Discrete Structures 2 [1]

A Study Guide for Students of Sorsogon State University - Bulan Campus[2]

JARRIAN VINCE G. GOJAR[3]

February 27, 2025

[1]A course in the Bachelor of Science in Computer Science

[2]This book is a study guide for students of Sorsogon State University - Bulan Campus taking up the course Discrete Structures 2.

[3]https://github.com/godkingjay

Sorsogon State University - Bulan Campus

Contents

List of Figures

List of Tables

Preface

"If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is."

– John von Neumann

Jarrian Vince G. Gojar
https://github.com/godkingjay

1

Boolean Algebra

1.1 Introduction

Circuits in computers are made up of millions of tiny switches that can be in one of two states: on or off. These switches are controlled by electrical signals that represent logical values. The behavior of these switches can be described using a mathematical system called Boolean Algebra. **Boolean Algebra** is a branch of mathematics that deals with logical values and operations on these values. It is widely used in computer science and engineering to design and analyze digital circuits. Computers uses the binary number system, which has only two digits: 0 and 1 which means "low voltage" and "high volt" respectively. These digits correspond to the logical values FALSE and TRUE, respectively.

1.2 History of Boolean Algebra

George Boole was an English mathematician and logician who lived in the 19th century. He was born in 1815 and died in 1864. Boole is best known for his work in the field of logic, which laid the foundation for modern computer science.

Boole's most famous work is his book *The Laws of Thought*, which was published in 1854. In this book, Boole introduced the concept of Boolean Algebra, which is a mathematical system for dealing with logical values. Boolean Algebra is based on the idea that logical values can be represented as either TRUE or FALSE.

In 1938, **Claude Shannon** showed that the two-valued Boolean Algebra could be used to describe the operation of electrical switches. This discovery laid the foundation for the design of digital circuits and computers.

1.3 Fundamental Operations

The three fundamental operations of Boolean Algebra are:

- **AND** - The AND operation takes two or more inputs and produces a 1 output only if all inputs are 1.
- **OR** - The OR operation takes two or more inputs and produces a 1 output if at least one input is 1.
- **NOT** - The NOT operation takes a single input and produces the opposite value. If the input is 1, the output is 0, and vice versa.

	Formal Logic	Set Theory	Boolean Algebra
Variables	p, q, r, ...	A, B, C, ...	x, y, z, ...
Operations	$\wedge$, $\vee$, $\neg$	$\cap$, $\cup$, $-$	$\bullet$, $+$, '
Special Elements	F, T	$\emptyset$, U	0, 1

Table 1.1: Comparison of Formal Logic, Set Theory, and Boolean Algebra

Table 1.1 shows a comparison of the notation used in formal logic, set theory, and Boolean Algebra. In formal logic, variables are represented by letters such as p, q, r, etc., and the logical operations are represented by symbols such as $\wedge$, $\vee$, and $\neg$. In set theory, variables are represented by capital letters such as A, B, C, etc., and the set operations are represented by symbols such as $\cap$, $\cup$, and $-$. In Boolean Algebra, variables are represented by letters such as x, y, z, etc., and the Boolean operations are represented by symbols such as $\cdot$, $+$, and $'$. The special elements in each system are F and T in formal logic, $\emptyset$ and U in set theory, and 0 and 1 in Boolean Algebra.

Though the notation used in each system is different, the underlying concepts are the same. For example, the AND operation in Boolean Algebra is similar to the logical conjunction operation in formal logic, where the output is TRUE only if all inputs are TRUE. Similarly, the OR operation in Boolean Algebra is similar to the logical disjunction operation in formal logic, where the output is TRUE if at least one input is TRUE. Compared to set theory, the AND operation in Boolean Algebra is similar to the intersection operation, where the output is the set of elements that are common to all input sets.

1.3.1 AND Operation

The AND operation is denoted by the symbol $\cdot$ or juxtaposition. The output of the AND operation is 1 only if all inputs are 1. In Boolean Algebra, the AND operation is represented by the multiplication symbol $\cdot$ or by juxtaposition. **Juxtaposition** is the act of placing two or more things side by side or close together.

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.2: Truth Table for the AND Operation

Table 1.2 shows the truth table for the AND operation. The output is 1 only if both inputs are 1; otherwise, the output is 0.

Suppose we have the variables x and y, and we want to represent the AND operation between them. We can write this as $x \cdot y$ or xy via juxtaposition. The output of this operation is 1 only if both x and y are 1.

Input 1	Input 2	Output
x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.3: Truth Table for the AND Operation with Variables

Table 1.3 shows the truth table for the AND operation with variables x and y. The output is 1 only if both x and y are 1; otherwise, the output is 0.

Exercise

1. Consider the three input AND operation xyz. Write the truth table for this operation and determine the output for each combination of inputs.

1.3.2 OR Operation

The OR operation is denoted by the symbol $+$. The output of the OR operation is 1 if at least one input is 1. In Boolean Algebra, the OR operation is represented by the addition symbol $+$.

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

Table 1.4: Truth Table for the OR Operation

Table 1.4 shows the truth table for the OR operation. The output is 1 if at least one input is 1; otherwise, the output is 0.

Suppose we have the variables x and y, and we want to represent the OR operation between them. We can write this as $x + y$. The output of this operation is 1 if at least one of x and y is 1.

Input 1	Input 2	Output
x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Table 1.5: Truth Table for the OR Operation with Variables

Table 1.5 shows the truth table for the OR operation with variables x and y. The output is 1 if at least one of x and y is 1; otherwise, the output is 0.

Exercise
Write the truth table for the following operations:
1. $f(x, y, z) = x + y + z$
2. $f(x, y, z) = (x + y)z$
3. $f(x, y, z) = x + yz$
4. $f(x, y, z) = (x + y)(y + z)$
5. $f(x, y, z) = x(xy + yz)$

1.3.3 NOT Operation

The NOT operation is denoted by the symbol $'$. The output of the NOT operation is the opposite of the input. If the input is 1, the output is 0, and vice versa. In Boolean Algebra, the NOT operation is represented by the prime symbol $'$ or by an overline.

Input	**Output**
0	1
1	0

Table 1.6: Truth Table for the NOT Operation

Table 1.6 shows the truth table for the NOT operation. The output is the opposite of the input. If the input is 1, the output is 0, and vice versa.

Suppose we have the variable x, and we want to represent the NOT operation on it. We can write this as x' or $\overline{x}$. The output of this operation is the opposite or complement of x.

Input	**Output**
x	x'
0	1
1	0

Table 1.7: Truth Table for the NOT Operation with Variables

Table 1.7 shows the truth table for the NOT operation with variable x. The output is the opposite of x. If x is 1, the output is 0; if x is 0, the output is 1.

Exercise
Write the truth table for the following operations:
1. $f(x) = (x')'$
2. $f(x, y) = (x + y)'$
3. $f(x, y) = (x \cdot y)'$
4. $f(x, y, z) = (x + yz)'$
5. $f(x, y, z) = (x \cdot y + z)'$
6. $f(x, y, z) = (x + y)'z$
7. $f(x, y, z) = x' + y' + z'$
8. $f(x, y, z) = (x + y + z)'$
9. $f(x, y, z) = x'yz$
10. $f(x, y) = x' + y'(xy + x')$

1.4 Other Operations

In addition to the fundamental operations of AND, OR, and NOT, there are several other operations in Boolean Algebra that are commonly used. These operations include:

- **XOR** - The XOR operation takes two inputs and produces a 1 output if the inputs are different.
- **NAND** - The NAND operation is the complement of the AND operation. The output of the NAND operation is 0 only if all inputs are 1.
- **NOR** - The NOR operation is the complement of the OR operation. The output of the NOR operation is 0 if at least one input is 1.
- **XNOR** - The XNOR operation is the complement of the XOR operation. The output of the XNOR operation is 1 if the inputs are the same.

1.4.1 XOR Operation

The XOR operation is denoted by the symbol $\oplus$. The output of the XOR operation is 1 if the inputs are different. In Boolean Algebra, the XOR operation is represented by the symbol $\oplus$. If the XOR operation takes more than two inputs, it is called the **parity function**. A parity function is a function that determines whether the number of inputs that are 1 is even or odd.

Input 1	**Input 2**	**Output**
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.8: Truth Table for the XOR Operation

Table 1.8 shows the truth table for the XOR operation. The output is 1 if the inputs are different; otherwise, the output is 0.

Suppose we have the variables x and y, and we want to represent the XOR operation between them. We can write this as $x \oplus y$. The output of this operation is 1 if x and y are different.

Input 1	**Input 2**	**Output**
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.9: Truth Table for the XOR Operation with Variables

Table 1.9 shows the truth table for the XOR operation with variables x and y. The output is 1 if x and y are different; otherwise, the output is 0.

Exercise
Write the truth table for the following operations:

1. $f(x, y, z) = (x \oplus y)z$
2. $f(x, y, z) = x \oplus yz$
3. $f(x, y, z) = (x \oplus y)(y \oplus z)(z \oplus x)$
4. $f(x, y, z) = x' \oplus y \oplus z'$

1.4.2 NAND Operation

The NAND operation is simply the complement of the AND operation. The output of the NAND operation is 0 only if all inputs are 1. In Boolean Algebra, the NAND operation is represented by putting an overline or $'$ over the AND operation.

Input 1	**Input 2**	**Output**
x	y	$(xy)'$
0	0	1
0	1	1
1	0	1
1	1	0

Table 1.10: Truth Table for the NAND Operation with Variables

Table 1.10 shows the truth table for the NAND operation with variables x and y. The output is 0 only if both x and y are 1; otherwise, the output is 1.

1.4.3 NOR Operation

The NOR operation is simply the complement of the OR operation. The output of the NOR operation is 0 if at least one input is 1. In Boolean Algebra, the NOR operation is represented by putting an overline or $'$ over the OR operation.

Input 1	**Input 2**	**Output**
x	y	$(x+y)'$
0	0	1
0	1	0
1	0	0
1	1	0

Table 1.11: Truth Table for the NOR Operation with Variables

Table 1.11 shows the truth table for the NOR operation with variables x and y. The output is 0 if at least one of x and y is 1; otherwise, the output is 1.

1.4.4 XNOR Operation

The XNOR operation is simply the complement of the XOR operation. The output of the XNOR operation is 1 if the inputs are the same. In Boolean Algebra, the XNOR operation is represented by putting an overline or $'$ over the XOR operation.

Input 1	**Input 2**	**Output**
x	y	$(x \oplus y)'$
0	0	1
0	1	0
1	0	0
1	1	1

Table 1.12: Truth Table for the XNOR Operation with Variables

Table 1.12 shows the truth table for the XNOR operation with variables x and y. The output is 1 if x and y are the same; otherwise, the output is 0.

1.5 Tautology and Fallacy

In Boolean Algebra, a **tautology** is a statement that is always TRUE, regardless of the values of its variables. A **fallacy** is a statement that is always FALSE, regardless of the values of its variables.

x	x'	$x + x'$	xx'
0	1	1	0
1	0	1	0

Table 1.13: Examples of Tautologies and Fallacies

Table 1.13 shows examples of tautologies and fallacies. The expression $x + x'$ is a tautology because it is always TRUE, regardless of the value of x. The expression xx' is a fallacy because it is always FALSE, regardless of the value of x.

1.6 Boolean Functions

A **Boolean function** is a function that takes one or more Boolean variables as input and produces a Boolean output. Boolean functions are used to represent logical operations in Boolean Algebra. The output of a Boolean function is determined by the values of its input variables and the logical operations applied to them. It is also known as a **switching function**. Boolean variables are variables that can take on one of two values: 0 or 1. In Boolean Algebra, variables are typically denoted by letters such as x, y, z, etc. The values of these variables represent logical values: 0 corresponds to FALSE, and 1 corresponds to TRUE.

A Boolean function is a function in the form $f : B^n \to B$, where $B = \{0, 1\}$ is the set of Boolean values, and n is the number of input variables and is called the **arity** of the function.

A **literal** is a variable or its complement. For example, x and x' are literals. In the boolean function $f = (x + yz) + x'$, there are three variables: x, y, and z. The literals in this function are x, y, z, and x' which are the variables and their complements.

Exercises
Write the truth table for the following Boolean functions:

1. $f(x, y) = [xy + (x + y)']'$
2. $f(x, y) = (x + y) \oplus (xy)'$
3. $f(x, y, z) = x(y + z')$
4. $f(x, y, z) = (x + y)(y + z)(z + x)$
5. $f(x, y, z) = x \oplus y \oplus z$

1.7 Laws of Boolean Algebra

The laws of Boolean Algebra are a set of rules that define the properties of logical operations in Boolean Algebra. These laws are used to simplify Boolean expressions and to prove the equivalence of different expressions. The laws of Boolean Algebra are based on the properties of logical operations such as AND, OR, and NOT.

Boolean algebra's AND operations associates to multiplication in arithmetic, while OR operations associates to addition. While the NOT operation is simply the complement of the input.

Laws	AND	OR
Identity	$x(1) = x$	$x + 0 = x$
Domination	$x(0) = 0$	$x + 1 = 1$
Commutative	$xy = yx$	$x + y = y + x$
Associative	$x(yz) = (xy)z$	$x + (y + z) = (x + y) + z$
Distributive	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$
Inverse	$xx' = 0$	$x + x' = 1$
Idempotent	$xx = x$	$x + x = x$
Absorption	$x(x + y) = x$	$x + xy = x$
	$x(x' + y) = xy$	$x + x'y = x + y$
De Morgan's Theorem	$(xy)' = x' + y'$	$(x + y)' = x'y'$
Involution (Double Negation)	$(x')' = x$	
Consensus	$xy + x'z + yz = xy + x'z$	

Table 1.14: Laws of Boolean Algebra

Table 1.14 shows the laws of Boolean Algebra. These laws are used to simplify Boolean expressions and to prove the equivalence of different expressions. The laws are based on the properties of logical operations such as AND, OR, and NOT.

Exercises

1. Verify the following laws of Boolean Algebra:
   (a) Identity Law: $x + 0 = x$
   (b) Commutative Law: $xy = yx$
   (c) Associative Law: $x(yz) = (xy)z$
   (d) Distributive Law: $x(y + z) = xy + xz$
   (e) Inverse Law: $xx' = 0$
   (f) Involution Law: $(x')' = x$
   (g) De Morgan's Theorem: $(xy)' = x' + y'$
2. Prove the following laws of Boolean Algebra:
   (a) Domination Law: $x + 1 = 1$
   (b) Idempotent Law: $x + x = x$
   (c) Absorption Law: $x(x + y) = x$

1.8 Simplifying Boolean Expressions

Boolean expressions can be simplified using the laws of Boolean Algebra. Simplifying a Boolean expression involves applying the laws of Boolean Algebra to reduce the expression to its simplest form. This process involves combining terms, eliminating redundant terms, and applying the laws of Boolean Algebra to simplify the expression.

Consider the Boolean expression $f(x, y) = x + x'y$. We can simplify this expression using the laws of Boolean Algebra as follows:

$$\begin{aligned} f(x, y) &= x + x'y \\ &= (x + x')(x + y) && \text{Distributive Law} \\ &= (1)(x + y) && \text{Inverse Law} \\ &= x + y && \text{Identity Law} f(x, y) && = x + x'y = x + y \end{aligned}$$

The expression $f(x, y) = x + x'y$ can be simplified to $f(x, y) = x + y$ using the laws of Boolean Algebra.

Consider the Boolean expression $f(x, y, z) = (x + yz') + (xy)'$. We can simplify this expression using the laws of Boolean Algebra as follows:

$$
\begin{aligned}
f(x, y, z) &= (x + yz') + (xy)' && \\
&= (x + yz') + x' + y' && \text{De Morgan's Theorem} \\
&= (x + x') + yz' + y' && \text{Commutative Law} \\
&= 1 + yz' + y' && \text{Inverse Law} \\
&= 1 && \text{Domination Law}
\end{aligned}
$$

The expression $f(x, y, z) = (x + yz') + (xy)'$ can be simplified to $f(x, y, z) = 1$ using the laws of Boolean Algebra.

Consider the Boolean expression $f(x, y) = (x + x'y) + (x + y')$. We can simplify this expression using the laws of Boolean Algebra as follows:

$$
\begin{aligned}
f(x, y) &= (x + x'y) + (x + y') && \\
&= [(x + x')(x + y)] + (x + y') && \text{Distributive Law} \\
&= [1(x + y)] + (x + y') && \text{Inverse Law} \\
&= (x + y) + (x + y') && \text{Identity Law} \\
&= (y + x) + (x + y') && \text{Commutative Law} \\
&= y + (x + x) + y' && \text{Associative Law} \\
&= y + x + y' && \text{Idempotent Law} \\
&= x + y + y' && \text{Commutative Law} \\
&= x + 1 && \text{Inverse Law} \\
&= 1 && \text{Domination Law}
\end{aligned}
$$

The expression $f(x, y) = (x + x'y) + (x + y')$ can be simplified to $f(x, y) = 1$ using the laws of Boolean Algebra.

Exercises
Simplify the following Boolean expressions using the laws of Boolean Algebra:

1. $f(x,y) = (x+y')(x+y)$
2. $f(w,x) = w + [w + (wx)]$
3. $f(x) = (x' + x')'$
4. $f(x) = (x + x')'$
5. $f(w,x,y,z) = w + (wx'yz)$
6. $f(w,x,y,z) = w'(wxyz)'$
7. $f(x,y,z) = [y + x'y + (x+y')]y'$
8. $f(w,x,y,z) = wx + (x'z') + (y+z')$
9. $f(x,y,z) = (x+y)(x+z)$
10. $f(x,y) = [xy + (x+y)']'$
11. $f(w,x,y,z) = (w + x' + y + z')y$
12. $f(x,y,z) = x + y + (x' + y + z)'$
13. $f(x,y,z) = xz + x'y + zy$
14. $f(x,y,z) = (x+z)(x'+y)(z+y)$
15. $f(x,y,z) = x' + y' + xyz'$

1.9 Principle of Duality

The **principle of duality** states that any theorem or identity in Boolean Algebra remains valid if we interchange the AND and OR operations and the constants 0 and 1. In other words, if we replace each AND operation with an OR operation, each OR operation with an AND operation, each 0 with a 1, and each 1 with a 0, the resulting expression is also valid.

Expression	**Dual Expression**
$x+y$	xy
$x(y+z)$	$x+yz$
$x(y+0)$	$x+y\cdot 1$
$x+1=1$	$x\cdot 0=0$
$x+x=x$	$x\cdot x=x$

Table 1.15: Examples of the Principle of Duality

Table 1.15 shows examples of the principle of duality. The dual expression of $x+y$ is xy, the dual expression of $x(y+z)$ is $x+yz$, and so on. The principle of duality states that any theorem or identity in Boolean Algebra remains valid if we interchange the AND and OR operations and the constants 0 and 1.

Exercises

Write the dual expression for the following Boolean expressions:

1. $x + yz$
2. $x(y + z)$
3. $x + y + z$
4. $x(y + z) + x'$
5. $x + y + z + 1$
6. $x + x'$
7. $x + y + z + 0$
8. $x(y + z) + 1$
9. $x + y + z + x'$
10. $x + y + z + x$

2

Logic Gates and Circuits

2.1 Introduction

A **logic gate** is an electronic device that performs a logical operation on one or more binary inputs and produces a single binary output. Logic gates are the building blocks of digital circuits and are used to implement Boolean functions.

Say for example, we have two binary inputs x and y, and we want to perform the AND operation on them. We can use an AND gate to perform this operation. The AND gate takes two binary inputs x and y and produces a single binary output that is the result of the AND operation on x and y.

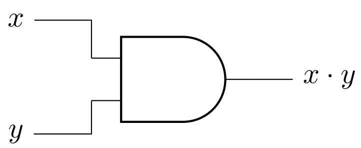

Figure 1: Example of a Logic Circuit for the AND Operation

Figure 1 shows an example of a logic circuit for the AND operation. The circuit takes two binary inputs x and y and produces a single binary output that is the result of the AND operation on x and y. This logic circuit is actually equivalent to the boolean expression $f(x, y) = x \cdot y$.

2.2 Logic Gates and Circuits

There are several types of logic gates that perform different logical operations. The most common logic gates are:

2.2.1 Buffer Gate

A **buffer gate** is a logic gate that takes a single input and produces a single output that is the same as the input.

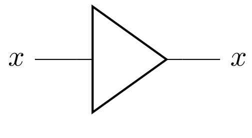

Figure 2: Logic Circuit for the Buffer Gate

Figure 2 shows a logic circuit for the buffer gate. The buffer gate takes a single binary input x and produces a single binary output that is the same as the input.

2.2.2 NOT Gate

A **NOT gate** is a logic gate that takes a single input and produces a single output that is the complement of the input. In boolean algebra, the NOT operation is represented by the prime symbol $'$ or by an overline.

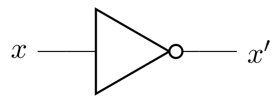

Figure 3: Logic Circuit for the NOT Gate

Figure 3 shows a logic circuit for the NOT gate. The NOT gate takes a single binary input x and produces a single binary output that is the complement of the input.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x) = (x')'$
2. $f(y) = ((y')')'$

2.2.3 AND Gate

An **AND gate** is a logic gate that takes two or more inputs and produces a single output that is the result of the AND operation on the inputs.

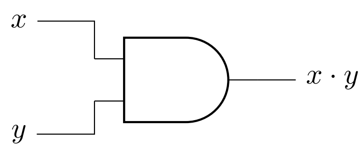

Figure 4: Logic Circuit for the AND Gate

Figure 4 shows a logic circuit for the AND gate. The AND gate takes two binary inputs x and y and produces a single binary output that is the result of the AND operation on x and y. This logic circuit is actually equivalent to the boolean expression $f(x, y) = x \cdot y$. The AND gate can take more than two inputs, and the output is TRUE only if all inputs are TRUE. Otherwise, the output is FALSE.

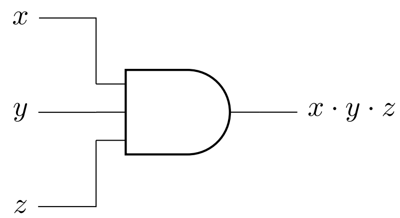

Figure 5: Logic Circuit for the AND Gate with 3 Inputs

Figure 5 shows a logic circuit for the AND gate with three inputs or the boolean expression $f(x, y, z) = x \cdot y \cdot z$. The AND gate takes three binary inputs x, y, and z and produces a single binary output that is the result of the AND operation on x, y, and z.

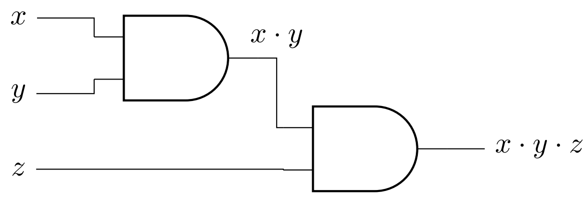

Figure 6: Logic Circuit for the AND Operation $x \cdot y \cdot z$

Figure 6 shows another implementation of the AND operation $x \cdot y \cdot z$ using two AND gates. The first AND gate takes two inputs x and y and produces the output $x \cdot y$. The second AND gate takes the output of the first AND gate and the third input z and produces the output $x \cdot y \cdot z$.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x, y) = xy'$
2. $f(x, y, z) = x'y'$
3. $f(x, y, z) = (xy)'z$

2.2.4 OR Gate

An **OR gate** is a logic gate that takes two or more inputs and produces a single output that is the result of the OR operation on the inputs.

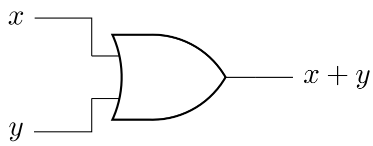

Figure 7: Logic Circuit for the OR Gate

Figure 7 shows a logic circuit for the OR gate for the boolean expression $f(x, y) = x + y$. The OR gate takes two binary inputs x and y and produces a single binary output that is the result of the OR operation on x and y.

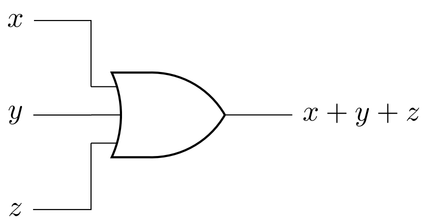

Figure 8: Logic Circuit for the OR Gate with 3 Inputs

Figure 8 shows a logic circuit for the OR gate with three inputs for the boolean expression $f(x, y, z) = x + y + z$. The OR gate takes three binary inputs x, y, and z and produces a single binary output that is the result of the OR operation on x, y, and z.

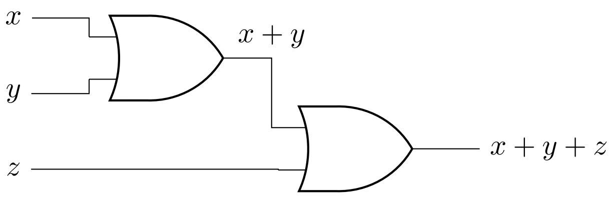

Figure 9: Logic Circuit for the OR Operation $x+y+z$

Figure 9 shows another implementation of the OR operation $x+y+z$ using two OR gates. The first OR gate takes two inputs x and y and produces the output $x+y$. The second OR gate takes the output of the first OR gate and the third input z and produces the output $x+y+z$.

Exercises
Draw the logic circuits for the following boolean expressions:
1. $f(x,y) = x + y'$
2. $f(x,y,z) = (x+y)'z$
3. $f(x,y,z) = xy + xz$
4. $f(x,y,z) = (x+y)(x+z)$

2.2.5 NAND Gate

A **NAND gate** is a logic gate that takes two or more inputs and produces a single output that is the complement of the result of the AND operation on the inputs.

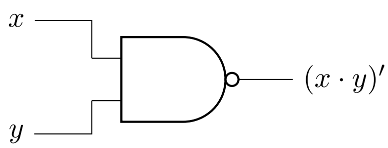

Figure 10: Logic Circuit for the NAND Gate

Figure 10 shows a logic circuit for the NAND gate for the boolean expression $f(x,y) = (x \cdot y)'$. The NAND gate takes two binary inputs x and y and produces a single binary output that is the complement of the result of the AND operation on x and y.

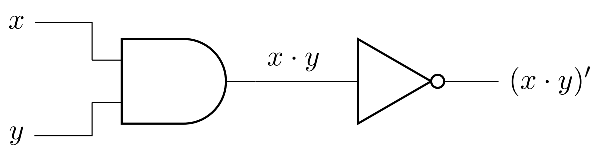

Figure 11: Logic Circuit for the NAND Gate using AND and NOT Gates

Figure 11 shows another implementation of the NAND gate using an AND gate and a NOT gate. The AND gate takes two inputs x and y and produces the output $x \cdot y$. The NOT gate takes the output of the AND gate and produces the complement of the result of the AND operation on x and y. This logic circuit is equivalent to the boolean expression $f(x,y) = (x \cdot y)'$.

Exercises
Draw the logic circuits for the following boolean expressions:
1. $f(x,y,z) = (xyz)'$
2. $f(x,y,z) = (xy)' + (xz)'$

2.2.6 NOR Gate

A **NOR gate** is a logic gate that takes two or more inputs and produces a single output that is the complement of the result of the OR operation on the inputs.

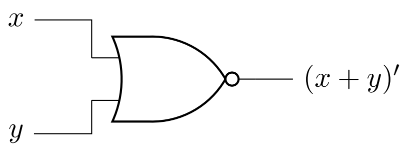

Figure 12: Logic Circuit for the NOR Gate

Figure 12 shows a logic circuit for the NOR gate for the boolean expression $f(x, y) = (x + y)'$. The NOR gate takes two binary inputs x and y and produces a single binary output that is the complement of the result of the OR operation on x and y.

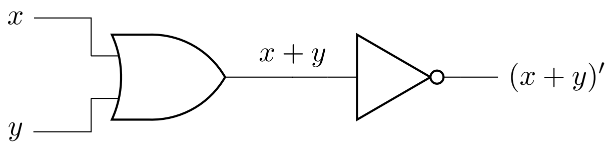

Figure 13: Logic Circuit for the NOR Gate using OR and NOT Gates

Figure 13 shows another implementation of the NOR gate using an OR gate and a NOT gate. The OR gate takes two inputs x and y and produces the output $x + y$. The NOT gate takes the output of the OR gate and produces the complement of the result of the OR operation on x and y. This logic circuit is equivalent to the boolean expression $f(x, y) = (x + y)'$.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x, y) = (x + y)'$
2. $f(x, y, z) = (x + y + z)'$
3. $f(x, y, z) = (x + y)(x + z)'$
4. $f(x, y, z) = (xy)'(x + z)'$

2.2.7 XOR Gate

An **XOR gate** is a logic gate that takes two or more inputs and produces a single output that is the result of the exclusive OR operation on the inputs.

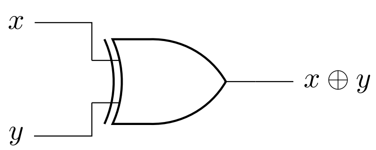

Figure 14: Logic Circuit for the XOR Gate

Figure 14 shows a logic circuit for the XOR gate for the boolean expression $f(x, y) = x \oplus y$. The XOR gate takes two binary inputs x and y and produces a single binary output that is the result of the exclusive OR operation on x and y. The output of the XOR gate is TRUE if the inputs are different and FALSE if the inputs are the same.

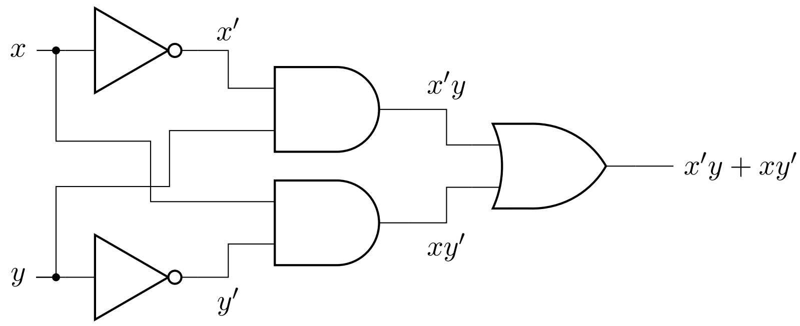

Figure 15: Logic Circuit for the XOR Gate using OR, AND, and NOT Gates

Figure 15 shows another implementation of the XOR gate using OR, AND, and NOT gates. The NOT gates take the inputs x and y and produce the complements x' and y'. The AND gates take the inputs x, y, x', and y' and produce the outputs $x'y$ and xy'. The OR gate takes the outputs of the AND gates and produces the output $x'y + xy'$. This logic circuit is equivalent to the boolean expression $f(x, y) = x \oplus y$. Figure 15 is a more complex implementation of the XOR gate using OR, AND, and NOT gates, but this circuit is equivalent to the XOR gate in Figure 14.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x, y) = x \oplus y'$
2. $f(x, y, z) = x \oplus y \oplus z$

2.2.8 XNOR Gate

An **XNOR gate** is a logic gate that takes two or more inputs and produces a single output that is the complement of the result of the exclusive NOR operation on the inputs.

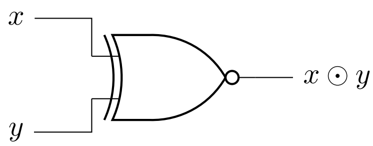

Figure 16: Logic Circuit for the XNOR Gate

Figure 16 shows a logic circuit for the XNOR gate for the boolean expression $f(x, y) = x \odot y$ or $f(x, y) = (x \oplus y)'$. The XNOR gate takes two binary inputs x and y and produces a single binary output that is the complement of the result of the exclusive NOR operation on x and y. The output of the XNOR gate is TRUE if the inputs are the same and FALSE if the inputs are different.

Exercises
Draw the logic circuits for the following boolean expressions:

1. $f(x, y) = x \odot y'$
2. $f(x, y, z) = x \odot y \odot z$
3. $f(x, y, z) = (x \oplus y) \odot z$
4. $f(x, y, z) = [x' + (y \odot z)]'$
5. $f(x, y, z) = (x \odot y) + (x' \odot z)$

2.2.9 Exercises

Exercises

Draw the logic circuits for the following boolean expressions:

1. $f(x, y) = (x + y')(x + y)$
2. $f(w, x) = w + [w + (wx)]$
3. $f(x) = (x' + x')'$
4. $f(x) = (x + x')'$
5. $f(w, x, y, z) = w + (wx'yz)$
6. $f(w, x, y, z) = w'(wxyz)'$
7. $f(x, y, z) = [y + x'y + (x + y')]y'$
8. $f(w, x, y, z) = wx + (x'z') + (y + z')$
9. $f(x, y, z) = (x \odot y)(x \oplus z)$
10. $f(x, y) = [xy + (x \oplus y)']'$
11. $f(w, x, y, z) = (w + x' + y + z')y$
12. $f(x, y, z) = x + y + (x' + y + z)'$
13. $f(x, y, z) = xz + x'y + zy$
14. $f(x, y, z) = (x \odot z)(x' + y)(z \oplus y)$
15. $f(x, y, z) = x' + y' + xyz'$

2.3 Minimization of Circuits

2.4 Binary Arithmetic and Representation

3

Graph Theory

3.1 Introduction

3.2 Graphs

3.2.1 Terms and Definitions

3.2.2 Paths and Cycles

3.2.3 Hamiltonian Cycles

3.2.4 Shortest Path Algorithms

3.2.5 Representation of Graphs

3.2.6 Isomorphism of Graphs

3.2.7 Planar Graphs

3.3 Trees

3.3.1 Terms and Definitions

3.3.2 Spanning Trees

3.3.3 Binary Trees

3.3.4 Tree Traversals

3.3.5 Decision Trees

3.3.6 Isomorphism of Trees

4

Network Models and Petri Nets

5

Automata, Grammars and Languages

5.1 Languages and Grammars

5.2 Finite State Automata

5.3 Regular Expressions

6

Computational Geometry

6.1 Basics of Computational Geometry

6.2 Closest-Pair Problem

6.3 Convex Hull Algorithm

6.4 Voronoi Diagrams

6.5 Line Segment Intersection

6.6 Applications in Computer Graphics and Geographical Information Systems

7

References

A. **Books**

-

B. **Other Sources**

-