided. ////

A list of some basic equivalent formulas which will be found useful is given in Table 1-2.15. In order to make the list complete, we use **T** and **F** as special variables in the sense that **T** can be replaced by only a tautology and **F** by only a contradiction.

Table 1-2.14

P	Q	$P \to Q$	$\neg P$	$\neg P \vee Q$	$(P \to Q) \rightleftarrows (\neg P \vee Q)$
T	T	T	F	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

In view of the associative laws, we can write $(P \vee Q) \vee R$ as $P \vee Q \vee R$, and $(P \wedge Q) \wedge R$ as $P \wedge Q \wedge R$.

In Table 1-2.15 we note that pairs of equivalent formulas are arranged two to a line such as

$$A_1 \Leftrightarrow B_1 \qquad A_2 \Leftrightarrow B_2$$

For each pair A_1, B_1 there is a corresponding pair A_2, B_2 in which $\vee$ is replaced by $\wedge$, $\wedge$ by $\vee$, **T** by **F**, and **F** by **T**. A_1 and A_2 are said to be duals of each other, and so are B_1 and B_2. Duality is discussed in Sec. 1-2.10.

In constructing substitution instances of a statement formula, we are allowed to substitute only for the variables appearing in the formula. Furthermore, the same formula is to be substituted for every occurrence of a particular variable. This rule ensures that substitution instances of a tautology are also tautologies. Consider now another process, called a *replacement process*, in which we replace any part of a statement formula which is itself a formula, be it atomic or molecular, by any other formula. For example, in the formula $(P \wedge Q) \to P$ we replace $(P \wedge Q)$ by $R \to (S \wedge \neg M)$ and the second P by $(P \wedge R) \to (\neg S \vee M)$ to obtain $(R \to (S \wedge \neg M)) \to ((P \wedge R) \to (\neg S \vee M))$. In general, a replacement yields a new formula, but it may not always be an interesting formula. However, if we impose the restriction that any part of a given formula that is to be replaced by another formula must be equivalent to that other formula, then the result is equivalent to the original formula. By this process one can obtain new formulas which are equivalent to the original formula. For example, we can replace P in $P \wedge Q$ by the formula $P \vee P$, since $P \vee P \Leftrightarrow P$, to get $(P \vee P) \wedge Q$ which is equivalent to $P \wedge Q$. Consequently, if we replace any part or parts of a tautology by formulas that are equivalent to these parts, we again get a tautology.

EXAMPLE 2 Show that $P \to (Q \to R) \Leftrightarrow P \to (\neg Q \vee R) \Leftrightarrow (P \wedge Q) \to R$.

SOLUTION Recall from Example 1 that $Q \to R \Leftrightarrow \neg Q \vee R$. Replacing $Q \to R$ by $\neg Q \vee R$, we get $P \to (\neg Q \vee R)$, which is equivalent to $\neg P \vee (\neg Q \vee R)$ by the same rule. Now

$$\begin{aligned}\neg P \vee (\neg Q \vee R) \Leftrightarrow (\neg P \vee \neg Q) \vee R &\Leftrightarrow \neg(P \wedge Q) \vee R \\ &\Leftrightarrow (P \wedge Q) \to R\end{aligned}$$

using associativity of $\vee$, De Morgan's law, and the previously used rule. ////

Table 1-2.15 EQUIVALENT FORMULAS

$P \lor P \Leftrightarrow P$	$P \land P \Leftrightarrow P$	(Idempotent laws)	(1)
$(P \lor Q) \lor R \Leftrightarrow P \lor (Q \lor R)$	$(P \land Q) \land R \Leftrightarrow P \land (Q \land R)$	(Associative laws)	(2)
$P \lor Q \Leftrightarrow Q \lor P$	$P \land Q \Leftrightarrow Q \land P$	(Commutative laws)	(3)
$P \lor (Q \land R) \Leftrightarrow (P \lor Q) \land (P \lor R)$	$P \land (Q \lor R) \Leftrightarrow (P \land Q) \lor (P \land R)$	(Distributive laws)	(4)
$P \lor \mathbf{F} \Leftrightarrow P$	$P \land \mathbf{T} \Leftrightarrow P$		(5)
$P \lor \mathbf{T} \Leftrightarrow \mathbf{T}$	$P \land \mathbf{F} \Leftrightarrow \mathbf{F}$		(6)
$P \lor \neg P \Leftrightarrow \mathbf{T}$	$P \land \neg P \Leftrightarrow \mathbf{F}$		(7)
$P \lor (P \land Q) \Leftrightarrow P$	$P \land (P \lor Q) \Leftrightarrow P$	(Absorption laws)	(8)
$\neg(P \lor Q) \Leftrightarrow \neg P \land \neg Q$	$\neg(P \land Q) \Leftrightarrow \neg P \lor \neg Q$	(De Morgan's laws)	(9)

EXAMPLE 3 Show that $(\neg P \wedge (\neg Q \wedge R)) \vee (Q \wedge R) \vee (P \wedge R) \Leftrightarrow R$.

SOLUTION

$$
\begin{aligned}
&(\neg P \wedge (\neg Q \wedge R)) \vee (Q \wedge R) \vee (P \wedge R) \\
&\quad\Leftrightarrow (\neg P \wedge (\neg Q \wedge R)) \vee ((Q \vee P) \wedge R) && (4)\\
&\quad\Leftrightarrow ((\neg P \wedge \neg Q) \wedge R) \vee ((Q \vee P) \wedge R) && (2)\\
&\quad\Leftrightarrow ((\neg P \wedge \neg Q) \vee (Q \vee P)) \wedge R && (4)\\
&\quad\Leftrightarrow (\neg (P \vee Q) \vee (P \vee Q)) \wedge R && (9), (3)\\
&\quad\Leftrightarrow \mathbf{T} \wedge R && (7)\\
&\quad\Leftrightarrow R && (5)
\end{aligned}
$$

The basic equivalent statement formulas used are denoted by the numbers on the right-hand side which correspond to numbers in Table 1-2.15. ////

EXAMPLE 4 Show that $((P \vee Q) \wedge \neg(\neg P \wedge (\neg Q \vee \neg R))) \vee (\neg P \wedge \neg Q) \vee (\neg P \wedge \neg R)$ is a tautology.

SOLUTION Using De Morgan's laws, we obtain

$$
\begin{aligned}
\neg P \wedge \neg Q \Leftrightarrow \neg(P \vee Q) \qquad & \neg P \wedge \neg R \Leftrightarrow \neg(P \vee R)\\
(\neg P \wedge \neg Q) \vee (\neg P \wedge \neg R) & \Leftrightarrow \neg(P \vee Q) \vee \neg(P \vee R)\\
& \Leftrightarrow \neg((P \vee Q) \wedge (P \vee R))
\end{aligned}
$$

Also

$$
\begin{aligned}
\neg(\neg P \wedge (\neg Q \vee \neg R)) & \Leftrightarrow \neg(\neg P \wedge \neg(Q \wedge R))\\
& \Leftrightarrow P \vee (Q \wedge R) \Leftrightarrow (P \vee Q) \wedge (P \vee R)\\
(P \vee Q) \wedge ((P \vee Q) \wedge (P \vee R)) & \Leftrightarrow (P \vee Q) \wedge (P \vee R)
\end{aligned}
$$

Consequently, the given formula is equivalent to

$$((P \vee Q) \wedge (P \vee R)) \vee \neg((P \vee Q) \wedge (P \vee R))$$

which is a substitution instance of $P \vee \neg P$. ////

The equivalences given in Table 1-2.15 also describe the properties of the operators $\wedge$, $\vee$, and $\neg$ on the set of statements in symbolic logic. It is shown in Chap. 4 that the set of all statements under the operations $\wedge$, $\vee$, and $\neg$ is an algebra called the *statement algebra* which is a particular example of a Boolean algebra. A comparison of the statement algebra and the set algebra is given in Chap. 2.

1-2.10 Duality Law

In this section we shall consider formulas which contain the connectives $\wedge$, $\vee$, and $\neg$. There is no loss of generality in restricting our consideration to these connectives since we shall see later that any formula containing any other connective can be replaced by an equivalent formula containing only these three connectives.

Two formulas, A and A^*, are said to be *duals* of each other if either one can be obtained from the other by replacing $\wedge$ by $\vee$ and $\vee$ by $\wedge$. The connectives $\wedge$ and $\vee$ are also called *duals* of each other. If the formula A contains the special variables **T** or **F**, then A^*, its dual, is obtained by replacing **T** by **F** and **F** by **T** in addition to the above-mentioned interchanges.

EXAMPLE 1 Write the duals of (*a*) $(P \vee Q) \wedge R$; (*b*) $(P \wedge Q) \vee \mathbf{T}$; (*c*) $\neg(P \vee Q) \wedge (P \vee \neg(Q \wedge \neg S))$.

SOLUTION The duals are (*a*) $(P \wedge Q) \vee R$, (*b*) $(P \vee Q) \wedge \mathbf{F}$, and (*c*) $\neg(P \wedge Q) \vee (P \wedge \neg(Q \vee \neg S))$. ////

The following theorem shows the equivalence of a formula and one that is obtained from its dual.

Theorem 1-2.1 Let A and A^* be dual formulas and let $P_1, P_2, \ldots, P_n$ be all the atomic variables that occur in A and A^*. That is to say, we may write A as $A(P_1, P_2, \ldots, P_n)$ and A^* as $A^*(P_1, P_2, \ldots, P_n)$. Then through the use of De Morgan's laws

$$P \wedge Q \Leftrightarrow \neg(\neg P \vee \neg Q) \qquad P \vee Q \Leftrightarrow \neg(\neg P \wedge \neg Q)$$

we can show

$$\neg A(P_1, P_2, \ldots, P_n) \Leftrightarrow A^*(\neg P_1, \neg P_2, \ldots, \neg P_n) \tag{1}$$

Thus the negation of a formula is equivalent to its dual in which every variable is replaced by its negation. As a consequence of this fact, we also have

$$A(\neg P_1, \neg P_2, \ldots, \neg P_n) \Leftrightarrow \neg A^*(P_1, P_2, \ldots, P_n) \tag{2}$$

EXAMPLE 2 Verify equivalence (1) if $A(P, Q, R)$ is $\neg P \wedge \neg(Q \vee R)$.

SOLUTION Now $A^*(P, Q, R)$ is $\neg P \vee \neg(Q \wedge R)$, and $A^*(\neg P, \neg Q, \neg R)$ is $\neg\neg P \vee \neg(\neg Q \wedge \neg R) \Leftrightarrow P \vee (Q \vee R)$. On the other hand, $\neg A(P, Q, R)$ is $\neg(\neg P \wedge \neg(Q \vee R)) \Leftrightarrow P \vee (Q \vee R)$. ////

We shall now prove an interesting theorem which states that if any two formulas are equivalent, then their duals are also equivalent to each other. In other words, if $A \Leftrightarrow B$, then $A^* \Leftrightarrow B^*$.

Theorem 1-2.2 Let $P_1, P_2, \ldots, P_n$ be all the atomic variables appearing in the formulas A and B. Given that $A \Leftrightarrow B$ means "$A \rightleftarrows B$ is a tautology," then the following are also tautologies.

$$A(P_1, P_2, \ldots, P_n) \rightleftarrows B(P_1, P_2, \ldots, P_n)$$
$$A(\neg P_1, \neg P_2, \ldots, \neg P_n) \rightleftarrows B(\neg P_1, \neg P_2, \ldots, \neg P_n)$$

Using (2), we get

$$\neg A^*(P_1, P_2, \ldots, P_n) \rightleftarrows \neg B^*(P_1, P_2, \ldots, P_n)$$

Hence $A^* \Leftrightarrow B^*$.

This theorem explains why in Table 1-2.15 we have for every pair of equivalent formulas an equivalent pair of formulas consisting of duals of the first pair.

EXAMPLE 3 Show that

(a) $\neg(P \wedge Q) \rightarrow (\neg P \vee (\neg P \vee Q)) \Leftrightarrow (\neg P \vee Q)$
(b) $(P \vee Q) \wedge (\neg P \wedge (\neg P \wedge Q)) \Leftrightarrow (\neg P \wedge Q)$

SOLUTION

(a)
$$\begin{aligned}&\neg(P \wedge Q) \rightarrow (\neg P \vee (\neg P \vee Q))\\ &\Leftrightarrow (P \wedge Q) \vee (\neg P \vee (\neg P \vee Q)) \qquad (3)\\ &\Leftrightarrow (P \wedge Q) \vee (\neg P \vee Q)\\ &\Leftrightarrow (P \wedge Q) \vee \neg P \vee Q\\ &\Leftrightarrow ((P \vee \neg P) \wedge (Q \vee \neg P)) \vee Q\\ &\Leftrightarrow (Q \vee \neg P) \vee Q \Leftrightarrow Q \vee \neg P \Leftrightarrow \neg P \vee Q\end{aligned}$$

From (3) it follows that

$$(P \wedge Q) \vee (\neg P \vee (\neg P \vee Q)) \Leftrightarrow \neg P \vee Q$$

Writing the duals, we obtain by Theorem 1-2.2 that

$$(P \vee Q) \wedge (\neg P \wedge (\neg P \wedge Q)) \Leftrightarrow \neg P \wedge Q \qquad ////$$

1-2.11 Tautological Implications

Recall the definition of the conditional as given in Table 1-2.8. The connectives $\wedge$, $\vee$, and $\rightleftarrows$ are symmetric in the sense that $P \wedge Q \Leftrightarrow Q \wedge P$, $P \vee Q \Leftrightarrow Q \vee P$, and $P \rightleftarrows Q \Leftrightarrow Q \rightleftarrows P$. On the other hand, $P \rightarrow Q$ is not equivalent to $Q \rightarrow P$.

For any statement formula $P \rightarrow Q$, the statement formula $Q \rightarrow P$ is called its *converse*, $\neg P \rightarrow \neg Q$ is called its *inverse*, and $\neg Q \rightarrow \neg P$ is called its *contrapositive*.

From Table 1-2.16 it is clear that

$$P \rightarrow Q \Leftrightarrow \neg Q \rightarrow \neg P \qquad Q \rightarrow P \Leftrightarrow \neg P \rightarrow \neg Q$$

A statement A is said to *tautologically imply* a statement B if and only if $A \rightarrow B$ is a tautology. We shall denote this idea by $A \Rightarrow B$ which is read as "A implies B." Similar to the case with $\Leftrightarrow$, we note that $\Rightarrow$ is not a connective nor is $A \Rightarrow B$ a statement formula. Just as $A \Leftrightarrow B$ states that A and B are equiv-

Table 1-2.16

P	Q	$\neg P$	$\neg Q$	$P \rightarrow Q$	$\neg Q \rightarrow \neg P$
T	T	F	F	T	T
T	F	F	T	F	F
F	T	T	F	T	T
F	F	T	T	T	T

alent or that $A \rightleftarrows B$ is a tautology, in a similar manner $A \Rightarrow B$ states that $A \rightarrow B$ is a tautology or A tautologically implies B.

We have avoided using the expression "imply" to translate the conditional, so that we shall abbreviate "tautologically imply" simply as "imply." Obviously $A \Rightarrow B$ guarantees that B has the truth value T whenever A has the truth value T.

One can determine whether $A \Rightarrow B$ by constructing the truth tables of A and B in the same manner as was done in the determination of $A \Leftrightarrow B$.

The implications in Table 1-2.17 have important applications. All of them can be proved by truth table or by other methods.

In order to show any of the given implications, it is sufficient to show that an assignment of the truth value T to the antecedent of the corresponding conditional leads to the truth value T for the consequent. This procedure guarantees that the conditional becomes a tautology, thereby proving the implication. In (9), if we assume that $\neg Q \wedge (P \rightarrow Q)$ has the truth value T, then both $\neg Q$ and $P \rightarrow Q$ have the truth value T, which means that Q has the value F. $P \rightarrow Q$ has the truth value T, and hence P must have the value F. Therefore the consequent $\neg P$ must have the value T.

In (12), we assume that the antecedent is true. This assumption means that $P \vee Q$, $P \rightarrow R$, and $Q \rightarrow R$ are true. If P is true, then R must be true because $P \rightarrow R$ is true. If Q is true, then R must also be true. But at least one of P or Q is true by our assumption that $P \vee Q$ is true, and so R is true.

Another method to show $P \Rightarrow Q$ is to assume that the consequent Q has the value F and then show that this assumption leads to P's having the value F. Then $P \rightarrow Q$ must have the value T.

In (9) assume that $\neg P$ is false, so that P is true. Then $\neg Q \wedge (P \rightarrow Q)$ must be false. This statement holds because if Q is true, then $\neg Q$ is false, while if Q is false, then $P \rightarrow Q$ is false. Hence the implication in (9) is shown.

Example 4 in Sec. 1-2.6 shows the equivalence of the statements $P \rightleftarrows Q$ and $(P \rightarrow Q) \wedge (Q \rightarrow P)$; it is easy to verify that ($P \Rightarrow Q$ and $Q \Rightarrow P$) iff $P \Leftrightarrow Q$. This statement is an alternative definition of the equivalence of two formulas. If each of the two formulas A and B implies the other, then A and B are equivalent.

Table 1-2.17 IMPLICATIONS

$P \wedge Q \Rightarrow P$	(1)
$P \wedge Q \Rightarrow Q$	(2)
$P \Rightarrow P \vee Q$	(3)
$\neg P \Rightarrow P \rightarrow Q$	(4)
$Q \Rightarrow P \rightarrow Q$	(5)
$\neg(P \rightarrow Q) \Rightarrow P$	(6)
$\neg(P \rightarrow Q) \Rightarrow \neg Q$	(7)
$P \wedge (P \rightarrow Q) \Rightarrow Q$	(8)
$\neg Q \wedge (P \rightarrow Q) \Rightarrow \neg P$	(9)
$\neg P \wedge (P \vee Q) \Rightarrow Q$	(10)
$(P \rightarrow Q) \wedge (Q \rightarrow R) \Rightarrow P \rightarrow R$	(11)
$(P \vee Q) \wedge (P \rightarrow R) \wedge (Q \rightarrow R) \Rightarrow R$	(12)

There are several important facts about implication and equivalence that should be observed. If a formula is equivalent to a tautology, then it must be a tautology. Similarly, if a formula is implied by a tautology, then it is a tautology.

Both implication and equivalence are transitive. To say that equivalence is transitive means if $A \Leftrightarrow B$ and $B \Leftrightarrow C$, then $A \Leftrightarrow C$. This statement follows from the definition of equivalence. To show that the implication is also transitive, assume that $A \Rightarrow B$ and $B \Rightarrow C$. Then $A \rightarrow B$ and $B \rightarrow C$ are tautologies. Hence $(A \rightarrow B) \wedge (B \rightarrow C)$ is also a tautology. But from (11), $(A \rightarrow B) \wedge (B \rightarrow C) \Rightarrow (A \rightarrow C)$. Hence $A \rightarrow C$ is a tautology.

The transitivity of implications can also be applied in several stages. In order to show that $A \Rightarrow C$, it may be convenient to introduce a series of formulas $B_1, B_2, \ldots, B_m$ such that $A \Rightarrow B_1, B_1 \Rightarrow B_2, \ldots, B_{m-1} \Rightarrow B_m$, and $B_m \Rightarrow C$.

Another important property of implication is that if $A \Rightarrow B$ and $A \Rightarrow C$, then $A \Rightarrow (B \wedge C)$. By our assumption, if A is true, then B and C are both true. Thus $B \wedge C$ is true, and hence $A \rightarrow (B \wedge C)$ is true.

We can extend our notion of implication $P \Rightarrow Q$ to the case where several formulas, say $H_1, H_2, \ldots, H_m$, jointly imply a particular formula Q; that is, $H_1, H_2, \ldots, H_m \Rightarrow Q$ means $(H_1 \wedge H_2 \wedge \cdots \wedge H_m) \Rightarrow Q$.

An important theorem which is used in Sec. 1-4.1 follows.

Theorem 1-2.3 If $H_1, H_2, \ldots, H_m$ and P imply Q, then $H_1, H_2, \ldots, H_m$ imply $P \rightarrow Q$.

PROOF From our assumption we have

$$(H_1 \wedge H_2 \wedge \cdots \wedge H_m \wedge P) \Rightarrow Q$$

This assumption means $(H_1 \wedge H_2 \wedge \cdots \wedge H_m \wedge P) \rightarrow Q$ is a tautology. Using the equivalence (see Example 2, Sec. 1-2.9)

$$P_1 \rightarrow (P_2 \rightarrow P_3) \Leftrightarrow (P_1 \wedge P_2) \rightarrow P_3$$

we can say that

$$(H_1 \wedge H_2 \wedge \cdots \wedge H_m) \rightarrow (P \rightarrow Q)$$

is a tautology. Hence the theorem. ////

EXERCISES 1-2.11

1 Show the following implications.
(*a*) $(P \wedge Q) \Rightarrow (P \rightarrow Q)$
(*b*) $P \Rightarrow (Q \rightarrow P)$
(*c*) $(P \rightarrow (Q \rightarrow R)) \Rightarrow (P \rightarrow Q) \rightarrow (P \rightarrow R)$

2 Show the following equivalences.
(*a*) $P \rightarrow (Q \rightarrow P) \Leftrightarrow \neg P \rightarrow (P \rightarrow Q)$
(*b*) $P \rightarrow (Q \vee R) \Leftrightarrow (P \rightarrow Q) \vee (P \rightarrow R)$
(*c*) $(P \rightarrow Q) \wedge (R \rightarrow Q) \Leftrightarrow (P \vee R) \rightarrow Q$
(*d*) $\neg(P \rightleftarrows Q) \Leftrightarrow (P \vee Q) \wedge \neg(P \wedge Q)$

3 Show the following implications without constructing the truth tables.
 (*a*) $P \to Q \Rightarrow P \to (P \wedge Q)$
 (*b*) $(P \to Q) \to Q \Rightarrow P \vee Q$
 (*c*) $((P \vee \neg P) \to Q) \to ((P \vee \neg P) \to R) \Rightarrow (Q \to R)$ (see Sec. 1-6.3)
 (*d*) $(Q \to (P \wedge \neg P)) \to (R \to (P \wedge \neg P)) \Rightarrow (R \to Q)$ (see Sec. 1-6.3)

4 Show that P is equivalent to the following formulas.

$$\neg\neg P \qquad P \wedge P \qquad P \vee P \qquad P \vee (P \wedge Q) \qquad P \wedge (P \vee Q)$$
$$(P \wedge Q) \vee (P \wedge \neg Q) \qquad (P \vee Q) \wedge (P \vee \neg Q)$$

5 Show the following equivalences.
 (*a*) $\neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q$
 (*b*) $\neg(P \vee Q) \Leftrightarrow \neg P \wedge \neg Q$
 (*c*) $\neg(P \to Q) \Leftrightarrow P \wedge \neg Q$
 (*d*) $\neg(P \rightleftarrows Q) \Leftrightarrow (P \wedge \neg Q) \vee (\neg P \wedge Q)$

1-2.12 Formulas with Distinct Truth Tables

Using all the connectives defined so far and the rules for constructing well-formed formulas, it is possible to construct an unlimited number of statement formulas. We shall try to determine how many of these formulas have distinct truth tables.

Let us consider all possible truth tables that can be obtained when the formulas involve only one variable P. These possible truth tables are shown in Table 1-2.18.

Any formula involving only one variable will have one of these four truth tables. Obviously the simplest formulas corresponding to the entries under 1, 2, 3, and 4 are P, $\neg P$, $P \vee \neg P$, and $P \wedge \neg P$ respectively. Every other formula depending upon P alone would then be equivalent to one of these four formulas.

If we consider formulas obtained by using two variables P and Q and any of the connectives defined, we also obtain an unlimited number of formulas. The number of distinct truth tables for formulas involving two variables is given by $2^{2^2} = 2^4 = 16$. Since there are 2^2 rows in the truth table and since each row could have any of the two entries T or F, we have 2^{2^2} possible tables, as shown in Table 1-2.19.

Any formula involving only two variables will have one of these 16 truth tables. All those formulas which have one of these truth tables are equivalent to each other.

A statement formula containing n variables must have as its truth table one of the 2^{2^n} possible truth tables, each of them having 2^n rows. This fact suggests that there are many formulas which may look very different from one another but are equivalent.

Table 1-2.18

P	1	2	3	4
T	T	F	T	F
F	F	T	T	F

Table 1-2.19

P	Q	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
T	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
F	T	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
F	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F

One method to determine whether two statement formulas A and B are equivalent is to construct their truth tables and compare them. This method is very tedious and difficult to implement even on a computer because the number of entries increases very rapidly as n increases (note that $2^{10} \simeq 1{,}000$). A better method would be to transform A and B to some standard forms A' and B' such that a simple comparison of A' and B' should establish whether $A \Leftrightarrow B$. This method is feasible; the standard forms are called canonical forms or normal forms and are discussed in Sec. 1-3.

1-2.13 Functionally Complete Sets of Connectives

So far we have defined the connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$, and $\rightleftarrows$. We introduce some other connectives in Sec. 1-2.14 because of their usefulness in certain applications. On the other hand, we show in this section that not all the connectives defined thus far are necessary. In fact, we can find certain proper subsets of these connectives which are sufficient to express any formula in an equivalent form. Any set of connectives in which every formula can be expressed in terms of an equivalent formula containing the connectives from this set is called a *functionally complete* set of connectives. It is assumed that such a functionally complete set does not contain any redundant connectives, i.e., a connective which can be expressed in terms of the other connectives.

In order to arrive at a functionally complete set, we first examine the following equivalence:

$$P \rightleftarrows Q \Leftrightarrow (P \rightarrow Q) \wedge (Q \rightarrow P)$$

This equivalence suggests that in any formula we can replace the part (here "part" means any part which is itself a formula) containing the biconditional by an equivalent formula not containing the biconditional. Thus all the biconditionals can be replaced in a formula.

EXAMPLE 1 Write an equivalent formula for $P \wedge (Q \rightleftarrows R) \vee (R \rightleftarrows P)$ which does not contain the biconditional.

SOLUTION

$$P \wedge (Q \rightleftarrows R) \vee (R \rightleftarrows P)$$
$$\Leftrightarrow P \wedge ((Q \rightarrow R) \wedge (R \rightarrow Q)) \vee ((R \rightarrow P) \wedge (P \rightarrow R))$$

Thus the equivalent formula is $P \wedge ((Q \rightarrow R) \wedge (R \rightarrow Q)) \vee ((R \rightarrow P) \wedge (P \rightarrow R))$. ////

Next we now consider the equivalence $P \rightarrow Q \Leftrightarrow \neg P \vee Q$. This equivalence suggests that the conditionals can also be eliminated by replacing those parts which contain conditionals by their equivalents.

EXAMPLE 2 Write an equivalent formula for $P \wedge (Q \rightleftarrows R)$ which contains neither the biconditional nor the conditional.

SOLUTION

$$\begin{aligned} P \wedge (Q \rightleftarrows R) &\Leftrightarrow P \wedge ((Q \to R) \wedge (R \to Q)) \\ &\Leftrightarrow P \wedge ((\neg Q \vee R) \wedge (\neg R \vee Q)) \end{aligned}$$

Thus the required formula is $P \wedge (\neg Q \vee R) \wedge (\neg R \vee Q)$. ////

Notice that from De Morgan's laws we have

$$P \wedge Q \Leftrightarrow \neg(\neg P \vee \neg Q) \qquad P \vee Q \Leftrightarrow \neg(\neg P \wedge \neg Q)$$

This first equivalence means that it is also possible to obtain a formula which is equivalent to a given formula in which conjunctions are eliminated. A similar procedure is possible for the elimination of disjunctions.

If we implement all the steps suggested above, we can first replace all biconditionals, then the conditionals, and finally all the conjunctions or all the disjunctions in any formula to obtain an equivalent formula which contains either the negation and disjunction only or the negation and conjunction only. This fact means that the sets of connectives $\{\wedge, \neg\}$ and $\{\vee, \neg\}$ are functionally complete.

One can show that $\{\neg\}$, $\{\wedge\}$, or $\{\vee\}$ are not functionally complete and neither is $\{\wedge, \vee\}$.

From the five connectives $\wedge$, $\vee$, $\neg$, $\to$, $\rightleftarrows$ we have obtained at least two sets of functionally complete connectives. A question arises whether there is any one connective which is functionally complete. The answer to such a question is no if only the above five connectives are considered. There are some connectives, which are defined in Sec. 1-2.14, that are functionally complete. The question of finding a functionally complete set with fewer connectives is not as theoretical as it may appear, because in physical two-state devices, which are described in Sec. 1-2.15, the connectives correspond to certain physical elements of the device. From the point of view of maintenance and economical production, it is sometimes necessary not to use a variety of different elements.

Note that if a given formula is replaced by an equivalent formula in which the number of different connectives is reduced, the resulting formula may become more complex. This is why we use a larger number of connectives than are needed. In Sec. 1-2.14 we define some more connectives which will be found useful.

EXERCISES 1-2.13

1 Write formulas which are equivalent to the formulas given below and which contain the connectives $\wedge$ and $\neg$ only.
(*a*) $\neg(P \rightleftarrows (Q \to (R \vee P)))$
(*b*) $((P \vee Q) \wedge R) \to (P \vee R)$

2 For each column in Table 1-2.19 write a formula, involving two variables P and Q, whose truth table corresponds to the truth values in the column chosen.

3 Show that $\{\wedge, \vee\}$, $\{\vee\}$, and $\{\neg\}$ are not functionally complete. (*Hint:* Write a formula which is a tautology.)

1-2.14 Other Connectives

It was shown earlier that not all connectives defined thus far are necessary for the description of the statement calculus. For any formula of the statement calculus, there exists an equivalent formula in which appear only those connectives belonging to one of the functionally complete sets. In spite of this fact, we did define other connectives because, by using them, some of the formulas become simpler. There are other connectives which serve similar purposes, and these will be defined in this section.

Let P and Q be any two formulas. Then the formula $P \,\bar{\vee}\, Q$, in which the connective $\bar{\vee}$ is called an *exclusive OR*, is true whenever either P or Q, but not both, is true. The exclusive OR is defined by Table 1-2.20. The exclusive OR is also called the *exclusive disjunction*. The following equivalences follow from its definition.

1 $P \,\bar{\vee}\, Q \Leftrightarrow Q \,\bar{\vee}\, P$ (symmetric)
2 $(P \,\bar{\vee}\, Q) \,\bar{\vee}\, R \Leftrightarrow P \vee (Q \,\bar{\vee}\, R)$ (associative)
3 $P \wedge (Q \,\bar{\vee}\, R) \Leftrightarrow (P \wedge Q) \,\bar{\vee}\, (P \wedge R)$ (distributive)
4 $(P \,\bar{\vee}\, Q) \Leftrightarrow (P \wedge \neg Q) \vee (\neg P \wedge Q)$
5 $(P \,\bar{\vee}\, Q) \Leftrightarrow \neg(P \rightleftarrows Q)$

One can also prove that if $P \,\bar{\vee}\, Q \Leftrightarrow R$, then $P \,\bar{\vee}\, R \Leftrightarrow Q$ and $Q \,\bar{\vee}\, R \Leftrightarrow P$, and $P \,\bar{\vee}\, Q \,\bar{\vee}\, R$ is a contradiction.

Given a formula in which $\bar{\vee}$ appears, it is possible to obtain an equivalent formula in which only the connectives $\wedge$, $\vee$, and $\neg$ appear by using the equivalence of the formulas in (4).

Other connectives which have useful applications in the design of computers are called *NAND* and *NOR*. The word "*NAND*" is a combination of "*NOT*" and "*AND*," where "*NOT*" stands for negation and "*AND*" for the conjunction. The connective *NAND* is denoted by the symbol $\uparrow$. For any two formulas P and Q

$$P \uparrow Q \Leftrightarrow \neg(P \wedge Q)$$

Another connective, useful in a similar context, is known as *NOR*, a combination of *NOT* and *OR*, where "*OR*" stands for the disjunction. The connective *NOR* is denoted by the symbol $\downarrow$. For any two formulas P and Q

$$P \downarrow Q \Leftrightarrow \neg(P \vee Q)$$

The connectives $\uparrow$ and $\downarrow$ have been defined in terms of the connectives $\wedge$, $\vee$, and $\neg$. Therefore, for any formula containing the connectives $\uparrow$ or $\downarrow$,

Table 1-2.20

P	Q	$P \,\bar{\vee}\, Q$
T	T	F
T	F	T
F	T	T
F	F	F

one can obtain an equivalent formula containing the connectives $\wedge$, $\vee$, and $\neg$ only. Note that $\uparrow$ and $\downarrow$ are duals of each other. Therefore, in order to obtain the dual of a formula which includes $\uparrow$ or $\downarrow$, we should interchange $\uparrow$ and $\downarrow$ in addition to making the other interchanges mentioned earlier.

We now show that each of the connectives $\uparrow$ and $\downarrow$ is functionally complete. In order to do this, it is sufficient to show that the sets of connectives $\{\wedge, \neg\}$ and $\{\vee, \neg\}$ can be expressed either in terms of $\uparrow$ alone or in terms of $\downarrow$ alone. The following equivalences express $\neg$, $\wedge$, and $\vee$ in terms of $\uparrow$ alone.

$$P \uparrow P \Leftrightarrow \neg(P \wedge P) \Leftrightarrow \neg P \vee \neg P \Leftrightarrow \neg P$$
$$(P \uparrow Q) \uparrow (P \uparrow Q) \Leftrightarrow \neg(P \uparrow Q) \Leftrightarrow P \wedge Q$$
$$(P \uparrow P) \uparrow (Q \uparrow Q) \Leftrightarrow \neg P \uparrow \neg Q \Leftrightarrow \neg(\neg P \wedge \neg Q) \Leftrightarrow P \vee Q$$

In a similar manner, the following equivalences express $\neg$, $\vee$, and $\wedge$ in terms of $\downarrow$ alone

$$P \downarrow P \Leftrightarrow \neg(P \vee P) \Leftrightarrow \neg P \wedge \neg P \Leftrightarrow \neg P$$
$$(P \downarrow Q) \downarrow (P \downarrow Q) \Leftrightarrow \neg(P \downarrow Q) \Leftrightarrow P \vee Q$$
$$(P \downarrow P) \downarrow (Q \downarrow Q) \Leftrightarrow \neg P \downarrow \neg Q \Leftrightarrow P \wedge Q$$

Because a single operator *NAND* or *NOR* is functionally complete, we call each of the sets $\{\uparrow\}$ and $\{\downarrow\}$ a *minimal functionally complete set*, or, in short, a *minimal set*.

The idea of a minimal set of connectives is useful in the economics of the design of electronic circuits. It may be noted that although we can express any formula by an equivalent formula containing a single connective $\uparrow$ or $\downarrow$, we seldom do so because such formulas are often complicated. This fact explains why programming languages as well as our symbolic language have a number of connectives available.

We now list some of the basic properties of the connectives *NAND* and *NOR*.

$$P \uparrow Q \Leftrightarrow Q \uparrow P \qquad P \downarrow Q \Leftrightarrow Q \downarrow P \qquad \text{(commutative)} \tag{1}$$
$$P \uparrow (Q \uparrow R) \Leftrightarrow P \uparrow \neg(Q \wedge R) \Leftrightarrow \neg(P \wedge \neg(Q \wedge R)) \Leftrightarrow \neg P \vee (Q \wedge R) \tag{2}$$

while

$$(P \uparrow Q) \uparrow R \Leftrightarrow (P \wedge Q) \vee \neg R$$

Thus the connective $\uparrow$ is not associative. Similarly

$$P \downarrow (Q \downarrow R) \Leftrightarrow \neg P \wedge (Q \vee R) \qquad (P \downarrow Q) \downarrow R \Leftrightarrow (P \vee Q) \wedge \neg R$$

It is possible to define $P \uparrow Q \uparrow R \Leftrightarrow \neg(P \wedge Q \wedge R)$. However, $P \uparrow Q \uparrow R$ is not equivalent to $P \uparrow (Q \uparrow R)$ or to $(P \uparrow Q) \uparrow R$ or to $Q \uparrow (P \uparrow R)$. This nonassociativity of the connectives *NAND* and *NOR* creates some difficulty in using them.

$$P \uparrow Q \Leftrightarrow \neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q \Leftrightarrow (\neg P \wedge Q) \vee (P \wedge \neg Q) \vee (\neg P \wedge \neg Q) \tag{3}$$

Similarly

$$P \downarrow Q \Leftrightarrow \neg(P \vee Q) \Leftrightarrow \neg P \wedge \neg Q \Leftrightarrow (\neg P \vee Q) \\ \wedge (P \vee \neg Q) \wedge (\neg P \vee \neg Q) \qquad (4)$$

EXERCISES 1-2.14

1 If $A(P, Q, R)$ is given by $P \uparrow (Q \wedge \neg(R \downarrow P))$, find its dual $A^*(P, Q, R)$. Also find formulas which are equivalent to A and A^*, but which contain the connectives $\wedge, \vee$, and $\neg$ only.

2 Express $P \to (\neg P \to Q)$ in terms of $\uparrow$ only. Express the same formula in terms of $\downarrow$ only.

3 Express $P \uparrow Q$ in terms of $\downarrow$ only.

1-2.15 Two-state Devices and Statement Logic

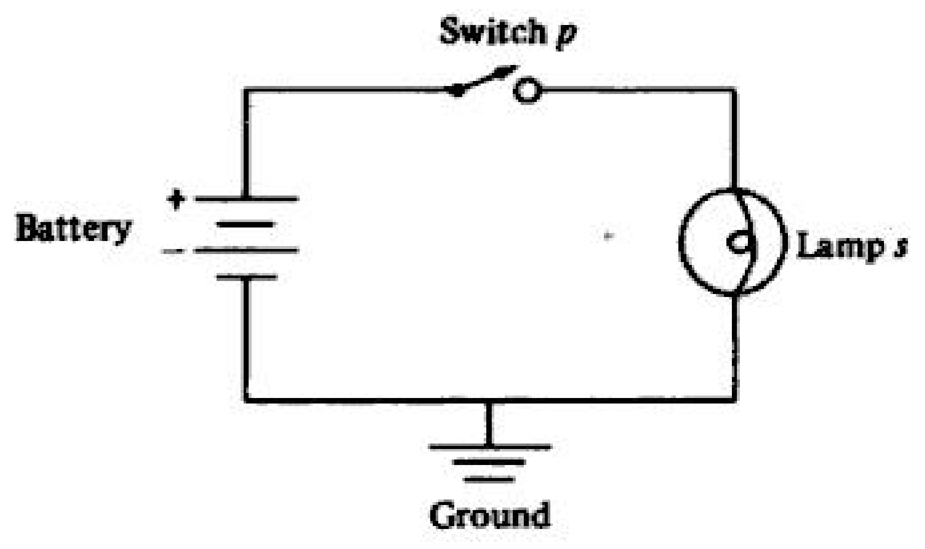
The statement logic that we have discussed so far is called *two-valued logic,* because we admit only those statements having a truth value of true or false. A similar situation exists in various electrical and mechanical devices which are assumed to be in one of two possible configurations; for this reason, they are called two-state devices. We first give several examples of such commonly known devices and then show their connection to two-valued logic.

An electric switch which is used for turning "on" and "off" an electric light is a two-state device. Normally, such a switch is operated manually; however, if it is operated automatically by electric power, we say the switch is relay-operated. A vacuum tube or a transistor is another two-state device in which the current is either passing (conducting) or not passing (nonconducting). A mechanical clutch can be engaged or disengaged. A small doughnut-shaped metal disc with a wire coil wrapped around it (called a magnetic core in computers) may be magnetized in one direction if the current is passed through the coil in one way and may be magnetized in the opposite direction if the current is reversed. Many other examples of two-state devices can be cited. A general discussion of such devices can be given by replacing the word "switch" by the word "gate" to mean a device which permits or stops the flow of not only electric current but any quantity that can go through the device, such as water, information, persons, etc.

Let us first consider the example of an electric lamp controlled by a mechanical switch. Such a circuit is displayed in Fig. 1-2.3. When the switch p is open, there is no current flowing in the circuit and the lamp s is "off." When p is closed, the lamp s is "on." The state of the switch and the lamp can be represented by the table of combinations in Fig. 1-2.3. Let us denote the statements as

P: The switch p is closed.

S: The lamp s is on.



State of switch p	State of light s
closed	on
open	off

FIGURE 1-2.3 A switch as a two-state device.

Then we can rewrite the table of Fig. 1-2.3 as

$p(P)$	$s(S)$
1	1
0	0

Throughout this section we shall denote the truth values of statements P, Q, ... by 1 and 0 in place of T and F respectively. At the same time, the input switches such as p or the output indicator such as the lamp s will be assigned the values 1 and 0 to correspond to the states when the current is flowing or not flowing. In such cases the table shown here can be understood to be either a truth table or a table that relates the input and the output values.

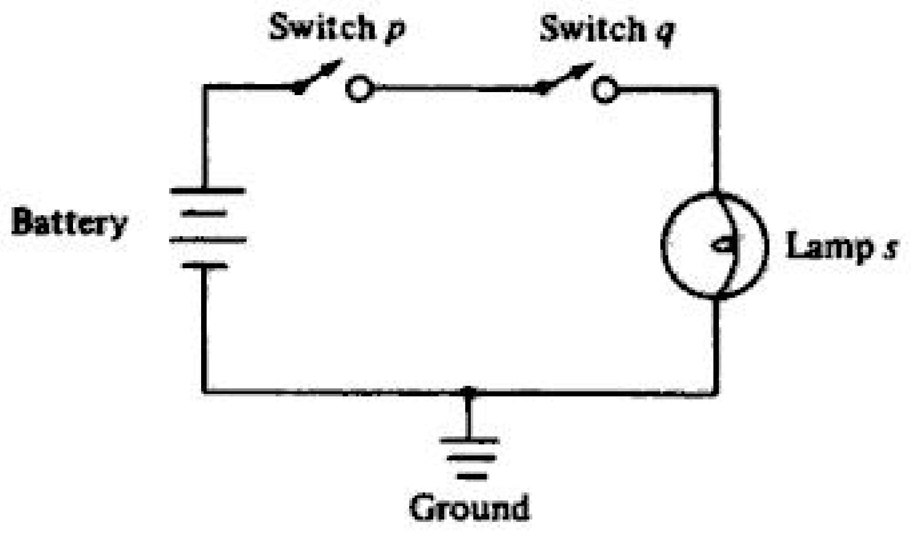
Next, consider an extension of the preceding circuit in which we have two switches p and q in series. The lamp s is turned on whenever both the switches p and q are closed. Such a circuit with its table of combinations is shown in Fig. 1-2.4. In the table, we have used the statements

P: The switch p is closed.

Q: The switch q is closed.

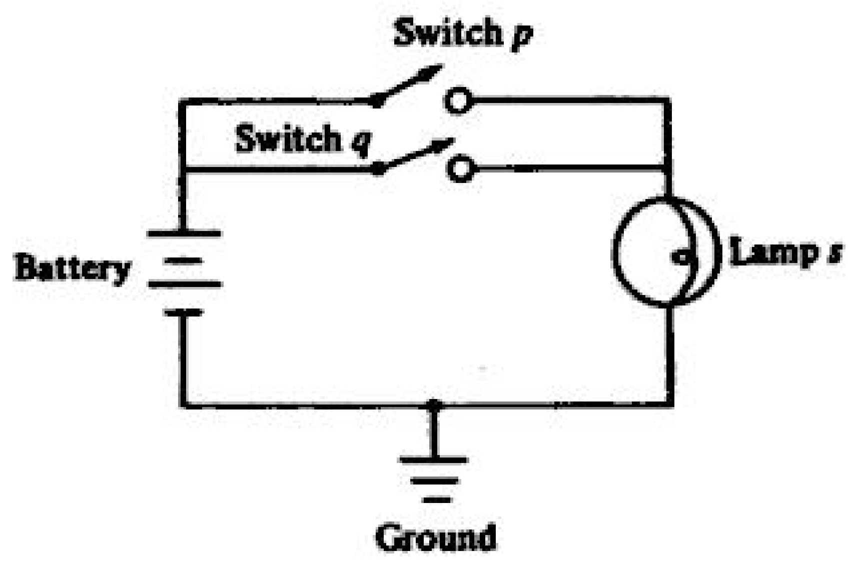
S: The light s is on.

From the table it is clear that $P \wedge Q \Leftrightarrow S$.



P	Q	S
1	1	1
1	0	0
0	1	0
0	0	0

FIGURE 1-2.4 A two-state device for AND logic.



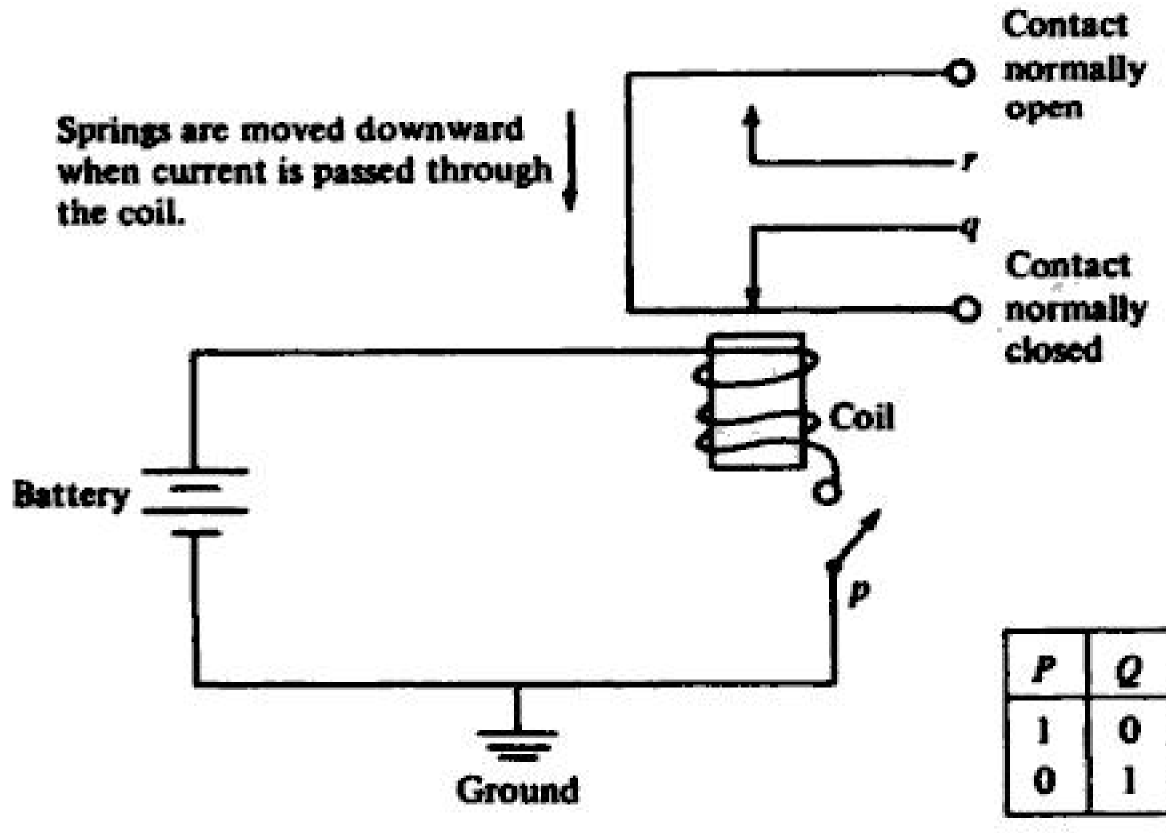
P	Q	S
1	1	1
1	0	1
0	1	1
0	0	0

FIGURE 1-2.5 A two-state device for OR logic.

Figure 1-2.5 contains a circuit and its associated table of combinations in which two switches are connected in parallel. From the table it is clear that $P \vee Q \Leftrightarrow S$.

We have just shown how the connectives $\wedge$ and $\vee$ correspond to switches connected in series and in parallel, respectively.

We now consider an example of a switch controlled by a relay. A simplified configuration and its associated table of combinations are given in Fig. 1-2.6. When the switch p is open (P is false, because we shall use P: The switch p is closed.), no current flows and the contact q which is normally closed remains closed and the contact r remains open. When p is closed, the current will flow from the battery through the coil which will cause the movement of a relay armature, which in turn causes the springs to move downward and the normally closed contact q to open while the normally open contact r closes. If p is opened, then the contact q closes and r opens because the spring moves upward to its original position. Thus with the statements P, Q, and R denoting the switches p, q, and r to be closed respectively, we can represent the operation of the device by the table of combinations in Fig. 1-2.6. In fact, the switches q and r are always in the opposite states, that is, $Q \Leftrightarrow \neg R$, $Q \Leftrightarrow \neg P$, and $R \Leftrightarrow P$. Note that the output Q is the negation of the input P.



P	Q	R
1	0	1
0	1	0

FIGURE 1-2.6 A relay-switching device.

Instead of representing the logical connectives $\neg$, $\vee$, and $\wedge$ by the circuits just given or by some other equivalent circuits (consisting of semiconductor devices, for example), they are generally represented by block diagrams or *gates*. Each gate has one or more input wires and one output wire.

The logical connectives $\neg$, $\vee$, and $\wedge$ will be denoted by the symbols $^-$, $+$, and $\cdot$ respectively in the remainder of this section. This denotation is in keeping with the terminology used in switching theory.

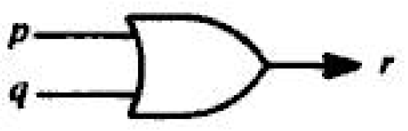
The block-diagram symbol for an *OR* gate is shown in Fig. 1-2.7 along with the table of combinations relating the inputs and output of the gate. Any such gate is also called a *module*.

Figure 1-2.8 shows the block-diagram symbol for an *AND* gate as well as its associated table of combinations.

The negation operator can be represented by the block diagram in Fig. 1-2.9.

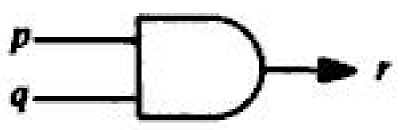
The block diagrams not only replace switches and relays but can also be used to represent "gates" in a more general sense. We may use p to denote voltage potential of an input which is "high" or "low" to allow a transistor to be in a conducting or nonconducting state. It is therefore convenient to use these modules and interpret the symbols according to the context. The number of inputs to *OR* gates and *AND* gates can be extended to more than 2.

The above modules, or gates, can be interconnected to realize various logical expressions. These systems of modules are known as logic or combinational networks. Figure 1-2.10*a* shows a logic network with three inputs a, b, and c and an output expression $(a + b)\cdot c$. Networks which form expressions $(a\cdot\bar{b}) + (\bar{a}\cdot b)$ and $(a + b)\cdot(\bar{a} + \bar{b})$ are given in Fig. 1-2.10*b* and *c* respectively.



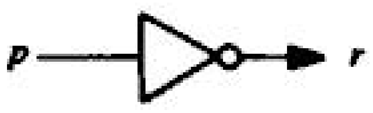
Input		Output
p	q	$r(p + q)$
1	1	1
1	0	1
0	1	1
0	0	0

FIGURE 1-2.7 An *OR* gate.



Input		Output
p	q	$r(p \cdot q)$
1	1	1
1	0	0
0	1	0
0	0	0

FIGURE 1-2.8 An *AND* gate.



Input	Output
p	$r(\bar{p})$
1	0
0	1

FIGURE 1-2.9 A *NOT* gate.

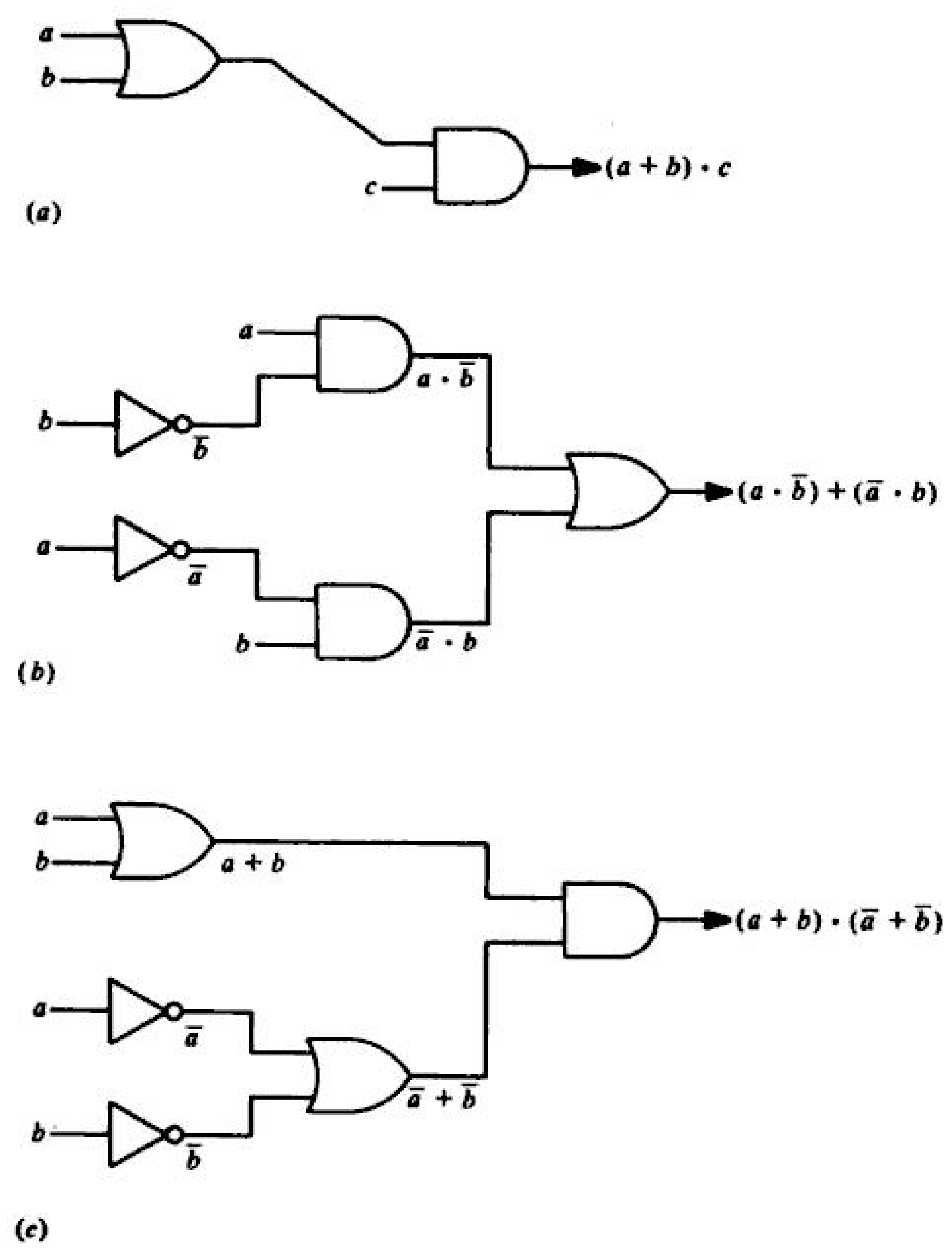


FIGURE 1-2.10 Logic networks. (*a*) Logic network for $(a + b) \cdot c$. (*b*) Logic network for $a \cdot \bar{b} + \bar{a} \cdot b$. (*c*) Logic network for $(a + b) \cdot (\bar{a} + \bar{b})$.

In the remainder of this section it is assumed that a variable and its negation are available without having to use an inverter. The mechanical relay was an example of a device which supplied both. There are other basic transistor devices which also supply both.

In order to simplify expressions by reducing the number of parentheses used, we specify that $\cdot$ has precedence over $+$. For example, $(a \cdot b) + (c \cdot d)$ will be written as $a \cdot b + c \cdot d$.

Consider the logical expression $a \cdot b + c \cdot d$, which consists of a disjunction of two conjunctions. A logic network to realize this expression can be a two-level network as shown in Fig. 1-2.11*a*. A *two-level network* is one in which the longest path through which information must pass from input to output is two gates. A two-level network consisting of *AND* gates at the input stage followed by *OR* gates at the output stage is sometimes called a *sum-of-products* (*AND*-to-OR) *network*.

Another possibility in a two-level network is to have *OR* gates at the input stage followed by *AND* gates at the output of the network. Such a configuration

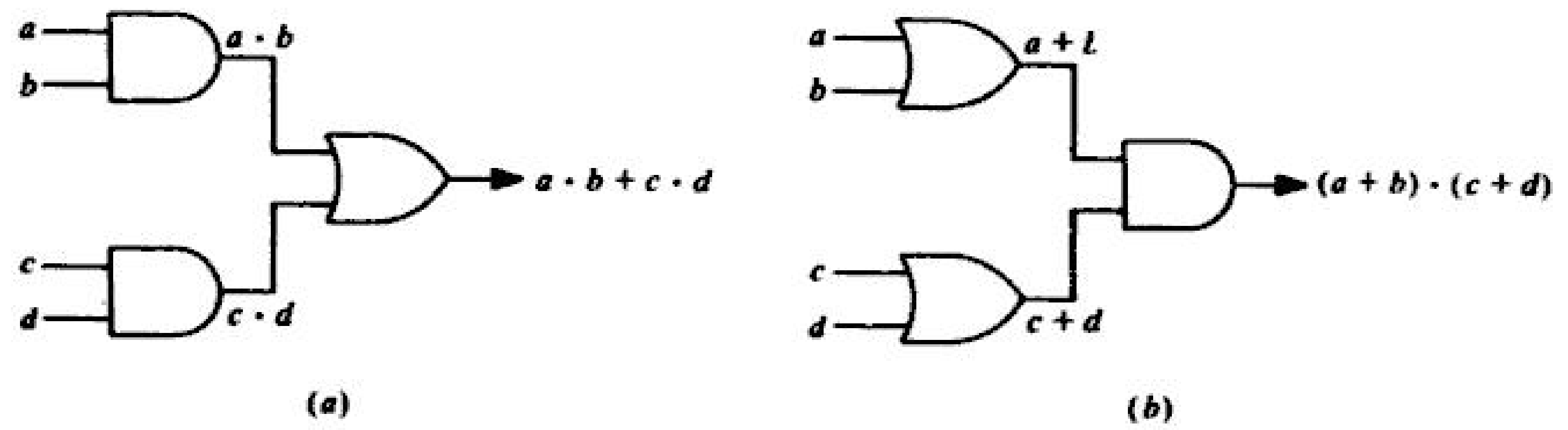


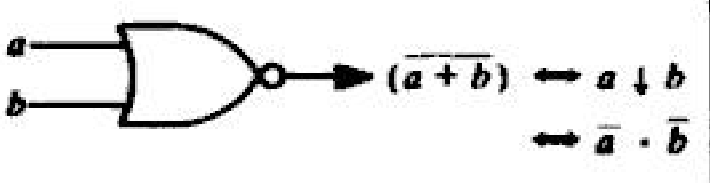
FIGURE 1-2.11 Two-level networks. (*a*) *AND*-to-*OR* logic network. (*b*) *OR*-to-*AND* logic network.

is shown for the expression $(a + b) \cdot (c + d)$ in Fig. 1-2.11*b* and is called a *product-of-sums* (*OR*-to-*AND*) *network*.

Other types of gates frequently used in computers are *NOR* gates and *NAND* gates. Figure 1-2.12*a* represents a *NOR* gate and its associated table. Figure 1-2.12*b* is an equivalent representation of a *NOR* gate consisting of an *OR* gate followed by an inverter.

Figure 1-2.13*a* shows a *NAND* gate with its table. The other part of the diagram is an equivalent *NAND* gate representation consisting of an *AND* gate followed by an inverter.

In Sec. 1-2.14 it was shown that each of the connectives *NAND* (↑) and *NOR* (↓) is functionally complete; either can be used to obtain the *AND*, *OR*, and *NOT* operations. The realizations of these operations in terms of *NAND* gates and *NOR* gates only are given in Figs. 1-2.14 and 1-2.15 respectively. The



a
b
$\overline{(a+b)} \to a \downarrow b$
$\to \bar{a} \cdot \bar{b}$
(a)

Input		Output
a	b	
1	1	0
1	0	0
0	1	0
0	0	1

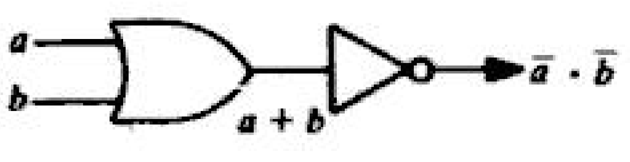
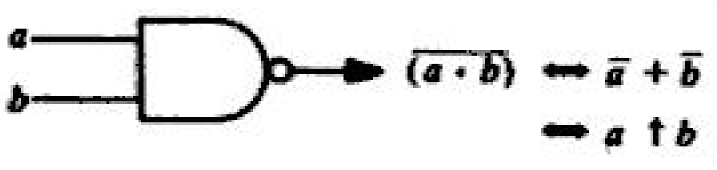


FIGURE 1-2.12 A *NOR* gate. (*a*) Block-diagram symbol and truth table for a *NOR* gate. (*b*) Equivalent *NOR* gate network.



Input		Output
a	b	
1	1	0
1	0	1
0	1	1
0	0	1

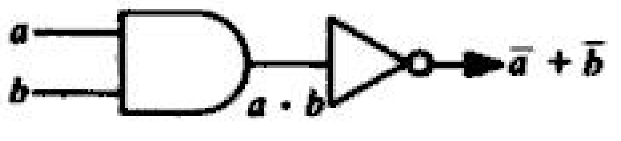


FIGURE 1-2.13 A *NAND* gate. (*a*) Block-diagram symbol and truth table for a *NAND* gate. (*b*) Equivalent *NAND* gate network.

use of a single type of gate such as *NAND* or *NOR* is often preferred to a variety of gate types because it is cheaper to produce and maintain just one type of gate.

One method used to produce a circuit containing only *NAND* or only *NOR* gates is to replace the *AND*, *OR*, and *NOT* gates by NAND or NOR gates, according to Figs. 1-2.14 and 1-2.15. This replacement often results in a circuit which contains more *NAND* or *NOR* gates than necessary. There are other techniques to generate equivalent circuits containing fewer *NAND* or *NOR* gates. As an example, each of the circuits in Fig. 1-2.11, if substituted according to Figs. 1-2.14 and 1-2.15, would require nine gates. But Fig. 1-2.16 contains different realizations of the same expressions, using only three gates.

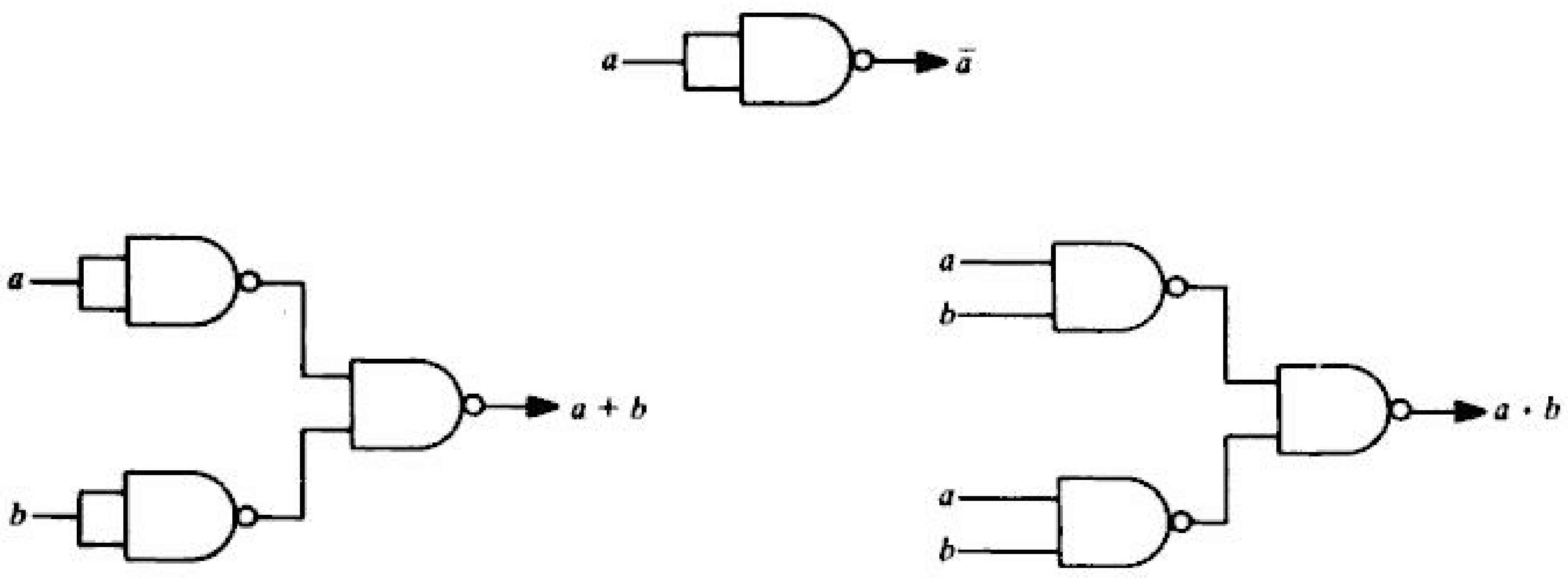


FIGURE 1-2.14. *NAND* gate generation of $+$, $\cdot$, and $^-$ operations.

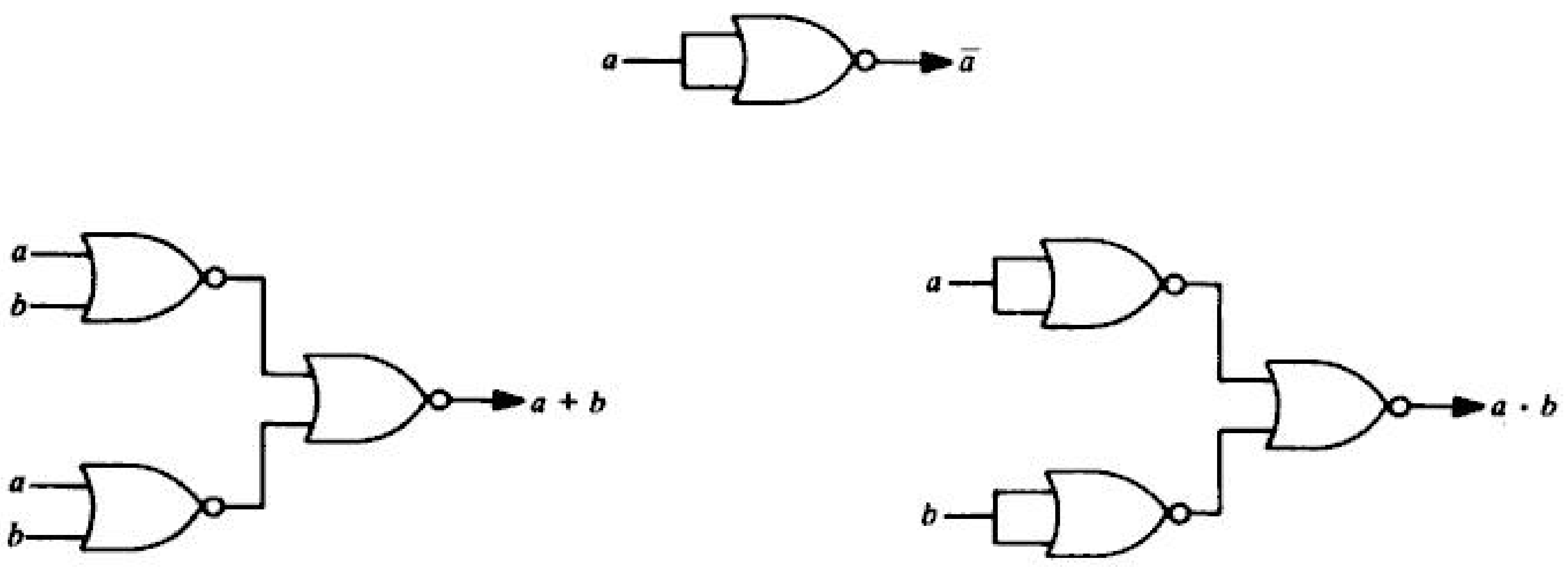


FIGURE 1-2.15. *NOR* gate generation of $+$, $\cdot$, and $^-$ operations.

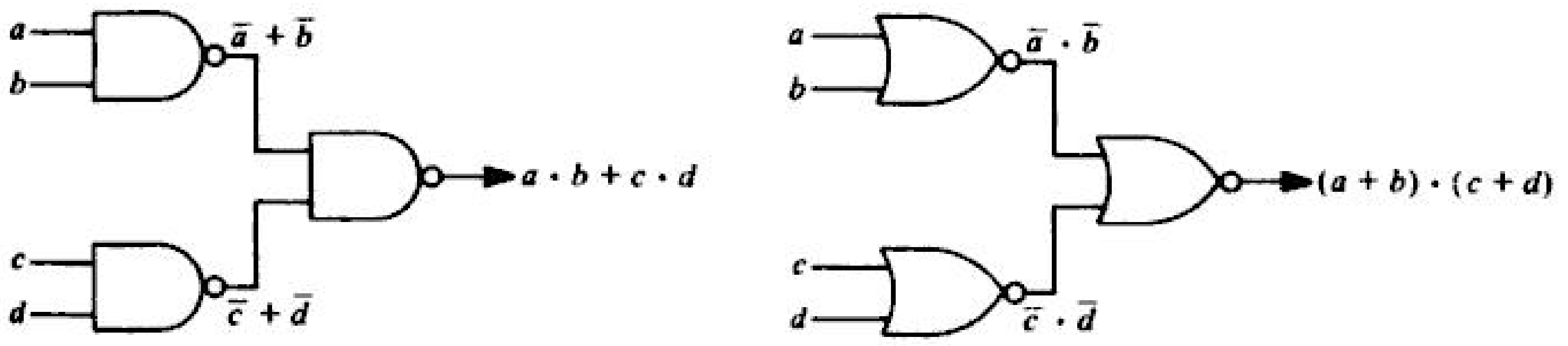


FIGURE 1-2.16. Two-level *NAND* and *NOR* gate networks.

So far we have discussed certain elements which correspond to some of the connectives of statement logic. Normally, if a formula containing these connectives is given, we can physically realize a circuit that corresponds to the formula by replacing the connectives by the appropriate gates and the variables by certain physical quantities such as voltage, current, etc. This method may, however, not yield the best design from the point of view of using a minimum number of gates, or the design it yields may not be a minimal design in some other sense. Sometimes even the formula may not be available and all we may have is its truth table. Even in this case, we may be required to physically realize the formula, not in any manner, but in some minimal way. The design procedure in both cases consists of the following steps:

1 Express the given information in terms of logical variables.
2 Get a formula for the required output in terms of the variables defined in step 1, using the logical connectives.
3 Obtain a formula which is logically equivalent to the one formed in step 2 and which will result in a least expensive (minimal) physical realization.
4 Replace the logical connectives in the formula in step 3 by proper logic blocks.

A discussion of step 3 is considered in some detail in Chap. 4. Steps 1 and 2 are explained by means of an example.

EXAMPLE 1 A certain government installation has an intruder alarm system which is to be operative only if a manual master switch situated at a security office is in the closed position. If this master switch is on (i.e., closed), an alarm will be sounded if a door to a restricted-access area within the installation is disturbed, or if the main gates to the installation are opened without the security officer first turning a special switch to the closed position. The restricted-access area door is equipped with a sensing device that causes a switch to set off the alarm if this door is disturbed in any way whenever the master switch is closed. However, the main gates are opened during daytime hours to allow the public to enter the installation grounds. Furthermore, at certain specified time intervals during a 24-hour period, the master switch is turned off to allow the authorized personnel to enter or leave the restricted-access area. During the period when the main gates are open and the master switch is turned off, it is required that some automatic recording instrument make an entry every time the door to the restricted-access area is opened.

SOLUTION *Step 1* Let us first designate each primary statement by a variable. This assignment will permit us to consider the assignment of all possible truth values to the variables. Thus let

A: The alarm will be given.
M: The manual master switch is closed.
G: The main gates to the installation are open.
R: The restricted-area door has been disturbed.
S: The special switch is closed.
E: The recording equipment is activated.

Step 2 The output variables are A and E. The conditions given in the problem require that

$$A \Leftrightarrow M \cdot (\bar{R} + (G \cdot \bar{S}))$$
$$E \Leftrightarrow \bar{M} \cdot G \cdot R \qquad ////$$

EXERCISES 1-2.15

1 Construct a circuit diagram for a simple elevator control circuit which operates as follows. When a person pushes the button to summon the elevator to a floor, the elevator responds to this request unless it is already answering a previous call to another floor. In this latter case, the request is ignored. Assume that there are only three floors in the building.
2 For the formula $(P \wedge Q) \vee (\neg R \wedge \neg P)$, draw a corresponding circuit diagram using (*a*) *NOT*, *AND*, and *OR* gates; (*b*) *NAND* gates only.

EXERCISES 1-2

1 Show the equivalences (1) to (9) listed in Table 1-2.15 of Sec. 1-2.9.
2 Show the implications (1) to (12) listed in Table 1-2.17 of Sec. 1-2.11.
3 Show the following

$$\neg(P \uparrow Q) \Leftrightarrow \neg P \downarrow \neg Q \qquad \neg(P \downarrow Q) \Leftrightarrow \neg P \uparrow \neg Q$$

4 Write a formula which is equivalent to the formula

$$P \wedge (Q \rightleftarrows R)$$

and contains the connective *NAND* ($\uparrow$) only. Obtain an equivalent formula whicl contains the connective *NOR* ($\downarrow$) only.
5 Show that $P \wedge \neg P \wedge Q \Rightarrow R$ and $R \Rightarrow P \vee \neg P \vee Q$.
6 Show the equivalences (1) to (3) given in Sec. 1-2.14.
7 Show that the set $\{\neg, \rightarrow\}$ is functionally complete.
8 A connective denoted by $\overset{\cdot}{\rightarrow}$ is defined by Table 1-2.21. Show that $\{\neg, \overset{\cdot}{\rightarrow}\}$ is functionally complete.
9 Show the following equivalences.
(*a*) $A \rightarrow (P \vee C) \Leftrightarrow (A \wedge \neg P) \rightarrow C$
(*b*) $(P \rightarrow C) \wedge (Q \rightarrow C) \Leftrightarrow (P \vee Q) \rightarrow C$
(*c*) $((Q \wedge A) \rightarrow C) \wedge (A \rightarrow (P \vee C)) \Leftrightarrow (A \wedge (P \rightarrow Q)) \rightarrow C$
(*d*) $((P \wedge Q \wedge A) \rightarrow C) \wedge (A \rightarrow (P \vee Q \vee C)) \Leftrightarrow (A \wedge (P \rightleftarrows Q)) \rightarrow C$
(See Sec. 1-4.4 for application of these equivalences.)

Table 1-2.21

P	Q	$P \overset{\cdot}{\rightarrow} Q$
T	T	T
T	F	T
F	T	F
F	F	T

10 Obtain formulas having the simplest possible form which are equivalent to the formulas given here.

(*a*) $((P \to Q) \rightleftarrows (\neg Q \to \neg P)) \wedge R$

(*b*) $P \vee (\neg P \vee (Q \wedge \neg Q))$

(*c*) $(P \wedge (Q \wedge S)) \vee (\neg P \wedge (Q \wedge S))$

11 A chemical plant produces sulfuric acid with a pH value in the range 2.5 to 3.0. There is a main tank in which the acid is held while it is being diluted or heated (concentrated) to put it within this range. If an output valve which empties the tank is open, no heating or diluting can take place because the current batch is being moved out of the tank for final storage. Dilution will occur when the pH is less than 2.5 (as long as the tank is not full) and also whenever the level of the liquid remains below a specified level (the tank must be filled somehow after it has been emptied). Heating will take place when the pH is greater than 3.0 as long as the tank is not full. (A full tank would spill over during the agitation caused by vigorous heating.)

Express the control circuit as block diagrams for both the heating control and the dilution control.

1-3 NORMAL FORMS

Let $A(P_1, P_2, \ldots, P_n)$ be a statement formula where $P_1, P_2, \ldots, P_n$ are the atomic variables. If we consider all possible assignments of the truth values to $P_1, P_2, \ldots, P_n$ and obtain the resulting truth values of the formula A, then we get the truth table for A. Such a truth table contains 2^n rows. The formula may have the truth value T for all possible assignments of the truth values to the variables $P_1, P_2, \ldots, P_n$. In this case, A is said to be identically true, or a tautology. If A has the truth value F for all possible assignments of the truth values to $P_1, P_2, \ldots, P_n$, then A is said to be identically false, or a contradiction. Finally, if A has the truth value T for at least one combination of truth values assigned to $P_1, P_2, \ldots, P_n$, then A is said to be *satisfiable*.

The problem of determining, in a finite number of steps, whether a given statement formula is a tautology or a contradiction or at least satisfiable is known as a *decision problem*. Obviously, the construction of truth tables involves a finite number of steps, and, as such, a decision problem in the statement calculus always has a solution. Similarly, decision problems can be posed for other logical systems, particularly for the predicate calculus. However, in the latter case, the solution of the decision problem may not be simple.

As was mentioned earlier, the construction of truth tables may not be practical, even with the aid of a computer. We therefore consider other procedures known as reduction to normal forms.

1-3.1 Disjunctive Normal Forms

It will be convenient to use the word "product" in place of "conjunction" and "sum" in place of "disjunction" in our current discussion.

A product of the variables and their negations in a formula is called an *elementary product*. Similarly, a sum of the variables and their negations is called an *elementary sum*.

Let P and Q be any two atomic variables. Then P, $\neg P \wedge Q$, $\neg Q \wedge P \wedge$

$\neg P$, $P \wedge \neg P$, and $Q \wedge \neg P$ are some examples of elementary products. On the other hand, P, $\neg P \vee Q$, $\neg Q \vee P \vee \neg P$, $P \vee \neg P$, and $Q \vee \neg P$ are examples of elementary sums of the two variables. Any part of an elementary sum or product which is itself an elementary sum or product is called a *factor* of the original elementary sum or product. Thus $\neg Q$, $P \wedge \neg P$, and $\neg Q \wedge P$ are some of the factors of $\neg Q \wedge P \wedge \neg P$. The following statements hold for elementary sums and products.

> A necessary and sufficient condition for an elementary product to be identically false is that it contain at least one pair of factors in which one is the negation of the other.
>
> A necessary and sufficient condition for an elementary sum to be identically true is that it contain at least one pair of factors in which one is the negation of the other.

We shall now prove the first of these two statements. The proof of the second will follow along the same lines.

We know that for any variable P, $P \wedge \neg P$ is identically false. Hence if $P \wedge \neg P$ appears in the elementary product, then the product is identically false. On the other hand, if an elementary product is identically false and does not contain at least one factor of this type, then we can assign truth values T and F to variables and negated variables, respectively, that appear in the product. This assignment would mean that the elementary product has the truth value T. But that statement is contrary to our assumption. Hence the statement follows.

A formula which is equivalent to a given formula and which consists of a sum of elementary products is called a *disjunctive normal form* of the given formula.

We shall first discuss a procedure by which one can obtain a disjunctive normal form of a given formula. It has already been shown that if the connectives $\rightarrow$ and $\rightleftarrows$ appear in the given formula, then an equivalent formula can be obtained in which "$\rightarrow$" and "$\rightleftarrows$" are replaced by $\wedge$, $\vee$, and $\neg$. This statement would be true of any other connective yet undefined. The truth of this statement will become apparent after our discussion of principal disjunctive normal forms. Therefore, there is no loss of generality in assuming that the given formula contains the connectives $\wedge$, $\vee$, and $\neg$ only.

If the negation is applied to the formula or to a part of the formula and not to the variables appearing in it, then by using De Morgan's laws an equivalent formula can be obtained in which the negation is applied to the variables only. After this step, the formula obtained may still fail to be in disjunctive normal form because there may be some parts of it which are products of sums. By repeated application of the distributive laws we obtain the required normal form.

EXAMPLE 1 Obtain disjunctive normal forms of (*a*) $P \wedge (P \rightarrow Q)$; (*b*) $\neg(P \vee Q) \rightleftarrows (P \wedge Q)$.

SOLUTION

(*a*) $P \wedge (P \rightarrow Q) \Leftrightarrow P \wedge (\neg P \vee Q) \Leftrightarrow (P \wedge \neg P) \vee (P \wedge Q)$

(*b*) $\neg(P \vee Q) \rightleftarrows (P \wedge Q)$

$$\Leftrightarrow (\neg(P \vee Q) \wedge (P \wedge Q)) \vee ((P \vee Q) \wedge \neg(P \wedge Q))$$

[using $R \rightleftarrows S \Leftrightarrow (R \wedge S) \vee (\neg R \wedge \neg S)$]

$$\begin{aligned}
&\Leftrightarrow (\neg P \wedge \neg Q \wedge P \wedge Q) \vee ((P \vee Q) \wedge (\neg P \vee \neg Q))\\
&\Leftrightarrow (\neg P \wedge \neg Q \wedge P \wedge Q) \vee ((P \vee Q) \wedge \neg P)\\
&\quad \vee ((P \vee Q) \wedge \neg Q)\\
&\Leftrightarrow (\neg P \wedge \neg Q \wedge P \wedge Q) \vee (P \wedge \neg P) \vee (Q \wedge \neg P)\\
&\quad \vee (P \wedge \neg Q) \vee (Q \wedge \neg Q)
\end{aligned}$$

which is the required disjunctive normal form. ////

The disjunctive normal form of a given formula is not unique. In fact, different disjunctive normal forms can be obtained for a given formula if the distributive laws are applied in different ways. Apart from this fact, the factors in each elementary product, as well as the factors in the sum, can be commuted. However, we shall not consider as distinct the various disjunctive normal forms obtained by reordering the factors either in the elementary products or in the sums.

Consider the formula $F \vee (Q \wedge R)$. Here the formula is already in the disjunctive normal form. However, we may write

$$\begin{aligned}
P \vee (Q \wedge R) \Leftrightarrow (P \vee Q) \wedge (P \vee R) \Leftrightarrow (P \wedge P)\\
\vee (P \wedge Q) \vee (P \wedge R) \vee (Q \wedge R)
\end{aligned}$$

the last equivalent formula being another equivalent disjunctive normal form. Of course, different disjunctive normal forms of the same formula are equivalent. In order to arrive at a unique normal form of a given formula, we introduce the principal disjunctive normal form in Sec. 1-3.3.

Finally, we remark that a given formula is identically false if every elementary product appearing in its disjunctive normal form is identically false. For the assumption to be true, every elementary product would have to have at least two factors, of which one is the negation of the other.

1-3.2 Conjunctive Normal Forms

A formula which is equivalent to a given formula and which consists of a product of elementary sums is called a *conjunctive normal form* of the given formula.

The method for obtaining a conjunctive normal form of a given formula is similar to the one given for disjunctive normal forms and will be demonstrated by examples. Again, the conjunctive normal form is not unique. Furthermore, a given formula is identically true if every elementary sum in its conjunctive normal form is identically true. This would be the case if every elementary sum appearing in the formula had at least two factors, of which one is the negation of the other.

EXAMPLE 1 Obtain a conjunctive normal form of each of the formulas given in Example 1 of Sec. 1-3.1.

SOLUTION

(a) $P \wedge (P \to Q) \Leftrightarrow P \wedge (\neg P \vee Q)$. Hence $P \wedge (\neg P \vee Q)$ is a required form.
(b) $\neg(P \vee Q) \rightleftarrows (P \wedge Q) \Leftrightarrow (\neg(P \vee Q) \to (P \wedge Q)) \wedge ((P \wedge Q) \to \neg(P \vee Q))$

[using $R \rightleftarrows S \Leftrightarrow (R \to S) \wedge (S \to R)$]

$$\begin{aligned}
&\Leftrightarrow ((P \vee Q) \vee (P \wedge Q)) \wedge (\neg(P \wedge Q) \vee (\neg P \wedge \neg Q)) \\
&\Leftrightarrow ((P \vee Q \vee P) \wedge (P \vee Q \vee Q)) \wedge ((\neg P \vee \neg Q) \vee (\neg P \wedge \neg Q)) \\
&\Leftrightarrow (P \vee Q \vee P) \wedge (P \vee Q \vee Q) \wedge (\neg P \vee \neg Q \vee \neg P) \wedge (\neg P \vee \neg Q \vee \neg Q)
\end{aligned}$$

////

EXAMPLE 2 Show that the formula $Q \vee (P \wedge \neg Q) \vee (\neg P \wedge \neg Q)$ is a tautology.

SOLUTION First we obtain a conjunctive normal form of the given formula.

$$\begin{aligned}
Q \vee (P \wedge \neg Q) \vee (\neg P \wedge \neg Q) &\Leftrightarrow Q \vee ((P \vee \neg P) \wedge \neg Q) \\
&\Leftrightarrow (Q \vee (P \vee \neg P)) \wedge (Q \vee \neg Q) \\
&\Leftrightarrow (Q \vee P \vee \neg P) \wedge (Q \vee \neg Q)
\end{aligned}$$

Since each of the elementary sums is a tautology, the given formula is a tautology. ////

1-3.3 Principal Disjunctive Normal Forms

Let P and Q be two statement variables. Let us construct all possible formulas which consist of conjunctions of P or its negation and conjunctions of Q or its negation. None of the formulas should contain both a variable and its negation. Furthermore, any formula which is obtained by commuting the formulas in the conjunction is not included in the list because such a formula will be equivalent to one included in the list. For example, either $P \wedge Q$ or $Q \wedge P$ is included, but not both. For two variables P and Q, there are 2^2 such formulas given by

$$P \wedge Q \qquad P \wedge \neg Q \qquad \neg P \wedge Q \qquad \text{and} \qquad \neg P \wedge \neg Q$$

These formulas are called *minterms* or Boolean conjunctions of P and Q. From the truth tables of these minterms, it is clear that no two minterms are equivalent. Each minterm has the truth value T for exactly one combination of the truth values of the variables P and Q. This fact is shown in Table 1-3.1.

We assert that if the truth table of any formula containing only the variables P and Q is known, then one can easily obtain an equivalent formula which consists of a disjunction of some of the minterms. This statement is demonstrated as follows.

For every truth value T in the truth table of the given formula, select the

Table 1-3.1

P	Q	$P \wedge Q$	$P \wedge \neg Q$	$\neg P \wedge Q$	$\neg P \wedge \neg Q$
T	T	T	F	F	F
T	F	F	T	F	F
F	T	F	F	T	F
F	F	F	F	F	T

minterm which also has the value T for the same combination of the truth values of P and Q. The disjunction of these minterms will then be equivalent to the given formula.

This discussion provides the basis for a proof that a formula containing any connective is equivalent to a formula containing $\wedge$, $\vee$, and $\neg$.

For a given formula, an equivalent formula consisting of disjunctions of minterms only is known as its *principal disjunctive normal form*. Such a normal form is also called the *sum-of-products canonical form*.

EXAMPLE 1 The truth tables for $P \rightarrow Q$, $P \vee Q$, and $\neg(P \wedge Q)$ are given in Table 1-3.2. Obtain the principal disjunctive normal forms of these formulas.

SOLUTION

$$P \rightarrow Q \Leftrightarrow (P \wedge Q) \vee (\neg P \wedge Q) \vee (\neg P \wedge \neg Q)$$

$$P \vee Q \Leftrightarrow (P \wedge Q) \vee (P \wedge \neg Q) \vee (\neg P \wedge Q)$$

$$\neg(P \wedge Q) \Leftrightarrow (P \wedge \neg Q) \vee (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \qquad ////$$

Note that the number of minterms appearing in the normal form is the same as the number of entries with the truth value T in the truth table of the given formula. Thus every formula which is not a contradiction has an equivalent principal disjunctive normal form. Further, such a normal form is unique, except for the rearrangements of the factors in the disjunctions as well as in each of the minterms. One can get a unique normal form by imposing a certain order in which the variables appear in the minterms as well as a definite order in which the minterms appear in the disjunction. In that case, if two given formulas are equivalent, then both of them must have identical principal disjunctive normal forms. Therefore, if we can devise a method other than the construction of truth tables to obtain the principal disjunctive normal form of a given formula, then

Table 1-3.2

P	Q	$P \rightarrow Q$	$P \vee Q$	$\neg(P \wedge Q)$
T	T	T	T	F
T	F	F	T	T
F	T	T	T	T
F	F	T	F	T

the same method can be used to determine whether two given formulas are equivalent.

Although our discussion of the principal disjunctive normal form was restricted to formulas containing only two variables, it is possible to define the minterms for three or more variables. Minterms for the three variables P, Q, and R are

$$\begin{array}{llll} P \wedge Q \wedge R & P \wedge Q \wedge \neg R & P \wedge \neg Q \wedge R & P \wedge \neg Q \wedge \neg R \\ \neg P \wedge Q \wedge R & \neg P \wedge Q \wedge \neg R & \neg P \wedge \neg Q \wedge R & \neg P \wedge \neg Q \wedge \neg R \end{array}$$

These minterms satisfy properties similar to those given for two variables. An equivalent principal disjunctive normal form of any formula which depends upon the variables P, Q, and R can be obtained. Note that there are 2^3 minterms for three variables or, more generally, 2^n minterms for n variables. For any formula containing n variables which are denoted by $P_1, P_2, \ldots, P_n$, an equivalent disjunctive normal form can be obtained by selecting appropriate minterms out of the set of 2^n possible minterms.

If a formula is a tautology, then obviously all the minterms appear in its principal disjunctive normal form; it is also possible to determine whether a given formula is a tautology by obtaining its principal disjunctive normal form.

In order to obtain the principal disjunctive normal form of a given formula without constructing its truth table, one may first replace the conditionals and biconditionals by their equivalent formulas containing only $\wedge$, $\vee$, and $\neg$. Next, the negations are applied to the variables by using De Morgan's laws followed by the application of distributive laws, as was done earlier in obtaining the disjunctive or conjunctive normal forms. Any elementary product which is a contradiction is dropped. Minterms are obtained in the disjunctions by introducing the missing factors. Identical minterms appearing in the disjunctions are deleted. This procedure is demonstrated by means of examples.

EXAMPLE 2 Obtain the principal disjunctive normal forms of (*a*) $\neg P \vee Q$; (*b*) $(P \wedge Q) \vee (\neg P \wedge R) \vee (Q \wedge R)$.

SOLUTION

(*a*) $\neg P \vee Q \Leftrightarrow (\neg P \wedge (Q \vee \neg Q)) \vee (Q \wedge (P \vee \neg P))$

$$(A \wedge \mathbf{T} \Leftrightarrow A)$$

$$\Leftrightarrow (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \vee (Q \wedge P) \vee (Q \wedge \neg P)$$

(distributive laws)

$$\Leftrightarrow (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \vee (P \wedge Q)$$

(commutative law and $P \vee P \Leftrightarrow P$)

(See Example 1.)

(*b*) $(P \wedge Q) \vee (\neg P \wedge R) \vee (Q \wedge R)$

$$\Leftrightarrow (P \wedge Q \wedge (R \vee \neg R)) \vee (\neg P \wedge R \wedge (Q \vee \neg Q)) \vee (Q \wedge R \wedge (P \vee \neg P))$$

$$\Leftrightarrow (P \wedge Q \wedge R) \vee (P \wedge Q \wedge \neg R) \vee (\neg P \wedge Q \wedge R) \vee (\neg P \wedge \neg Q \wedge R)$$

////

EXAMPLE 3 Show that the following are equivalent formulas.

(*a*) $P \vee (P \wedge Q) \Leftrightarrow P$
(*b*) $P \vee (\neg P \wedge Q) \Leftrightarrow P \vee Q$

SOLUTION We write the principal disjunctive normal form of each formula and compare these normal forms.

$$(a)\quad P \vee (P \wedge Q) \Leftrightarrow (P \wedge (Q \vee \neg Q)) \vee (P \wedge Q) \Leftrightarrow (P \wedge Q) \vee (P \wedge \neg Q)$$
$$P \Leftrightarrow P \wedge (Q \vee \neg Q) \Leftrightarrow (P \wedge Q) \vee (P \wedge \neg Q)$$
$$(b)\quad P \vee (\neg P \wedge Q) \Leftrightarrow (P \wedge (Q \vee \neg Q)) \vee (\neg P \wedge Q)$$
$$\Leftrightarrow (P \wedge Q) \vee (P \wedge \neg Q) \vee (\neg P \wedge Q)$$
$$P \vee Q \Leftrightarrow (P \wedge (Q \vee \neg Q)) \vee (Q \wedge (P \vee \neg P))$$
$$\Leftrightarrow (P \wedge Q) \vee (P \wedge \neg Q) \vee (\neg P \wedge Q) \qquad ////$$

EXAMPLE 4 Obtain the principal disjunctive normal form of

$$P \rightarrow ((P \rightarrow Q) \wedge \neg(\neg Q \vee \neg P))$$

SOLUTION Using $P \rightarrow Q \Leftrightarrow \neg P \vee Q$ and De Morgan's law, we obtain

$$P \rightarrow ((P \rightarrow Q) \wedge \neg(\neg Q \vee \neg P))$$
$$\Leftrightarrow \neg P \vee ((\neg P \vee Q) \wedge (Q \wedge P))$$
$$\Leftrightarrow \neg P \vee (\neg P \wedge (Q \wedge P)) \vee (Q \wedge (Q \wedge P))$$
$$\Leftrightarrow \neg P \vee (Q \wedge P)$$
$$\Leftrightarrow (\neg P \wedge (Q \vee \neg Q)) \vee (Q \wedge P)$$
$$\Leftrightarrow (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \vee (P \wedge Q) \qquad ////$$

The procedure described above becomes tedious if the given formula is complicated and contains more than two or three variables. When the number of variables is large, even a comparison of two principal disjunctive normal forms becomes cumbersome. In Sec. 1-3.5, we describe an ordering procedure for the variables and a notation which make such a comparison easy. We also discuss in Chap. 2 a computer program to obtain the sum-of-products canonical form for a given formula.

1-3.4 Principal Conjunctive Normal Forms

In order to define the principal conjunctive normal form, we first define formulas which are called maxterms. For a given number of variables, the *maxterm* consists of disjunctions in which each variable or its negation, but not both, appears only once. Thus the maxterms are the duals of minterms. Either from the duality principle or directly from the truth tables, it can be ascertained that each of the maxterms has the truth value F for exactly one combination of the truth values of the variables. Also different maxterms have the truth value F for different combinations of the truth values of the variables.

For a given formula, an equivalent formula consisting of conjunctions of the maxterms only is known as its *principal conjunctive normal form*. This normal form is also called the *product-of-sums canonical form*. Every formula which is not a tautology has an equivalent principal conjunctive normal form which is unique except for the rearrangement of the factors in the maxterms as well as in their conjunctions. The method for obtaining the principal conjunctive normal form for a given formula is similar to the one described previously for the principal disjunctive normal form. In fact, all the assertions made for the principal disjunctive normal forms can also be made for the principal conjunctive normal forms by the duality principle.

If the principal disjunctive (conjunctive) normal form of a given formula A containing n variables is known, then the principal disjunctive (conjunctive) normal form of $\neg A$ will consist of the disjunction (conjunction) of the remaining minterms (maxterms) which do not appear in the principal disjunctive (conjunctive) normal form of A. From $A \Leftrightarrow \neg\neg A$ one can obtain the principal conjunctive (disjunctive) normal form of A by repeated applications of De Morgan's laws to the principal disjunctive (conjunctive) normal form of $\neg A$. This procedure will be illustrated by an example.

In order to determine whether two given formulas A and B are equivalent, one can obtain any of the principal normal forms of the two formulas and compare them. It is not necessary to assume that both formulas have the same variables. In fact, each formula can be assumed to depend upon all the variables that appear in both formulas, by introducing the missing variables and then reducing them to their principal normal forms.

EXAMPLE 1 Obtain the principal conjunctive normal form of the formula S given by $(\neg P \to R) \wedge (Q \rightleftarrows P)$.

SOLUTION

$$\begin{aligned}
&(\neg P \to R) \wedge (Q \rightleftarrows P)\\
&\quad \Leftrightarrow (P \vee R) \wedge ((Q \to P) \wedge (P \to Q))\\
&\quad \Leftrightarrow (P \vee R) \wedge (\neg Q \vee P) \wedge (\neg P \vee Q)\\
&\quad \Leftrightarrow (P \vee R \vee (Q \wedge \neg Q)) \wedge (\neg Q \vee P \vee (R \wedge \neg R))\\
&\qquad\quad \wedge (\neg P \vee Q \vee (R \wedge \neg R))\\
&\quad \Leftrightarrow (P \vee Q \vee R) \wedge (P \vee \neg Q \vee R) \wedge (P \vee \neg Q \vee \neg R)\\
&\qquad\quad \wedge (\neg P \vee Q \vee R) \wedge (\neg P \vee Q \vee \neg R)
\end{aligned}$$

Now the conjunctive normal form of $\neg S$ can easily be obtained by writing the conjunction of the remaining maxterms; thus, $\neg S$ has the principal conjunctive normal form

$$(P \vee Q \vee \neg R) \wedge (\neg P \vee \neg Q \vee R) \wedge (\neg P \vee \neg Q \vee \neg R)$$

By considering $\neg\neg S$, we obtain

$$\begin{aligned}
&\neg(P \vee Q \vee \neg R) \vee \neg(\neg P \vee \neg Q \vee R) \vee \neg(\neg P \vee \neg Q \vee \neg R)\\
&\quad \Leftrightarrow (\neg P \wedge \neg Q \wedge R) \vee (P \wedge Q \wedge \neg R) \vee (P \wedge Q \wedge R)
\end{aligned}$$

which is the principal disjunctive normal form of S. ////

EXAMPLE 2 The truth table for a formula A is given in Table 1-3.3. Determine its disjunctive and conjunctive normal forms.

SOLUTION By choosing the minterms corresponding to each T value of A, we obtain

$$A \Leftrightarrow (P \wedge \neg Q \wedge R) \vee (\neg P \wedge Q \wedge R) \vee (\neg P \wedge Q \wedge \neg R) \vee (\neg P \wedge \neg Q \wedge \neg R)$$

Similarly

$$A \Leftrightarrow (\neg P \vee \neg Q \vee \neg R) \wedge (\neg P \vee \neg Q \vee R) \wedge (\neg P \vee Q \vee R) \wedge (P \vee Q \vee \neg R)$$

Here the maxterms appearing in the normal form correspond to the F values of A. The maxterms are written down by including the variable if its truth value is F and its negation if the value is T. ////

1-3.5 Ordering and Uniqueness of Normal Forms

Given any n statement variables, let us first arrange them in some fixed order. If capital letters are used to denote the variables, then they may be arranged in alphabetical order. If subscripted letters are also used, then the following is an illustration of the order that may be used:

$$A, B, \ldots, Z, A_1, B_1, \ldots, Z_1, A_2, B_2, \ldots$$

As an example, if the variables are P_1, Q, R_3, S_1, T_2, and Q_3, then they may be arranged as

$$Q, P_1, S_1, T_2, Q_3, R_3$$

Once the variables have been arranged in a particular order, it is possible to designate them as the first variable, second variable, and so on.

Let us assume that n variables are given and are arranged in a particular order. The 2^n minterms corresponding to the n variables can be designated by $m_0, m_1, \ldots, m_{2^n-1}$. If we write the subscript of any particular minterm in binary and add an appropriate number of zeros on the left (if necessary) so that the number of digits in the subscript is exactly n, then we can obtain the corresponding minterm in the following manner. If in the ith location from the left there

Table 1-3.3

P	Q	R	A
T	T	T	F
T	T	F	F
T	F	T	T
T	F	F	F
F	T	T	T
F	T	F	T
F	F	T	F
F	F	F	T

appears 1, then the *i*th variable appears in the conjunction. If 0 appears in the *i*th location from the left, then the negation of the *i*th variable appears in the conjunction forming the minterm. Thus each of $m_0, m_1, \ldots, m_{2^n-1}$ corresponds to a unique minterm, which can be determined from the binary representation of the subscript. Conversely, given any minterm, one can find which of m_0, $m_1, \ldots, m_{2^n-1}$ designates it.

Let P, Q, and R be three variables arranged in that order. The corresponding minterms are denoted by $m_0, m_1, \ldots, m_7$. We can write the subscript 5 in binary as 101, and the minterm m_5 is $P \wedge \neg Q \wedge R$. Similarly m_0 corresponds to $\neg P \wedge \neg Q \wedge \neg R$. To obtain the minterm m_3, we write 3 as 11 and append a zero on the left to get 011, and m_3 is $\neg P \wedge Q \wedge R$.

If we have six variables $P_1, P_2, \ldots, P_6$, then there are $2^6 = 64$ minterms denoted by $m_0, m_1, \ldots, m_{63}$. To get a minterm, m_{38} say, we write 38 in binary as 100110; then the minterm m_{38} is $P_1 \wedge \neg P_2 \wedge \neg P_3 \wedge P_4 \wedge P_5 \wedge \neg P_6$.

Having developed a notation for the representation of the minterms, which can be further simplified by writing only the subscripts of $m_0, m_1, \ldots, m_{2^n-1}$, we designate the disjunction (sum) of minterms by the compact notation $\sum$. Using such a notation, the sum-of-products canonical form representing the disjunction of m_i, m_j, and m_k can be written down as $\sum i,j,k$. As an example, it is known that

$$(P \wedge Q) \vee (\neg P \wedge R) \vee (Q \wedge R) \Leftrightarrow (\neg P \wedge \neg Q \wedge R)$$
$$\vee (\neg P \wedge Q \wedge R) \vee (P \wedge Q \wedge \neg R) \vee (P \wedge Q \wedge R)$$

Thus we denote the principal disjunctive normal form of

$$(P \wedge Q) \vee (\neg P \wedge R) \vee (Q \wedge R)$$

as $\sum 1,3,6,7$. With this type of representation and simplification of notation, it is easy to compare two principal disjunctive normal forms.

We now proceed to obtain a similar representation for the product-of-sums (principal conjunctive normal) forms. We denote the maxterms of n variables by $M_0, M_1, \ldots, M_{2^n}-1$. Again, the maxterm corresponding to M_j, say, is obtained by writing j in binary and appending the required number of zeros to the left in order to get n digits. If 0 appears in the *i*th location from the left of this binary number, then the *i*th variable appears in the disjunction, while if 1 appears in the *i*th location, then the negation of the *i*th variable appears. Thus the binary representation of the subscript uniquely determines the maxterm, and, conversely, every binary representation of numbers between 0 and $2^n - 1$ determines a maxterm. Note that the convention regarding 1 and 0 here is the opposite of what was used for minterms. This convention is adopted with a view to connect the two principal normal forms of any given formula.

The maxterms, $M_0, M_1, \ldots, M_7$, corresponding to three variables P, Q, and R, are

$$\begin{array}{llll} P \vee Q \vee R & P \vee Q \vee \neg R & P \vee \neg Q \vee R & P \vee \neg Q \vee \neg R \\ \neg P \vee Q \vee R & \neg P \vee Q \vee \neg R & \neg P \vee \neg Q \vee R & \neg P \vee \neg Q \vee \neg R \end{array}$$

As before, further simplification is introduced by using $\prod$ to denote the conjunction (product) of maxterms. Thus $\prod i,j,k$ represents the conjunction of maxterms M_i, M_j, M_k.

To illustrate this discussion, we consider $(P \wedge Q) \vee (\neg P \wedge R)$. We obtain its principal conjunctive normal form as follows.

$$\begin{aligned}
&(P \wedge Q) \vee (\neg P \wedge R)\\
&\qquad \Leftrightarrow ((P \wedge Q) \vee \neg P) \wedge ((P \wedge Q) \vee R)\\
&\qquad \Leftrightarrow (P \vee \neg P) \wedge (Q \vee \neg P) \wedge (P \vee R) \wedge (Q \vee R)\\
&\qquad \Leftrightarrow (Q \vee \neg P \vee (R \wedge \neg R)) \wedge (P \vee R \vee (Q \wedge \neg Q))\\
&\qquad\quad \wedge (Q \vee R \vee (P \wedge \neg P))\\
&\qquad \Leftrightarrow (Q \vee \neg P \vee R) \wedge (Q \vee \neg P \vee \neg R) \wedge (P \vee R \vee Q)\\
&\qquad\quad \wedge (P \vee R \vee \neg Q) \wedge (Q \vee R \vee P) \wedge (Q \vee R \vee \neg P)\\
&\qquad \Leftrightarrow (\neg P \vee Q \vee R) \wedge (\neg P \vee Q \vee \neg R) \wedge (P \vee Q \vee R)\\
&\qquad\quad \wedge (P \vee \neg Q \vee R)
\end{aligned}$$

Thus the product-of-sums canonical form of $(P \wedge Q) \vee (\neg P \wedge R)$ can be represented as $\prod 0,2,4,5$. Note that its disjunctive normal form is

$$\begin{aligned}
&(P \wedge Q \wedge R) \vee (P \wedge Q \wedge \neg R) \vee (\neg P \wedge Q \wedge R)\\
&\qquad \vee (\neg P \wedge \neg Q \wedge R) \Leftrightarrow \sum 1,3,6,7
\end{aligned}$$

More generally, given any formula containing n variables and using the above notations to represent the equivalent principal disjunctive and conjunctive normal forms, we see clearly that all numbers lying between 0 and $2^n - 1$ which do not appear in one normal form will appear in the other. This statement follows from the principle of duality and the discussion given earlier regarding the relation between these two principal normal forms.

EXERCISES 1-3.5

1 Write equivalent forms for the following formulas in which negations are applied to the variables only.

(*a*) $\neg(P \vee Q)$ (*d*) $\neg(P \rightleftarrows Q)$
(*b*) $\neg(P \wedge Q)$ (*e*) $\neg(P \uparrow Q)$
(*c*) $\neg(P \to Q)$ (*f*) $\neg(P \downarrow Q)$

Obtain the principal conjunctive normal forms of (*a*), (*c*), and (*d*).

2 Obtain the product-of-sums canonical forms of the following formulas.

(*a*) $(P \wedge Q \wedge R) \vee (\neg P \wedge R \wedge Q) \vee (\neg P \wedge \neg Q \wedge \neg R)$
(*b*) $(\neg S \wedge \neg P \wedge R \wedge Q) \vee (S \wedge P \wedge \neg R \wedge \neg Q) \vee (\neg S \wedge P \wedge R \wedge \neg Q) \vee$ $(Q \wedge \neg P \wedge \neg R \wedge S) \vee (P \wedge \neg S \wedge \neg R \wedge Q)$
(*c*) $(P \wedge Q) \vee (\neg P \wedge Q) \vee (P \wedge \neg Q)$
(*d*) $(P \wedge Q) \vee (\neg P \wedge Q \wedge R)$

3 Obtain the principal disjunctive and conjunctive normal forms of the following formulas.

(*a*) $(\neg P \vee \neg Q) \to (P \rightleftarrows \neg Q)$ (*d*) $(P \to (Q \wedge R)) \wedge (\neg P \to (\neg Q \wedge \neg R))$
(*b*) $Q \wedge (P \vee \neg Q)$ (*e*) $P \to (P \wedge (Q \to P))$
(*c*) $P \vee (\neg P \to (Q \vee (\neg Q \to R)))$ (*f*) $(Q \to P) \wedge (\neg P \wedge Q)$

Which of the above formulas are tautologies?

1-3.6 Completely Parenthesized Infix Notation and Polish Notation

In this section we discuss the problem of translating a given logical expression or a statement formula into some target language such as machine language or assembly language. Also, from a given statement formula we wish to determine mechanically its truth value for various assignments of truth values to its variables. In order to do such an evaluation, it is first necessary for us to examine our notations, symbols, and rules for writing a well-formed formula.

In writing a statement formula, parentheses have been used whenever necessary. In fact, our definition of well-formed formulas as given in Sec. 1-2.7 imposes upon us the condition that a certain number of parentheses must be used. In order that the number of parentheses not become excessively large, we developed certain conventions. The first such convention developed and incorporated in the definition of the well-formed formula stated that if A is a well-formed formula, then $\neg A$ is a well-formed formula. No parentheses were used with A. This means $\neg$ has the highest precedence and applies to any well-formed formula following it. In this way, $\neg P \vee Q$ is understood to mean $(\neg P) \vee Q$, not $\neg(P \vee Q)$. Also $\neg(P \vee Q) \wedge R$ stands for $(\neg(P \vee Q) \wedge R)$. The next convention suggested was that the outermost parentheses of an expression be dropped, so that $(P \wedge Q) \vee R$ was taken to be a well-formed formula instead of $((P \wedge Q) \vee R)$ as required by the original definition. Nested parentheses have also been used. It is understood that the proper subexpression to be considered first is obtained by scanning from left to right up to the first right parenthesis encountered and then scanning left until a left parenthesis is detected. If an evaluation of such an expression is to be done mechanically, the number of parentheses should be reduced by adopting certain conventions so that an excessive number of scannings of the expression is avoided.

One method of reducing the number of parentheses further is to prescribe an order of precedence for the connectives. Once this order is prescribed, further reductions can be made by requiring that for any two binary connectives appearing in a formula which have the same precedence, the left one is evaluated first. The same requirement can be stated by saying that the binary connectives are *left-associative,* while $\neg$ is *right-associative.* Such a convention is commonly used in arithmetic; for example, $4 + 6 \times 3 - 7$ stands for $[4 + (6 \times 3)] - 7$.

Let us restrict ourselves to formulas containing the connectives $\neg$, $\wedge$, and $\vee$ and assume this order of precedence: $\neg$, then $\wedge$, and then $\vee$. First we consider an expression which does not contain any parentheses, such as

$$\underbrace{\underbrace{P \vee \underbrace{Q \wedge R}_{1}}_{3} \vee \underbrace{S \wedge T}_{2}}_{4}$$

According to our convention, the above expression stands for $(P \vee (Q \wedge R)) \vee (S \wedge T)$. For the evaluation of this expression we must scan from left to right repeatedly. The numbers below the subexpressions indicate the steps of such an evaluation. This process of evaluation is inefficient because of the repeated scanning that must be done.

If there are parentheses in an expression, then the order of precedence is altered by the parentheses. For example, in $(P \vee Q) \wedge R$ we evaluate first $P \vee Q$ and then $(P \vee Q) \wedge R$. In fact, it is possible to write expressions which make the order of evaluation of subexpressions independent of the precedence of the connectives. This task is accomplished by parenthesizing subexpressions in such a way that corresponding to each connective there is a pair of parentheses. This pair encloses the operator (connective) and its operands. We now define the *parenthetical level* of an operator as the total number of pairs of parentheses that surround it. A pair of parentheses has the same parenthetical level as that of the operator to which it corresponds, i.e., of the operator which is immediately enclosed by this pair. Such an expression is called a *fully parenthesized expression*. For example, in the fully parenthesized expression

$$(P \underset{1}{\vee} ((Q \underset{3}{\wedge} R) \underset{2}{\wedge} (\underset{3}{\neg} S)))$$

the integers below the operators specify the parenthetical level of each operator. When such an expression is evaluated, the subexpression containing the operator with the highest parenthetical level is evaluated first. In the case where more than one operator has the highest parenthetical level (as in the above example), we evaluate them one after the other from left to right. Once the subexpressions containing operators at the highest parenthetical level have been evaluated, the subexpressions containing the operators at the next highest level are evaluated in the same way. Thus, in the above example the subexpressions are evaluated in the following order

$$(Q \wedge R) \qquad (\neg S) \qquad ((Q \wedge R) \wedge (\neg S)) \qquad (P \vee ((Q \wedge R) \wedge (\neg S)))$$

As mentioned earlier, for a fully parenthesized expression no convention regarding the order of precedence of an operator is needed.

In the one case when the order of precedence of the operators is prescribed and the expressions are partly parenthesized, or in the other case when the expressions are fully parenthesized, a repeated scanning from left to right is still needed in order to evaluate an expression. The reason is that the operators appear along with the operands inside the expression. The notation used so far was to write the operator (at least the binary) between the operands, for example, $P \wedge Q$. Such a notation is called an *infix notation*. Repeated scanning is avoided if the infix expression is converted first to an equivalent parenthesis-free *suffix* or *prefix* expression in which the subexpressions have the form

operand operand operator

or

operator operand operand

in place of an infix form where we have

operand operator operand

This type of notation is known as *Łukasiewiczian notation* (due to the Polish logician Jan Łukasiewicz) or "reverse Polish" or "Polish" notation. For example, the expressions given in each row of Table 1-3.4 are equivalent.

Table 1-3.4

Infix	Suffix	Prefix
A	A	A
$A \vee B$	$AB\vee$	$\vee AB$
$A \vee B \vee C$	$AB\vee C\vee$	$\vee\vee ABC$
$A \vee (B \vee C)$	$ABC\vee\vee$	$\vee A\vee BC$
$A \vee B \wedge C$	$ABC\wedge\vee$	$\vee A\wedge BC$
$A \wedge (B \vee C)$	$ABC\vee\wedge$	$\wedge A\vee BC$
$A \wedge B \wedge C$	$AB\wedge C\wedge$	$\wedge\wedge ABC$
$A \wedge \neg B$	$AB\neg\wedge$	$\wedge A\neg B$
$A \wedge (B \vee \neg C)$	$ABC\neg\vee\wedge$	$\wedge A\vee B\neg C$

Note that in both the suffix and prefix equivalents of an infix expression the variables are in the same relative position. The only differences are that the expressions in suffix or prefix form are parenthesis-free and that the operators are rearranged according to the rules of precedence for the operators and to the overruling of the precedence rules by the use of parentheses.

A fully parenthesized infix expression can be directly translated to prefix notation by beginning with the conversion of the inner parenthesized subexpression and then proceeding toward the outside of the expression. In the case of the fully parenthesized expression

$$(A \underset{1}{\vee} ((B \underset{3}{\wedge} C) \underset{2}{\wedge} D))$$

the innermost parenthesized subexpression of level 3 is

$$(B \wedge C)$$

and it is converted to $\wedge BC$. This prefix subexpression becomes the first operand of the operator $\wedge$ at level 2. Therefore the subexpression $\wedge BC \wedge D$ of level 2 is converted to the prefix equivalent of $\wedge\wedge BCD$, and finally at level 1 the term $A \vee \wedge\wedge BCD$ is converted to the final prefix form of $\vee A \wedge\wedge BCD$.

Programmers, of course, do not program expressions in fully parenthesized form. Certain FORTRAN and other compilers initially convert partially parenthesized expressions to a fully parenthesized form before conversion to a suffix form is performed. (Suffix form seems to be more convenient for some compilers.)

Let us consider the problem of mechanically converting a *parenthesis-free* expression (containing $\vee$, $\wedge$, and $\neg$) into prefix form. As was mentioned previously, only the operators are rearranged in order to obtain the prefix Polish equivalent. In scanning right to left, the rightmost operator having the highest precedence will be the first operator to be encountered in the prefix string. The next highest precedence operator will be the second operator to be encountered in the expression. Note that for infix expressions if we do not specify that a leftmost (or rightmost in the case of the negation operator) operator has precedence over other operators of equal precedence, then the prefix equivalent is not unique. For example, the expression $A \vee B \vee C$ would be converted to $\vee A \vee BC$ or $\vee\vee ABC$ if no mention was made that the leftmost operator $\vee$ in the infix string has precedence over the remaining operator. From this fact it is clear that when we scan a prefix expression from right to left, we encounter the operators in the same order in which we would have evaluated them by following the precedence convention for operators in the infix expression. For example, the prefix equivalent of $A \vee B \wedge C$, $\vee A \wedge BC$, where the operator $\wedge$ is encountered before $\vee$ in a right-to-left scan, indicates that the conjunction is to be evaluated before the disjunction.

In practice it is often necessary to evaluate expressions, i.e., to determine their truth value for a given set of truth values assigned to their variables. This evaluation can be done more easily by using a prefix (suffix) representation of the expression, because scanning of the expression is required in only one direction, viz., from right to left (left to right) and only once, whereas for the infix expression the scanning has to be done several times and in both directions. For example, to evaluate the prefix expression $\vee A \wedge BC$, we scan this string from right to left until we encounter $\wedge$. The two operands, viz., B and C, which appear immediately to the right of this operator are its operands, and the expression $\wedge BC$ is replaced by its truth value. Let us assume that this value is denoted by T_1. Note that T_1 is either T or F. This fact reduces the original prefix string to $\vee AT_1$. Continuing the scanning beyond T_1, the next operator we encounter is $\vee$ whose operands are A and T_1, and the evaluation results in a value which we will denote by T_2.

This method of evaluating prefix expressions can be summarized by the following four rules which are repeatedly applied until all operators have been processed.

1 Find the rightmost operator in the expression.
2 Select the two operands immediately to the right of the operator found.
3 Perform the indicated operation.
4 Replace the operator and operands with the result.

As a further example, the prefix expression $\vee A \wedge \wedge BCD$ which corresponds to the infix expression $A \vee (B \wedge C) \wedge D$ is evaluated here for values of $A = F$, $B = T$, $C = T$, and $D = T$.

Prefix form	*Current operator*	*Current operands*	*Computed value*
$\vee A \wedge \wedge BCD$	$\wedge$	B, C	$T_1 = T$
$\vee A \wedge T_1 D$	$\wedge$	T_1, D	$T_2 = T$
$\vee AT_2$	$\vee$	A, T_2	$T_3 = T$
T_3	—	—	

We will return to this topic in Sec. 3-4.

Note that the preceding discussion applies equally well to suffix expressions. All that is needed is to change prefix to suffix, left to right, and right to left.

EXERCISES 1-3.6

1 Write the following formulas in prefix and suffix form. The following precedence is assumed: $\rightleftarrows, \rightarrow, \vee, \wedge, \neg$ ($\neg$ having the highest precedence).

(a) $P \to Q \vee R \vee S$
(b) $Q \wedge \neg(R \rightleftarrows P \vee Q)$
(c) $P \wedge \neg R \to Q \rightleftarrows P \wedge Q$
(d) $\neg\neg P \vee Q \wedge R \vee \neg Q$

2 Convert the following prefix and suffix formulas into completely parenthesized form. Also write them in an infix form, using the above order of precedence to minimize the number of parentheses.
(a) $\to \neg P \vee Q \rightleftarrows R \neg S$
(b) $\to\to PQ \to\to QR \to PR$
(c) $P \neg P \to P \to P \to$
(d) $PQ \to RQ \to \wedge PR \vee \wedge Q \to$

1-4 THE THEORY OF INFERENCE FOR STATEMENT CALCULUS

The main function of logic is to provide rules of inference, or principles of reasoning. The theory associated with such rules is known as inference theory because it is concerned with the inferring of a conclusion from certain premises. When a conclusion is derived from a set of premises by using the accepted rules of reasoning, then such a process of derivation is called a *deduction* or a *formal proof.* In a formal proof, every rule of inference that is used at any stage in the derivation is acknowledged. In mathematical literature, the proofs given are generally *informal* in the sense that many steps in the derivation are either omitted or considered to be understood.

An important difference between the reasoning used in any general discussion and that used in mathematics is that the premises used are believed to be true either from experience or from faith, and if proper rules are followed, then one expects the conclusion to be true. In mathematics, one is solely concerned with the conclusion which is obtained by following the rules of logic. This conclusion, called a theorem, can be inferred from a set of premises, called the axioms of the theory. The truth value plays no part in the theory.

Now we come to the questions of what we mean by the rules and theory of inference. The rules of inference are criteria for determining the validity of an argument. These rules are stated in terms of the forms of the statements (premises and conclusions) involved rather than in terms of the actual statements or their truth values. Therefore, the rules will be given in terms of statement formulas rather than in terms of any specific statements. These rules are not arbitrary in the sense that they allow us to indicate as valid at least those arguments which we would normally expect to be valid. In addition, neither do they characterize as valid those arguments which we would normally consider as invalid.

In any argument, a conclusion is admitted to be true provided that the premises (assumptions, axioms, hypotheses) are accepted as true and the reasoning used in deriving the conclusion from the premises follows certain accepted rules of logical inference. Such an argument is called *sound.* In any argument, we are always concerned with its soundness. In logic the situation is slightly different, and we concentrate our attention on the study of the rules of inference by which conclusions are derived from premises. Any conclusion which is arrived at by following these rules is called a *valid conclusion,* and the argument is called a *valid argument.* The actual truth values of the premises do not play any part in the determination of the validity of the argument. In short, in logic we are concerned with the validity but not necessarily with the soundness of an argument.

1-4.1 Validity Using Truth Tables

Let A and B be two statement formulas. We say that "B *logically follows from* A" or "B is a *valid conclusion* (*consequence*) of the premise A" iff $A \rightarrow B$ is a tautology, that is, $A \Rightarrow B$.

Just as the definition of implication was extended to include a set of formulas rather than a single formula, we say that from a set of premises $\{H_1, H_2, \ldots, H_m\}$ a conclusion C follows logically iff

$$H_1 \wedge H_2 \wedge \cdots \wedge H_m \Rightarrow C \qquad (1)$$

Given a set of premises and a conclusion, it is possible to determine whether the conclusion logically follows (we shall simply say "follows") from the given premises by constructing truth tables as follows.

Let $P_1, P_2, \ldots, P_n$ be all the atomic variables appearing in the premises $H_1, H_2, \ldots, H_m$ and the conclusion C. If all possible combinations of truth values are assigned to $P_1, P_2, \ldots, P_n$ and if the truth values of $H_1, H_2, \ldots, H_m$ and C are entered in a table, then it is easy to see from such a table whether (1) is true. We look for the rows in which all $H_1, H_2, \ldots, H_m$ have the value T. If, for every such row, C also has the value T, then (1) holds. Alternatively, we may look for the rows in which C has the value F. If, in every such row, at least one of the values of $H_1, H_2, \ldots, H_m$ is F, then (1) also holds. We call such a method a "truth table technique" for the determination of the validity of a conclusion and demonstrate this technique by examples.

EXAMPLE 1 Determine whether the conclusion C follows logically from the premises H_1 and H_2.

(*a*) $H_1: P \rightarrow Q \quad H_2: P \quad C: Q$

(*b*) $H_1: P \rightarrow Q \quad H_2: \neg P \quad C: Q$

(*c*) $H_1: P \rightarrow Q \quad H_2: \neg(P \wedge Q) \quad C: \neg P$

(*d*) $H_1: \neg P \quad H_2: P \rightleftarrows Q \quad C: \neg(P \wedge Q)$

(*e*) $H_1: P \rightarrow Q \quad H_2: Q \quad C: P$

SOLUTION We first construct the appropriate truth table, as shown in Table 1.4.1. For (*a*) we observe that the first row is the only row in which both

Table 1-4.1

P	Q	$P \rightarrow Q$	$\neg P$	$\neg Q$	$\neg(P \wedge Q)$	$P \rightleftarrows Q$
T	T	T	F	F	F	T
T	F	F	F	T	T	F
F	T	T	T	F	T	F
F	F	T	T	T	T	T

the premises have the value T. The conclusion also has the value T in that row. Hence it is valid. In (b) observe the third and fourth rows. The conclusion Q is true only in the third row, but not in the fourth, and hence the conclusion is not valid. Similarly, we can show that the conclusions are valid in (c) and (d) but not in (e).

The conclusion P in (e) does not follow logically from the premises $P \to Q$ and Q, no matter which statements in English are translated as P and Q or what the truth value of the conclusion P may be. As a particular case, consider the argument

H_1: If Canada is a country, then New York is a city.: $(P \to Q)$
H_2: New York is a city.: (Q)
Conclusion C: Canada is a country.: (P)

Note that both the premises and the conclusion have the truth value T. However, the conclusion does not follow logically from the premises. This example is chosen to emphasize the fact that we are not so much concerned with the conclusion's being true or false as we are with determining whether the conclusion follows from the premises, i.e., whether the argument is valid or invalid. ////

Theoretically, it is possible to determine in a finite number of steps whether a conclusion follows from a given set of premises by constructing the appropriate truth table. However, this method becomes tedious when the number of atomic variables present in all the formulas representing the premises and conclusion is large. This disadvantage, coupled with the fact that the inference theory is applicable in more general situations where the truth table technique is no longer available, suggests that we should investigate other possible methods. This investigating will be done in the following sections.

EXERCISES 1-4.1

1 Show that the conclusion C follows from the premises $H_1, H_2, \ldots$ in the following cases.

(a)	H_1: $P \to Q$			C: $P \to (P \land Q)$
(b)	H_1: $\neg P \lor Q$	H_2: $\neg(Q \land \neg R)$	H_3: $\neg R$	C: $\neg P$
(c)	H_1: $\neg P$	H_2: $P \lor Q$		C: Q
(d)	H_1: $\neg Q$	H_2: $P \to Q$		C: $\neg P$
(e)	H_1: $P \to Q$	H_2: $Q \to R$		C: $P \to R$
(f)	H_1: R	H_2: $P \lor \neg P$		C: R

2 Determine whether the conclusion C is valid in the following, when $H_1, H_2, \ldots$ are the premises.

(a)	H_1: $P \to Q$	H_2: $\neg Q$		C: P
(b)	H_1: $P \lor Q$	H_2: $P \to R$	H_3: $Q \to R$	C: R
(c)	H_1: $P \to (Q \to R)$	H_2: $P \land Q$		C: R
(d)	H_1: $P \to (Q \to R)$	H_2: R		C: P
(e)	H_1: $\neg P$	H_2: $P \lor Q$		C: $P \land Q$

3 Without constructing a truth table, show that $A \land E$ is not a valid consequence of

$$A \rightleftarrows B \qquad B \rightleftarrows (C \land D) \qquad C \rightleftarrows (A \lor E) \qquad A \lor E$$

Also show that $A \lor C$ is not a valid consequence of

$$A \rightleftarrows (B \to C) \qquad B \rightleftarrows (\neg A \lor \neg C) \qquad C \rightleftarrows (A \lor \neg B) \qquad B$$

4 Show that $L \vee M$ follows from

$$P \wedge Q \wedge R \qquad (Q \rightleftarrows R) \rightarrow (L \vee M)$$

5 Show without constructing truth tables that the following statements cannot all be true simultaneously.
(a) $P \rightleftarrows Q \qquad Q \rightarrow R \qquad \neg R \vee S \qquad \neg P \rightarrow S \qquad \neg S$
(b) $R \vee M \qquad \neg R \vee S \qquad \neg M \qquad \neg S$

1-4.2 Rules of Inference

We now describe the process of derivation by which one demonstrates that a particular formula is a valid consequence of a given set of premises. Before we do this, we give two rules of inference which are called rules **P** and **T**.

Rule **P**: A premise may be introduced at any point in the derivation.
Rule **T**: A formula S may be introduced in a derivation if S is tautologically implied by any one or more of the preceding formulas in the derivation.

Before we proceed with the actual process of derivation, we list some important implications and equivalences that will be referred to frequently.

Not all the implications and equivalences listed in Tables 1-4.2 and 1-4.3 respectively are independent of one another. One could start with only a minimum number of them and derive the others by using the above rules of inference. Such an axiomatic approach will not be followed. We list here most of the important implications and equivalences and show how some of them are used in a derivation. Those which are used more often than the others are given special names because of their importance.

EXAMPLE 1 Demonstrate that R is a valid inference from the premises $P \rightarrow Q$, $Q \rightarrow R$, and P.

SOLUTION

{1}	(1)	$P \rightarrow Q$	Rule **P**
{2}	(2)	P	Rule **P**
{1, 2}	(3)	Q	Rule **T**, (1), (2), and I_{11} (modus ponens)
{4}	(4)	$Q \rightarrow R$	Rule **P**
{1, 2, 4}	(5)	R	Rule **T**, (3), (4), and I_{11}

The second column of numbers designates the formula as well as the line of derivation in which it occurs. The set of numbers in braces (the first column) for each line shows the premises on which the formula in the line depends. On the right, **P** or **T** represents the rule of inference, followed by a comment showing from which formulas and tautology that particular formula has been obtained. For example, if we follow this notation, the third line shows that the formula in this line is numbered (3) and has been obtained from premises in (1) and (2). The comment on the right says that the formula Q has been introduced using rule **T** and also indicates the details of the application of rule **T**. ////

Table 1-4.2 IMPLICATIONS

I_1	$P \wedge Q \Rightarrow P$	(simplification)
I_2	$P \wedge Q \Rightarrow Q$	(simplification)
I_3	$P \Rightarrow P \vee Q$	(addition)
I_4	$Q \Rightarrow P \vee Q$	(addition)
I_5	$\neg P \Rightarrow P \rightarrow Q$	
I_6	$Q \Rightarrow P \rightarrow Q$	
I_7	$\neg(P \rightarrow Q) \Rightarrow P$	
I_8	$\neg(P \rightarrow Q) \Rightarrow \neg Q$	
I_9	$P, Q \Rightarrow P \wedge Q$	
I_{10}	$\neg P, P \vee Q \Rightarrow Q$	(disjunctive syllogism)
I_{11}	$P, P \rightarrow Q \Rightarrow Q$	(modus ponens)
I_{12}	$\neg Q, P \rightarrow Q \Rightarrow \neg P$	(modus tollens)
I_{13}	$P \rightarrow Q, Q \rightarrow R \Rightarrow P \rightarrow R$	(hypothetical syllogism)
I_{14}	$P \vee Q, P \rightarrow R, Q \rightarrow R \Rightarrow R$	(dilemma)

Table 1-4.3 EQUIVALENCES

E_1	$\neg\neg P \Leftrightarrow P$	(double negation)
E_2	$P \wedge Q \Leftrightarrow Q \wedge P$	(commutative laws)
E_3	$P \vee Q \Leftrightarrow Q \vee P$	(commutative laws)
E_4	$(P \wedge Q) \wedge R \Leftrightarrow P \wedge (Q \wedge R)$	(associative laws)
E_5	$(P \vee Q) \vee R \Leftrightarrow P \vee (Q \vee R)$	(associative laws)
E_6	$P \wedge (Q \vee R) \Leftrightarrow (P \wedge Q) \vee (P \wedge R)$	(distributive laws)
E_7	$P \vee (Q \wedge R) \Leftrightarrow (P \vee Q) \wedge (P \vee R)$	(distributive laws)
E_8	$\neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q$	(De Morgan's laws)
E_9	$\neg(P \vee Q) \Leftrightarrow \neg P \wedge \neg Q$	(De Morgan's laws)
E_{10}	$P \vee P \Leftrightarrow P$	
E_{11}	$P \wedge P \Leftrightarrow P$	
E_{12}	$R \vee (P \wedge \neg P) \Leftrightarrow R$	
E_{13}	$R \wedge (P \vee \neg P) \Leftrightarrow R$	
E_{14}	$R \vee (P \vee \neg P) \Leftrightarrow \mathbf{T}$	
E_{15}	$R \wedge (P \wedge \neg P) \Leftrightarrow \mathbf{F}$	
E_{16}	$P \rightarrow Q \Leftrightarrow \neg P \vee Q$	
E_{17}	$\neg(P \rightarrow Q) \Leftrightarrow P \wedge \neg Q$	
E_{18}	$P \rightarrow Q \Leftrightarrow \neg Q \rightarrow \neg P$	
E_{19}	$P \rightarrow (Q \rightarrow R) \Leftrightarrow (P \wedge Q) \rightarrow R$	
E_{20}	$\neg(P \rightleftarrows Q) \Leftrightarrow P \rightleftarrows \neg Q$	
E_{21}	$P \rightleftarrows Q \Leftrightarrow (P \rightarrow Q) \wedge (Q \rightarrow P)$	
E_{22}	$(P \rightleftarrows Q) \Leftrightarrow (P \wedge Q) \vee (\neg P \wedge \neg Q)$	

EXAMPLE 2 Show that $R \vee S$ follows logically from the premises $C \vee D$, $(C \vee D) \rightarrow \neg H$, $\neg H \rightarrow (A \wedge \neg B)$, and $(A \wedge \neg B) \rightarrow (R \vee S)$.

SOLUTION

{1}	(1)	$(C \vee D) \rightarrow \neg H$	**P**
{2}	(2)	$\neg H \rightarrow (A \wedge \neg B)$	**P**
{1, 2}	(3)	$(C \vee D) \rightarrow (A \wedge \neg B)$	**T**, (1), (2), and I_{13}
{4}	(4)	$(A \wedge \neg B) \rightarrow (R \vee S)$	**P**
{1, 2, 4}	(5)	$(C \vee D) \rightarrow (R \vee S)$	**T**, (3), (4), and I_{13}
{6}	(6)	$C \vee D$	**P**
{1, 2, 4, 6}	(7)	$R \vee S$	**T**, (5), (6), and I_{11}

The two tautologies frequently used in the above derivations are I_{13}, known as hypothetical syllogism, and I_{11}, known as modus ponens. ////

EXAMPLE 3 Show that $S \vee R$ is tautologically implied by $(P \vee Q) \wedge (P \to R) \wedge (Q \to S)$.

SOLUTION

{1}	(1)	$P \vee Q$	P
{1}	(2)	$\neg P \to Q$	T, (1), E_1, and E_{16}
{3}	(3)	$Q \to S$	P
{1, 3}	(4)	$\neg P \to S$	T, (2), (3), and I_{13}
{1, 3}	(5)	$\neg S \to P$	T, (4), E_{18}, and E_1
{6}	(6)	$P \to R$	P
{1, 3, 6}	(7)	$\neg S \to R$	T, (5), (6), and I_{13}
{1, 3, 6}	(8)	$S \vee R$	T, (7), E_{16}, and E_1 ////

EXAMPLE 4 Show that $R \wedge (P \vee Q)$ is a valid conclusion from the premises $P \vee Q$, $Q \to R$, $P \to M$, and $\neg M$.

SOLUTION

{1}	(1)	$P \to M$	P
{2}	(2)	$\neg M$	P
{1, 2}	(3)	$\neg P$	T, (1), (2), and I_{12}
{4}	(4)	$P \vee Q$	P
{1, 2, 4}	(5)	Q	T, (3), (4), and I_{10}
{6}	(6)	$Q \to R$	P
{1, 2, 4, 6}	(7)	R	T, (5), (6), and I_{11}
{1, 2, 4, 6}	(8)	$R \wedge (P \vee Q)$	T, (4), (7), and I_9 ////

EXAMPLE 5 Show I_{12}: $\neg Q, P \to Q \Rightarrow \neg P$.

SOLUTION

{1}	(1)	$P \to Q$	P
{1}	(2)	$\neg Q \to \neg P$	T, (1), and E_{18}
{3}	(3)	$\neg Q$	P
{1, 3}	(4)	$\neg P$	T, (2), (3), and I_{11} ////

We shall now introduce a third inference rule, known as rule **CP** or rule of conditional proof.

Rule **CP** If we can derive S from R and a set of premises, then we can derive $R \to S$ from the set of premises alone.

Rule **CP** is not new for our purpose here because it follows from the equivalence E_{19} which states that

$$(P \wedge R) \rightarrow S \Leftrightarrow P \rightarrow (R \rightarrow S)$$

Let P denote the conjunction of the set of premises and let R be any formula. The above equivalence states that if R is included as an additional premise and S is derived from $P \wedge R$, then $R \rightarrow S$ can be derived from the premises P alone.

Rule **CP** is also called the *deduction theorem* and is generally used if the conclusion is of the form $R \rightarrow S$. In such cases, R is taken as an additional premise and S is derived from the given premises and R.

EXAMPLE 6 Show that $R \rightarrow S$ can be derived from the premises $P \rightarrow (Q \rightarrow S)$, $\neg R \vee P$, and Q.

SOLUTION Instead of deriving $R \rightarrow S$, we shall include R as an additional premise and show S first.

{1}	(1)	$\neg R \vee P$	**P**
{2}	(2)	R	**P** (assumed premise)
{1, 2}	(3)	P	**T**, (1), (2), and I_{10}
{4}	(4)	$P \rightarrow (Q \rightarrow S)$	**P**
{1, 2, 4}	(5)	$Q \rightarrow S$	**T**, (3), (4), and I_{11}
{6}	(6)	Q	**P**
{1, 2, 4, 6}	(7)	S	**T**, (5), (6), and I_{11}
{1, 4, 6}	(8)	$R \rightarrow S$	**CP**

////

These examples show that a derivation consists of a sequence of formulas, each formula in the sequence being either a premise or tautologically implied by formulas appearing before.

In Sec. 1-3.1 we discussed the decision problem in terms of determining whether a given formula is a tautology. We can extend this notion to the determination of validity of arguments. Accordingly, if one can determine in a finite number of steps whether an argument is valid, then the decision problem for validity is solvable.

One solution to the decision problem for validity is provided by the truth table method. Use of this method is often not practical. The method of derivation just discussed provides only a partial solution to the decision problem, because if an argument is valid, then it is possible to show by this method that the argument is valid. On the other hand, if an argument is not valid, then it is very difficult to decide, after a finite number of steps, that this is the case. There are other methods of derivation which do allow one to determine, after a finite number of steps, whether an argument is or is not valid. One such method is described in Sec. 1-4.4, and its computer implementation is given later in Sec. 2-7.

We shall now give some examples of derivation involving statements in

English. We first symbolize the given statements and then use the method of derivation just discussed.

EXAMPLE 7 "If there was a ball game, then traveling was difficult. If they arrived on time, then traveling was not difficult. They arrived on time. Therefore, there was no ball game." Show that these statements constitute a valid argument.

SOLUTION Let

P: There was a ball game.

Q: Traveling was difficult.

R: They arrived on time.

We are required to show that from the premises $P \rightarrow Q$, $R \rightarrow \neg Q$, and R the conclusion $\neg P$ follows. (Complete the rest of the derivation.) ////

EXAMPLE 8 If A works hard, then either B or C will enjoy themselves. If B enjoys himself, then A will not work hard. If D enjoys himself, then C will not. Therefore, if A works hard, D will not enjoy himself.

SOLUTION Let A: A works hard; B: B will enjoy himself; C: C will enjoy himself; D: D will enjoy himself. Show that $A \rightarrow \neg D$ follows from $A \rightarrow B \vee C$, $B \rightarrow \neg A$, and $D \rightarrow \neg C$. Since the conclusion is given in the form of a condition $A \rightarrow \neg D$, include A as an additional premise and show that $\neg D$ follows logically from all the premises including A. Finally, use rule **CP** to obtain the result. ////

1-4.3 Consistency of Premises and Indirect Method of Proof

A set of formulas $H_1, H_2, \ldots, H_m$ is said to be consistent if their conjunction has the truth value T for some assignment of the truth values to the atomic variables appearing in $H_1, H_2, \ldots, H_m$. If, for every assignment of the truth values to the atomic variables, at least one of the formulas $H_1, H_2, \ldots, H_m$ is false, so that their conjunction is identically false, then the formulas $H_1, H_2, \ldots, H_m$ are called *inconsistent*.

Alternatively, a set of formulas $H_1, H_2, \ldots, H_m$ is inconsistent if their conjunction implies a contradiction, that is,

$$H_1 \wedge H_2 \wedge \cdots \wedge H_m \Rightarrow R \wedge \neg R$$

where R is any formula. Note that $R \wedge \neg R$ is a contradiction, and it is necessary and sufficient for the implication that $H_1 \wedge H_2 \wedge \cdots \wedge H_m$ be a contradiction.

The notion of inconsistency is used in a procedure called *proof by contradiction* or *reductio ad absurdum* or *indirect method of proof.* In order to show that a conclusion C follows logically from the premises $H_1, H_2, \ldots, H_m$, we assume that C is false and consider $\neg C$ as an additional premise. If the new set of premises is inconsistent, so that they imply a contradiction, then the assumption that $\neg C$ is true does not hold simultaneously with $H_1 \wedge H_2 \wedge \cdots \wedge H_m$ being true. Therefore, C is true whenever $H_1 \wedge H_2 \wedge \cdots \wedge H_m$ is true. Thus, C follows logically from the premises $H_1, H_2, \ldots, H_m$.

EXAMPLE 1 Show that $\neg(P \land Q)$ follows from $\neg P \land \neg Q$.

SOLUTION We introduce $\neg\neg(P \land Q)$ as an additional premise and show that this additional premise leads to a contradiction.

{1}	(1)	$\neg\neg(P \land Q)$	**P** (assumed)	
{1}	(2)	$P \land Q$	**T**, (1), and E_1	
{1}	(3)	P	**T**, (2), and I_1	
{4}	(4)	$\neg P \land \neg Q$	**P**	
{4}	(5)	$\neg P$	**T**, (4), I_1	
{1, 4}	(6)	$P \land \neg P$	**T**, (3), (5), I_9	////

EXAMPLE 2 Show that the following premises are inconsistent.

1 If Jack misses many classes through illness, then he fails high school.
2 If Jack fails high school, then he is uneducated.
3 If Jack reads a lot of books, then he is not uneducated.
4 Jack misses many classes through illness and reads a lot of books.

SOLUTION

E: Jack misses many classes.

S: Jack fails high school.

A: Jack reads a lot of books.

H: Jack is uneducated.

The premises are $E \to S$, $S \to H$, $A \to \neg H$, and $E \land A$.

{1}	(1)	$E \to S$	**P**	
{2}	(2)	$S \to H$	**P**	
{1, 2}	(3)	$E \to H$	**T**, (1), (2), and I_{13}	
{4}	(4)	$A \to \neg H$	**P**	
{4}	(5)	$H \to \neg A$	**T**, (4), E_{18}	
{1, 2, 4}	(6)	$E \to \neg A$	**T**, (3), (5), I_{13}	
{1, 2, 4}	(7)	$\neg E \lor \neg A$	**T**, (6), E_{16}	
{1, 2, 4}	(8)	$\neg(E \land A)$	**T**, (7), E_8	
{9}	(9)	$E \land A$	**P**	
{1, 2, 4, 9}	(10)	$(E \land A) \land \neg(E \land A)$	**T**, (8), (9), I_9	////

Proof by contradiction is sometimes convenient. However, it can always be eliminated and replaced by a conditional proof (**CP**). Observe that

$$P \to (Q \land \neg Q) \Rightarrow \neg P \tag{1}$$

In the proof by contradiction we show

$$H_1, H_2, \ldots, H_m \Rightarrow C$$

by showing

$$H_1, H_2, \ldots, H_m, \neg C \Rightarrow R \wedge \neg R \tag{2}$$

Now (2) can be converted to the following by using rule **CP**

$$H_1, H_2, \ldots, H_m \Rightarrow \neg C \rightarrow (R \wedge \neg R) \tag{3}$$

From (3) and (1) and E_1, we can show

$$H_1, H_2, \ldots, H_m \Rightarrow C$$

which is the required derivation.

1-4.4 Automatic Theorem Proving

In this section we reformulate the rules of inference theory for statement calculus and describe a procedure of derivation which can be conducted mechanically. The formulation given earlier could not be used for this purpose because the construction of a derivation depends heavily upon the skill, experience, and ingenuity of the person to make the right decision at every step.

Let us first examine the shortcomings of procedures used in any derivation of Sec. 1-4.2. Rule **P** permits the introduction of a premise at any point in the derivation, but does not suggest either the premise or the step at which it should be introduced. Rule **T** allows us to introduce any formula which follows from the previous steps. However, there is no definite choice of such a formula, nor is there any guidance for the use of any particular equivalence. Similarly, rule **CP** does not tell anything about the stages at which an antecedent is to be introduced as an assumed premise, nor does it indicate the stage at which it is again incorporated into the conditional. At every step, such decisions are taken from a large number of alternatives, with the ultimate aim of reaching the conclusion. Such a procedure is far from mechanical.

We shall now describe a set of rules and a procedure which allow one to construct each step of derivation in a specified manner without recourse to any ingenuity and finally to arrive at a last step which clearly indicates whether a given conclusion follows from the premises. In this sense, not only is our procedure here mechanical, but it also becomes a full decision process for validity.

The formulation described here is essentially based upon the work of Hao Wang.[1] Our system consisting of 10 rules, an axiom schema, and rules of well-formed sequents and formulas will now be described.

1 Variables: The capital letters $A, B, C, \ldots, P, Q, R, \ldots$ are used as statement variables. They are also used as statement formulas; however, in such cases the context will clearly indicate this usage.

2 Connectives: The connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\rightleftarrows$ appear in the formulas with the order of precedence as given; viz., $\neg$ has the highest precedence, followed by $\wedge$, and so on. The concept of well-formed formula is the same as

[1] Hao Wang, Towards Mechanical Mathematics, *IBM J. Res. Devel.*, **4(1)**: 2–22 (1960).

given in Sec. 1-2.7 with the additional assumption of the precedence and associativity of the connectives needed in order to reduce the number of parentheses appearing in a formula.

3 String of Formulas: A *string of formulas* is defined as follows.

(*a*) Any formula is a string of formulas.
(*b*) If α and β are strings of formulas, then α, β and β, α are strings of formulas.
(*c*) Only those strings which are obtained by steps (*a*) and (*b*) are strings of formulas, with the exception of the empty string which is also a string of formulas.

For our purpose here, the order in which the formulas appear in any string is not important, and so the strings A, B, C; B, C, A; A, C, B; etc., are the same.

4 Sequents: If α and β are strings of formulas, then $\alpha \overset{s}{\rightarrow} \beta$ is called a *sequent* in which α is denoted the antecedent and β the consequent of the sequent.

A sequent $\alpha \overset{s}{\rightarrow} \beta$ is true iff either at least one of the formulas of the antecedent is false or at least one of the formulas of the consequent is true. Thus $A, B, C \overset{s}{\rightarrow} D, E, F$ is true iff $A \wedge B \wedge C \rightarrow D \vee E \vee F$ is true. In this sense, the symbol "$\overset{s}{\rightarrow}$" is a generalization of the connective $\rightarrow$ to strings of formulas.

In the same manner, we shall use the symbol "$\overset{s}{\Rightarrow}$," applied to strings of formulas, as a generalization of the symbol "$\Rightarrow$." Thus $A \Rightarrow B$ means "A implies B" or "$A \rightarrow B$ is a tautology," while $\alpha \overset{s}{\Rightarrow} \beta$ means that $\alpha \overset{s}{\rightarrow} \beta$ is true. For example, $P, Q, R \overset{s}{\Rightarrow} P, N$.

Occasionally we have sequents which have empty strings of formulas as antecedent or as consequent. The empty antecedent is interpreted as the logical constant "*true*" or **T**, and the empty consequent is interpreted as the logical constant "*false*" or **F**.

5 Axiom Schema: If α and β are strings of formulas such that every formula in both α and β is a variable only, then the sequent $\alpha \overset{s}{\rightarrow} \beta$ is an *axiom* iff α and β have at least one variable in common. As an example, $A, B, C \overset{s}{\rightarrow} P, B, R$, where A, B, C, P, and R are variables, is an axiom.

Note that if $\alpha \overset{s}{\rightarrow} \beta$ is an axiom, then $\alpha \overset{s}{\Rightarrow} \beta$.

6 Theorem: The following sequents are theorems of our system.

(*a*) Every axiom is a theorem.
(*b*) If a sequent α is a theorem and a sequent β results from α through the use of one of the 10 rules of the system which are given below, then β is a theorem.
(*c*) Sequents obtained by (*a*) and (*b*) are the only theorems.

Note that we have used α and β temporarily to denote sequents for the purpose of the above description.

7 Rules: The following rules are used to combine formulas within strings by introducing connectives. Corresponding to each of the connectives there are two rules, one for the introduction of the connective in the antecedent and the other for its introduction in the consequent. In the description of these rules, $\alpha, \beta, \gamma, \ldots$ are strings of formulas while X and Y are formulas to which the connectives are applied.

Antecedent Rules

Rule $\neg\Rightarrow$: If $\alpha, \beta \overset{s}{\Rightarrow} X, \gamma$, then $\alpha, \neg X, \beta \overset{s}{\Rightarrow} \gamma$.

Rule $\wedge\Rightarrow$: If $X, Y, \alpha, \beta \overset{s}{\Rightarrow} \gamma$, then $\alpha, X \wedge Y, \beta \overset{s}{\Rightarrow} \gamma$.

Rule $\vee\Rightarrow$: If $X, \alpha, \beta \overset{s}{\Rightarrow} \gamma$ and $Y, \alpha, \beta \overset{s}{\Rightarrow} \gamma$, then $\alpha, X \vee Y, \beta \overset{s}{\Rightarrow} \gamma$.

Rule $\rightarrow\Rightarrow$: If $Y, \alpha, \beta \overset{s}{\Rightarrow} \gamma$ and $\alpha, \beta \overset{s}{\Rightarrow} X, \gamma$, then $\alpha, X \rightarrow Y, \beta \overset{s}{\Rightarrow} \gamma$.

Rule $\rightleftarrows\Rightarrow$: If $X, Y, \alpha, \beta \overset{s}{\Rightarrow} \gamma$ and $\alpha, \beta \overset{s}{\Rightarrow} X, Y, \gamma$, then $\alpha, X \rightleftarrows Y, \beta \overset{s}{\Rightarrow} \gamma$.

Consequent Rules

Rule $\Rightarrow\neg$: If $X, \alpha \overset{s}{\Rightarrow} \beta, \gamma$, then $\alpha \overset{s}{\Rightarrow} \beta, \neg X, \gamma$.

Rule $\Rightarrow\wedge$: If $\alpha \overset{s}{\Rightarrow} X, \beta, \gamma$ and $\alpha \overset{s}{\Rightarrow} Y, \beta, \gamma$, then $\alpha \overset{s}{\Rightarrow} \beta, X \wedge Y, \gamma$.

Rule $\Rightarrow\vee$: If $\alpha \overset{s}{\Rightarrow} X, Y, \beta, \gamma$, then $\alpha \overset{s}{\Rightarrow} \beta, X \vee Y, \gamma$.

Rule $\Rightarrow\rightarrow$: If $X, \alpha \overset{s}{\Rightarrow} Y, \beta, \gamma$, then $\alpha \overset{s}{\Rightarrow} \beta, X \rightarrow Y, \gamma$.

Rule $\Rightarrow\rightleftarrows$: If $X, \alpha \overset{s}{\Rightarrow} Y, \beta, \gamma$ and $Y, \alpha \overset{s}{\Rightarrow} X, \beta, \gamma$, then $\alpha \overset{s}{\Rightarrow} \beta, X \rightleftarrows Y, \gamma$.

Note that the order in which the formulas and strings of formulas appear in a string in any of the above rules is unimportant. However, some kind of ordering is necessary in writing down the strings of formulas when an algorithm is written for computer implementation of the procedure described here. We have used a certain order in writing these rules with this point in mind. A computer algorithm is given in Sec. 2-7.

The system described here is equivalent to the one described earlier except that the procedures and techniques of derivation are different. This difference does not affect the validity of an argument. Some of the rules just given correspond to certain equivalences given earlier (see Problem 9 in Exercises 1-2).

The description of our system is complete. We shall now describe the procedure used in practice.

In the method of derivation given in Sec. 1-4, we showed that a conclusion C follows from the premises $H_1, H_2, \ldots, H_m$ by constructing a derivation whose last step was C, and $H_1, H_2, \ldots, H_m$ were introduced at various stages by using rule **P**. This method essentially means showing

$$H_1, H_2, \ldots, H_m \Rightarrow C \tag{1}$$

Another way of stating (1) is

$$H_1 \rightarrow (H_2 \rightarrow (H_3 \cdots (H_m \rightarrow C) \cdots)) \tag{2}$$

is a tautology (see Sec. 1-2.11).

Our new formulation is premise-free, so that in order to show that C follows from $H_1, H_2, \ldots, H_m$, we establish that

$$\overset{s}{\rightarrow} H_1 \rightarrow (H_2 \rightarrow (H_3 \cdots (H_m \rightarrow C) \cdots)) \tag{3}$$

is a theorem. We must show that

$$\overset{s}{\Rightarrow} H_1 \rightarrow (H_2 \rightarrow (H_3 \cdots (H_m \rightarrow C) \cdots)) \tag{4}$$

Our procedure involves showing (3) to be a theorem. For this purpose, we first assume (4) and then show that this assumption is or is not justified. This task is accomplished by working backward from (4), using the rules, and showing that (4) holds if some simpler sequent is a theorem. (By a simpler sequent we mean a sequent in which some connective is eliminated in one of the formulas appearing in the antecedent or the consequent.) We continue working backward until we arrive at the simplest possible sequents, i.e., those which do not have any connectives. If these sequents are axioms, then we have justified our assumption of (4). If at least one of the simplest sequents is not an axiom, then the assumption of (4) is not justified and C does not follow from $H_1, H_2, \ldots, H_m$. In the case when C follows from $H_1, H_2, \ldots, H_m$, the derivation of (4) is easily constructed by simply working through the same steps, starting from the axioms obtained. We now demonstrate this procedure by means of examples.

EXAMPLE 1 Show that $P \vee Q$ follows from P.

SOLUTION We need to show that

$$
\begin{array}{lll}
(1) & \overset{s}{\Rightarrow} P \rightarrow (P \vee Q) & \\
(1) & \text{if } (2)\ P \overset{s}{\Rightarrow} P \vee Q & (\Rightarrow\rightarrow) \\
(2) & \text{if } (3)\ P \overset{s}{\Rightarrow} P, Q & (\Rightarrow\vee)
\end{array}
$$

We first eliminate the connective $\rightarrow$ in (1). Using the rule $\Rightarrow\rightarrow$ we have "if $P \overset{s}{\Rightarrow} P \vee Q$ then $\overset{s}{\Rightarrow} P \rightarrow (P \vee Q)$." Here we have named $P \overset{s}{\Rightarrow} P \vee Q$ by (2). Each line of derivation thus introduces the name as well as gives a rule. Note also that "(1) if (2)" means "if (2) then (1)." The chain of arguments is then given by (1) holds if (2), and (2) holds if (3). Finally (3) is a theorem, because it is an axiom. The actual derivation is simply a reversal of these steps in which (3) is an axiom that leads to $\overset{s}{\Rightarrow} P \rightarrow (P \vee Q)$ as shown.

$$
\begin{array}{lll}
(a) & P \overset{s}{\Rightarrow} P, Q & \text{Axiom} \\
(b) & P \overset{s}{\Rightarrow} P \vee Q & \text{Rule } (\Rightarrow\vee),\ (a) \\
(c) & \overset{s}{\Rightarrow} P \rightarrow (P \vee Q) & \text{Rule } (\Rightarrow\rightarrow),\ (b)
\end{array}
$$

////

EXAMPLE 2 Show that $\overset{s}{\Rightarrow} (\neg Q \wedge (P \rightarrow Q)) \rightarrow \neg P$.

SOLUTION

$$
\begin{array}{lll}
(1) & \overset{s}{\Rightarrow} (\neg Q \wedge (P \rightarrow Q)) \rightarrow \neg P & \\
(1) & \text{if } (2)\ \neg Q \wedge (P \rightarrow Q) \overset{s}{\Rightarrow} \neg P & (\Rightarrow\rightarrow) \\
(2) & \text{if } (3)\ \neg Q, P \rightarrow Q \overset{s}{\Rightarrow} \neg P & (\wedge\Rightarrow) \\
(3) & \text{if } (4)\ P \rightarrow Q \overset{s}{\Rightarrow} \neg P, Q & (\neg\Rightarrow) \\
(4) & \text{if } (5)\ Q \overset{s}{\Rightarrow} \neg P, Q \text{ and } (6) \overset{s}{\Rightarrow} P, \neg P, Q & (\rightarrow\Rightarrow) \\
(5) & \text{if } (7)\ P, Q \overset{s}{\Rightarrow} Q & (\Rightarrow\neg) \\
(6) & \text{if } (8)\ P \overset{s}{\Rightarrow} P, Q & (\Rightarrow\neg)
\end{array}
$$

Now (7) and (8) are axioms, hence the theorem (1) follows. We omit the derivation, which is easily obtained by starting with the axioms (7) and (8) and retracing the steps. ////

Note that in the solution of Example 2 in the second step we have two alternatives available, viz., to eliminate either $\wedge$ in the antecedent of (2) or $\neg$ in the consequent. It is immaterial which elimination is carried out first. When the process is carried out on a computer, we shall adopt the convention that for any number of possible connectives that are available for elimination at a stage, the one on the left is eliminated first. This procedure was actually followed in the solution. We shall not follow this convention when the solution is obtained by hand.

EXAMPLE 3 Does P follow from $P \vee Q$?

SOLUTION We investigate whether $\overset{s}{\rightarrow} (P \vee Q) \rightarrow P$ is a theorem. Assume (1) $\overset{s}{\Rightarrow} (P \vee Q) \rightarrow P$.

(1)	if (2) $P \vee Q \overset{s}{\Rightarrow} P$	$(\Rightarrow\rightarrow)$
(2)	if (3) $P \overset{s}{\Rightarrow} P$ and (4) $Q \overset{s}{\Rightarrow} P$	$(\vee\Rightarrow)$

Note that (3) is an axiom, but (4) is not. Hence P does not follow from $P \vee Q$. ////

In some cases the derivation is longer if this procedure is used. The reason is that for every connective appearing there is at least one step introduced. In some steps a branching appears, and then we have to pursue two steps. We shall now consider an example for which the derivation was given earlier in Sec. 1-4.1.

EXAMPLE 4 Show that $S \vee R$ is tautologically implied by $(P \vee Q) \wedge (P \rightarrow R) \wedge (Q \rightarrow S)$ (see Example 3, Sec. 1-4.1).

SOLUTION To show

(1)	$\overset{s}{\Rightarrow} ((P \vee Q) \wedge (P \rightarrow R) \wedge (Q \rightarrow S)) \rightarrow (S \vee R)$	
(1)	if (2) $(P \vee Q) \wedge (P \rightarrow R) \wedge (Q \rightarrow S) \overset{s}{\Rightarrow} (S \vee R)$	$(\Rightarrow\rightarrow)$
(2)	if (3) $(P \vee Q) \wedge (P \rightarrow R) \wedge (Q \rightarrow S) \overset{s}{\Rightarrow} S, R$	$(\Rightarrow\vee)$
(3)	if (4) $(P \vee Q), (P \rightarrow R), (Q \rightarrow S) \overset{s}{\Rightarrow} S, R$	$(\wedge\Rightarrow$ twice)
(4)	if (5) $P, P \rightarrow R, Q \rightarrow S \overset{s}{\Rightarrow} S, R$	
	and (6) $Q, P \rightarrow R, Q \rightarrow S \overset{s}{\Rightarrow} S, R$	$(\vee\Rightarrow)$
(5)	if (7) $P, R, Q \rightarrow S \overset{s}{\Rightarrow} S, R$ and (8) $P, Q \rightarrow S \overset{s}{\Rightarrow} P, S, R$	$(\rightarrow\Rightarrow)$
(7)	if (9) $P, R, S \overset{s}{\Rightarrow} S, R$ and (10) $P, R \overset{s}{\Rightarrow} S, R, Q$	$(\rightarrow\Rightarrow)$
(8)	if (11) $P, S \overset{s}{\Rightarrow} P, S, R$ and(12) $P \overset{s}{\Rightarrow} P, S, R, Q$	$(\rightarrow\Rightarrow)$
(6)	if (13) $Q, R, Q \rightarrow S \overset{s}{\Rightarrow} S, R$ and (14) $Q, Q \rightarrow S \overset{s}{\Rightarrow} S, R, P$	$(\rightarrow\Rightarrow)$
(13)	if (15) $Q, R, S \overset{s}{\Rightarrow} S, R$ and (16) $Q, R \overset{s}{\Rightarrow} S, R, Q$	$(\rightarrow\Rightarrow)$
(14)	if (17) $Q, S \overset{s}{\Rightarrow} S, R, P$ and (18) $Q \overset{s}{\Rightarrow} S, R, P, Q$	$(\rightarrow\Rightarrow)$

Now, (9) to (12) and (15) to (18) are all axioms; therefore the result follows. The nesting of steps shows how the branching occurs. ////

EXERCISES 1-4

1 Show the validity of the following arguments, for which the premises are given on the left and the conclusion on the right.

(*a*) $\neg(P \wedge \neg Q), \neg Q \vee R, \neg R$	$\neg P$
(*b*) $(A \to B) \wedge (A \to C), \neg(B \wedge C), D \vee A$	D
(*c*) $\neg J \to (M \vee N), (H \vee G) \to \neg J, H \vee G$	$M \vee N$
(*d*) $P \to Q, (\neg Q \vee R) \wedge \neg R, \neg(\neg P \wedge S)$	$\neg S$
(*e*) $(P \wedge Q) \to R, \neg R \vee S, \neg S$	$\neg P \vee \neg Q$
(*f*) $P \to Q, Q \to \neg R, R, P \vee (J \wedge S)$	$J \wedge S$
(*g*) $B \wedge C, (B \rightleftarrows C) \to (H \vee G)$	$G \vee H$
(*h*) $(P \to Q) \to R, P \wedge S, Q \wedge \mathbf{T}$	R

2 Derive the following, using rule **CP** if necessary.
(*a*) $\neg P \vee Q, \neg Q \vee R, R \to S \Rightarrow P \to S$
(*b*) $P, P \to (Q \to (R \wedge S)) \Rightarrow Q \to S$
(*c*) $P \to Q \Rightarrow P \to (P \wedge Q)$
(*d*) $(P \vee Q) \to R \Rightarrow (P \wedge Q) \to R$
(*e*) $P \to (Q \to R), Q \to (R \to S) \Rightarrow P \to (Q \to S)$

3 Prove $\neg P \wedge (P \vee Q) \Rightarrow Q$, using E_{12}, E_6, E_3, and I_2 only.

4 Show that the following sets of premises are inconsistent.
(*a*) $P \to Q, P \to R, Q \to \neg R, P$
(*b*) $A \to (B \to C), D \to (B \wedge \neg C), A \wedge D$
Hence show that $P \to Q, P \to R, Q \to \neg R, P \Rightarrow M$ and

$$A \to (B \to C), D \to (B \wedge \neg C), A \wedge D \Rightarrow P$$

5 Show the following (use indirect method if needed).
(*a*) $(R \to \neg Q), R \vee S, S \to \neg Q, P \to Q \Rightarrow \neg P$
(*b*) $S \to \neg Q, S \vee R, \neg R, \neg R \rightleftarrows Q \Rightarrow \neg P$
(*c*) $\neg(P \to Q) \to \neg(R \vee S), ((Q \to P) \vee \neg R), R \Rightarrow P \rightleftarrows Q$

6 Show the following, using the system given in Sec. 1-4.4.
(*a*) $P \Rightarrow (\neg P \to Q)$
(*b*) $P \wedge \neg P \wedge Q \Rightarrow R$
(*c*) $R \Rightarrow P \vee \neg P \vee Q$
(*d*) $P, \neg P \vee (P \wedge Q) \Rightarrow Q$
(*e*) $\neg(P \wedge Q) \Rightarrow \neg P \vee \neg Q$

1-5 THE PREDICATE CALCULUS

So far our discussion of symbolic logic has been limited to the consideration of statements and statement formulas. The inference theory was also restricted in the sense that the premises and conclusions were all statements. The symbols $P, Q, R, \ldots, P_1, Q_1, \ldots$ were used for statements or statement variables. The statements were taken as basic units of statement calculus, and no analysis of any atomic statement was admitted. Only compound formulas were analyzed, and this analysis was done by studying the forms of the compound formulas, i.e., the connections between the constituent atomic statements. It was not possible to express the fact that any two atomic statements have some features in common. In order to investigate questions of this nature, we introduce the concept of a predicate in an atomic statement. The logic based upon the analysis of predicates in any statement is called *predicate logic*.

1-5.1 Predicates

Let us first consider the two statements

John is a bachelor.

Smith is a bachelor.

Obviously, if we express these statements by symbols, we require two different symbols to denote them. Such symbols do not reveal the common features of these two statements; viz., both are statements about two different individuals who are bachelors. If we introduce some symbol to denote "is a bachelor" and a method to join it with symbols denoting the names of individuals, then we will have a symbolism to denote statements about any individual's being a bachelor. The part "is a bachelor" is called a *predicate*. Another consideration which leads to some similar device for the representation of statements is suggested by the following argument.

All human beings are mortal.

John is a human being.

Therefore, John is a mortal.

Such a conclusion seems intuitively true. However, it does not follow from the inference theory of the statement calculus developed earlier. The reason for this deficiency is the fact that the statement "All human beings are mortal" cannot be analyzed to say anything about an individual. If we could separate the part "are mortal" from the part "All human beings," then it might be possible to consider any particular human being.

We shall symbolize a predicate by a capital letter and the names of individuals or objects in general by small letters. We shall soon see that using capital letters to symbolize statements as well as predicates will not lead to any confusion. Every predicate describes something about one or more objects (the word "object" is being used in a very general sense to include individuals as well). Therefore, a statement could be written symbolically in terms of the predicate letter followed by the name or names of the objects to which the predicate is applied.

We again consider the statements

1 John is a bachelor.
2 Smith is a bachelor.

Denote the predicate "is a bachelor" symbolically by the predicate letter B, "John" by j, and "Smith" by s. Then Statements (1) and (2) can be written as $B(j)$ and $B(s)$ respectively. In general, any statement of the type "p is Q" where Q is a predicate and p is the subject can be denoted by $Q(p)$.

A statement which is expressed by using a predicate letter must have at least one name of an object associated with the predicate. When an appropriate number of names are associated with a predicate, then we get a statement. Using a capital letter to denote a predicate may not indicate the appropriate number of names associated with it. Normally, this number is clear from the context or from the notation being used. This numbering can also be accomplished by attaching a superscript to a predicate letter, indicating the number of names that are to be appended to the letter. A predicate requiring m $(m > 0)$ names is called an m-place predicate. For example, B in (1) and (2) is a 1-place predicate. Another example is that "L: is less than" is a 2-place predicate. In order to extend our definition to $m = 0$, we shall call a statement a 0-place predicate because no names are associated with a statement.

Let R denote the predicate "is red" and let p denote "This painting." Then the statement

3 This painting is red.

can be symbolized by $R(p)$. Further, the connectives described earlier can now be used to form compound statements such as "John is a bachelor, and this painting is red," which can be written as $B(j) \wedge R(p)$. Other connectives can also be used to form statements such as

$$B(j) \rightarrow R(p) \qquad \neg R(p) \qquad B(j) \vee R(p) \qquad \text{etc.}$$

Consider, now, statements involving the names of two objects, such as

4 Jack is taller than Jill.
5 Canada is to the north of the United States.

The predicates "is taller than" and "is to the north of" are 2-place predicates because names of two objects are needed to complete a statement involving these predicates. If the letter G symbolizes "is taller than," j_1 denotes "Jack," and j_2 denotes "Jill," then Statement (4) can be translated as $G(j_1, j_2)$. Note that the order in which the names appear in the statement as well as in the predicate is important. Similarly, if N denotes the predicate "is to the north of," c: Canada, and s: United States, then (5) is symbolized as $N(c, s)$. Obviously, $N(s, c)$ is the statement "The United States is to the north of Canada."

Examples of 3-place predicates and 4-place predicates are:

6 Susan sits between Ralph and Bill.
7 Green and Miller played bridge against Johnston and Smith.

In general, an n-place predicate requires n names of objects to be inserted in fixed positions in order to obtain a statement. The position of these names is important. If S is an n-place predicate letter and $a_1, a_2, \ldots, a_n$ are the names of objects, then $S(a_1, a_2, \ldots, a_n)$ is a statement. If we use this convention, every predicate symbol is followed by an appropriate number of letters, which are the names of objects, enclosed in parentheses and separated by commas. Occasionally, the parentheses and the commas are dropped. The definition does not require that the names be chosen from any fixed set. For example, if B denotes the predicate "is a bachelor" and t denotes "This table," then $B(t)$ symbolizes "This table is a bachelor." In our everyday language, the only admissible name in this context would be that of an individual. However, such restrictions are not necessary according to the rules given above. We show a method of imposing such restrictions in Sec. 1-5.5.

1-5.2 The Statement Function, Variables, and Quantifiers

Let H be the predicate "is a mortal," b the name "Mr. Brown," c "Canada," and s "A shirt." Then $H(b)$, $H(c)$, and $H(s)$ all denote statements. In fact, these statements have a common form. If we write $H(x)$ for "x is mortal," then $H(b), H(c), H(s)$, and others having the same form can be obtained from $H(x)$ by replacing x by an appropriate name. Note that $H(x)$ is not a statement, but it results in a statement when x is replaced by the name of an object. The letter x used here is a placeholder. From now on we shall use small letters as individual or object variables as well as names of objects.

A *simple statement function* of one variable is defined to be an expression consisting of a predicate symbol and an individual variable. Such a statement function becomes a statement when the variable is replaced by the name of any object. The statement resulting from a replacement is called a *substitution instance* of the statement function and is a formula of statement calculus.

The word "simple" in the above definition distinguishes the simple statement function from those statement functions which are obtained from combining one or more simple statement functions and the logical connectives. For example, if we let $M(x)$ be "x is a man" and $H(x)$ be "x is a mortal," then we can form *compound statement functions* such as

$$M(x) \land H(x) \qquad M(x) \to H(x) \qquad \neg H(x) \qquad M(x) \lor \neg H(x) \qquad \text{etc.}$$

An extension of this idea to the statement functions of two or more variables is straightforward. Consider, for example, the statement function of two variables:

1 $G(x, y)$: x is taller than y.

If both x and y are replaced by the names of objects, we get a statement. If m represents Mr. Miller and f Mr. Fox, then we have

$G(m, f)$: Mr. Miller is taller than Mr. Fox.

and

$G(f, m)$: Mr. Fox is taller than Mr. Miller.

It is possible to form statement functions of two variables by using statement functions of one variable. For example, given

$M(x)$: x is a man.

$H(y)$: y is a mortal.

then we may write

$M(x) \land H(y)$: x is a man and y is a mortal.

It is not possible, however, to write every statement function of two variables using statement functions of one variable.

One way of obtaining statements from any statement function is to replace the variables by the names of objects. There is another way in which statements can be obtained. In order to understand this alternative method, we first consider some familiar equations in elementary algebra.

2 $x + 2 = 5$
3 $x^2 + 1 = 0$
4 $(x - 1) * (x - \frac{1}{2}) = 0$
5 $x^2 - 1 = (x - 1) * (x + 1)$

In algebra, it is conventional to assume that the variable x is to be replaced by numbers (real, complex, rational, integer, etc.). In the above equations, we would not normally consider substituting for x the name of a person or object instead of numbers. We may state this idea by saying that the *universe* of the variable x is the set of real numbers or complex numbers or integers, etc. The restriction depends upon the problem under consideration. For example, we may be interested in only the real solution or the positive solution in a particular case. In Statement (2), if x is replaced by a real number, we get a statement. The resulting statement is true when 3 is substituted for x, while, for every other substitution, the resulting statement is false. In (3) there is no real number which, when substituted for x, gives a true statement. If, however, the universe of x includes complex numbers as well, then we find that there are two substitution instances which give true statements. In (4), if the universe of x is assumed to be integers, then there is only one number which produces a true statement when substituted. The situation is slightly different in (5) in the sense that if any number is substituted for x, then the resulting statement is true. Therefore, we may say that

6 For any number x, $x^2 - 1 = (x - 1) * (x + 1)$.

Note that (6) is a statement and not a statement function even though a variable x appears in it. In fact, the addition of the phrase "For any number x," has changed the situation. The letter x, as used in (6), is different from the variable x used in Statements (2) to (5). In (6) the variable x need not be replaced by any name to obtain a statement. In mathematics this distinction is often not made. Occasionally when a statement involves an equality, a distinction is made by using the symbol $\equiv$ instead of the equality sign to show that it is a statement. In this case, (6) would be written as $x^2 - 1 \equiv (x - 1) * (x + 1)$. A similar situation occurs when a statement function does not involve an equality, and a distinction is necessary in logic between these two different uses of the variables.

Let us first consider the following statements. Each one is a statement about all individuals or objects belonging to a certain set.

7 All men are mortal.
8 Every apple is red.
9 Any integer is either positive or negative.

Let us paraphrase these in the following manner.

7a For all x, if x is a man, then x is a mortal.
8a For all x, if x is an apple, then x is red.
9a For all x, if x is a integer, then x is either positive or negative.

We have already shown how statement functions such as "x is a man," "x is an apple," or "x is red" can be written by using predicate symbols. If we introduce a symbol to denote the phrase "For all x," then it would be possible to symbolize Statements (7*a*), (8*a*), and (9*a*).

We symbolize "For all x" by the symbol "$(\forall x)$" or by "(x)" with an understanding that this symbol be placed before the statement function to which this phrase is applied. Using

$M(x)$: x is man. $H(x)$: x is a mortal.

$A(x)$: x is an apple. $R(x)$: x is red.

$N(x)$: x is an integer. $P(x)$: x is either positive or negative.

we write (7*a*), (8*a*), and (9*a*) as

7b $(x)(M(x) \to H(x))$
8b $(x)(A(x) \to R(x))$
9b $(x)(N(x) \to P(x))$

Sometimes $(x)(M(x) \to H(x))$ is also written as $(\forall x)(M(x) \to H(x))$. The symbols (x) or $(\forall x)$ are called *universal quantifiers*. Strictly speaking, the quantification symbol is "( )" or "($\forall$)," and it contains the variable which is to be quantified. It is now possible for us to quantify any statement function of one variable to obtain a statement. Thus $(x)M(x)$ is a statement which can be translated as

10 For all x, x is a man.
10a For every x, x is a man.
10b Everything is a man.

In order to determine the truth values of any one of these statements involving a universal quantifier, one may be tempted to consider the truth values of the statement function which is quantified. This method is not possible for two reasons. First, statement functions do not have truth values. When the variables are replaced by the names of objects, we get statements which have a truth value. Second, in most cases there is an infinite number of statements that can be produced by such substitutions.

Note that the particular variable appearing in the statements involving a quantifier is not important because the statements remain unchanged if x is replaced by y throughout. Thus the statements $(x)(M(x) \to H(x))$ and $(y)(M(y) \to H(y))$ are equivalent.

Sometimes it is necessary to use more than one universal quantifier in a statement. For example consider

$G(x, y)$: x is taller than y.

We can state that "For any x and any y, if x is taller than y, then y is not taller than x" or "For any x and y, if x is taller than y, then it is not true that y is taller than x." This statement can now be symbolized as

$$(x)(y)(G(x, y) \to \neg G(y, x))$$

The universal quantifier was used to translate expressions such as "for all," "every," and "for any." Another quantifier will now be introduced to symbolize expressions such as "for some," "there is at least one," or "there exists some" (note that "some" is used in the sense of "at least one").

Consider the following statements:

11 There exists a man.
12 Some men are clever.
13 Some real numbers are rational.

The first statement can be expressed in various ways, two such ways being

11a There exists an x such that x is a man.
11b There is at least one x such that x is a man.

Similarly, (12) can be written as

12a There exists an x such that x is a man and x is clever.
12b There exists at least one x such that x is a man and x is clever.

Such a rephrasing allows us to introduce the symbol "$(\exists x)$," called the *existential quantifier,* which symbolizes expressions such as "there is at least one x such that" or "there exists an x such that" or "for some x." Writing

$M(x)$: x is a man.
$C(x)$: x is clever.
$R_1(x)$: x is a real number.
$R_2(x)$: x is rational.

and using the existential quantifier, we can write the Statements (11) to (13) as

11c $(\exists x)(M(x))$
12c $(\exists x)(M(x) \land C(x))$
13c $(\exists x)(R_1(x) \land R_2(x))$

It may be noted that a conjunction has been used in writing the statements of the form "Some A are B," while a conditional was used in writing statements of the form "All A are B." To a beginner this usage may appear confusing. We show in Sec. 1-5.5 why these connectives are the right ones to be used in these cases.

1-5.3 Predicate Formulas

Recall that capital letters were first used to denote some definite statements. Subsequently they were used as placeholders for the statements, and, in this sense, they were called statement variables. These statement variables were also considered as special cases of statement formulas.

In Secs. 1-5.1 and 1-5.2 the capital letters were introduced as definite predicates. It was suggested that a superscript n be used along with the capital letters in order to indicate that the capital letter is used as an n-place predicate. However, this notation was not necessary because an n-place predicate symbol must be followed by n object variables. Such variables are called *object* or *individual variables* and are denoted by lowercase letters. When used as an n-place predicate, the capital letter is followed by n individual variables which are enclosed in parentheses and separated by commas. For example, $P(x_1, x_2, \ldots, x_n)$ denotes an n-place predicate formula in which the letter P is an n-place predicate and

$x_1, x_2, \ldots, x_n$ are individual variables. In general, $P(x_1, x_2, \ldots, x_n)$ will be called an *atomic formula* of predicate calculus. It may be noted that our symbolism includes the atomic formulas of the statement calculus as special cases ($n = 0$). The following are some examples of atomic formulas.

$R \qquad Q(x) \qquad P(x, y) \qquad A(x, y, z) \qquad P(a, y)$ and $A(x, a, z)$

A well-formed formula of predicate calculus is obtained by using the following rules.

1 An atomic formula is a well-formed formula.

2 If A is a well-formed formula, then $\neg A$ is a well-formed formula.

3 If A and B are well-formed formulas, then $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, and $(A \rightleftarrows B)$ are also well-formed formulas.

4 If A is a well-formed formula and x is any variable, then $(x)A$ and $(\exists x)A$ are well-formed formulas.

5 Only those formulas obtained by using rules (1) to (4) are well-formed formulas.

Since we will be concerned with only well-formed formulas, we shall use the term "formula" for "well-formed formula." We shall follow the same conventions regarding the use of parentheses as was done in the case of statement formulas.

1-5.4 Free and Bound Variables

Given a formula containing a part of the form $(x)P(x)$ or $(\exists x)P(x)$, such a part is called an x-bound part of the formula. Any occurrence of x in an x-bound part of a formula is called a *bound occurrence* of x, while any occurrence of x or of any variable that is not a bound occurrence is called a *free occurrence*. Further, the formula $P(x)$ either in $(x)P(x)$ or in $(\exists x)P(x)$ is described as the *scope* of the quantifier. In other words, the scope of a quantifier is the formula immediately following the quantifier. If the scope is an atomic formula, then no parentheses are used to enclose the formula; otherwise parentheses are needed. As illustrations, consider the following formulas:

$$(x)P(x, y) \tag{1}$$

$$(x)(P(x) \rightarrow Q(x)) \tag{2}$$

$$(x)(P(x) \rightarrow (\exists y)R(x, y)) \tag{3}$$

$$(x)(P(x) \rightarrow R(x)) \vee (x)(P(x) \rightarrow Q(x)) \tag{4}$$

$$(\exists x)(P(x) \wedge Q(x)) \tag{5}$$

$$(\exists x)P(x) \wedge Q(x) \tag{6}$$

In (1), $P(x, y)$ is the scope of the quantifier, and both occurrences of x are bound occurrences, while the occurrence of y is a free occurrence. In (2), the scope of the universal quantifier is $P(x) \rightarrow Q(x)$, and all occurrences of x are bound. In (3), the scope of (x) is $P(x) \rightarrow (\exists y)R(x, y)$, while the scope of $(\exists y)$ is $R(x, y)$. All occurrences of both x and y are bound occurrences. In (4), the scope of the first quantifier is $P(x) \rightarrow R(x)$, and the scope of the second is

$P(x) \rightarrow Q(x)$. All occurrences of x are bound occurrences. In (5), the scope of $(\exists x)$ is $P(x) \wedge Q(x)$. However, in (6) the scope of $(\exists x)$ is $P(x)$, and the last occurrence of x in $Q(x)$ is free.

It is useful to note that in the bound occurrence of a variable, the letter which is used to represent the variable is not important. In fact, any other letter can be used as a variable without affecting the formula, provided that the new letter is not used elsewhere in the formula. Thus the formulas

$$(x)P(x, y) \qquad \text{and} \qquad (z)P(z, y)$$

are the same. Further, the bound occurrence of a variable cannot be substituted by a constant; only a free occurrence of a variable can be. For example, $(x)P(x) \wedge Q(a)$ is a substitution instance of $(x)P(x) \wedge Q(y)$. In fact, $(x)P(x) \wedge Q(a)$ can be expressed in English as "Every x has the property P, and a has the property Q." A change of variables in the bound occurrence is not a substitution instance. Sometimes it is useful to change the variables in order to avoid confusion. In (6), it is better to write $(y)P(y) \wedge Q(x)$ instead of $(x)P(x) \wedge Q(x)$, so as to separate the free and bound occurrences of variables. Occasionally, one may come across a formula of the type $(x)P(y)$ in which the occurrence of y is free and the scope of (x) does not contain an x; in such a case, we have a vacuous use of (x). Finally, it may be mentioned that in a statement every occurrence of a variable must be bound, and no variable should have a free occurrence. In the case where a free variable occurs in a formula, then we have a statement function.

EXAMPLE 1 Let

$$P(x): x \text{ is a person.}$$
$$F(x, y): x \text{ is the father of } y.$$
$$M(x, y): x \text{ is the mother of } y.$$

Write the predicate "x is the father of the mother of y."

SOLUTION In order to symbolize the predicate, we name a person called z as the mother of y. Obviously we want to say that x is the father of z and z the mother of y. It is assumed that such a person z exists. We symbolize the predicate as

$$(\exists z)(P(z) \wedge F(x, z) \wedge M(z, y)) \qquad ////$$

EXAMPLE 2 Symbolize the expression "All the world loves a lover."

SOLUTION First note that the quotation really means that everybody loves a lover. Now let

$$P(x): x \text{ is a person.}$$
$$L(x): x \text{ is a lover.}$$
$$R(x, y): x \text{ loves } y.$$

The required expression is

$$(x)(P(x) \rightarrow (y)(P(y) \wedge L(y) \rightarrow R(x, y))) \qquad ////$$

1-5.5 The Universe of Discourse

Example 2 in Sec. 1-5.4 shows that the process of symbolizing a statement in predicate calculus can be quite complicated. However, some simplification can be introduced by limiting the class of individuals or objects under consideration. This limitation means that the variables which are quantified stand for only those objects which are members of a particular set or class. Such a restricted class is called the *universe of discourse* or the *domain* of individuals or simply the *universe*. If the discussion refers to human beings only, then the universe of discourse is the class of human beings. In elementary algebra or number theory, the universe of discourse could be numbers (real, complex, rational, etc.).

EXAMPLE 1 Symbolize the statement "All men are giants."

SOLUTION Using

$$G(x): x \text{ is a giant.}$$
$$M(x): x \text{ is a man.}$$

the given statement can be symbolized as $(x)(M(x) \rightarrow G(x))$. However, if we restrict the variable x to the universe which is the class of men, then the statement is

$$(x)G(x) \qquad ////$$

EXAMPLE 2 Consider the statement "Given any positive integer, there is a greater positive integer." Symbolize this statement with and without using the set of positive integers as the universe of discourse.

SOLUTION Let the variables x and y be restricted to the set of positive integers. Then the above statement can be paraphrased as follows: For all x, there exists a y such that y is greater than x. If $G(x, y)$ is "x is greater than y," then the given statement is $(x)(\exists y)G(y, x)$. If we do not impose the restriction on the universe of discourse and if we write $P(x)$ for "x is a positive integer," then we can symbolize the given statement as $(x)(P(x) \rightarrow (\exists y)(P(y) \wedge G(y, x)))$. ////

The universe of discourse, if any, must be explicitly stated, because the truth value of a statement depends upon it. For instance, consider the predicate

$$Q(x): x \text{ is less than 5.}$$

and the statements $(x)Q(x)$ and $(\exists x)Q(x)$. If the universe of discourse is given by the sets

1 $\{-1, 0, 1, 2, 4\}$
2 $\{3, -2, 7, 8, -2\}$
3 $\{15, 20, 24\}$

then $(x)Q(x)$ is true for the universe of discourse (1) and false for (2) and (3). The statement $(\exists x)Q(x)$ is true for both (1) and (2), but false for (3).

It may be noted that there are two ways of obtaining a 0-place predicate from an n-place predicate. The first way is to substitute names of objects from

of the predicate calculus with the understanding that the atomic variables there stand for prime predicate formulas.

The valid formulas obtained in this manner do not exhaust all possible valid formulas. There are several other valid formulas, particularly those involving quantifiers, which are useful. Such valid formulas are obtained in Sec. 1-6.4 by using the inference theory of predicate logic.

1-6.2 Some Valid Formulas over Finite Universes

In this and in the following section we denote predicate formulas by capital letters such as A, B, C, $\cdots$ followed by object variables x, y, $\cdots$. Thus $A(x)$, $A(x, y)$, $B(y)$, and $C(x, y, z)$ are examples of predicate formulas. Some clarification is necessary at this stage. In the formula $A(x)$, we wish to say that A is a predicate formula in which x is one of the free variables. This variable x is of interest to us, and we want to emphasize the dependence of A on it. For example, we may write $B(x)$ for $(y)P(y) \vee Q(x)$.

If in a formula $A(x)$ we replace each free occurrence of the variable x by another variable y, then we say that y is *substituted* for x in the formula, and the resulting formula is denoted by $A(y)$. For such a substitution, the formula $A(x)$ must be free for y. A formula $A(x)$ is said to be *free for* y if no free occurrence of x is in the scope of the quantifiers (y) or $(\exists y)$. If $A(x)$ is not free for y, then it is necessary to change the variable y, appearing as a bound variable, to another variable before substituting y for x. If y is to be substituted, then it is usually a good idea to make all the bound variables different from y. The following examples show what $A(y)$ is for a given $A(x)$.

$A(x)$	$A(y)$
$P(x,y) \wedge (\exists y)Q(y)$	$P(y,y) \wedge (\exists y)Q(y)$ or $P(y,y) \wedge (\exists z)Q(z)$
$(S(x) \wedge S(y)) \vee (x)R(x)$	$(S(y) \wedge S(y)) \vee (x)R(x)$ or $(S(y) \wedge S(y)) \vee (z)R(z)$

The following formulas are not free for y.

$$P(x, y) \wedge (y)Q(x, y) \qquad (y)(S(y) \rightarrow S(x))$$

In order to substitute y in place of the variable x in these formulas, it is necessary to first make them free for y as follows:

$A(x)$	$A(y)$
$P(x,y) \wedge (z)Q(x,z)$	$P(y,y) \wedge (z)Q(y,z)$
$(z)(S(z) \rightarrow S(x))$	$(z)(S(z) \rightarrow S(y))$

If the universe of discourse is a finite set, then all possible substitutions of the object variables can be enumerated. However, it is not possible to enumerate all possible substitutions if the universe of discourse is infinite. We shall now give some equivalences which hold for a finite universe. Later we show that these equivalences also hold for an arbitrary universe.

1-6.4 Theory of Inference for The Predicate Calculus

The method of derivation involving predicate formulas uses the rules of inference given for the statement calculus and also certain additional rules which are required to deal with the formulas involving quantifiers. The rules **P and T**, regarding the introduction of a premise at any stage of derivation and the introduction of any formula which follows logically from the formulas already introduced, remain the same. If the conclusion is given in the form of a conditional, we shall also use the rule of conditional proof called **CP**. Occasionally, we may use the indirect method of proof in introducing the negation of the conclusion as an additional premise in order to arrive at a contradiction.

The equivalences and implications of the statement calculus can be used in the process of derivation as before, except that the formulas involved are generalized to predicates. But these formulas do not have any quantifiers in them, while some of the premises or the conclusion may be quantified. In order to use the equivalences and implications, we need some rules on how to eliminate quantifiers during the course of derivation. This elimination is done by *rules of specification* called rules **US** and **ES**. Once the quantifiers are eliminated, the derivation proceeds as in the case of the statement calculus, and the conclusion is reached. It may happen that the desired conclusion is quantified. In this case, we need *rules of generalization* called rules **UG** and **EG**, which can be used to attach a quantifier.

The rules of generalization and specification follow. Here $A(x)$ is used to denote a formula with a free occurrence of x. $A(y)$ denotes a formula obtained by the substitution of y for x in $A(x)$. Recall that for such a substitution $A(x)$ must be free for y.

Rule **US** (Universal Specification) From $(x)A(x)$ one can conclude $A(y)$.

Rule **ES** (Existential Specification) From $(\exists x)A(x)$ one can conclude $A(y)$ provided that y is not free in any given premise and also not free in any prior step of the derivation. These requirements can easily be met by choosing a new variable each time **ES** is used. (The conditions of **ES** are more restrictive than ordinarily required, but they do not affect the possibility of deriving any conclusion.)

Rule **EG** (Existential Generalization) From $A(x)$ one can conclude $(\exists y)A(y)$.

Rule **UG** (Universal Generalization) From $A(x)$ one can conclude $(y)A(y)$ provided that x is not free in any of the given premises and provided that if x is free in a prior step which resulted from use of **ES**, then no variables introduced by that use of **ES** appear free in $A(x)$.

We shall now show, by means of an example, how an invalid conclusion could be arrived at if the second restriction on rule **UG** were not imposed. The other restrictions on **ES** and **UG** are easy to understand.

Let $D(u, v)$: u is divisible by v. Assume that the universe of discourse is $\{5, 7, 10, 11\}$, so that the statement $(\exists u)D(u, 5)$ is true because both $D(5, 5)$

fiers occur in combinations. These combinations are possible even in the case of 1-place predicates, and they become particularly important in the case of n-place predicates ($n \geq 2$). For example, if $P(x, y)$ is a 2-place predicate formula, then the following possibilities exist:

$$
\begin{array}{lll}
(x)(y)P(x, y) & (x)(\exists y)P(x, y) & (\exists x)(y)P(x, y) \\
(\exists x)(\exists y)P(x, y) & (y)(x)P(x, y) & (\exists y)(x)P(x, y) \\
(y)(\exists x)P(x, y) & (\exists y)(\exists x)P(x, y) &
\end{array}
$$

It is understood that $(x)(y)P(x, y)$ stands for $(x)((y)P(x, y))$ and $(\exists x)(y)P(x, y)$ for $(\exists x)((y)P(x, y))$. The brackets are not used because even without them there is no possibility of misunderstanding the meaning. From the meaning of the quantifiers, the following formulas can be obtained.

$$(x)(y)P(x, y) \Leftrightarrow (y)(x)P(x, y) \tag{1}$$
$$(x)(y)P(x, y) \Rightarrow (\exists y)(x)P(x, y) \tag{2}$$
$$(y)(x)P(x, y) \Rightarrow (\exists x)(y)P(x, y) \tag{3}$$
$$(\exists y)(x)P(x, y) \Rightarrow (x)(\exists y)P(x, y) \tag{4}$$
$$(\exists x)(y)P(x, y) \Rightarrow (y)(\exists x)P(x, y) \tag{5}$$
$$(x)(\exists y)P(x, y) \Rightarrow (\exists y)(\exists x)P(x, y) \tag{6}$$
$$(y)(\exists x)P(x, y) \Rightarrow (\exists x)(\exists y)P(x, y) \tag{7}$$
$$(\exists x)(\exists y)P(x, y) \Leftrightarrow (\exists y)(\exists x)P(x, y) \tag{8}$$

Figure 1-6.1 shows implications (2) to (7) and equivalences (1) and (8). One can also prove these implications and equivalences using the method of derivation given in the previous section.

The negation of any of the above formulas can be obtained by repeated applications of the equivalences E_{25} and E_{26} of Sec. 1-6.3. For example,

$$\neg(\exists y)(x)P(x, y) \Leftrightarrow (y)(\neg(x)P(x, y)) \Leftrightarrow (y)(\exists x)\neg P(x, y)$$

The negations of other formulas of this type are obtained in a similar manner.

The inference rules and the method of derivation as given in Sec. 1-6.4 also apply to n-place predicate formulas. Obviously, some special care would

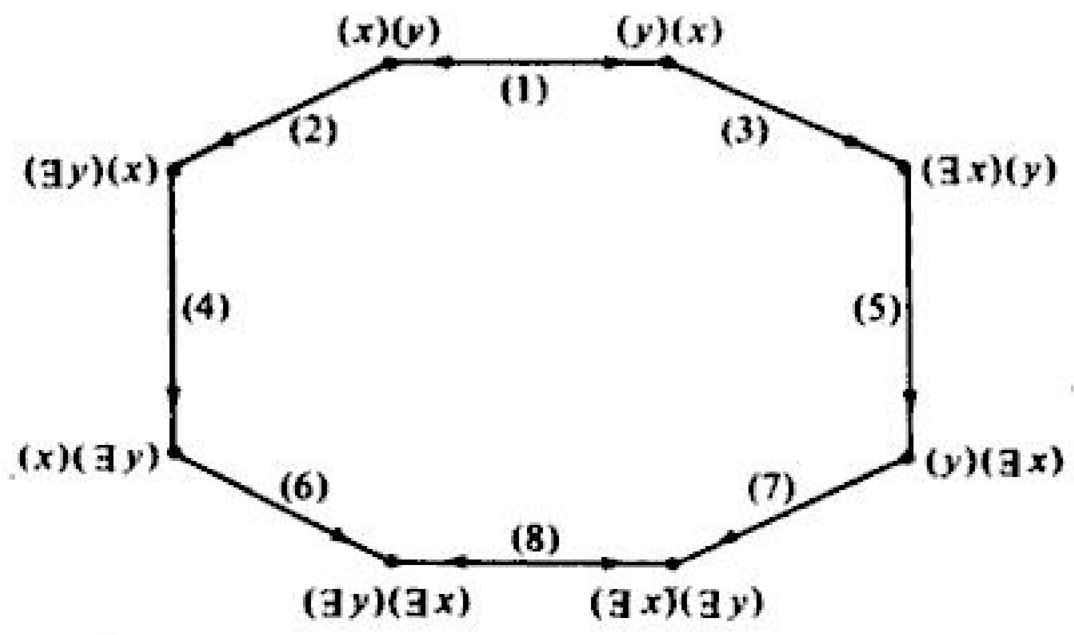


FIGURE 1-6.1 Graphical representation of relationships among formulas involving two quantifiers.

2

SET THEORY

INTRODUCTION

The concept of a set is used in various disciplines and particularly often in mathematics. In this chapter, we introduce elementary set theory. An axiomatic approach to the discussion and questions of a philosophical nature will be avoided. Although our presentation remains informal, we try to indicate formal proofs which use the notation and the rules of inference given in Chap. 1. As we proceed, an analogy will be drawn between the statement calculus and the set operations leading to a set algebra which is similar to the statement algebra given earlier. Initially, the notation of set theory is introduced and certain operations are defined. Then follows an introduction to the representation of discrete structures. The concepts of relations, orderings, and functions are presented after a discussion of the algebra of sets. A particular type of function known as a binary operation prepares us for a discussion of algebraic structures, which form the subject matter of Chap. 3. A special function known as a hashing function, which maps a name into an integer and permits us to handle nonnumeric data, is discussed. The natural numbers are introduced, and the principle of mathematical induction is given. A discussion of recursive functions then follows. Certain applications

It is enough to note at this stage that set inclusion is reflexive and transitive. These terms are explained in Sec. 2-3.2. The proof of Statement (1) is obvious, while Statement (2) can be proved by using the implication

$$(x)(x \in A \rightarrow x \in B) \wedge (x)(x \in B \rightarrow x \in C) \Rightarrow (x)(x \in A \rightarrow x \in C)$$

(see Example 2, Sec. 1-6.4). For two sets A and B, note that $A \subseteq B$ does not necessarily imply $B \subseteq A$ except for the following case.

Definition 2-1.2 Two sets A and B are said to be *equal* (extensionally equal) iff $A \subseteq B$ and $B \subseteq A$, or symbolically,

$$A = B \Leftrightarrow (A \subseteq B \wedge B \subseteq A)$$

From the equivalence

$$(x)((P(x) \rightarrow Q(x)) \wedge (Q(x) \rightarrow P(x))) \Leftrightarrow (x)(P(x) \rightleftarrows Q(x))$$

we can alternatively define the equality of two sets as

$$A = B \Leftrightarrow (x)(x \in A \rightleftarrows x \in B)$$

We now give some examples of sets that are equal and sets that are not equal.

$\{1, 2, 4\} = \{1, 2, 2, 4\}$.

$\{1, 4, 2\} = \{1, 2, 4\}$.

If $P = \{\{1, 2\}, 4\}$ and $Q = \{1, 2, 4\}$, then $P \neq Q$.

$\{\{1\}\} \neq \{1\}$ because $\{1\} \in \{\{1\}\}$ while $1 \in \{1\}$.

If $A = \{x \mid x(x - 1) = 0\}$ and $B = \{0, 1\}$, then $A = B$.

$\{1, 3, 5, \ldots\} = \{x \mid x \text{ is an odd positive integer}\}$.

From the definition of equality of sets it is clear that

$$A = B \Leftrightarrow B = A$$

The equality of sets is reflexive, symmetric, and transitive.

Definition 2-1.3 A set A is called a *proper subset* of a set B if $A \subseteq B$ and $A \neq B$. Symbolically it is written as $A \subset B$, so that

$$A \subset B \Leftrightarrow (A \subseteq B \wedge A \neq B)$$

$A \subset B$ is also called a proper inclusion.

A proper inclusion is not reflexive; however, it is transitive, i.e.,

$$(A \subset B) \wedge (B \subset C) \Rightarrow (A \subset C)$$

We shall now introduce two special sets, of which one includes every set under discussion while the other is included in every set under discussion.

say here that operations on one or more sets produce other sets according to certain rules.

Definition 2-1.8 The *intersection* of any two sets A and B, written as $A \cap B$, is the set consisting of all the elements which belong to both A and B. Symbolically,

$$A \cap B = \{x \mid (x \in A) \wedge (x \in B)\}$$

From the definition of intersection it follows that for any sets A and B,

$$A \cap B = B \cap A \qquad A \cap A = A \qquad \text{and } A \cap \varnothing = \varnothing \tag{1}$$

The first of these equalities shows that the intersection is commutative. The importance of the other two will be discussed later. The commutativity of intersection can be proved in the following manner. For any x,

$$\begin{aligned} x \in A \cap B &\Leftrightarrow x \in \{x \mid (x \in A) \wedge (x \in B)\} \\ &\Leftrightarrow (x \in A) \wedge (x \in B) \\ &\Leftrightarrow (x \in B) \wedge (x \in A) \\ &\Leftrightarrow x \in \{x \mid (x \in B) \wedge (x \in A)\} \\ &\Leftrightarrow x \in B \cap A \end{aligned}$$

The other two equalities in Eq. (1) can be proved in a similar manner.

Since $A \cap B$ is a set, we can consider its intersection with another set C, so that

$$(A \cap B) \cap C = \{x \mid x \in A \cap B \wedge x \in C\}$$

Using $(x \in A \wedge x \in B) \wedge x \in C \Leftrightarrow x \in A \wedge (x \in B \wedge x \in C)$, we can easily show that

$$(A \cap B) \cap C = A \cap (B \cap C) \qquad \text{(associative)} \tag{2}$$

In view of Eq. (2), we can write $(A \cap B) \cap C$ as $A \cap B \cap C$.

For an indexed set $A = \{A_1, A_2, \ldots, A_n\} = \{A_i\}_{i \in J}$ where $\mathbf{I}_n = \{1, 2, \ldots, n\}$, we write

$$A_1 \cap A_2 \cap \cdots \cap A_n = \bigcap_{i=1}^{n} A_i = \bigcap_{i \in J_n} A_i$$

In general, for any index set J,

$$\bigcap_{i \in J} A_i = \{x \mid x \in A_i \text{ for all } i \in J\}$$

Definition 2-1.9 Two sets A and B are called *disjoint* iff $A \cap B = \varnothing$, that is, A and B have no element in common.

Definition 2-1.10 A collection of sets is called a *disjoint collection* if, for every pair of sets in the collection, the two sets are disjoint. The elements of a disjoint collection are said to be *mutually disjoint.*

4 Show that $A \subseteq B \Leftrightarrow A \cup B = B$.
5 If $S = \{a, b, c\}$, find nonempty disjoint sets A_1 and A_2 such that $A_1 \cup A_2 = S$. Find other solutions to this problem.
6 Prove the equalities in Eqs. (4) and (5).
7 Given $A = \{2, 3, 4\}$, $B = \{1, 2\}$, and $C = \{4, 5, 6\}$, find $A + B$, $B + C$, $A + B + C$, and $(A + B) + (B + C)$.

2-1.5 Venn Diagrams

Introduction of the universal set permits the use of a pictorial device to study the connection between the subsets of a universal set and their intersection, union, difference, and other operations. The diagrams used are called Venn diagrams. A *Venn diagram* is a schematic representation of a set by a set of points. The universal set E is represented by a set of points in a rectangle (or any other figure), and a subset, say A, of E is represented by the interior of a circle or some other simple closed curve inside the rectangle. In Fig. 2-1.1 the shaded areas represent the sets indicated below each figure. The Venn diagram for $A \subseteq B$ and $A \cap B = \varnothing$ are also given. From some of the Venn diagrams it is

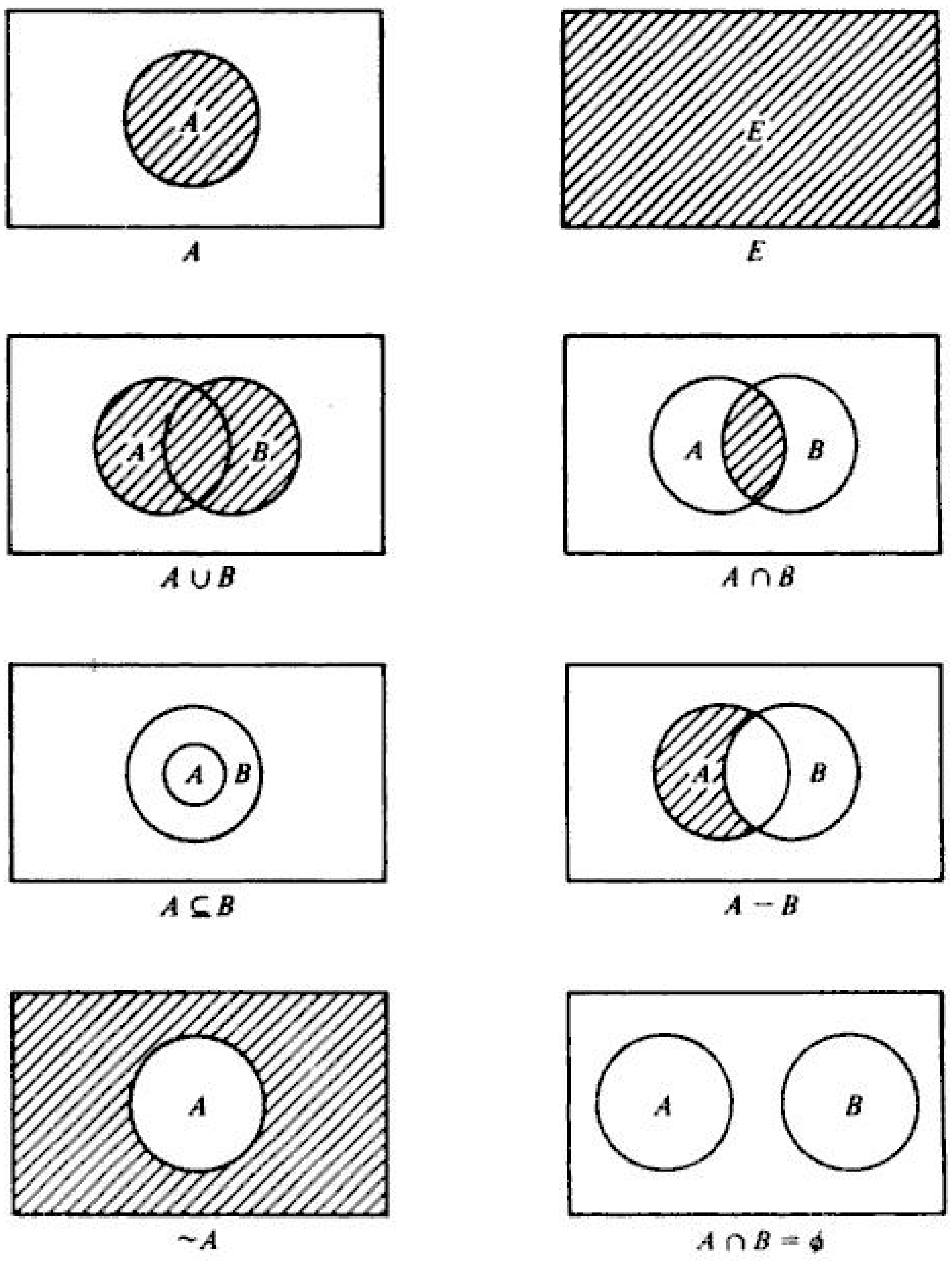


FIGURE 2-1.1 Venn diagrams.

All the identities just given are presented in pairs except for the identity (11). This pairing is done because a principle of duality similar to the one given for statement algebra (see Sec. 1-2.10) also holds in the case of set algebra. In fact, the principle of duality holds for any Boolean algebra. At present it is sufficient to note that given any identity of the set algebra, one can obtain another identity by interchanging $\cup$ with $\cap$ and E with $\varnothing$.

Assuming identities (4) to (6), we shall prove the absorption laws. First note that

$$A \cup (A \cap B) = (A \cup A) \cap (A \cup B) = A \cap (A \cup B)$$

from the distributive and idempotent laws. Now

$$\begin{aligned} A \cup (A \cap B) &= (A \cap E) \cup (A \cap B) && \text{from (5)} \\ &= A \cap (E \cup B) && \text{from (4)} \\ &= A \cap E && \text{from (6)} \\ &= A && \text{from (5)} \end{aligned}$$

Alternatively one can prove it as follows. For any x,

$$\begin{aligned} x \in A \cup (A \cap B) &\Leftrightarrow x \in \{x \mid (x \in A) \vee ((x \in A) \wedge (x \in B))\} \\ &\Leftrightarrow x \in \{x \mid x \in A\} \\ &\Leftrightarrow x \in A \end{aligned}$$

using the absorption laws of predicate calculus.

In order to complete our discussion, we list some implications and certain set inclusions

$$(A \cup B \neq \varnothing) \Rightarrow (A \neq \varnothing) \vee (B \neq \varnothing) \qquad (12)$$

$$(A \cap B \neq \varnothing) \Rightarrow (A \neq \varnothing) \wedge (B \neq \varnothing) \qquad (13)$$

To prove the implication (12), let us assume that $A \neq \varnothing \vee B \neq \varnothing$ is *false*. This requires that $A \neq \varnothing$ is *false* and also that $B \neq \varnothing$ is *false*, that is, $A = B = \varnothing$. But then $A \cup B = \varnothing$, so that $A \cup B \neq \varnothing$ is also *false*. Hence the implication is proved. One could also have proved (12) by assuming that $A \cup B \neq \varnothing$ is *true* and showing that this assumption requires $A \neq \varnothing \vee B \neq \varnothing$ to be *true*. Implication (13) can be proved in a similar manner.

The following inclusions follow from the definition and have been proved earlier in this section.

$$A \cap B \subseteq A \qquad A \cap B \subseteq B \qquad A \subseteq A \cup B \qquad A - B \subseteq A \qquad (14)$$

Let A be a family of indexed sets over an index set $\mathbf{I}$ such that $A = \{A_1, A_2, \ldots\} = \{A_i\}_{i \in \mathbf{I}}$. Then

$$\bigcup_{i \in \mathbf{I}} A_i = \{x \mid x \in A_i \text{ for some } i \in \mathbf{I}\} \qquad (15)$$

$$\bigcap_{i \in \mathbf{I}} A_i = \{x \mid x \in A_i \text{ for every } i \in \mathbf{I}\} \qquad (16)$$

and B and is written as $A \times B$. Accordingly,

$$A \times B = \{\langle x, y\rangle \mid (x \in A) \wedge (y \in B)\}$$

EXAMPLE 1 If $A = \{\alpha, \beta\}$ and $B = \{1, 2, 3\}$, what are $A \times B$, $B \times A$, $A \times A$, $B \times B$, and $(A \times B) \cap (B \times A)$?

SOLUTION

$$\begin{aligned}
A \times B &= \{\langle\alpha, 1\rangle, \langle\alpha, 2\rangle, \langle\alpha, 3\rangle, \langle\beta, 1\rangle, \langle\beta, 2\rangle, \langle\beta, 3\rangle\}\\
B \times A &= \{\langle 1, \alpha\rangle, \langle 2, \alpha\rangle, \langle 3, \alpha\rangle, \langle 1, \beta\rangle, \langle 2, \beta\rangle, \langle 3, \beta\rangle\}\\
A \times A &= \{\langle\alpha, \alpha\rangle, \langle\alpha, \beta\rangle, \langle\beta, \alpha\rangle, \langle\beta, \beta\rangle\}\\
B \times B &= \{\langle 1, 1\rangle, \langle 1, 2\rangle, \langle 1, 3\rangle, \langle 2, 1\rangle, \langle 2, 2\rangle, \langle 2, 3\rangle, \langle 3, 1\rangle,\\
&\quad \langle 3, 2\rangle, \langle 3. 3\rangle\}
\end{aligned}$$

$(A \times B) \cap (B \times A) = \varnothing$ ////

EXAMPLE 2 If $A = \varnothing$ and $B = \{1, 2, 3\}$ what are $A \times B$ and $B \times A$?

SOLUTION

$$A \times B = \varnothing = B \times A$$ ////

Before we consider the cartesian product of more than two sets let us consider the expressions $(A \times B) \times C$ and $A \times (B \times C)$. From the definition it follows that

$$\begin{aligned}
(A \times B) \times C &= \{\langle\langle a, b\rangle, c\rangle \mid (\langle a, b\rangle \in A \times B) \wedge (c \in C)\}\\
&= \{\langle a, b, c\rangle \mid (a \in A) \wedge (b \in B) \wedge (c \in C)\} \qquad (1)
\end{aligned}$$

The last step follows from our definition of the ordered triple given in Sec. 2-1.8. Next,

$$A \times (B \times C) = \{\langle a, \langle b, c\rangle\rangle \mid (a \in A) \wedge (\langle b, c\rangle \in B \times C)\}$$

Here $\langle a, \langle b, c\rangle\rangle$ is not an ordered triple. If we consider $(A \times B) \times C$ as an ordered pair, then the first member is an ordered pair and the second member is an element of C. On the other hand, $A \times (B \times C)$ is an ordered pair in which the first member is an element of A while the second member is an ordered pair. This fact shows that

$$(A \times B) \times C \neq A \times (B \times C)$$

Before defining the cartesian product of any finite number of sets, we shall show that the cartesian product satisfies the following distributive properties.

For any three sets A, B, and C

$$\begin{aligned}
A \times (B \cup C) &= (A \times B) \cup (A \times C)\\
A \times (B \cap C) &= (A \times B) \cap (A \times C) \qquad (2)
\end{aligned}$$

We now prove the first of these two identities.

2-2.4 Pointers and Linked Allocation

The previous section discussed at some length how the address of an element in a data structure could be obtained by direct computation. The data structures discussed were linearly ordered, and this ordering relation was specified in the corresponding storage structures by using sequential allocation. There was no need for an element to specify where the next element would be found.

Consider a list consisting of elements which individually vary in size. The task of directly computing the address of a particular element becomes much more difficult. An obvious method of obtaining the address of a node (element) is to store it in the computer memory. This address was previously defined as a link or pointer address. If the list in question has n nodes, we can store the address of each node in a vector consisting of n elements. The first element of the vector will contain the address of the first node of the list, the second element the address of the second node, and so on.

There are many applications which, by their very nature, have data which are continually being updated (additions, deletions, etc.). Each time a change occurs, significant manipulation of the data is required. The representation of the data by sequentially allocated lists in some of these cases results in an inefficient use of memory and wasted computational time, and, indeed, for certain problems this method of allocation is totally unacceptable.

The interpretation of a pointer as an address is a natural one. Most computers use addresses to find the next instruction to be executed and its operand(s). In many hardware configurations special registers are used to store such addresses.

A pointer can be regarded as a general type of structure because when a pointer to a data structure is given, then its contents become accessible. Pointers are always of the same length (usually no longer than a half-word), and this property enables the manipulation of pointers to be performed in a uniform manner by using simple allocation techniques regardless of the configurations of the structures they may point to.

In the sequential-allocation method one is able to compute an address of an element provided that the storage structure is organized in some uniform manner. Pointers permit the referencing of structures in a uniform way regardless of the organization of the structure being referenced. Pointers are capable of representing a much more complex relationship between elements of a structure than a linear order.

The use of pointers or links to refer to elements of a data structure (which is linearly ordered) implies that elements which are adjacent because of the linear ordering need not be physically adjacent in memory. This type of allocation scheme is called *linked allocation.* We now turn to the problem of representing structures by this type of allocation.

A plex has been defined to consist of an ordered set of elements which may vary in number. The previous subsection was concerned with representation of the relationship of adjacency between elements in a plex. There are many other structures that can be represented by a plex where the relationships that exist between elements are much more complex than that of adjacency.

The simplest form that can be used to represent a linear plex is to expand

a number of subprograms which are written in the FORTRAN language. We will make use of the PL/I language to represent certain common operations performed on lists.

There are a number of classes of operations which are associated with linked lists. The first class of such operations is independent of the data contained in the nodes of a list. These operations include the insertion, deletion, and selection of nodes. Programming languages which possess list processing capabilities usually have these operations built in.

Another class of operations associated with list structures is the operation which converts the raw data from a human-readable form to a corresponding machine form. The inverse operation of converting an internal structure to a suitable human-readable form is also required. These operations are clearly data-dependent, and attention must be given to the interpretation that is associated with the structures. List processing languages will have some standard basic routines for such operations, but any additional routines must be programmed.

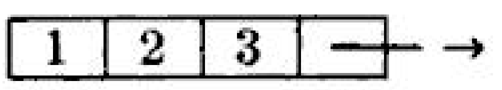
Finally, there are operations that must be programmed to manipulate the data according to what is required in a particular application at hand. In the case of polynomial manipulation, for example, such operations would include the addition, subtraction, multiplication, division, differentiation, and integration of polynomials. Once a programmer has access to all the routines for the three classes mentioned above, the task of programming an algorithm is much simpler.

Let us consider the problem of describing singly linked linear list representations and operations in an algorithmic notation and also in the PL/I language. A node consists of a number of fields, each of which can represent an integer, a real number, etc., except for one field (usually the last for purposes of illustration), called a pointer, which contains the location of the next node in the list. This location is specified as a computer memory address.

Consider the example of representing a term of a polynomial in the variables x and y. A typical node will be represented as

XEXP	*YEXP*	*COEF*	*LINK*

which consists of four sequentially allocated fields that we will collectively refer to as *TERM*. The first two fields represent the power of the variables x and y respectively. The third and fourth fields represent the coefficient of the term in the polynomial and the address of the next term in the polynomial, respectively. For example, the term $3xy^2$ would be represented as



The selection of a particular field within a node for the example of our polynomial is an easy matter. Our algorithmic notation will allow the referencing of any field of a node, given the pointer P to that node. $COEF(P)$ denotes the coefficient field of a node pointed to by P. Similarly, the exponents of x and y are given by $XEXP(P)$ and $YEXP(P)$ respectively, and the pointer to the next node is given by $LINK(P)$.

```
POLYST:
     PROCEDURE OPTIONS(MAIN);
/*CONSTRUCT A LINKED LIST REPRESENTATION OF A POLYNOMIAL USING
  THE INSERT PROCEDURE. */
     DECLARE
          1 TERM BASED(P),
               2 XEXP BINARY FIXED,
               2 YEXP BINARY FIXED,
               2 COEF BINARY FLOAT,
               2 LINK POINTER,
          Q POINTER,
          POLY POINTER,
          INSERT ENTRY(BIN FIXED,BIN FIXED,BIN FLOAT,PTR) RETURNS(PTR);

     POLY = NULL; /* INITIALIZE */

     POLY = INSERT(0,2,1,POLY); /* INSERT LAST TERM OF POLYNOMIAL */

     POLY = INSERT(1,1,3,POLY); /* INSERT SECOND TERM OF POLYNOMIAL */

     POLY = INSERT(2,0,1,POLY); /* INSERT FIRST TERM OF POLYNOMIAL */

END POLYST;
```

FIGURE 2-2.5 Construction of the polynomial $x^2 + 3xy + y^2$.

is given by variable X. Auxiliary pointer variables $NEXT$ and $PRED$ are used The list is composed of nodes with structure previously described as $TERM$.

1 [Empty list?] If $FIRST = NULL$ then write 'underflow' and Exit.

2 [Delete first node?] If $X = FIRST$ then set $FIRST \leftarrow LINK(FIRST)$ and go to step 8.

3 [Initiate search for predecessor of X.] Set $NEXT \leftarrow FIRST$.

4 [Update pointers.] Set $PRED \leftarrow NEXT$ and $NEXT \leftarrow LINK(NEXT)$.

5 [End of list?] If $NEXT = NULL$ then write 'node not found' and Exit.

6 [Is this node X?] If $NEXT \neq X$ then go to step 4.

7 [Delete X.] Set $LINK(PRED) \leftarrow LINK(X)$.

8 [Return node X.] Restore node X to the availability area and Exit.

The first step in the algorithm checks for an underflow. The second step determines whether the node to be deleted is the first node of the list, and if it is, then the second node of the list becomes the new first node. In the case of a list containing a single node, the pointer variable $FIRST$ will assume the null value as a result of the deletion.

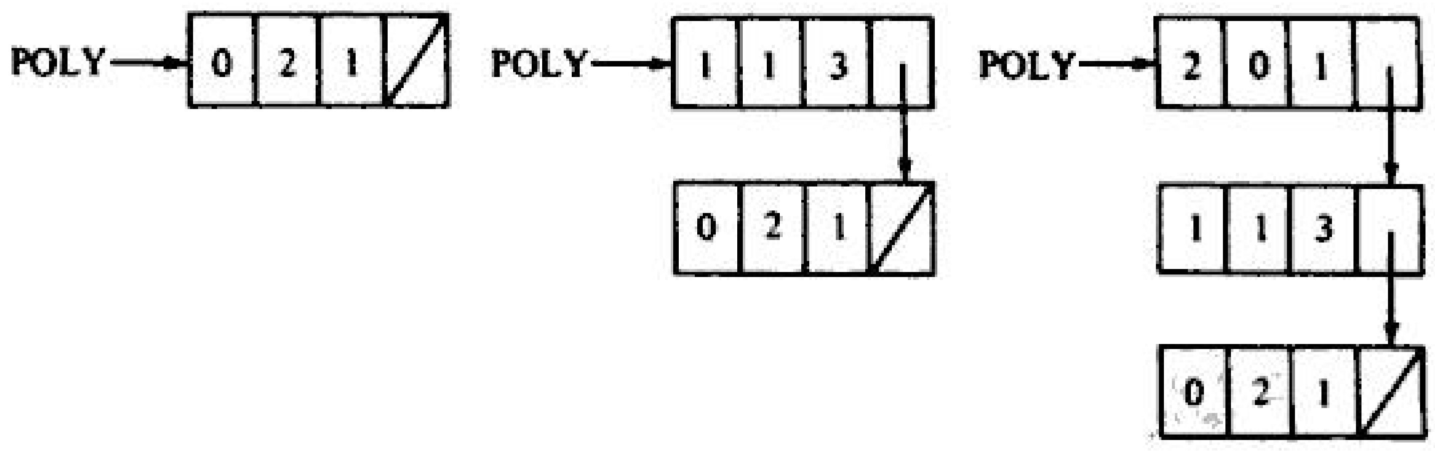


FIGURE 2-2.6 Trace of the construction of a linked list for polynomial $x^2 + 3xy + y^2$.

the predicate

$$x \text{ is a commerce student.}$$

We can find the extension set of the predicate

$$x \text{ is a male commerce student.}$$

by finding the intersection of the two preceding extension sets. This is done by performing the logical operation

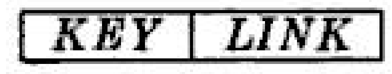
SEX_FILE(1)&COLLEGE_FILE(2)

```
BITSETS:
    PROCEDURE OPTIONS(MAIN);
/*THIS PROGRAM INPUTS CODED INFORMATION ABOUT STUDENTS, CREATING A
  MASTER FILE AND THREE BIT STRING ARRAYS.  THE MASTER FILE CONSISTS OF
  AN ARRAY OF STRUCTURES NAMED STUDENT.  EACH ELEMENT OF THIS ARRAY
  HAS FOUR FIELDS NAMED AND CONTAINING CODES AS FOLLOWS:
      NUMBER          - A SIX DIGIT STUDENT NUMBER
      SEX             - 1 - MALE
                      - 2 - FEMALE
      COLLEGE         - 1 - ARTS AND SCIENCE
                      - 2 - COMMERCE
                      - 3 - ENGINEERING
                      - 4 - GRADUATE STUDIES
                      - 5 - HOME ECONOMICS
                      - 6 - AGRICULTURE
      MARITAL_STATUS  - 1 - SINGLE
                      - 2 - MARRIED
                      - 3 - OTHER
  FOR THE FIELDS SEX, COLLEGE, AND MARITAL_STATUS, THE ARRAYS SEX_FILE,
  COLLEGE_FILE, AND STATUS_FILE ARE ESTABLISHED AS REPRESENTATIONS OF
  MUTUALLY EXCLUSIVE SUBSETS OF THE MASTER FILE.  THE ELEMENTS OF THESE
  ARRAYS ARE BIT STRINGS HAVING A LENGTH EQUAL TO THE NUMBER OF RECORDS
  IN THE MASTER FILE.  THERE ARE A TOTAL OF 11 BIT STRINGS, ONE FOR
  EACH OF THE ABOVE CODES.  EACH STRING INITIALLY CONSISTS ENTIRELY OF
  '0' BITS.
  THE FIRST PART OF THE MAIN PROGRAM INPUTS THE STUDENT FILE AND
  CONSTRUCTS THE BIT REPRESENTED SETS AS FOLLOWS:  FOR THE ITH RECORD,
  IF THE CODES J, K, AND M ARE IN THE FIELDS SEX, COLLEGE, AND MARITAL_
  STATUS RESPECTIVELY, THEN THE ITH BIT OF BIT STRINGS SEX_FILE(J),
  COLLEGE_FILE(K), AND STATUS_FILE(M) ARE SET TO '1'B.
  BEGINNING AT THE LABEL QUERY, A NUMBER OF EXTENSION SETS OF CERTAIN
  PREDICATES ARE PRINTED USING PROCEDURE OUTPUT.  THE FIRST INVOCATION
  PRINTS EVERY RECORD AND THE NEXT FIVE PRINT CERTAIN SUBSETS OF THE
  MASTER FILE.  THE USE OF LOGICAL CONNECTIVES IN FINDING
  INTERSECTIONS, UNIONS, OR COMPLEMENTATIONS OF EXTENSION SETS IS
  DEMONSTRATED.                                                      */
    GET LIST(N);
BEGIN; /* AUTOMATIC STORAGE ALLOCATION */
    DECLARE
        1 STUDENT(N),
            2 NUMBER FIXED(6),
            2 SEX FIXED(1),
            2 COLLEGE FIXED(1),
            2 MARITAL_STATUS FIXED(1),
        (SEX_FILE(2),COLLEGE_FILE(6),STATUS_FILE(3)) BIT(N)
            INITIAL((11)(1)'0'B),
        SEX_WORD(2) CHAR(10) INITIAL('MALE','FEMALE'),
        COLLEGE_WORD(6) CHAR(20) INITIAL('ARTS AND SCIENCE','COMMERCE',
            'ENGINEERING','GRADUATE STUDIES','HOME ECONOMICS',
            'AGRICULTURE'),
        STATUS_WORD(3) CHAR(7) INITIAL('SINGLE','MARRIED','OTHER'),
        I BINARY FIXED,
        HEADINGS CHAR(54) INITIAL
            ('NUMBER    SEX        COLLEGE              MARITAL STATUS');
```

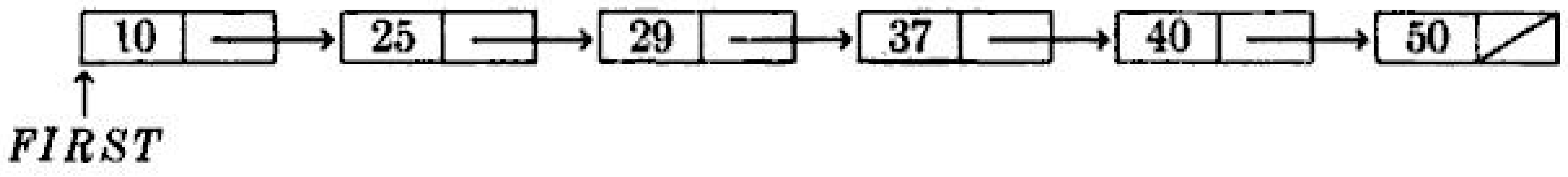
FIGURE 2-2.8 PL/I program applying bit represented sets.

and the pointer variable *BEGIN* is to denote the first node. The variables *INFO* and *FIELD* represent the information content of a node, while *LINK* and *PTR* are variables containing the address of the next node. Give the name *COPY* to the algorithm.

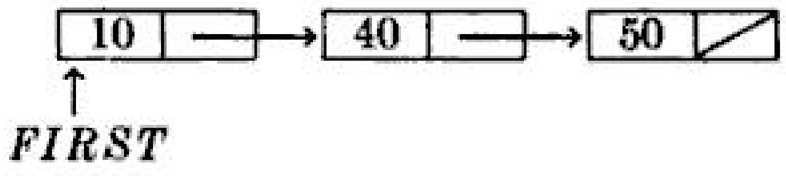
10 Suppose that you are given a simple linked list whose first node is denoted by the pointer variable *FIRST* and whose typical node is represented by



where the variables *KEY* and *LINK* represent the information and link fields of the node, respectively. The list is ordered on the field *KEY* such that the first and last nodes contain the smallest and largest values of the field. It is desired to delete a number of consecutive nodes whose *KEY* values are greater than or equal to *KMIN* and less than *KMAX*. For example, an initial list with *KMIN* and *KMAX* having values of 25 and 40, respectively, could look like this:



After deleting the designated nodes, the updated list would reduce to the following:



In this example, we dropped the nodes whose *KEY* field values are 25, 29, and 37. Formulate an algorithm and write a program which will accomplish the deletion operation for an arbitrary linked list.

2-3 RELATIONS AND ORDERING

The concept of a relation is a basic concept in everyday life as well as in mathematics. We have already used various relations. Associated with a relation is the act of comparing objects which are related to one another. The ability of a computer to perform different tasks based upon the result of a comparison is one of its most important attributes which is used several times during the execution of a typical program. In this section we first formalize the concept of a relation and then discuss methods of representing a relation by using a matrix or its graph. The relation matrix is useful in determining the properties of a relation and also in representing a relation on a computer. Various basic properties of relations are given, and certain important classes of relations are introduced. Among these, the compatibility relation and the equivalence relation have useful applications in the design of digital computers and other sequential machines. Partial ordering and its associated terminology are introduced next. The material in Chap. 4 is based upon these notions. Several relations given as examples in this section are used throughout the book. Algorithms to determine certain properties of relations are also given.

member of each of the following sets:

$$R_4 = R_1 \cap R_2 \cap R_3$$
$$= \{\langle x, y\rangle \mid \langle x, y\rangle \in \mathbf{R} \times \mathbf{R} \wedge x * y \geq 1 \wedge x^2 + y^2 \leq 9 \wedge y^2 < x\}$$
$$R_5 = R_2 \cap (R_1 \cup R_3) \cap \sim(R_1 \cap R_3)$$
$$= \{\langle x, y\rangle \mid \langle x, y\rangle \in \mathbf{R} \times \mathbf{R} \wedge x^2 + y^2 \leq 9 \wedge (x * y \geq 1 \vee y^2 < x) \wedge \sim(x * y \geq 1 \wedge y^2 < x)\}$$
$$R_6 = R_1 \cap \sim R_2 \cap R_3.$$
$$= \{\langle x, y\rangle \mid \langle x, y\rangle \in \mathbf{R} \times \mathbf{R} \wedge x * y \geq 1 \wedge \sim(x^2 + y^2 \leq 9) \wedge y^2 < x\}$$
$$R_7 = \sim(R_1 \cup R_3) \cap R_2$$
$$= \{\langle x, y\rangle \mid \langle x, y\rangle \in \mathbf{R} \times \mathbf{R} \wedge \sim(x * y \geq 1 \vee y^2 < x) \wedge x^2 + y^2 \leq 9\}$$

R_4 includes all points lying within the circle and the parabola and above the hyperbola of the first quadrant. R_5 includes all points within the circle which lie either within the parabola or above the hyperbola of the first quadrant, but not both, and all points within the circle and below the hyperbola in the third quadrant. R_6 includes all points lying above the hyperbola and within the parabola in the first quadrant. R_7 includes all points lying within the circle and between the hyperbolic curves but not within the parabola.

These newly defined sets can pictorially be represented as shown in Fig. 2-3.2. The program given in Fig. 2-3.3 reads a number of coordinate points and determines whether these points lie in the sets R_4 to R_7. Note that the relations R_4 to R_7 are written as predicates P_4 to P_7 in the program.

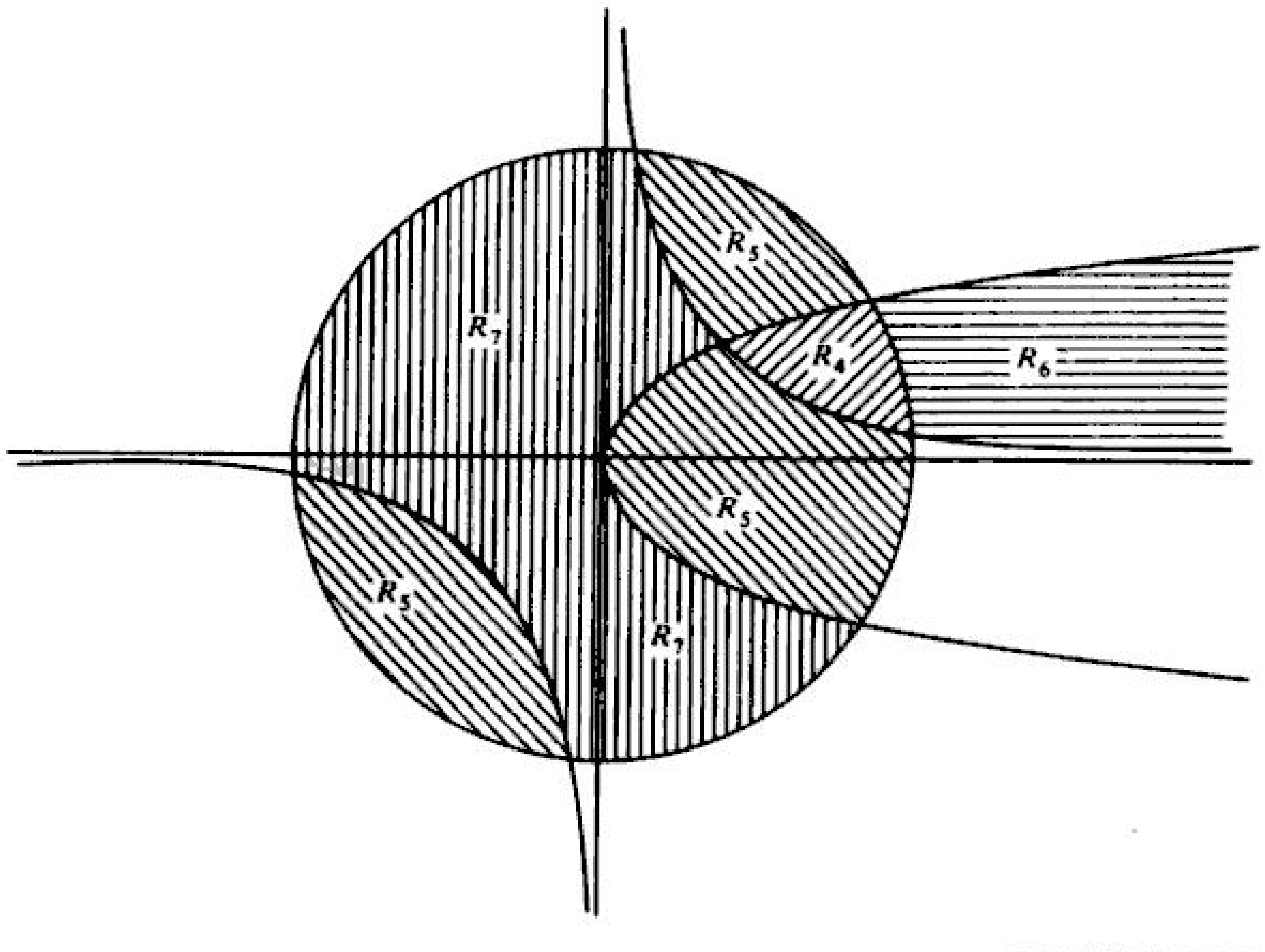


FIGURE 2-3.2

show that R is not transitive. Find a relation $R_1 \supseteq R$ such that R_1 is transitive. Can you find another relation $R_2 \supseteq R$ which is also transitive?

7 Given $S = \{1, 2, \ldots, 10\}$ and a relation R on S where

$$R = \{\langle x, y\rangle \mid x + y = 10\}$$

what are the properties of the relation R?

8 Let R be a relation on the set of positive real numbers so that its graphical representation consists of points in the first quadrant of the cartesian plane. What can we expect if R is (*a*) reflexive, (*b*) symmetric, and (*c*) transitive?

9 Show that the relations L and D given in Problem 3 of Exercises 2-3.1 are both reflexive, antisymmetric, and transitive. Give another example of such a relation. Draw the graphs of these relations as defined in Sec. 2-3.3.

2-3.3 Relation Matrix and the Graph of a Relation

A relation R from a finite set X to a finite set Y can also be represented by a matrix called the *relation matrix* of R.

Let $X = \{x_1, x_2, \ldots, x_m\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, and R be a relation from X to Y. The relation matrix of R can be obtained by first constructing a table whose columns are preceded by a column consisting of successive elements of X and whose rows are headed by a row consisting of the successive elements of Y. If $x_i \, R \, y_j$, then we enter a 1 in the ith row and jth column. If $x_k \not R \, x_l$, then we enter a zero in the kth row and lth column. As a special case, consider $m = 3$, $n = 2$, and R given by

$$R = \{\langle x_1, y_1\rangle, \langle x_2, y_1\rangle, \langle x_3, y_2\rangle, \langle x_2, y_2\rangle\} \tag{1}$$

The required table for R is Table 2-3.1.

If we assume that the elements of X and Y appear in a certain order, then the relation R can be represented by a matrix whose elements are 1s and 0s. This matrix can be written down from the table constructed or can be defined in the following manner.

$$r_{ij} = \begin{cases} 1 & \text{if } x_i \, R \, y_j \\ 0 & \text{if } x_i \not R \, y_j \end{cases}$$

where r_{ij} is the element in the ith row and jth column. The matrix obtained in this way is called the relation matrix. If X has m elements and Y has n elements, then the relation matrix is an $m \times n$ matrix. For the relation R given in Eq. (1), the relation matrix is

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Table 2-3.1

	y_1	y_2
x_1	1	0
x_2	1	1
x_3	0	1

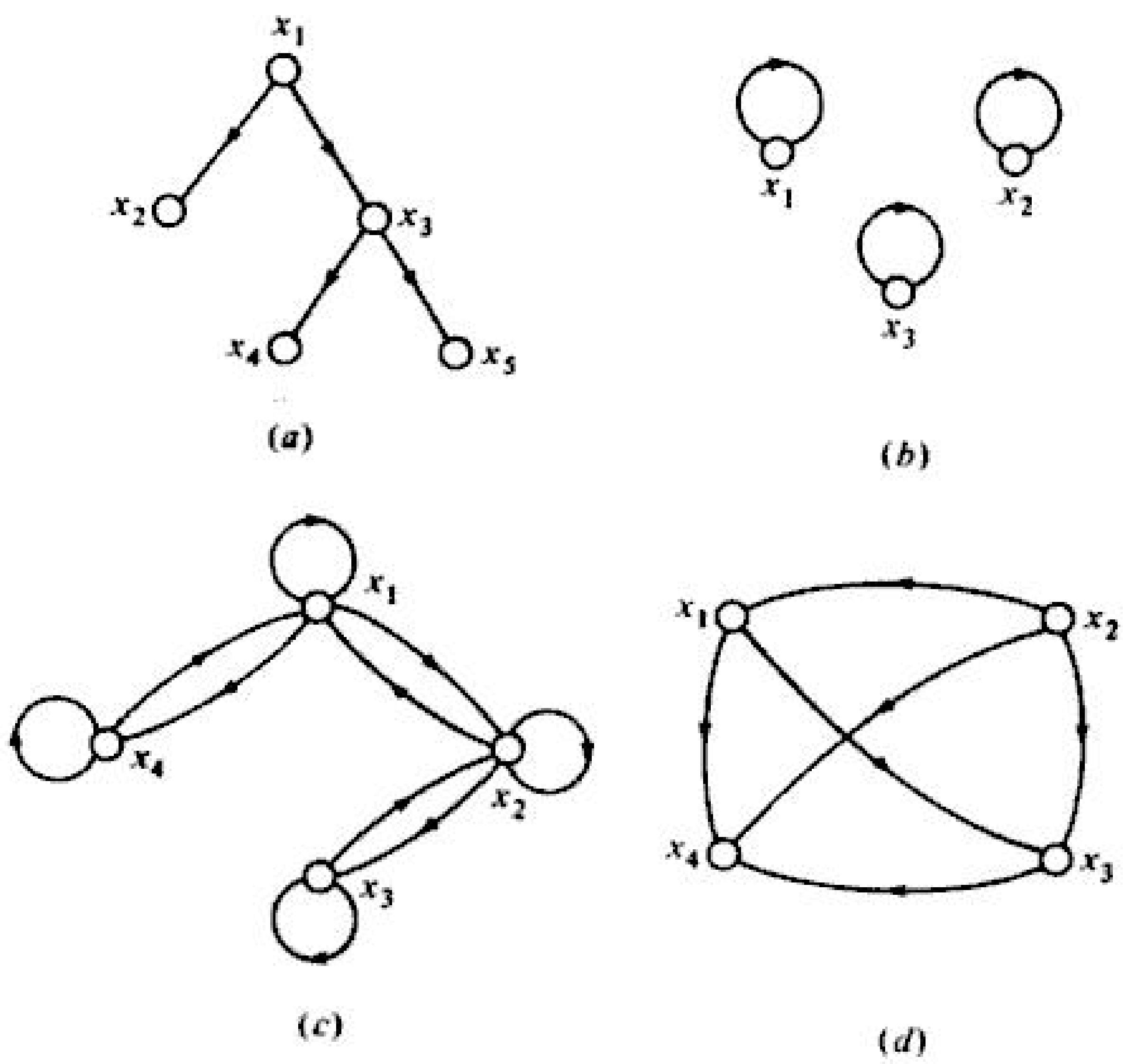


FIGURE 2-3.8

representations of the relation become unwieldy. In these cases, however, the matrix representation can be easily represented on a computer. When a relation matrix is available, it is easy to determine whether a given relation is reflexive or symmetric. It is not always easy to determine from the matrix whether the relation is transitive. We now present two algorithms. The first determines from a relation matrix whether the relation is reflexive and symmetric. The second algorithm then determines whether the relation is also transitive. Relations which are reflexive, symmetric, and transitive are called *equivalence relations*. Equivalence relations are discussed in Sec. 2-3.5.

The entries of the relation matrix are denoted by T and F instead of 1 and 0 in order to conserve storage. Note that in FORTRAN only 1 byte is needed for each logical entry, but at least 2 bytes are required for an integer entry.

Algorithm *REFSYM* Given a relation matrix R representing a relation in the set of positive integers from 1 to n inclusive, it is required to determine if the relation represented by R is symmetric and reflexive. If it is, the variable *FLAG* which is initially *false* is given the truth value *true*; otherwise *FLAG* remains *false*.

```
1 [Scan each row.] Repeat steps 2 and 3 for $i = 1, 2, \ldots, n$.
2 [Reflexive?] If $R[i, i] = F$ then Exit.
3 [Symmetric?] Repeat for $j = i + 1, i + 2, \ldots, n$:
      If $R[i\ j] \rightleftarrows R[j, i] = F$ then Exit.
4 [Successful test.] Set $FLAG \leftarrow T$ and Exit.
```

This algorithm scans each row of the matrix from the diagonal element to the right. If a diagonal element has the truth value F, then the algorithm is terminated in step 2 with *FLAG* remaining *false*. Step 3 scans each row in the

Let A, B, and C be three subsets of E and let the 2^3 minterms, denoted by $I_0, I_1, \ldots, I_7$ (see Fig. 2-3.11*c*), be as follows:

$$\begin{array}{ll} I_0 = \sim A \cap \sim B \cap \sim C & I_1 = \sim A \cap \sim B \cap C \\ I_2 = \sim A \cap B \cap \sim C & I_3 = \sim A \cap B \cap C \\ I_4 = A \cap \sim B \cap \sim C & I_5 = A \cap \sim B \cap C \\ I_6 = A \cap B \cap \sim C & I_7 = A \cap B \cap C \end{array}$$

The subscript of I shows indirectly the minterm under consideration. In order to obtain the minterm, first we write the subscript as a binary integer containing three digits (since there are three subsets under consideration). The appearance of 1 or 0 in the first position on the left indicates the presence of A or $\sim A$, respectively. This relation also holds for the second and third positions. The notation is similar to the one used in Secs. 1-3.5 and 2-1.3. For example, $I_5 = A \cap \sim B \cap C$ since 5 written as a binary integer is 101.

In general, if $A_1, A_2, \ldots, A_n$ are any n subsets of the universal set E, then the complete intersections or minterms generated by these n subsets are denoted by $I_0, I_1, \ldots, I_{2^n-1}$ (see Sec. 2-1.3). These are mutually disjoint and are such that

$$E = \bigcup_{j=0}^{2^n-1} I_j$$

One can recognize a similarity between the minterms defined here and those given in the statement calculus. We shall return to a general discussion of this in Chap. 4.

EXERCISES 2-3.4

1 Define a well-formed formula of set theory in the same manner as in the definition given in Sec. 1-2.7, using the operators $\cap$, $\cup$, and $\sim$ only.

2 Show that for any formula in set theory involving set variables A and B and the operations $\cap$, $\cup$, and $\sim$, one can obtain another formula which is equal to the given formula and which contains the union of minterms only.

3 Show that the set of operations $\{\cup, \sim\}$ is functionally complete for formulas in set theory (*Hint*: Follow the same procedure used in Sec. 1-2.13).

4 Write the duals of minterms and discuss some of their important properties.

2-3.5 Equivalence Relations

Definition 2-3.9 A relation R in a set X is called an *equivalence relation* if it is reflexive, symmetric, and transitive.

If R is an equivalence relation in a set X, then $D(R)$, the domain of R, is X itself. Therefore R will be called a relation on X. The following are some examples of equivalence relations.

1 Equality of numbers on a set of real numbers

2 Equality of subsets of a universal set

there is a set $C_1 \in C$ such that $x \in C_1$; also x does not belong to any other element of C. We now take all the elements of $C_1 \times C_1$ as members of a relation R. Thus every element of X that is in C_1 is an R-relative of every other member of C_1. Furthermore, no other member of X which is not in C_1 is related to the elements of C_1. Similarly, for every other member of the partition C, we form members of the relation R. If $C = \{C_1, C_2, C_3, \ldots, C_m\}$, then $R = (C_1 \times C_1) \cup (C_2 \times C_2) \cup \cdots \cup (C_m \times C_m)$. It is easy to see that R is an equivalence relation. Thus for every partition C we can define an equivalence relation.

EXAMPLE 5 Let $X = \{a, b, c, d, e\}$ and let $C = \{\{a, b\}, \{c\}, \{d, e\}\}$. Show that the partition C defines an equivalence relation on X.

SOLUTION

$R = \{\langle a, a\rangle, \langle b, b\rangle, \langle a, b\rangle, \langle b, a\rangle, \langle c, c\rangle, \langle d, d\rangle, \langle e, e\rangle, \langle d, e\rangle, \langle e, d\rangle\}$ ////

It has been shown that an equivalence relation on a set generates a partition of the set, and conversely. It may happen that two relations, which may have been defined in different ways, generate the same partition. Since a relation is a set, any two relations consisting of equal sets are indistinguishable for our purpose. This statement will be true of every partition of the set as well. The following serves as an illustration.

Let $X = \{1, 2, \ldots, 9\}$ and $R_1 = \{\langle x, y\rangle \mid x \in X \wedge y \in X \wedge (x - y)$ is divisible by $3\}$. Further, let

$R_2 = \{\langle x, y\rangle \mid x \in X \wedge y \in X$ and x, y are in same column of matrix A$\}$

where

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Although R_1 and R_2 have been defined differently, $R_1 = R_2$.

In Sec. 2-3.3 we have already given algorithms to determine whether a given relation R on a set is an equivalence relation. Once it is determined, our next task is to obtain the equivalence classes. Before giving an algorithm for this purpose, let us discuss the technique that will be used for the representation of the equivalence classes in the algorithm.

Given a set $\{1, 2, \ldots, n\}$ and an equivalence relation R on it, the equivalence classes can be represented by means of two vectors, each having n elements. These vectors are called *FIRST* and *MEMBER*. The ith component of *FIRST* for $1 \leq i \leq n$ contains the number which is the first element in the equivalence class to which i belongs. The ith component of *MEMBER* contains the number which follows i in the equivalence class, unless i is the last element, in which case *MEMBER*[i] is equal to zero.

As an example, let the set be $\{1, 2, 3, 4, 5, 6\}$ and the equivalence classes be $\{1, 3, 6\}$, $\{2\}$, and $\{4, 5\}$. The vectors *FIRST* and *MEMBER* representing these equivalence classes are shown in Fig. 2-3.14.

```
                  RELATION MATRIX

     1   2   3   4   5   6   7   8   9  10
 1   T   T   F   F   F   F   F   T   F   F
 2   T   T   F   F   F   F   F   T   F   F
 3   F   F   T   F   F   F   F   F   F   F
 4   F   F   F   T   F   T   T   F   F   F
 5   F   F   F   F   T   F   F   F   F   F
 6   F   F   F   T   F   T   T   F   F   F
 7   F   F   F   T   F   T   T   F   F   F
 8   T   T   F   F   F   F   F   T   F   F
 9   F   F   F   F   F   F   F   F   T   F
10   F   F   F   F   F   F   F   F   F   T

     THE RELATION IS AN EQUIVALENCE RELATION.

     EQUIVALENCE CLASSES
        1  2  8
        3
        4  6  7
        5
        9
        10

     FIRST       1   1   3   4   5   4   4   1   9  10

     MEMBER      2   8   0   6   0   7   0   0   0   0
```

FIGURE 2-3.16*b*.

Then R is a compatibility relation, and x, y are called compatible if $x \, R \, y$. A compatibility relation is sometimes denoted by $\approx$. Note that ball $\approx$ bed, bed $\approx$ egg, but ball $\not\approx$ egg. Thus $\approx$ is not transitive. Denoting "ball" by x_1, "bed" by x_2, "dog" by x_3, "let" by x_4, and "egg" by x_5, the graph of $\approx$ is given in Fig. 2-3.17*a*.

Since $\approx$ is a compatibility relation, it is not necessary to draw the loops at each element nor is it necessary to draw both $x \, R \, y$ and $y \, R \, x$. Thus we can simplify the graph of $\approx$, as shown in Fig. 2-3.17*b*. Note that the elements in each of the sets $\{x_1, x_2, x_4\}$ and $\{x_2, x_3, x_5\}$ are related to each other, i.e., the elements are mutually compatible. Further, these two sets define a covering of X. The set $\{x_2, x_4, x_5\}$ also has elements compatible to each other.

The relation matrix of a compatibility relation is symmetric and has its diagonal elements unity. It is, therefore, sufficient to give only the elements of the lower triangular part of the relation matrix in such a case. For the compatibility relation we have been discussing, the relation matrix can be obtained from Table 2-3.2.

2-3.7 Composition of Binary Relations

Since a binary relation is a set of ordered pairs, the usual operations such as union, intersection, etc., on these sets produce other relations. This topic was discussed in Sec. 2-3.1. We shall now consider another operation on relations—relations which are formed in two or more stages. Familiar examples of such relations are the relation of being a nephew or a brother's or sister's son, the relation of an uncle or a father's or mother's brother, and the relation of being a grandfather which is a father's or mother's father. These relations can be produced in the following manner.

Definition 2-3.13 Let R be a relation from X to Y and S be a relation from Y to Z. Then a relation written as $R \circ S$ is called a *composite relation* of R and S where

$$R \circ S = \{\langle x, z\rangle \mid x \in X \wedge z \in Z \wedge (\exists y)(y \in Y \wedge \langle x, y\rangle \in R \wedge \langle y, z\rangle \in S\}$$

The operation of obtaining $R \circ S$ from R and S is called *composition* of relations.

Note that $R \circ S$ is empty if the intersection of the range of R and the domain of S is empty. $R \circ S$ is nonempty if there is at least one ordered pair $\langle x, y\rangle \in R$ such that the second member $y \in Y$ of the ordered pair is a first member in an ordered pair in S. For the relation $R \circ S$, the domain is a subset of X and the range is a subset of Z. In fact, the domain is a subset of the domain of R, and its range is a subset of the range of S. From the graphs of R and S one can easily construct the graph of $R \circ S$. As an example, see Fig. 2-3.21.

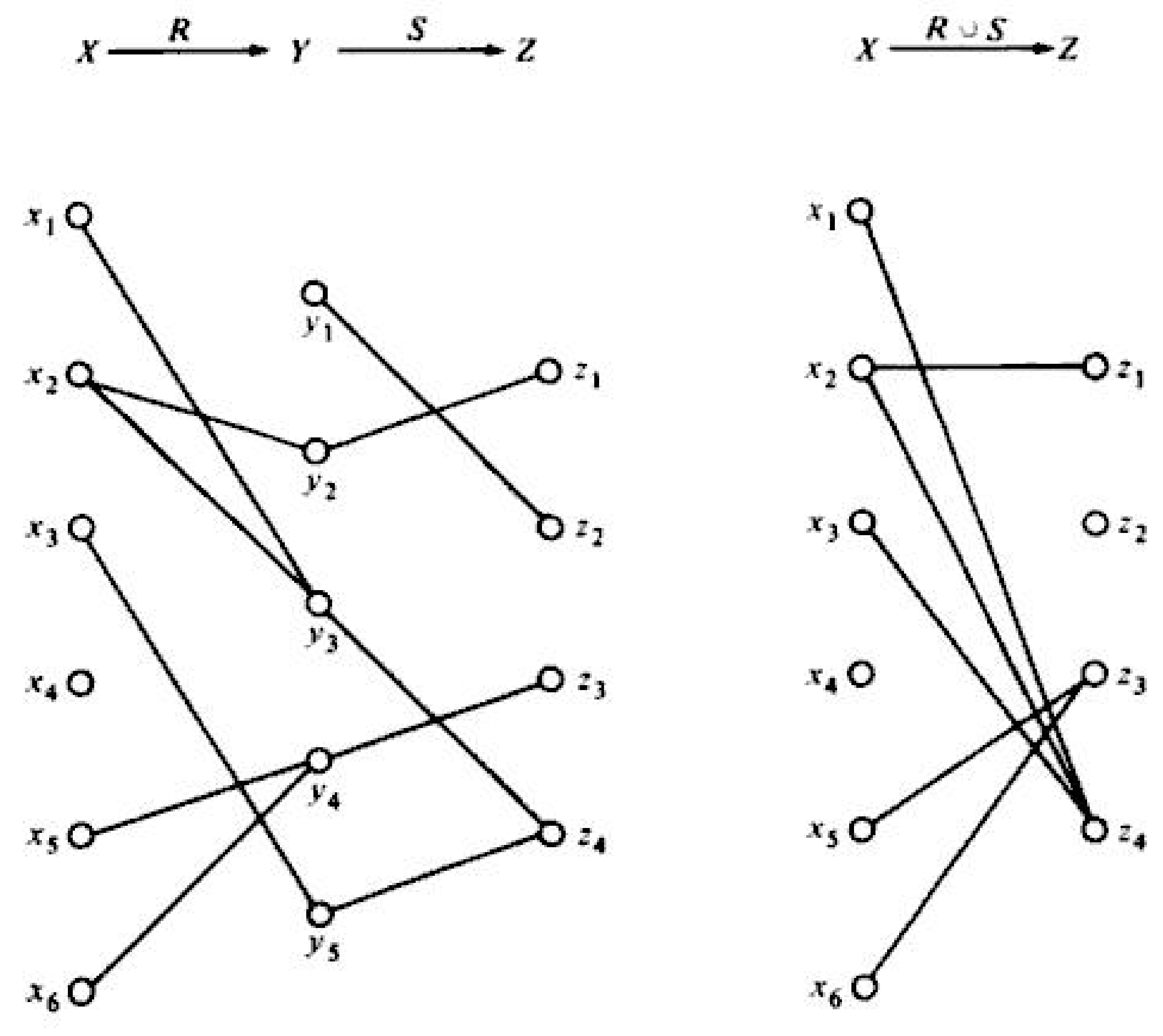


FIGURE 2-3.21 Relations R, S, and $R \circ S$.

SOLUTION

$$M_{\tilde{R}} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \text{transpose of } M_R$$

$$M_{\tilde{S}} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \text{transpose of } M_S$$

$$M_{R\circ S} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad M_{\widetilde{R\circ S}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$M_{\tilde{S}\circ\tilde{R}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} = M_{\widetilde{R\circ S}}$$

////

The following hold for any relations R and S.

1 $\tilde{\tilde{R}} = R$
2 $R = S \Leftrightarrow \tilde{R} = \tilde{S}$
3 $R \subseteq S \Leftrightarrow \tilde{R} \subseteq \tilde{S}$
4 $R \mathbin{\tilde{\cup}} S = \tilde{R} \cup \tilde{S}$
5 $R \mathbin{\tilde{\cap}} S = \tilde{R} \cap \tilde{S}$

We shall leave the proofs as exercises.

Let us now consider some distinct relations R_1, R_2, R_3, R_4 in a set

It is easy to write the elements of the relation $\subseteq$. Note that $\{a\}$ and $\{b, c\}$, $\{a, b\}$ and $\{a, c\}$, etc., are incomparable.

3 Divides and Integral Multiple: If a and b are positive integers, then we say "a divides b," written $a \mid b$, iff there is an integer c such that $ac = b$. Alternatively, we say that "b is an *integral multiple of* a." The relation "*divides*" is a partial order relation. Let X be the set of positive integers. The relations "divides" and "integral multiple of" are partial orderings on X, and each is the converse of the other.

As a special case, let $X = \{2, 3, 6, 8\}$ and let $\leq$ be the relation "divides" on X. Then

$$\leq \; = \{\langle 2, 2\rangle, \langle 3, 3\rangle, \langle 6, 6\rangle, \langle 8, 8\rangle, \langle 2, 8\rangle, \langle 2, 6\rangle, \langle 3, 6\rangle\}$$

The relation "integral multiple of," written as $\geq$, is given by

$$\geq \; = \{\langle 2, 2\rangle, \langle 3, 3\rangle, \langle 6, 6\rangle, \langle 8, 8\rangle, \langle 8, 2\rangle, \langle 6, 2\rangle, \langle 6, 3\rangle\}$$

4 Lexicographic Ordering: A useful example of simple or total ordering is the lexicographic ordering. We shall define it for certain ordered pairs first and then generalize it.

Let R be the set of real numbers and let $P = R \times R$. The relation $\geq$ on R is assumed to be the usual relation of "greater than or equal to." For any two ordered pairs $\langle x_1, y_1\rangle$ and $\langle x_2, y_2\rangle$ in P, we define the total ordering relation S as follows:

$$\langle x_1, y_1\rangle \, S \, \langle x_2, y_2\rangle \Leftrightarrow (x_1 > x_2) \vee ((x_1 = x_2) \wedge (y_1 \geq y_2))$$

It is clear that if $\langle x_1, y_1\rangle \not S \langle x_2, y_2\rangle$, then we must have $\langle x_2, y_2\rangle \, S \, \langle x_1, y_1\rangle$, so that S is a total ordering on P. The partial ordering S is called the *lexicographic ordering*. The significance of the terminology will become clear after we generalize the above ordering relation. The following are some of the ordered pairs of P which are S-related:

$$\langle 2, 2\rangle \, S \, \langle 2, 1\rangle$$
$$\langle 3, 1\rangle \, S \, \langle 1, 5\rangle$$
$$\langle 2, 2\rangle \, S \, \langle 2, 2\rangle$$
$$\langle 3, 2\rangle \, S \, \langle 1, 1\rangle$$

We now generalize this concept. For this purpose, let R be a total ordering relation on a set X and let

$$P = X \cup X^2 \cup X^3 \cup \cdots \cup X^n = \bigcup_i X^i \qquad (n = 1, 2, 3, \ldots)$$

This equation means that the set P consists of strings of elements of X of length less than or equal to n. We may assume some fixed value of n. A string of length p may be considered as an ordered p-tuple. We now define a total ordering S on P called lexicographic ordering. For this purpose, let $\langle u_1, u_2, \ldots, u_p\rangle$ and $\langle v_1, v_2, \ldots, v_q\rangle$, with $p \leq q$, be any two elements of P. Note that before starting, to compare two strings to determine the ordering in P, the strings are interchanged if necessary so that $p \leq q$. Now

$$\langle u_1, u_2, \ldots, u_p\rangle \, S \, \langle v_1, v_2, \ldots, v_q\rangle$$

EXAMPLE 2 Let A be a given finite set and $\rho(A)$ its power set. Let $\subseteq$ be the inclusion relation on the elements of $\rho(A)$. Draw Hasse diagrams of $\langle \rho(A), \subseteq \rangle$ for (*a*) $A = \{a\}$; (*b*) $A = \{a, b\}$; (*c*) $A = \{a, b, c\}$; (*d*) $A = \{a, b, c, d\}$.

SOLUTION The required Hasse diagrams are given in Fig. 2-3.25*a* to *d*. ////

The following points may be noted about Hasse diagrams in general. For a given partially ordered set, a Hasse diagram is not unique, as can be seen from Fig. 2-3.25*b*. From a Hasse diagram of $\langle P, \leq \rangle$, the Hasse diagram of $\langle P, \geq \rangle$, which is the dual of $\langle P, \leq \rangle$, can be obtained by rotating the diagram through 180° so that the points at the top become the points at the bottom. Some Hasse diagrams have a unique point which is above all the other points, and similarly some Hasse diagrams have a unique point which is below all other points. Such was the case for all the Hasse diagrams given in Example 2, while the Hasse diagram given in Example 1 does not possess this property. The Hasse diagrams become more complicated when the number of elements in the partially ordered set is large.

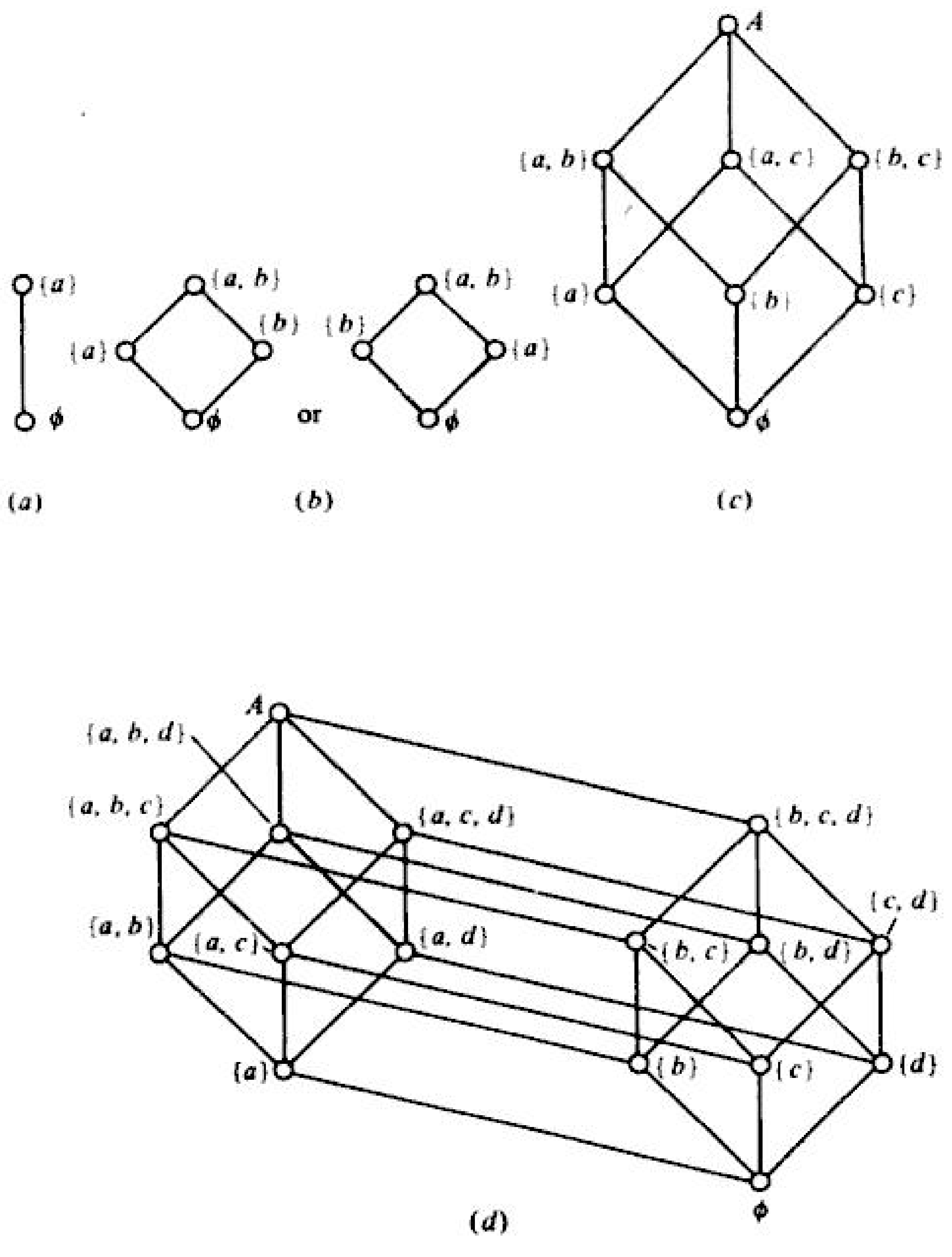


FIGURE 2-3.25 Hasse diagrams of $\langle \rho(A), \subseteq \rangle$.

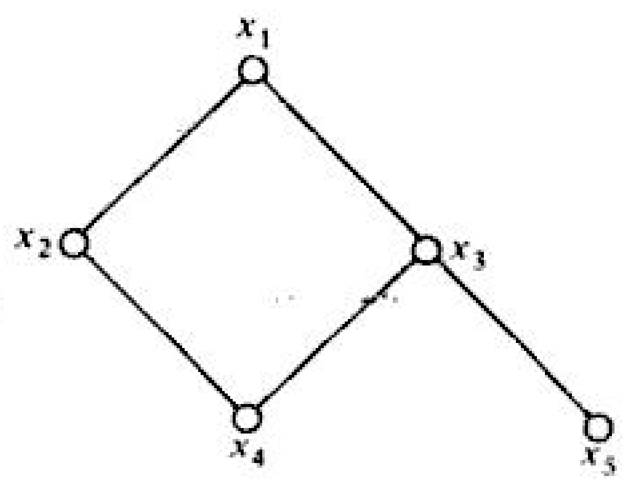


FIGURE 2-3.28

3 Give an example of a set X such that $\langle \rho(X), \subseteq \rangle$ is a totally ordered set.

4 Give a relation which is both a partial ordering relation and an equivalence relation on a set.

5 Let S denote the set of all the partial ordering relations on a set P. Define a partial ordering relation on S and interpret this relation in terms of the elements of P.

6 Figure 2-3.28 gives the Hasse diagram of a partially ordered set $\langle P, R \rangle$, where $P = \{x_1, x_2, \ldots, x_5\}$. Find which of the following are true: $x_1 \, R \, x_2$, $x_4 \, R \, x_1$, $x_3 \, R \, x_5$, $x_2 \, R \, x_5$, $x_1 \, R \, x_1$, $x_2 \, R \, x_3$, and $x_4 \, R \, x_5$. Find the least and greatest members in P if they exist. Also find the maximal and minimal elements of P. Find the upper and lower bounds of $\{x_2, x_3, x_4\}$, $\{x_3, x_4, x_5\}$, and $\{x_1, x_2, x_3\}$. Also indicate the LUB and GLB of these subsets if they exist.

7 Show that there are only five distinct Hasse diagrams for partially ordered sets that contain three elements.

2-4 FUNCTIONS

In this section we study a particular class of relations called functions. We are primarily concerned with discrete functions which transform a finite set into another finite set. There are several such transformations involved in the computer implementation of any program. Computer output can be considered as a function of the input. A compiler transforms a program into a set of machine language instructions (the object program). After introducing the concept of function in general, we discuss unary and binary operations which form a class of functions. Such operations have important applications in the study of algebraic structures in Chaps. 3 and 4. Also discussed is a special class of functions known as hashing functions that are used in organizing files on a computer, along with other techniques associated with such organizations. A PL/I program for the construction of a symbol table is also given.

2-4.1 Definition and Introduction

Definition 2-4.1 Let X and Y be any two sets. A relation f from X to Y is called a *function* if for every $x \in X$ there is a unique $y \in Y$ such that $\langle x, y \rangle \in f$.

Note that the definition of function requires that a relation must satisfy two additional conditions in order to qualify as a function. The first condition is that every $x \in X$ must be related to some $y \in Y$, that is, the domain of f must

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

3

ALGEBRAIC STRUCTURES

INTRODUCTION

In this chapter we shall first explain what is meant by an algebraic system and then give several examples of familiar algebraic systems and discuss some of their properties. These examples show that different algebraic systems may have several properties in common. This observation provides a motivation for the study of abstract algebraic systems. For such algebraic systems, certain properties are taken as axioms of the system. Any result that is valid for an abstract system holds for all those algebraic systems for which the axioms are true.

Throughout the chapter we shall introduce certain important and useful concepts associated with algebraic systems. For example, the concept of isomorphism shows that two algebraic systems which are isomorphic to one another are structurally indistinguishable and that the results of operations in one system can be obtained from those of the other by simply relabeling the names of the elements and symbols for operations. This concept has useful applications in the sense that the results of one system permit an identical interpretation in the other system. Another important concept is that of a congruence relation which has a useful property known as substitution.

with any one system will also be true for the other system after the labels are changed. We shall now formalize these ideas for any two algebraic systems.

Definition 3-1.1 Let $\langle X, \circ\rangle$ and $\langle Y, *\rangle$ be two algebraic systems of the same type in the sense that both $\circ$ and $*$ are binary (n-ary) operations. A mapping $g: X \to Y$ is called a *homomorphism*, or simply morphism, from $\langle X, \circ\rangle$ to $\langle Y, *\rangle$ if for any $x_1, x_2 \in X$

$$g(x_1 \circ x_2) = g(x_1) * g(x_2) \tag{2}$$

If such a function g exists, then it is customary to call $\langle Y, *\rangle$ a homomorphic image of $\langle X, \circ\rangle$, although we must note that $g(X) \subseteq Y$.

The concept of homomorphism is not restricted to algebraic systems with one binary operation. One can extend this definition to any two algebraic systems of the same type. Since in a homomorphism the operations are preserved, we shall see that several properties of the operations are also preserved.

For the algebraic systems $\langle F, \circ\rangle$ and $\langle \mathbf{Z}_4, +_4\rangle$, the mapping $\psi: F \to \mathbf{Z}_4$ given by Eq. (1) is a homomorphism. Any mapping which satisfies the condition given by Eq. (2) is a homomorphism. In the example of the algebraic systems $\langle F, \circ\rangle$ and $\langle \mathbf{Z}_4, +_4\rangle$, the mapping is bijective, which is a special case of homomorphism as can be seen from Definition 3-1.2 which follows. It is possible to have more than one homomorphic mapping from one algebraic system to another.

Definition 3-1.2 Let g be a homomorphism from $\langle X, \circ\rangle$ to $\langle Y, *\rangle$. If $g: X \to Y$ is onto, then g is called an *epimorphism*. If $g: X \to Y$ is one-to-one, then g is called a *monomorphism*. If $g: X \to Y$ is one-to-one onto, then g is called an *isomorphism*.

Definition 3-1.3 Let $\langle X, \circ\rangle$ and $\langle Y, *\rangle$ be two algebraic systems of the same type. If there exists an isomorphic mapping $g: X \to Y$, then $\langle X, \circ\rangle$ and $\langle Y, *\rangle$ are said to be *isomorphic*.

In the case when $\langle X, \circ\rangle$ and $\langle Y, *\rangle$ are isomorphic, then the two algebraic systems are structurally indistinguishable in the sense that they differ only in the labels used to denote the elements of the sets and the operations involved. It is easy to see that the inverse of an isomorphism is also an isomorphism. Also all the properties of the operations are preserved in an isomorphism.

Definition 3-1.4 Let $\langle X, \circ\rangle$ and $\langle Y, *\rangle$ be two algebraic systems such that $Y \subseteq X$. A homomorphism g from $\langle X, \circ\rangle$ to $\langle Y, *\rangle$ in such a case is called an *endomorphism*. If $Y = X$, then an isomorphism from $\langle X, \circ\rangle$ to $\langle Y, *\rangle$ is called an *automorphism*.

EXAMPLE 5 Show that the algebraic systems of $\langle F, \circ\rangle$ and $\langle \mathbf{Z}_4, +_4\rangle$ given in Examples 3 and 4 are isomorphic.

SOLUTION The mapping $\psi: F \to \mathbf{Z}_4$ defined by Eq. (1) is one-to-one onto and is a homomorphism; hence ψ is an isomorphism. ////

3-2.2 Homomorphism of Semigroups and Monoids

The concept of homomorphism for algebraic systems was introduced in Sec. 3-1.2. Now we apply this concept to semigroups and monoids. Homomorphisms of semigroups and monoids have useful applications in the economical design of sequential machines and in formal languages.

Definition 3-2.3 Let $\langle S, *\rangle$ and $\langle T, \Delta\rangle$ be any two semigroups. A mapping $g\colon S \to T$ such that for any two elements $a, b \in S$,

$$g(a * b) = g(a) \,\Delta\, g(b) \tag{1}$$

is called a *semigroup homomorphism.*

As before, a semigroup homomorphism is called a semigroup monomorphism, epimorphism, or isomorphism depending on whether the mapping is one-to-one, onto, or one-to-one onto respectively. Two semigroups $\langle S, *\rangle$ and $\langle T, \Delta\rangle$ are said to be isomorphic if there exists a semigroup isomorphic mapping from S to T.

We have already seen in Example 8, Sec. 3-1.2, that there exists a semigroup homomorphism g from $\langle \mathbf{N}, +\rangle$ to $\langle \mathbf{Z}_m, +_m\rangle$ in which the identity of $\langle \mathbf{N}, +\rangle$ is mapped into the identity $[0]$ of $\langle \mathbf{Z}_m, +_m\rangle$.

Let us now examine some of the implications of Eq. (1). For this purpose, let us assume that $\langle S, *\rangle$ is a semigroup and $\langle T, \Delta\rangle$ is an algebraic structure of the same type, that is, Δ is a binary operation on T but it is not necessarily associative. If there exists an onto mapping $g\colon S \to T$ such that for any $a, b \in S$ Eq. (1) is satisfied, then we can show that Δ must be associative and hence $\langle T, \Delta\rangle$ must be a semigroup. In order to see this, let $a, b, c \in S$

$$\begin{aligned} g((a * b) * c) &= g(a * b) \,\Delta\, g(c) \\ &= (g(a) \,\Delta\, g(b)) \,\Delta\, (g(c)) \end{aligned}$$

On the other hand,

$$g(a * (b * c)) = g((a * b) * c)$$

but $g(a * (b * c))$ can be shown to be equal to $g(a) \,\Delta\, (g(b) \,\Delta\, g(c))$ by a similar argument. Hence Δ is associative, and $\langle T, \Delta\rangle$ must be a semigroup. This result shows that Eq. (1) preserves the semigroup character because it preserves associativity.

Next, note that if g is a semigroup homomorphism from $\langle S, *\rangle$ to $\langle T, \Delta\rangle$, then for any element $a \in S$ which is idempotent, we must have $g(a)$ idempotent, because

$$g(a * a) = g(a) = g(a) \,\Delta\, g(a)$$

The property of idempotency is preserved under the semigroup homomorphism. In a similar manner commutativity is also preserved.

If $\langle S, *\rangle$ is a semigroup with an identity e, that is, $\langle S, *, e\rangle$ is a monoid and g is a homomorphism from $\langle S, *\rangle$ to a semigroup $\langle T, \Delta\rangle$, then for any $a \in S$,

$$g(a * e) = g(e * a) = g(a) \,\Delta\, g(e) = g(e) \,\Delta\, g(a) = g(a)$$

cording to Theorem 3-5.1. Equation (4) can be written as

$$f(a * b) = f(a) \diamond f(b)$$

showing that f is an isomorphism. ////

Example 2 of Sec. 3-2.2 is an illustration of the representation theorem. This representation theorem is also known as Cayley's representation theorem. It was proposed by Arthur Cayley in 1854. This theorem shows that the structure of a group is determined solely by its composition table.

EXERCISES 3-5.2

1 Find all the subgroups of (*a*) $\langle \mathbf{Z}_{12}, +_{12}\rangle$; (*b*) $\langle \mathbf{Z}_5, +_5\rangle$; (*c*) $\langle \mathbf{Z}_7^*, \times_7\rangle$; and (*d*) $\langle \mathbf{Z}_{11}^*, \times_{11}\rangle$.

2 Find the group of rigid rotations of a rectangle which is not a square. Show that this is a subgroup of $\langle D_4, \diamond\rangle$ given in Table 3-5.7.

3 Find all the subgroups of S_4 generated by the permutations

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 2 & 4 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 4 & 2 \end{pmatrix}$$

4 Show that the set of all elements a of a group $\langle G, *\rangle$ such that $a * x = x * a$ for every $x \in G$ is a subgroup of G.

5 Show that if $\langle G, *\rangle$ is a cyclic group, then every subgroup of $\langle G, *\rangle$ must be cyclic.

6 Show that $\langle \{1, 4, 13, 16\}, \times_{17}\rangle$ is a subgroup of $\langle \mathbf{Z}_{17}^*, \times_{17}\rangle$.

7 Let $\langle G, *\rangle$ be a group and $a \in G$. Let $f\colon G \to G$ be given by $f(x) = a * x * a^{-1}$ for every $x \in G$. Prove that f is an isomorphism of G onto G.

8 Show that the groups $\langle G, *\rangle$ and $\langle S, \Delta\rangle$ given by the following table are isomorphic.

$*$	p_1	p_2	p_3	p_4
p_1	p_1	p_2	p_3	p_4
p_2	p_2	p_1	p_4	p_3
p_3	p_3	p_4	p_1	p_2
p_4	p_4	p_3	p_2	p_1

Δ	q_1	q_2	q_3	q_4
q_1	q_3	q_4	q_1	q_2
q_2	q_4	q_3	q_2	q_1
q_3	q_1	q_2	q_3	q_4
q_4	q_2	q_1	q_4	q_3

3-5.3 Cosets and Lagrange's Theorem

From the definition of a subgroup it is clear that not every subset of a group is a subgroup. The problem that we try to solve here is to find those subsets which can qualify to become subgroups. An important relationship exists between the subgroups and the group itself. This relationship is explained by a theorem known as Lagrange's theorem, which is proved in this section. This theorem has important applications in the development of efficient group codes required in the transmission of information. Such group codes are discussed in Sec. 3-8. Another application of subgroups is in the construction of computer modules which perform group operations. Such modules are constructed by joining various subgroup modules that do operations in subgroups. The application of these to the design of fast adders is discussed in Sec. 3-7.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

ing identities:

(C-1) $a * a' = 0$ **(C-1)′** $a \oplus a' = 1$

(C-2) $0' = 1$ **(C-2)′** $1' = 0$

(C-3) $(a * b)' = a' \oplus b'$ **(C-3)′** $(a \oplus b)' = a' * b'$

(See Definition 4-1.9 and Prob. 5, Exercises 4-1.5.)

5 There exists a partial ordering relation $\leq$ on B such that

(P-1) $a * b = \text{GLB}\,\{a, b\}$ **(P-1)′** $a \oplus b = \text{LUB}\,\{a, b\}$

(P-2) $a \leq b \Leftrightarrow a * b = a \Leftrightarrow a \oplus b = b$

(P-3) $a \leq b \Leftrightarrow a * b' = 0 \Leftrightarrow b' \leq a' \Leftrightarrow a' \oplus b = 1$

(See Theorem 4-1.1 and Prob. 6, Exercises 4-1.5.)

As pointed out earlier, not all the identities given here are independent of one another. These identities arose by looking at a Boolean algebra as a special lattice. It is possible to define a Boolean algebra as an abstract algebraic system satisfying certain properties which are independent of each other. In fact, even the two binary operations $*$ and $\oplus$, the unary operation $'$, and the two distinguished elements are not all independent. One can define a Boolean algebra in terms of the operations $*$ and $'$ and a set of independent properties satisfied by these operations. We shall not, however, concern ourselves with this approach.

EXAMPLE 1 Let $B = \{0, 1\}$ be a set. The operations $*$, $\oplus$, and $'$ on B are given by Table 4-2.1. The algebra $\langle B, *, \oplus, ', 0, 1 \rangle$ satisfies all the properties listed here and is one of the simplest examples of a two-element Boolean algebra. A two-element Boolean algebra is the only Boolean algebra whose diagram is a chain.

EXAMPLE 2 Let S be a nonempty set and $\rho(S)$ be its power set. The set algebra $\langle \rho(S), \cap, \cup, \sim, \varnothing, S \rangle$ is a Boolean algebra in which the complement of any subset $A \subseteq S$ is $\sim A = S - A$, the relative complement of the set A. If S has n elements, then $\rho(S)$ has 2^n elements and the diagram of the Boolean algebra is an n cube. The partial ordering relation on $\rho(S)$ corresponding to the operations $\cap$ and $\cup$ is the subset relation $\subseteq$. The diagrams for the Boolean algebra $\langle \rho(S), \cap, \cup \rangle$ when S has 1, 2, and 3 elements are given in Fig. 4-2.1. If S is an empty set, then $\rho(S)$ has only one element, viz., $\varnothing$, so that $\varnothing = 0 = 1$, and the corresponding Boolean algebra is a degenerate Boolean algebra. We shall consider nondegenerate Boolean algebras only.

Table 4-2.1

$*$	0	1	$\oplus$	0	1	x	x'
0	0	0	0	0	1	0	1
1	0	1	1	1	1	1	0

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

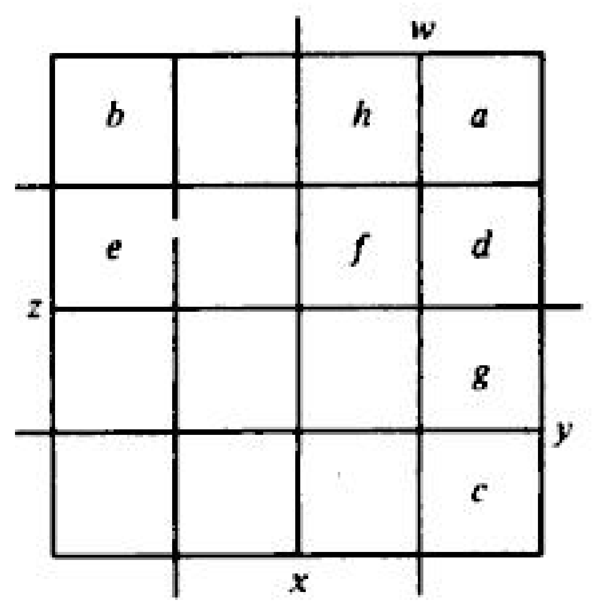


FIGURE 4-4.7

on its inner surface in the same orientation. Similar remarks hold for higher-order maps.

To this point we have discussed Karnaugh map structures, indicating what form they take for functions of different numbers of variables and describing how they are to be interpreted. To represent a Boolean function by means of a Karnaugh map, we select the Karnaugh map structure appropriate for that function and place 0s and 1s in the cells according to whether the functional value is 0 or 1 for the input combination associated with that cell. (Because one half of the map is associated with a 1 input value for a variable and the other half of the map with a 0 input value for that same variable, the input combinations are mapped one-to-one onto the cells of the Karnaugh map structure. The input combination serves as a sort of "address" for a cell within a Karnaugh map structure.) The Karnaugh map for the Boolean function $f = x_1 \cdot [x_2 + (x_3 \cdot \bar{x}_4)]$ is given in Fig. 4-4.8 (see also Fig. 4-4.5). Observe that only the functional values of 1 have been written in the appropriate cells; a blank cell is presumed to have a 0 in it. Also, for convenience, the map of the function f has had each of its cells labeled with the input combination to which it corresponds.

The last method for representing Boolean functions is one that was previously encountered in Sec. 1-2.15, namely, *circuit diagrams*. This representation does seem appropriate since Boolean functions can express the functioning of circuits. Because a circuit diagram actually shows which circuits are to be connected to which other circuits, it is occasionally possible to make use of a circuit diagram to eliminate unnecessary connectives and thus yield a simpler circuit.

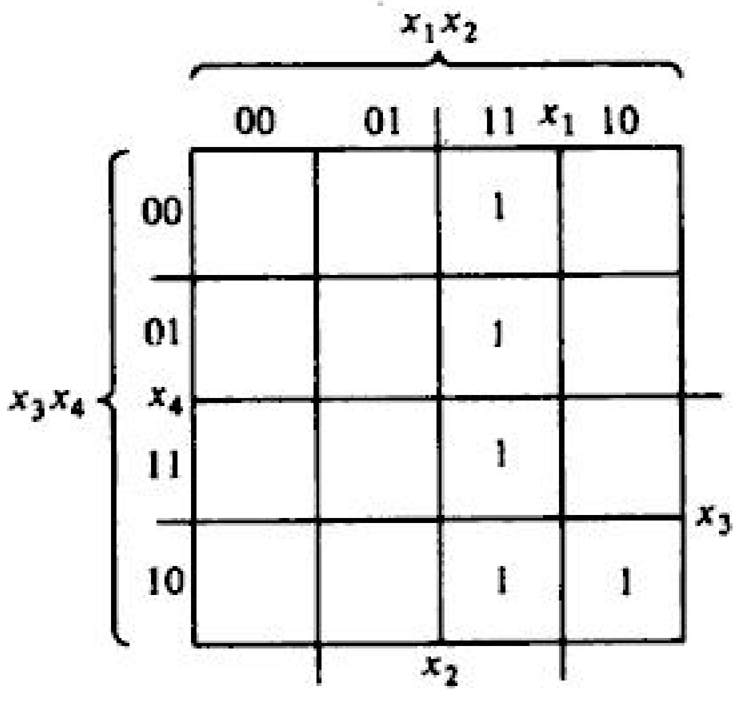


FIGURE 4-4.8

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

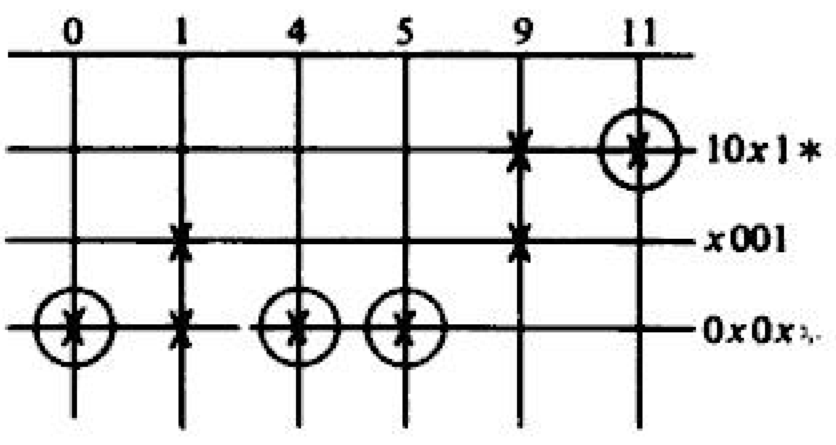


FIGURE 4-4.13 Prime implicant chart for $f = \sum(0, 1, 4, 5, 9, 11)$.

An alternate method for selecting a suitable minimum sum from the group of prime implicants is to construct a prime-implicant chart which for this problem is given in Fig. 4-4.13. The chart contains a number of columns, each of which has a number at the top that corresponds to a minterm in the sum-of-products form of the function. Each row corresponds to one of the prime implicants, as identified by 10x0, x001, and 0x0x at the right. In each row we mark a cross under each minterm contained in the prime implicant represented by that row. Thus, in the example the first prime implicant 10x1 contains or covers minterms 11 and 9. The remainder of the chart is completed in a similar manner.

The first step in using the prime-implicant chart is to examine the columns to see whether any column has exactly one cross in it. This is true for columns 0, 4, 5, and 11. We place circles around each of the crosses which stand alone in a column, and then we rule a line through all the crosses in each row containing a circled cross. The significance of this maneuver is that the particular prime implicant which has been marked is the only one which can cover the required encircled minterm. However, since it also covers all other minterms designated by a cross in the same row, no other prime implicants need be chosen to cover these minterms. A single asterisk is placed at the end of each prime implicant thus required. Such rows are called *primary basis rows*. In addition to covering the minterms under which the crosses are circled, each primary basis row covers other minterms where other crosses lie in that row. Thus row 0x0x in Fig. 4-4.13 covers not only minterms 0, 4, and 5 but also minterm 1.

We continue this process for the columns containing crosses which are in the other primary basis row, and at the end we have covered all minterms. Hence we have determined the minimum sum $\bar{x}_1\bar{x}_3 + x_1\bar{x}_2x_4$.

A general method that can be used to generate automatically all the prime implicants that cover a given set of minterms is the *Quine-McCluskey algorithm*. First, the 0 cubes are used to generate all the possible 1 cubes; then these 1 cubes are used to generate all possible 2 cubes; this process continues until the r cubes are generated for some r such that there are no $(r + 1)$ cubes. The cubes that remain after the elimination of all cubes covered by higher-order cubes are the prime implicants.

Consider the example of $g = x_1 \cdot (x_2 + x_3\bar{x}_4)$ which can be represented by the standard sum $\sum$ (10, 12, 13, 14, 15). Writing the minterms in cube form, grouping them in order of increasing number of 1s in each 0 cube, and finally applying the Quine-McCluskey algorithm to this set of ordered cubes gives the result shown in Fig. 4-4.14. The arrows in the diagram have been drawn and labeled to indicate which cubes generated the higher-order cubes. Observe that

f_1

0	1	3	7	15	16	18	19	23	31	
X	X									0000x
	X	X								000x1
		X	X				X	X		x0x11
			X	X				X	X	xx111
						X	X			1001x
					X	X				100x0
X					X					x0000

f_2

0	1	3	5	7	15	19	21	23	31	
X	X									0000x
		X				X				x0011
			X				X			x0101
			X	X						001x1
				X	X			X	X	xx111
	X	X	X	X						00$xx$1

FIGURE 4-4.20

will find all possible minimal covers of a set of 0 cubes, where minimal means that the cover does not contain redundant cubes.

The first step that should be taken in your algorithm is to determine the essential prime implicants. Then the problem reduces to finding the minimal covers of nonessential prime implicants for cubes not yet covered. A brute-force solution would be to test all combinations of nonessential prime implicants. There are other solutions to this problem however.

4-5 DESIGN EXAMPLES USING BOOLEAN ALGEBRA

In this section we illustrate how Boolean algebra is used in the design of some simple switching circuits that perform various arithmetic operations on numbers. We are concerned with fixed-length binary numbers, since in general most digital computers manipulate binary numbers of fixed length.

Initially, we give a brief discussion of arithmetic operations that can be performed on integral binary numbers. Let $a = \langle a_1, a_2, \ldots, a_n \rangle$ and $b = \langle b_1, b_2, \ldots, b_n \rangle$ denote two binary numbers. The binary addition table given in Table 4-5.1 is simple. Note, however, that in the case of two 1s, a 2-digit sequence is

```
    5         00101        00101       -9       10110
 + -9     + -01001     + 10110     + -5     + 11010
 ----      -------      -----       ----     ------
   -4      Change to    11011       -14      110000 → 10001
                                             ↑
                                             end-around carry,
                                             no overflow

    7     00111     -7      11000     -7     11000      11000
 + +9   + 01001  + -9    + 10110   - -9   - 10110    + 01001
 ----    -----    ----    ------    ----   -----      ------
   16    10000            101110       2  Change to  100001 ⇒ 00010
         ↑                ↑                          ↑
         overflow         end-around carry           end-around
                          but also overflow          carry
```

Another form by which signed binary numbers can be represented is the 2s complement representation. Again, numbers represented in this fashion have all the bits (including the sign bit) taking part in the arithmetic operation, and, as in the case of 1s complement, subtraction is unnecessary. Addition of the second operand in 2s complement form yields the same result as the subtraction of the second operand from the first operand. There are no end-around carries. The sign bit always comes out correct, except in the case of an overflow, for which no correction is possible. Except for one case, overflows are detected by checking the signs of the operands. If, when actually adding, the two operands have the same sign and the result has the opposite sign, then an overflow has occurred. The one exceptional case that cannot be detected by this method is the following: if a and b are n-bit words representing negative binary numbers (that is, 1 sign bit and $n - 1$ magnitude bits), then if $|a| + |b| = 2^{n-1}$, an unused word is generated $(100\cdots0)$ with a carry of 1 out of the sign bit. To detect this overflow, we must check for the magnitude bits being all 0 and for the carry-out bit value of 1.

To form the 2s complement representation of a number, first form the 1s complement representation of it (including the sign bit in the scope of the 1s complementation operation) and then add 1 to the low-order bit position. Another method is to copy bits from the original number, starting at the low-order end until the first 1 bit is encountered. After copying this first 1 bit, then the complements of each of the remaining bits are copied. Thus to form the 2s complement representation of +9, we have 01001 ⇒ 10110 + 00001 = 10111. Using the alternate method, we would have

```
01001 ⇒     1    (copy until first 1 is copied)
        10111    (copy complements of remaining bits)
```

Again, in the 2s complement representation, positive numbers are identical to positive numbers in sign-magnitude form or in 1s complement form. Negative numbers are represented as the result of performing a 2s complementation of the corresponding positive number of the same magnitude. Some examples of 2s com-

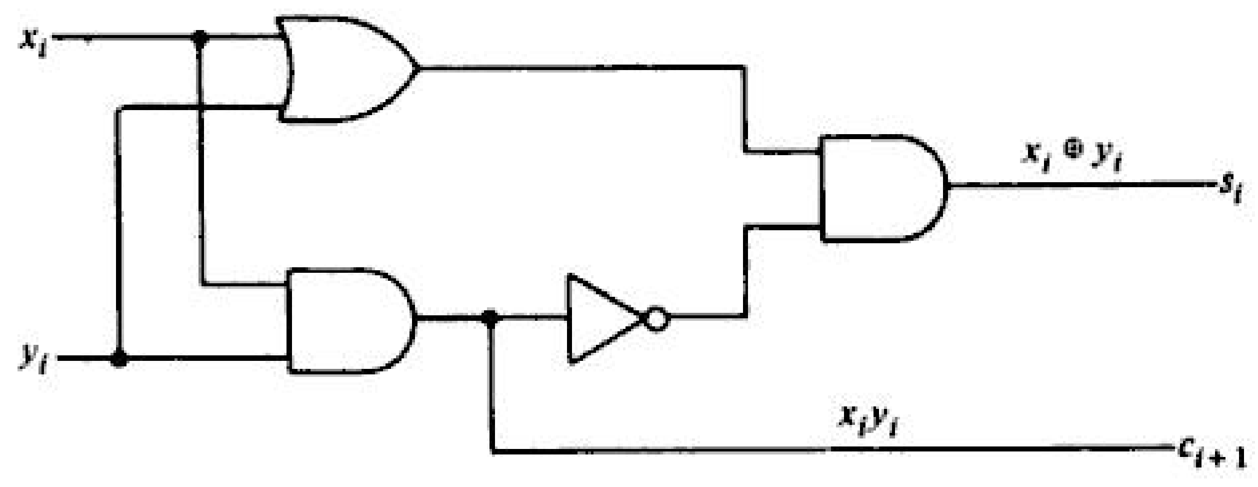


FIGURE 4-5.5 Circuit diagram for a half-adder module.

Again, we can use the circuit developed for the 1s complementer. In fact, that circuit with one extra connection is the required half-adder circuit. This circuit is given in Fig. 4-5.5.

Half-adders are not of much use by themselves in actually performing arithmetic operations because they cannot handle carry bits, and, in general, carry bits must be processed when an addition is performed. However, half-adders are very useful as components from which we can construct larger units that do perform arithmetic operations, e.g., a full-adder.

A full-adder module accepts two input data bits and a carry bit from the preceding full-adder module and generates the proper sum and carry bits. Figure 4-5.6 illustrates such a module with its interface to other modules in the iterative network. In tabular form, the requirements for a full-adder module are given in Table 4-5.6. From this table, we can construct equations as follows:

$$s_i = \bar{x}_i\bar{y}_ic_i + \bar{x}_iy_i\bar{c}_i + x_i\bar{y}_i\bar{c}_i + x_iy_ic_i$$

$$c_{i+1} = \bar{x}_iy_ic_i + x_i\bar{y}_ic_i + x_iy_i\bar{c}_i + x_iy_ic_i$$

These equations can be simplified or modified in such a manner that previous circuits, such as half-adders, can be used to construct new circuits. Formally,

$$\begin{aligned} s_i &= \bar{x}_i\bar{y}_ic_i + \bar{x}_iy_i\bar{c}_i + x_i\bar{y}_i\bar{c}_i + x_iy_ic_i \\ &= \bar{x}_i(\bar{y}_ic_i + y_i\bar{c}_i) + x_i(\bar{y}_i\bar{c}_i + y_ic_i) && \text{(using the distributive law)} \\ &= \bar{x}_i(y_i \oplus c_i) + x_i(\bar{y}_i\bar{c}_i + y_ic_i) && \text{(from the definition of } \oplus\text{)} \\ &= x_i \oplus (y_i \oplus c_i) && \text{[using Eq. (2)]} \end{aligned}$$

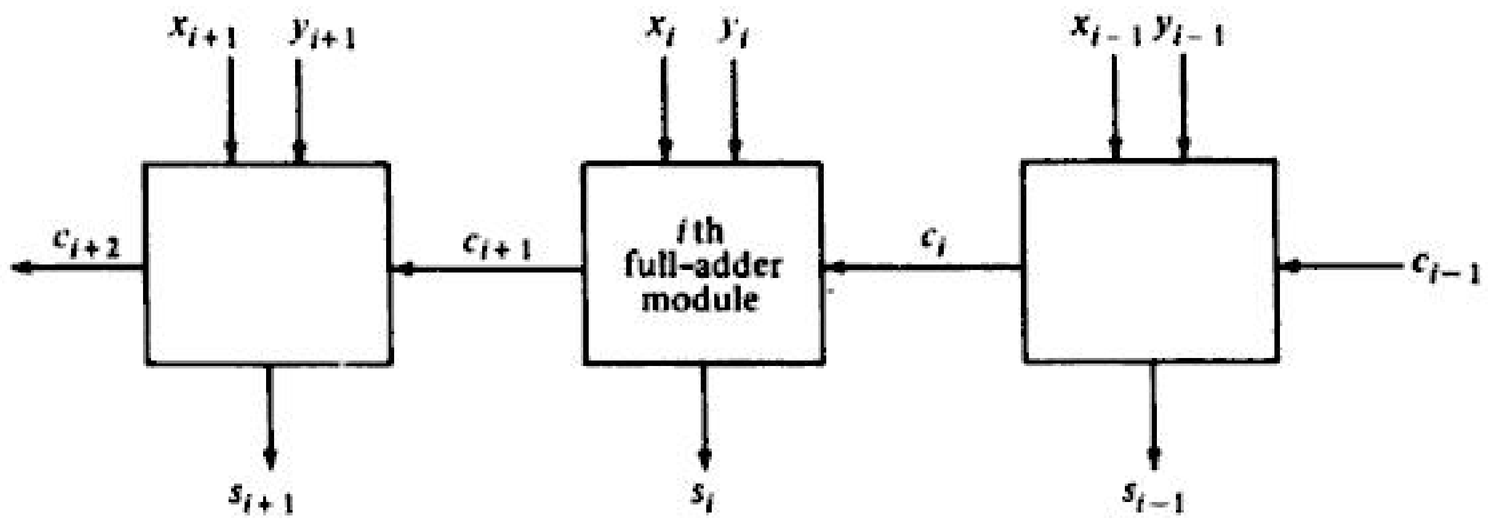


FIGURE 4-5.6 Typical module interface in a full-adder.

(1) $x - y$
(2) $x + y$
(3) $-x + y$

(*b*) Repeat part (*a*), performing subtraction by complementing and adding.
(*c*) Repeat part (*a*), using 2s complement representation of signed numbers.

3 Design a gate network that will have as input two 2-bit numbers and present the 4-bit product on four output terminals.

4 One form of an old puzzle requires a farmer (f) to transport a wolf (w), a goose (g), and a sack of corn (c) across a river, subject to the condition that if left unattended, the wolf will eat the goose, or the goose will eat the corn. Let $f = 0$ indicate the presence of the farmer on the west bank, $f = 1$ indicate his presence on the east bank, and similarly for w, g, and c.

(*a*) Write the table of combinations describing a switching circuit which has a transmission of 1 if and only if the farmer is in danger of losing the goose or corn. Assume that if an object is not on one side of the river, it must be on the other side.
(*b*) Design a two-level minimal circuit having the required transmission, and show the circuit.

5 Design a circuit which determines whether four signals on four input lines represent a valid BCD (Binary Coded Decimal) code word. The BCD code is given in Table 4-5.9.

Table 4-5.9

	Inputs			
Decimal number	*a*	*b*	*c*	*d*
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

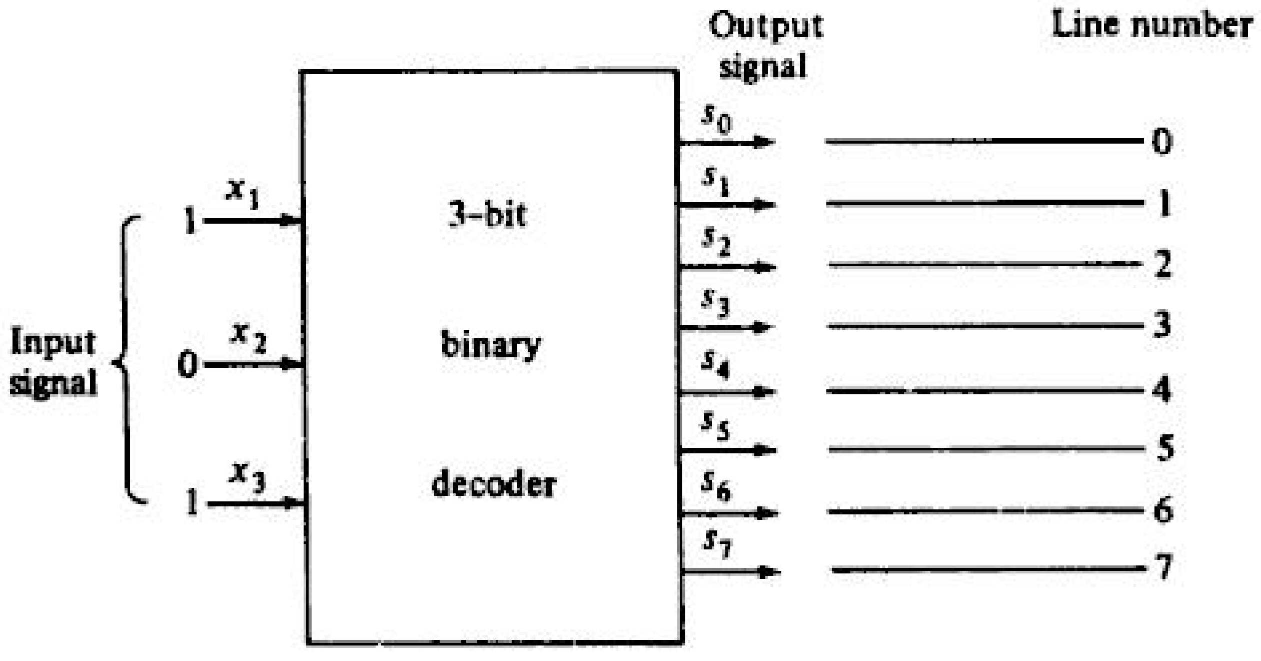


FIGURE 4-5.13 A 3-bit binary decoder.

delay element will be represented by c, and it will assume a value of 0 or 1, corresponding to the absence or presence of a carry bit. Since the present value at the input of the unit delay at time t_i is equal to its output at time t_{i+1}, this input is called the next state of the delay element. A block-diagram representation of the adder is given in Fig. 4-6.2, where an additional input line denoting the clock signal is also included. This additional input indicates that the serial adder is a synchronous circuit and that all events occur at discrete points in time.

We have given a very brief introduction to sequential circuits. In the next subsection we formalize these ideas and proceed with the development of an algorithm for determining a minimal equivalent sequential circuit for a given sequential circuit.

4-6.2 Equivalence of Finite-state Machines

In the previous subsection the basic notions of sequential circuits were introduced. We now wish to formalize these concepts by defining a finite-state machine. Furthermore, the important question of equivalent machines is discussed. In particular, we shall define what is meant by equivalent machines and show that for any given machine there exists a minimal equivalent machine. This reduced machine is homomorphic to the given machine. An algorithm for obtaining a minimal machine is developed. Finite-state machines have interesting algebraic properties based on the theory of semigroups, but we will not be concerned with such properties in this subsection. Finite-state machines can do many things, but there are certain operations such as multiplication which are beyond their range. We now proceed to the definition of a finite-state machine.

Definition 4-6.1 A *sequential machine*, or *finite-state machine*, is a system $N = \langle I, S, O, \delta, \lambda \rangle$, where the finite sets I, S, and O are alphabets that represent the input, state, and output symbols of the machine respectively. The alphabets I and O are not necessarily disjoint, but $I \cap S = O \cap S = \varnothing$. We shall denote the alphabets by

$$I = \{a_0, a_1, \ldots, a_n\} \qquad S = \{s_0, s_1, \ldots, s_m\} \qquad O = \{o_0, o_1, \ldots, o_r\}$$

δ is a mapping of $S \times I \to S$ which denotes the next-state function, and λ is a mapping $S \times I \to O$ which denotes the output function. We assume that the machine is in an initial state s_0.

Formally, a finite-state machine therefore consists of three not necessarily distinct alphabets and two functions. An abstract representation of a finite-state machine is given in Fig. 4-6.3. The machine reads a sequence of input symbols that are stored on an *input tape* and stores a sequence of output symbols on an *output tape*. Let the machine be in some state s_i and reading the input symbol a_p under its reading head. The mapping λ is then applied to s_i and a_p, thus causing the writing head to record a symbol o_k on the output tape. The function δ then causes the machine to go into state s_j. The machine proceeds to read the next input symbol and continues its operation until all symbols on the input tape are processed. Observe that the tapes are allowed to move only in one direction. In Fig. 4-6.3 the input symbols are processed from left to right. It should be

Definition 4-6.7 A finite-state machine $M = \langle I, S, O, \delta, \lambda \rangle$ is said to be *reduced* if and only if $s_i \equiv s_j$ implies that $s_i = s_j$ for all states $s_i, s_j \in S$.

In other words, a reduced finite-state machine is one in which each state is equivalent to itself and to no other. The partition of S in such a machine has all its equivalence classes consisting of a single element.

We shall now show how to construct a reduced finite-state machine M' which is equivalent to some given machine M. Let S in M be partitioned in a set of equivalence classes $[s]$ such that $P = \cup\, [s]$. Let the function ϕ be defined on the partition P such that $\phi([s]) = s'$, where s' is an arbitrary fixed element of $[s]$, called a representative. It is clear that $s' \equiv s$ in M. Let S' in M' be defined as

$$S' = \{s' \mid (\exists s)[s \in S \text{ and } \phi([s]) = s']\}$$

and let $I' = I$ and $O' = O$; that is, both machines will have the same input and output alphabets. The functions δ' and λ' are defined as follows:

$$\delta'(s', a) = \phi([\delta(s', a)])$$

and

$$\lambda'(s', a) = \lambda(s', a)$$

where s' is both in S and S'. Therefore the reduced machine is $M' = \langle I, S', O, \delta', \lambda' \rangle$.

Applying this procedure to the machine given in Table 4-6.3 gives the equivalent reduced machine in Table 4-6.4.

We shall now state a theorem without proof which shows the existence of a reduced equivalent machine.

Theorem 4-6.5 Let $M = \langle I, S, O, \delta, \lambda \rangle$ be a finite-state machine. Then there exists an equivalent machine M' with a set of states S' such that $S' \subseteq S$ and M' is reduced.

The idea of one finite-state machine simulating another is very important in a number of applications. This notion is formalized in the next definition.

Definition 4-6.8 Let $M = \langle I, S, O, \delta, \lambda \rangle$ and $M' = \langle I, S', O, \delta', \lambda' \rangle$ be two finite-state machines. Let function ϕ be a mapping from S into S'. A *finite-state homomorphism* is defined as

$$\left.\begin{aligned} \phi(\delta(s, a)) &= \delta'(\phi(s), a) \\ \lambda(s, a) &= \lambda'(\phi(s), a) \end{aligned}\right\} \quad \text{for all } a \in I$$

If ϕ is a one-one and onto function, then M ir *isomorphic* to M'.

Finite-state machines are often used in compilers where they usually perform the task of a scanner (Sec. 3-3). The machine in such a case does lower-level syntax analysis such as identifying variable names, operators, constants, etc. A machine which performs this scanning task is called an *acceptor*. In Chap. 6 we show that the set of languages that can be recognized by an acceptor is exactly the set of those languages that can be generated by a regular grammar.

5-1 BASIC CONCEPTS OF GRAPH THEORY

The terminology used in graph theory is not standard. It is not uncommon to find several different terms being used as synonyms. This situation, however, becomes more complicated when we find that a particular term is used by different authors to describe different concepts. This situation is natural because of the diversity of the fields in which graph theory is applied. Wherever possible, we shall indicate the alternative terms which are frequently used. We shall generally select alternatives that are often used in the literature in computer science.

In this section we shall define a graph as an abstract mathematical system. However, in order to provide some motivation for the terminology used and also to develop some intuitive feelings, we shall represent graphs diagrammatically. Any such diagram will also be called a graph. Our definitions and terms are not restricted to those graphs which can be represented by means of diagrams, even though this may appear to be the case because these terms have strong associations with such a representation. We shall see later on that a diagrammatic representation is only suitable in some very simple cases. Alternative methods of representing graphs will also be discussed. After introducing the terminology, we shall also discuss some of the basic results and theorems of graph theory.

5-1.1 Basic Definitions

Recall that in Chap. 2 a binary relation in a set V was defined as a subset of $V \times V$. It was shown that such a relation could be represented at least in some cases by a diagram which was called the graph of the relation. An alternative method of representing a relation was given by means of a relation or an incidence matrix. In this section we shall extend these ideas and in some ways generalize them.

We first consider several graphs which are represented by means of diagrams. Some of these graphs may be considered as graphs of certain relations, but there are others which cannot be interpreted in this manner.

Consider the diagrams shown in Fig. 5-1.1. For our purpose here, these diagrams represent graphs. Notice that every diagram consists of a set of points which are shown by dots or circles and are sometimes labeled $v_1, v_2, \ldots,$ or $1, 2, \ldots$. Also in every diagram certain pairs of such points are connected by lines or arcs. The other details, such as the geometry of the arcs, their lengths, the position of the points etc., are of no importance at present. Notice that every arc starts at one point and ends at another point. A definition of the graph which is essentially an abstract mathematical system will now be given. Such a mathematical system is an abstraction of the graphs given in Fig. 5-1.1.

Definition 5-1.1 A *graph* $G = \langle V, E, \phi \rangle$ consists of a nonempty set V called the set of *nodes* (*points, vertices*) of the graph, E is said to be the set of *edges* of the graph, and ϕ is a mapping from the set of edges E to a set of ordered or unordered pairs of elements of V

We shall assume throughout that both the sets V and E of a graph are finite. It would be convenient to write a graph G as $\langle V, E \rangle$, or simply as G. Notice

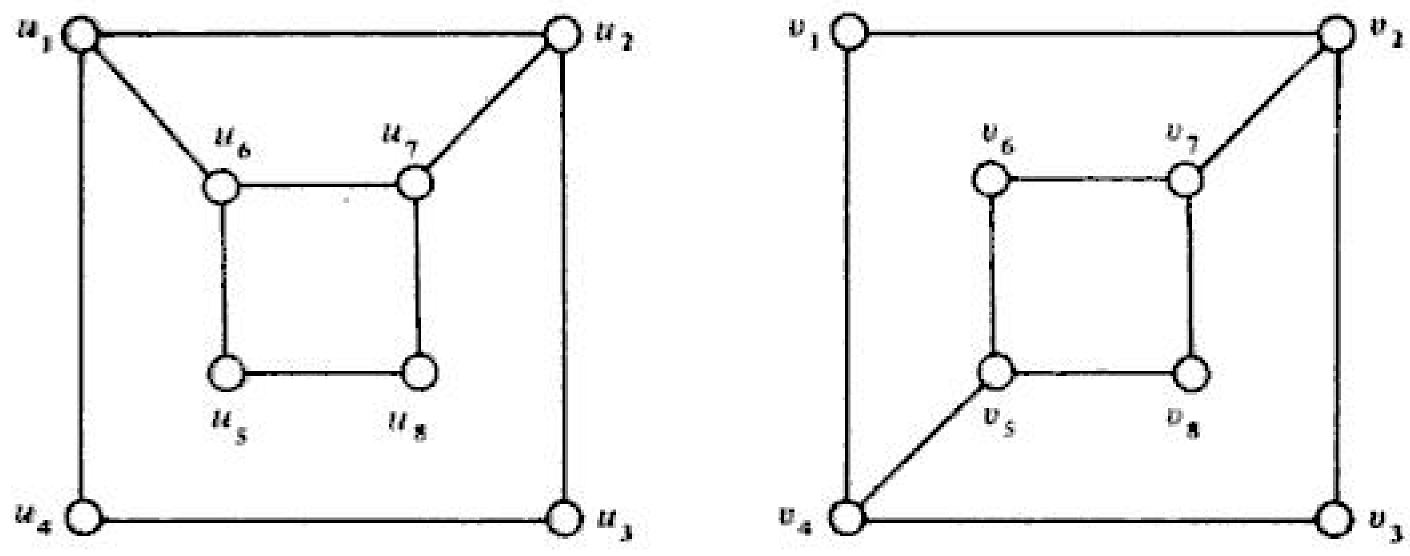


FIGURE 5-1.7

State the properties of these digraphs such as symmetry, transitivity, antisymmetry, etc.

3 Show that the digraphs given in Fig. 5-1.5*a* and *b* are isomorphic.
4 Show that the digraphs given in Fig. 5-1.6*a* and *b* are isomorphic.
5 Show that the digraphs in Fig. 5-1.7 are not isomorphic.
6 A simple digraph $G = \langle V, E \rangle$ is said to be *complete* if every node is adjacent to all other nodes of the graph. Show that a complete digraph with n nodes has the maximum number of edges viz., $n(n-1)$ edges, assuming that there are no loops.
7 The *complement* of a simple digraph $G = \langle V, E \rangle$ is the digraph $\bar{G} = \langle V, \bar{E} \rangle$ where $\bar{E} = V \times V - E$. Find the complements of the graphs given in Prob. 2.

5-1.2 Paths, Reachability, and Connectedness

In this section we introduce some additional terminology associated with a simple digraph. During the course of our discussion we shall also indicate how the same terminology and concepts can be extended to simple undirected graphs as well as to multigraphs.

Let $G = \langle V, E \rangle$ be a simple digraph. Consider a sequence of edges of G such that the terminal node of any edge in the sequence is the initial node of the next edge, if any, in the sequence. An example of such a sequence is

$$(\langle v_{i_1}, v_{i_2} \rangle, \langle v_{i_2}, v_{i_3} \rangle, \ldots, \langle v_{i_{k-2}}, v_{i_{k-1}} \rangle, \langle v_{i_{k-1}}, v_{i_k} \rangle)$$

where it is assumed that all the nodes and edges appearing in the sequence are in V and E respectively. It is customary to write such a sequence as

$$(v_{i_1}, v_{i_2}, \ldots, v_{i_{k-1}}, v_{i_k})$$

Note that not all edges and nodes appearing in a sequence need be distinct. Also, for a given graph any arbitrary set of nodes written in any order do not give a sequence as required. In fact each node appearing in the sequence must be adjacent to the nodes appearing just before and after it in the sequence, except in the case of the first and last nodes.

Definition 5-1.5 Any sequence of edges of a digraph such that the terminal node of any edge in the sequence is the initial node of the edge, if any, appearing next in the sequence defines a *path* of the graph.

A path is said to *traverse* through the nodes appearing in the sequence, *orig-*

3 Given a simple digraph $G = \langle V, E \rangle$, under what condition is the equation

$$d(v_1, v_2) + d(v_2, v_3) = d(v_1, v_3)$$

satisfied for v_1, v_2, and $v_3 \in V$?

4 Find the reachable sets of $\{v_1, v_4\}$, $\{v_4, v_5\}$, and $\{v_3\}$ for the digraph given in Fig. 5-1.14.

5 Find a node base for each of the digraphs given in Figs. 5-1.13 and 5-1.14.

6 Explain why no node in a node base is reachable from another node in the node base.

7 Prove that in an acyclic simple digraph a node base consists of only those nodes whose indegree is zero.

8 For the digraphs given in Figs. 5-1.13 and 5-1.14, determine whether they are strongly, weakly, or unilaterally connected.

9 Show that a simple digraph is strongly connected iff there is a cycle in G which includes each node at least once and no isolated node.

10 The *diameter* of a simple digraph $G = \langle V, E \rangle$ is given by δ, where

$$\delta = \max_{u,\, v \in V} d(u, v)$$

Find the diameter of the digraphs given in Figs. 5-1.13 and 5-1.14.

11 Find the strong components of the digraph given in Fig. 5-1.14. Also find its unilateral and weak components.

12 Show that every node and edge of a graph are contained in exactly one weak component.

5-1.3 Matrix Representation of Graphs

A diagrammatic representation of a graph has limited usefulness. Furthermore, such a representation is only possible when the number of nodes and edges is reasonably small. In this subsection we shall present an alternative method of representing graphs using matrices. Such a method of representation has several advantages. It is easy to store and manipulate matrices and hence the graphs represented by them in a computer. Well-known operations of matrix algebra can be used to calculate paths, cycles, and other characteristics of a graph.

Given a simple digraph $G = \langle V, E \rangle$, it is necessary to assume some kind of ordering of the nodes of the graph in the sense that a particular node is called a first node, another a second node, and so on. Our matrix representation of G depends upon the ordering of the nodes.

Definition 5-1.12 Let $G = \langle V, E \rangle$ be a simple digraph in which $V = \{v_1, v_2, \ldots, v_n\}$ and the nodes are assumed to be ordered from v_1 to v_n. An $n \times n$ matrix A whose elements a_{ij} are given by

$$a_{ij} = \begin{cases} 1 & \text{if } \langle v_i, v_j \rangle \in E \\ 0 & \text{otherwise} \end{cases}$$

is called the *adjacency matrix* of the graph G.

Recall that the adjacency matrix is the same as the relation matrix or the incidence matrix of the relation E in V. Any element of the adjacency matrix is either 0 or 1. Any matrix whose elements are either 0 or 1 is called a *bit matrix*

length from v_i to v_j. In order to decide this, with the help of the adjacency matrix we would have to consider all possible A^r for $r = 1, 2, \ldots$. This method is neither practical nor necessary, as we shall show.

Recall that in a simple digraph with n nodes, the length of an elementary path or cycle does not exceed n (see Theorem 5-1.1). Also for a path between any two nodes one can obtain an elementary path by deleting certain parts of the path which are cycles. Similarly (for cycles), we can always obtain an elementary cycle from a given cycle. If we are interested in determining whether there exists a path from v_i to v_j, all we need to examine are the elementary paths of length less than or equal to $n - 1$. In the case of $v_i = v_j$ and the path is a cycle, we need to examine all possible elementary cycles of length less than or equal to n. Such cycles or paths are easily determined from the matrix B_n where

$$B_n = A + A^2 + A^3 + \cdots + A^n$$

The element in the ith row and jth column of B_n shows the number of paths of length n or less which exist from v_i to v_j. If this element is nonzero, then it is clear that v_j is reachable from v_i. Of course, in order to determine reachability, we need to know the existence of a path, and not the number of paths between any two nodes. In any case, the matrix B_n furnishes the required information about the reachability of any node of the graph from any other node.

Definition 5-1.13 Let $G = \langle V, E \rangle$ be a simple digraph in which $|V| = n$ and the nodes of G are assumed to be ordered. An $n \times n$ matrix P whose elements are given by

$$p_{ij} = \begin{cases} 1 & \text{if there exists a path from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

is called the *path matrix* (*reachability matrix*) of the graph G.

Note that the path matrix only shows the presence or absence of at least one path between a pair of points and also the presence or absence of a cycle at any node. It does not, however, show all the paths that may exist. In this sense a path matrix does not give complete information about a graph as does the adjacency matrix. The path matrix is important in its own right.

The path matrix can be calculated from the matrix B_n by choosing $p_{ij} = 1$ if the element in the ith row and jth column of B_n is nonzero and $p_{ij} = 0$ otherwise. We shall apply this method of calculating the path matrix to our sample problem, whose graph is given in Fig. 5-1.15. The adjacency matrix $A = A_1$ and its powers A^2, A^3, A^4 have already been calculated. We thus have B_4 and the path matrix P given by

$$B_4 = \begin{pmatrix} 3 & 4 & 2 & 3 \\ 5 & 5 & 4 & 6 \\ 7 & 7 & 4 & 7 \\ 3 & 2 & 1 & 2 \end{pmatrix} \qquad P = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

matrix is given by

$$\begin{array}{ll} d_{ij} = \infty & \text{if } \langle v_i, v_j \rangle \notin E \\ d_{ii} = 0 & \text{for all } i = 1, 2, \ldots, n \\ d_{ij} = k & \text{where } k \text{ is the smallest integer for which} \\ & a_{ij}^{(k)} \neq 0 \end{array}$$

Determine the distance matrix of the digraph given in Fig. 5-1.17. What does $d_{ij} = 1$ mean?

7 Show that a digraph G is strongly connected if all the entries of the distance matrix except the diagonal entries are nonzero. How will you obtain the path matrix from a distance matrix? How will you modify the diagonal entries?

8 Modify algorithm *MINIMA* so that all minimal paths are computed.

5-1.4 Trees

An important class of diagraphs called directed trees will be introduced in this section along with the terminology associated with such trees. Trees are useful in describing any structure which involves hierarchy. Familiar examples of such structures are family trees, the decimal classification of books in a library, the hierarchy of positions in an organization, an algebraic expression involving operations for which certain rules of precedence are prescribed, etc. We shall describe here how trees can be represented by diagrams and other means. Representation of trees in a computer is discussed in Sec. 5-2.1. Applications of trees to grammars is given in Sec. 5-3.1.

Definition 5-1.14 A *directed tree* is an acyclic digraph which has one node called its *root* with indegree 0, while all other nodes have indegree 1.

Note that every directed tree must have at least one node. An isolated node is also a directed tree.

Definition 5-1.15 In a directed tree, any node which has outdegree 0 is called a *terminal node* or a *leaf*; all other nodes are called *branch nodes*. The *level* of any node is the length of its path from the root.

The level of the root of a directed tree is 0, while the level of any node is equal to its distance from the root. Observe that all the paths in a directed tree are elementary, and the length of a path from any node to another node, if such a path exists, is the distance between the nodes, because a directed tree is acyclic.

Figure 5-1.18 shows three different diagrams of a directed tree. Several other diagrams of the same tree can be drawn by choosing different relative positions of the nodes with respect to its root. The directed tree of our example has two nodes at level 1, five nodes at level 2, and three nodes at level 3. Figure 5-1.18*a* shows a natural way of representation, viz., the way a tree grows from its root up and ending in leaves at different levels. Figure 5-1.18*b* shows the same tree drawn upside down. This is a convenient way of drawing a directed tree and is commonly used in the literature. Figure 5-1.18*c* differs from *b* in the order in which the nodes appear at any level from left to right. According to our defini-

represented by a string over the alphabet $\{0, 1\}$, the root being represented by an empty string. Any son of a node u has a string which is prefixed by the string of u. The string of any terminal node is not prefixed to the string of any other node. The set of strings which correspond to terminal nodes form a *prefix* code. Thus the prefix code of the binary tree in Fig. 5-1.21*b* is $\{000, 001, 01, 10, 110, 111\}$. A similar representation of nodes of a positional m-ary tree by means of strings over an alphabet $\{0, 1, \ldots, m - 1\}$ is possible.

The string representation of the nodes of a positional binary tree immediately suggests a natural method of representing a binary tree in a computer. It is sufficient for our purpose at this stage simply to recognize that such a natural representation exists.

Binary trees are useful in several applications. We shall now show that every tree can be uniquely represented by a binary tree, so that for the computer representation of a tree it is possible to consider the representation of its

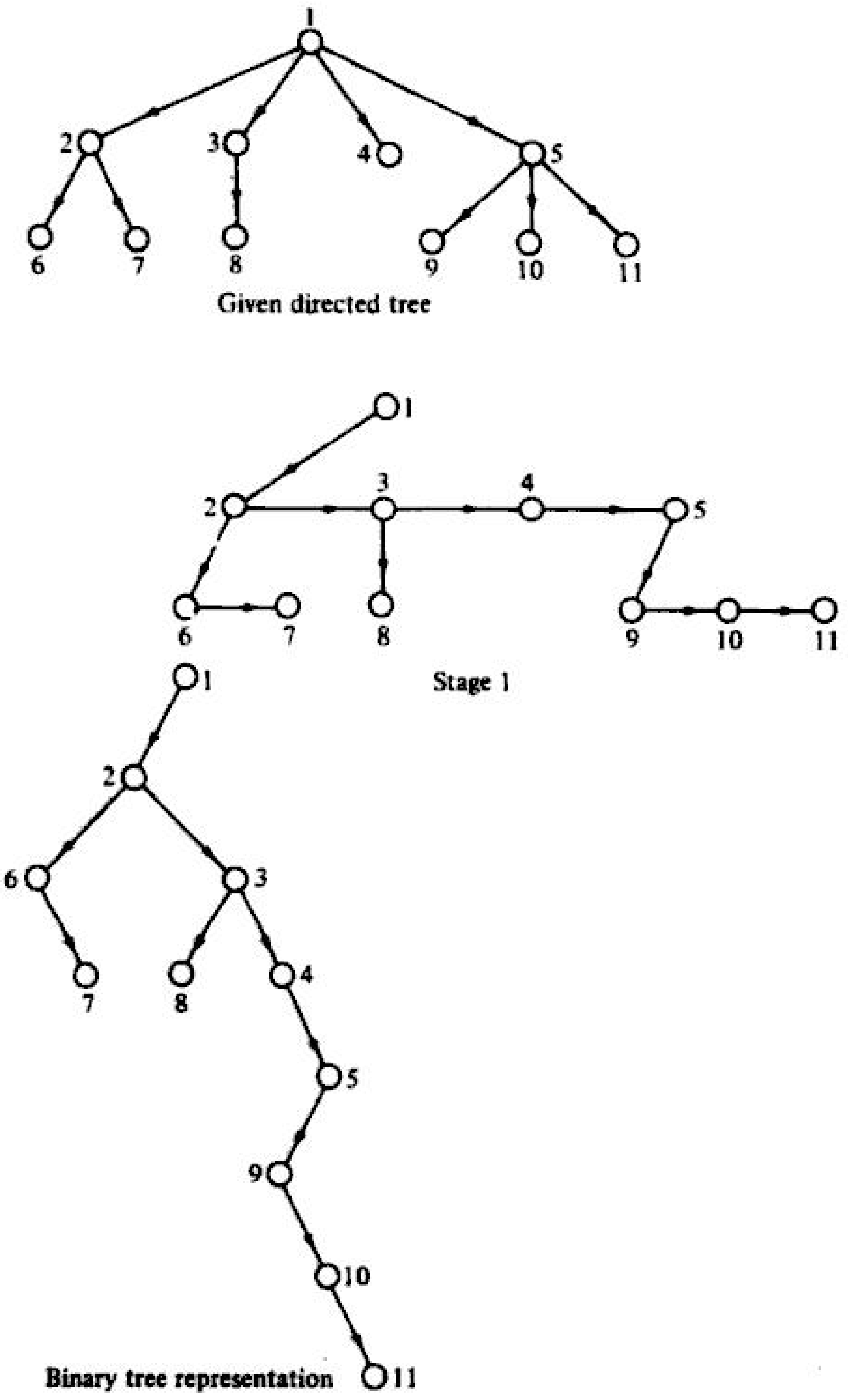


FIGURE 5-1.22

If a particular subtree is empty (i.e., when a node has no left or right descendant), the traversal is performed by doing nothing. In other words, a null subtree is considered to be fully traversed when it is encountered.

If the words "left" and "right" are interchanged in the preceding definitions, then we have three new traversal methods which are called *converse preorder*, *converse inorder*, and *converse postorder*, respectively.

The preorder, inorder, and postorder traversals of the tree given in Fig. 5-2.1 will process the nodes in the following order:

ABCDEFGH	(preorder)
CBDAEGHF	(inorder)
CDBHGFEA	(postorder)

(The respective converse traversals would be *AEFGHBDC*, *FHGEADBC*, and *HGFEDCBA*.)

Although recursive algorithms would probably be the simplest to write for the traversals of binary trees, we will formulate algorithms which are nonrecursive. Since in traversing a tree it is required to descend and subsequently ascend parts of the tree, pointer information which will permit movement up the tree must be temporarily stored. Observe that the structural information that is already present in the tree permits downward movement from the root of the tree. Because movement up the tree must be made in a reverse manner from that taken in descending the tree, a stack is required to save pointer values as the tree is traversed. We will now give an algorithm for traversing a tree in preorder.

Algorithm *PREORDER* Given a binary tree whose root node address is given by a variable T and whose node structure is the same as previously described, this algorithm traverses the tree in preorder. An auxiliary stack S is used, and TOP is the index of the top element of S. P is a temporary variable which denotes where we are in the tree.

1 [Initialize] If $T = NULL$, then Exit (the tree has no root and therefore is not a proper binary tree); otherwise set $P \leftarrow T$ and $TOP \leftarrow 0$.

2 [Visit node, stack right branch address, and go left] Process node P. If $RLINK(P) \neq NULL$, then set $TOP \leftarrow TOP + 1$ and $S[TOP] \leftarrow RLINK(P)$. Set $P \leftarrow LLINK(P)$.

3 [End of chain?] If $P \neq NULL$, then go to step 2.

4 [Unstack a right branch address] If $TOP = 0$, then Exit; otherwise set $P \leftarrow S[TOP]$, $TOP \leftarrow TOP - 1$, and go to step 2.

In the second and third steps of the algorithm, we visit and process a node. The address of the right branch of such a node, if it exists, is stacked, and a chain of left branches is followed until this chain ends. At this point, we enter step 4 and delete from the stack the address of the root node of the most recently encountered right subtree and process it according to steps 2 and 3. A trace of the algorithm for the binary tree given in Fig. 5-2.1 appears in Table 5-2.1, where the rightmost element in the stack is considered to be its top element and the

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

in preorder, while the suffix form is generated by an inorder traversal of the binary tree (not a postorder traversal!).

As an application of binary trees, we will formulate an algorithm that will maintain a tree-structured symbol table. (See Sec. 2-4.6.) One of the criteria that a symbol table routine must meet is that table searching be performed efficiently. This requirement originates in the compilation phase where many references to the entries of a symbol table are made. The two required operations that must be performed on a symbol table are insertion and "look-up," each of which involves searching. A binary tree structure was chosen for two reasons. The first reason is because if the symbol entries as encountered are randomly distributed according to lexicographic order, then table searching becomes approximately equivalent to a binary search as long as the tree is maintained in lexicographic order. Second, a binary tree structure is easily maintained in lexicographic order (in the sense that only a few pointers need be changed).

For simplicity, we assume a relatively sophisticated system which allows variable-length character strings to be used without much effort on the part of the programmer to handle them. We further assume that the symbol table routine is used to create trees that are local to a block of program code. This implies that an attempt to insert a duplicate entry is an error. In a global context, duplicate entries would be permitted as long as they were at different block levels. In a sense, the symbol table is a set of trees, one for each block level.

A binary tree will be constructed whose typical node is of the form

LLINK	*SYMBOLS*	*INFO*	*RLINK*

where *LLINK* and *RLINK* are pointer fields, *SYMBOLS* is the field for the character string which is the identifier or variable name (note that string descriptors might well be used here to allow fixed-length nodes, but it is assumed that this use is clear to the user), and *INFO* is some set of fields containing additional information about the identifier, such as its type. A node will be created by the execution of the statement $P \leftarrow NODE$ where the address of the new node is stored in P.

Finally, it is assumed that prior to any use of the symbol table routine at a particular block level, the appropriate tree head node is created with the *SYMBOLS* field set to a value that is greater lexicographically than any valid identifier. $HEAD[n]$ will point to this node where n designates the nth block level. Hence, the existence of an appropriate main routine which administers to the creation of tree heads as a new block is entered and to the deletion of tree heads as a block is exited, is assumed.

Because both the insertion and look-up operations involve many of the same actions (e.g., searching), we will actually produce only one routine, *TABLE*, and distinguish between insertion and look-up by the value of a global logical variable, *FLAG*. On invoking algorithm *TABLE*, if *FLAG* is *true*, then the requested operation is insertion; *NAME* and *DATA* contain the identifier name and additional information respectively. If the insertion is successful, then *FLAG* retains its original value; otherwise the value of *FLAG* is negated to indicate an error (because the identifier is already present in the table at that level), and an exit from the algorithm is made. On the other hand, if the algorithm is invoked with *FLAG* set to *false*, then the requested operation is look-up. In this

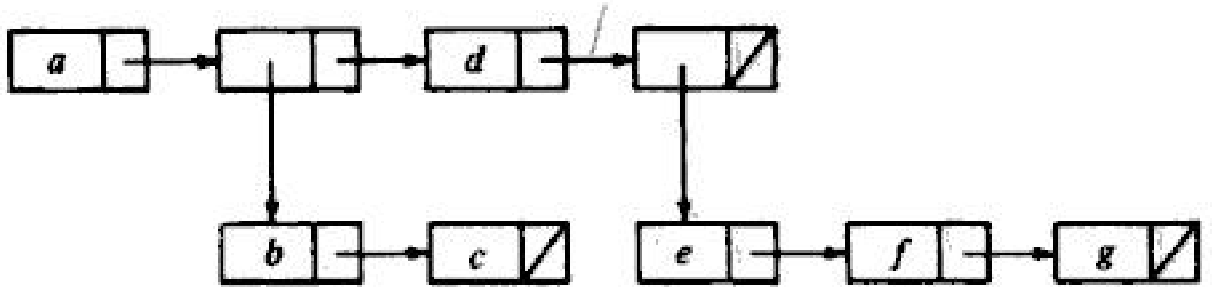


FIGURE 5-2.6 Storage representation of $(a, (b, c), d, (e, f, g))$.

has a depth of 2. The depth of a list is the number of levels it contains. The elements a and d are at level 1, and b and c are at level 2. The number of pairs of parentheses surrounding an element indicates its level. The element d in the list $(a, (b, (c, (d))))$ has a level of 4.

Order and depth are more easily understood in terms of our storage representation, where order is indicated by horizontal arrows and depth by vertical (downward-pointing) arrows. Thus, the list $(a, (b, c), d, (e, f, g))$ would be represented by the storage structure in Fig. 5-2.6. In storage, several lists may share common sublists. For example, the lists $(a, (b, c), d)$ and $(1, 5.2, (b, c))$ could be represented as in Fig. 5-2.7.

The recursive list M, where M is (a, b, M), can be represented as shown in Fig. 5-2.8. We would naturally use this storage representation where the common structure is shared rather than generating an infinite number of nodes to correspond to an infinite graph, but great care must be taken when manipulating such recursive structures in order to avoid programming oneself into an infinite loop.

A list structure occurs quite frequently in the processing of information, although it is not always evident. Consider a simple English sentence which consists of a subject, verb, and object. Any such sentence can be interpreted as a three-element list, whose elements can be atoms (single words) or lists (word phrases). The following sentences and their corresponding list representations are examples:

Man bites dog. = (Man, bites, dog)

The man bites the dog. = ((The, man), bites, (the, dog))

The big man is biting the small dog. = ((The, big, man), (is, biting), (the, small, dog))

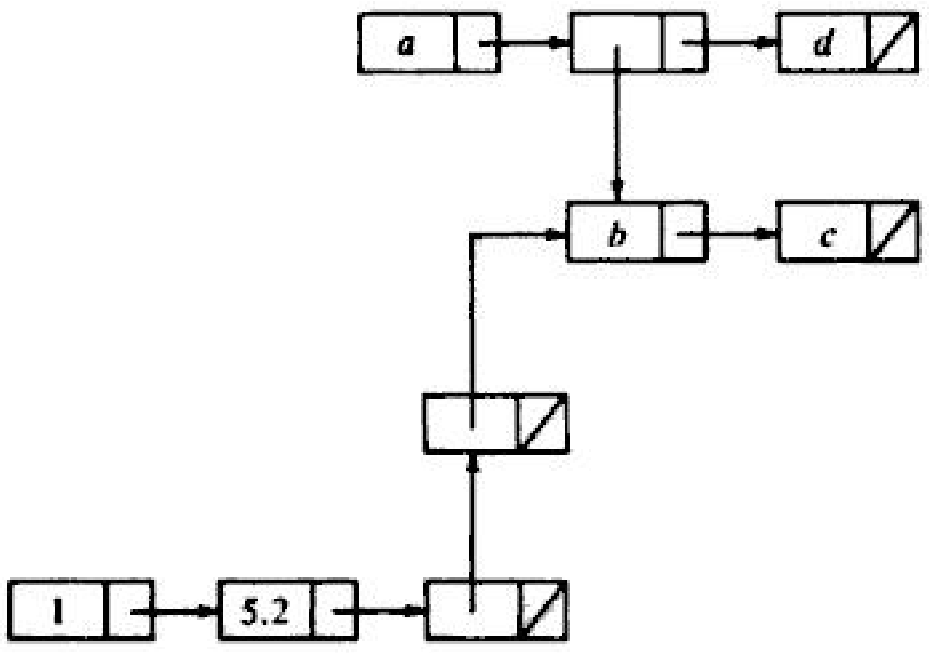


FIGURE 5-2.7 Two lists sharing a common sublist.

where Φ is

$$\langle\text{identifier}\rangle ::= \langle\text{letter}\rangle \mid \langle\text{identifier}\rangle\langle\text{letter}\rangle \mid \langle\text{identifier}\rangle\langle\text{digit}\rangle$$
$$\langle\text{letter}\rangle ::= a \mid b \mid \cdots \mid y \mid z$$
$$\langle\text{digit}\rangle ::= 0 \mid 1 \mid \cdots \mid 8 \mid 9$$

The existence of the rule ⟨identifier⟩ : : = ⟨letter⟩ does not imply that ⟨letter⟩ is a phrase since we cannot generate ⟨digit⟩⟨identifier⟩⟨digit⟩ from the starting symbol ⟨identifier⟩. What are the phrases of ⟨letter⟩ 1? A derivation for this sentential form is

$$\langle\text{identifier}\rangle \Rightarrow \langle\text{identifier}\rangle\langle\text{digit}\rangle \Rightarrow \langle\text{letter}\rangle\langle\text{digit}\rangle \Rightarrow \langle\text{letter}\rangle\, 1$$

Therefore

$$\langle\text{identifier}\rangle \overset{*}{\Rightarrow} \langle\text{letter}\rangle\langle\text{digit}\rangle \qquad \text{and} \qquad \langle\text{digit}\rangle \overset{+}{\Rightarrow} 1$$

Consequently, 1 is a simple phrase. Another derivation for the given sentential form is

$$\langle\text{identifier}\rangle \Rightarrow \langle\text{identifier}\rangle\langle\text{digit}\rangle \Rightarrow \langle\text{identifier}\rangle\, 1 \Rightarrow \langle\text{letter}\rangle\, 1$$

where

$$\langle\text{identifier}\rangle \overset{*}{\Rightarrow} \langle\text{identifier}\rangle\, 1 \qquad \text{and} \qquad \langle\text{identifier}\rangle \overset{+}{\Rightarrow} \langle\text{letter}\rangle$$

Again, the only phrase which is also a simple phrase is ⟨letter⟩.

In the subsequent discussion the leftmost simple phrase of a sentential form will be required. We therefore formulate the following definition.

Definition 5-3.2 The *handle* of a sentential form is its leftmost simple phrase.

In the current example, we have two possible simple phrases, namely, ⟨letter⟩ and 1, and since ⟨letter⟩ is the leftmost phrase, it is also the handle.

As previously discussed in Sec. 3-3.3, syntax trees are an important aid to understanding the syntax of a sentence. A syntax tree for a sentence of some language has a distinguished node called its *root* which is labeled by the starting symbol of the grammar. The leaf nodes of the syntax tree represent the terminal symbols in the sentence being diagramed. All nonleaf nodes correspond to nonterminal symbols. Each nonterminal node has a number of branches emanating downward, each of which represents a symbol in the right side of the production being applied at that point in the syntax tree.

The syntax tree corresponding to the following derivation of the sentence $c1$ in grammar G_1 is given in Fig. 5-3.1.

$$\langle\text{identifier}\rangle \Rightarrow \langle\text{identifier}\rangle\langle\text{digit}\rangle \Rightarrow \langle\text{letter}\rangle\langle\text{digit}\rangle \Rightarrow c\langle\text{digit}\rangle \Rightarrow c1$$

Note that another possible derivation for the same sentence is

$$\langle\text{identifier}\rangle \Rightarrow \langle\text{identifier}\rangle\langle\text{digit}\rangle \Rightarrow \langle\text{identifier}\rangle\, 1 \Rightarrow \langle\text{letter}\rangle\, 1 \Rightarrow c1$$

and that this derivation has the same syntax tree as that given in Fig. 5-3.1. Therefore for each syntax tree there exists at least one derivation.

If $A \rightarrow \beta$ is a rule of the grammar and $\phi_2 \in V_T^*$. Again the relation is easily extended to $\overset{+}{\underset{R}{\Rightarrow}}$ and $\overset{*}{\underset{R}{\Rightarrow}}$ where the relation $\psi \overset{+}{\underset{R}{\Rightarrow}} \sigma$ may be read as "ψ right-produces σ." The relation $\overset{+}{\underset{R}{\Rightarrow}}$ specifies that there is exactly one $\underset{R}{\Rightarrow}$ derivation. For example, the leftmost and rightmost derivations for $i + i * i$ in G_4 are

$$\begin{aligned}\langle\text{expression}\rangle &\underset{L}{\Rightarrow} \langle\text{expression}\rangle + \langle\text{expression}\rangle \underset{L}{\Rightarrow} \langle\text{term}\rangle + \langle\text{expression}\rangle\\ &\underset{L}{\Rightarrow} \langle\text{factor}\rangle + \langle\text{expression}\rangle \underset{L}{\Rightarrow} i + \langle\text{expression}\rangle\\ &\underset{L}{\Rightarrow} i + \langle\text{term}\rangle \underset{L}{\Rightarrow} i + \langle\text{term}\rangle * \langle\text{factor}\rangle\\ &\underset{L}{\Rightarrow} i + \langle\text{factor}\rangle * \langle\text{factor}\rangle \underset{L}{\Rightarrow} i + i * \langle\text{factor}\rangle\\ &\underset{L}{\Rightarrow} i + i * i\end{aligned}$$

and

$$\begin{aligned}\langle\text{expression}\rangle &\underset{R}{\Rightarrow} \langle\text{expression}\rangle + \langle\text{expression}\rangle \underset{R}{\Rightarrow} \langle\text{expression}\rangle + \langle\text{term}\rangle\\ &\underset{R}{\Rightarrow} \langle\text{expression}\rangle + \langle\text{term}\rangle * \langle\text{factor}\rangle\\ &\underset{R}{\Rightarrow} \langle\text{expression}\rangle + \langle\text{term}\rangle * i\\ &\underset{R}{\Rightarrow} \langle\text{expression}\rangle + \langle\text{factor}\rangle * i\\ &\underset{R}{\Rightarrow} \langle\text{expression}\rangle + i * i \underset{R}{\Rightarrow} \langle\text{term}\rangle + i * i\\ &\underset{R}{\Rightarrow} \langle\text{factor}\rangle + i * i \underset{R}{\Rightarrow} i + i * i\end{aligned}$$

respectively. The leftmost and the rightmost derivations correspond to a left-to-right and right-to-left top-down parse, respectively.

The general problem of parsing is to start with a string σ of terminal symbols, for example, $i + i * i$, and to find a sequence of productions such that $\langle\text{expression}\rangle \overset{*}{\Rightarrow} \sigma$. The bottom-up method (proceeding from left to right) attacks the problem by first "reducing" the above string to

$$\langle\text{factor}\rangle + i * i$$

then reducing this string to

$$\langle\text{term}\rangle + i * i$$

and then $\quad \langle\text{expression}\rangle + i * i$

and $\quad \langle\text{expression}\rangle + \langle\text{factor}\rangle * i$

etc., where a reduction is the opposite of a production. This process continues until everything is reduced to ⟨expression⟩ or until it is shown that it cannot be done. Note that the sequence of reductions is the reverse of the right canonical derivation. In general, bottom-up parsing from left to right proceeds by right reductions.

In a left-to-right bottom-up parse, the handle is to be reduced at each step in the parse. The questions which arise are, How do we find the handle of a sentential form and to what do we reduce it?

One obvious approach is merely to select one of the possible alternatives. If a mistake is subsequently detected, then we must retrace our steps to the location of the error and try some other alternative. This process is called "back-up," and it can be very time-consuming.

A more efficient solution involves looking at the context around the sub-

$A \rightarrow x$

$B \rightarrow Ax)$

Using only syntax trees, obtain as many of the precedence relations as possible.

6 Using the formal definition of precedence relations, obtain the precedence matrix for Prob. 5.

7 Prove Theorem 5-3.2.

8 Prove Theorem 5-3.3.

9 Prove Theorem 5-3.4.

10 Prove that $\cdot\!> = (R^+)^T \circ (\doteq) \circ (L^*)$.

11 Using the parsing algorithm of the text and the grammar of Prob. 5, give a tree of the parse for the strings $y(xx)y$, $(((xx)x)x)y$, $y((xx))x)y$.

12 Formulate an algorithm and write a program that will obtain a precedence matrix.

13 Write a program to use the precedence matrix obtained in Prob. 12 and parse a string according to the parsing algorithm of the text.

14 Can you obtain the precedence functions for the grammar of Prob. 5?

15 Is the following grammar a simple precedence grammar? If not, why not?

$E \rightarrow a$	$B \rightarrow R$
$E \rightarrow b$	$B \rightarrow (B)$
$E \rightarrow (E + E)$	$R \rightarrow E = E$

5-4 FAULT DETECTION IN COMBINATIONAL SWITCHING CIRCUITS

The synthesis and analysis of combinational switching circuits were discussed in the previous chapter. In this section, we discuss a different but equally important problem, fault detection and fault diagnosis. Fault detection is concerned with determining whether a fault exists in a circuit. In fault diagnosis, one tries to locate a specific fault in a system.

The diagnosis of faults in modern computing systems is becoming more important as the complexity of such systems increases. Rapid and effective diagnosis of faults in computer systems is desirable since once these diagnoses are in service, down time can be kept at a minimum.

The detection and diagnosis of faults in early computers were performed by technicians. Although these computers were physically very large, the number of components in them was rather small as compared to present computers, and a skillful engineer could locate a fault in a reasonable period of time. With the advent of transistors and integrated circuits, the number of components in a modern computer has greatly increased and fault detection and diagnosis have become increasingly difficult tasks.

The logical conclusion is that if a computer is at all operational, it should diagnose itself, or at worst another computer should be used to perform the diagnosis.

In this section we shall consider a combinational circuit diagram to be a directed graph whose nodes are the gates and whose edges are the connections between gates. From this representation an algorithm will be given to generate a fault table. The purpose of such a table is to be able to detect and more generally diagnose faults. The algorithm will use certain notions of relations such as their

The McGraw·Hill Companies

Tata McGraw-Hill
Publishing Company Limited
7 West Patel Nagar, New Delhi 110 008

Visit our website at : www.tatamcgrawhill.com

ISBN-13: 978-0-07-463113-3
ISBN-10: 0-07-463113-6

9 780074 631133