# Chapter 4

# Boolean Expressions and Circuits

George Boole (1815-1864) was among the first to explore the connections between logic and algebra, which is the topic of this section. Many of the objects that we will study bear his name. We will define what are called *Boolean variables* and *Boolean expressions*, and we will learn how to manipulate them. Sections 4.2–4.4 will apply this to computer circuitry. Although sections 4.1–4.4 are not needed for the remainder of this text, they do provide context for Chapter 22.

## 4.1 Boolean Expressions

The discussion begins with truth tables. In constructing truth tables, we saw how logic expressions such as $P \Leftrightarrow (Q \vee R)$ can be true or false, depending on whether $P$, $Q$ and $R$ are true or false. In doing this, we have regarded $P$, $Q$ and $R$ as statements (or open sentences) that could potentially be true or false. But (at least in writing truth tables) we tend not to assign any particular meaning to $P$, $Q$ and $R$. It is as if they are *variables* that can take on one of the two values T or F.

This outlook can be very convenient. A **Boolean variable** is a symbol that is allowed to have either the value T (true) or F (false). In this text we will use the upper case letters at the end of the alphabet $(W, X, Y, Z)$ for Boolean variables. Thus we might write $P \Leftrightarrow (Q \vee R)$ alternatively as $X \Leftrightarrow (Y \vee Z)$ and interpret this as an expression involving variables rather than statements.

It is common (especially in computer applications) to write 0 for F and 1 for T, so that a Boolean variable can have either the value 0 or 1. We will follow this convention for the rest of this chapter (though afterwards we revert to T and F). So instead of $T \vee F = F$ and $T \wedge F = T$ we will write $1 \vee 0 = 0$ and $1 \wedge 0 = 1$, etc. Following this convention, the truth tables for $\wedge$, $\vee$ and $\neg$ are as follows.

| $X$ | $Y$ | $X \wedge Y$ |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

| $X$ | $Y$ | $X \vee Y$ |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

| $X$ | $\neg X$ |
|---|---|
| 1 | 0 |
| 0 | 1 |

A **Boolean expression** is an expression formed by combining Boolean variables with $\wedge$, $\vee$ and $\neg$. Thus $\neg X \vee Y$ and $(X \wedge Y) \vee (\neg X \wedge \neg Y)$ are Boolean expressions, as is $W \wedge (X \vee Y)$. Single variables, like $X$ or $Y$, are also a Boolean expressions.

Boolean expressions do not use $\Rightarrow$ and $\Leftrightarrow$, as these operations can be expressed with $\wedge$, $\vee$ and $\neg$. Indeed, Exercise 3 of the previous section showed $X \Rightarrow Y = \neg X \vee Y$. Similarly, the truth table on page 59 shows that $X \Leftrightarrow Y = (X \wedge Y) \vee (\neg X \wedge \neg Y)$. So as long as we have $\wedge$, $\vee$ and $\neg$, we don't technically need $\Rightarrow$ and $\Leftrightarrow$.

Be aware that 0 and 1 are each themselves considered to be Boolean expressions. The reason is that $X \wedge \neg X = 0$, and $X \vee \neg X = 1$. no matter whether $X$ is 0 or 1. So since 0 and 1 each equal Boolean expressions, they are Boolean expressions.

Given a Boolean expression, you should have no trouble writing its truth table. For example, below is the truth table for $(X \wedge Y) \vee (\neg X \wedge \neg Y)$.

| $X$ | $Y$ | $(X \wedge Y) \vee (\neg X \wedge \neg Y)$ |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

You can think of a Boolean expression, like $(X \wedge Y) \vee (\neg X \wedge \neg Y)$, as being a *function* $f(X, Y)$ of two variables, given by the rule

$$f(X, Y) = (X \wedge Y) \vee (\neg X \wedge \neg Y).$$

For a given input, such as $X = 0$, $Y = 1$, the output is

$$f(0, 1) = (0 \wedge 1) \vee (\neg 0 \wedge \neg 1) = (0 \wedge 1) \vee (1 \wedge 0) = \mathbf{0}.$$

The truth table for $(X \wedge Y) \vee (\neg X \wedge \neg Y)$ talleys the output value for every possible input of $X$ and $Y$. Similarly, an expression like $X \vee (Y \wedge \neg Z)$ is a function of *three* variables, and so on.

Next we consider a type of problem that in a sense is the *reverse* of writing a truth table. Suppose we have in mind a set of outputs for a Boolean expression, but we don't know the expression. For instance, consider the outputs in this table.

| $X$ | $Y$ | ? |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

Can we find a Boolean expression $f(X, Y)$ for the top of the third column, so that this is its truth table?

The only rows with output 1 are the first and third row. The Boolean expression $X \wedge Y$ equals 1 on the first row and 0 on all others. Likewise $\neg X \wedge Y$ equals 1 on the third row and 0 on all others. Therefore $(X \wedge Y) \vee (\neg X \wedge Y)$ equals 1 on the first and third row, but it equals 0 on the other two rows. This gives our answer. Head the third column of the table with the Boolean expression $(X \wedge Y) \vee (\neg X \wedge Y)$, as shown below.

| $X$ | $Y$ | $(X \wedge Y) \vee (\neg X \wedge Y)$ |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

**Example 4.1.** Find a Boolean expression for the following table.

| $X$ | $Y$ | ? |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

**Solution.** This table has output is 1 in the first, second and fourth row, but 0 in the third. The expression $X \wedge Y$ equals 1 on the first row and 0 on all others. Likewise, $X \wedge \neg Y$ equals 1 on the second row and 0 on all others. Also $\neg X \wedge \neg Y$ equals 1 on the fourth row but 0 on all others. Therefore $(X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge \neg Y)$ equals 1 on every row except the third. So this expression can replace the "?" in the table above. ✍

The answer $(X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge \neg Y)$ to this example is not the only answer that works. The simpler expression $X \vee \neg Y$ also works, and would perhaps be a preferable answer. The point is that we have a technique that gives a viable expression for a table. Section 4.3 addresses the issue of simplifying such expressions.

**Example 4.2.** Find a Boolean expression for the following table.

| $X$ | $Y$ | $Z$ | ? |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 |

**Solution.** This table has output 1 only on the fourth, fifth and sixth rows. The expression $X \wedge \neg Y \wedge \neg Z$ equals 1 on the fourth row, but 0 on all other rows. The expression $\neg X \wedge Y \wedge Z$ equals 1 on the fifth row, but is 0 on all other rows. Finally, $\neg X \wedge Y \wedge \neg Z$ equals 1 on the sixth row, but is 0 on all other rows. Therefore

$$(X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge Y \wedge Z) \vee (\neg X \wedge Y \wedge \neg Z)$$

equals 1 on the on the fourth, fifth and sixth rows, but is 0 on all others. This is a Boolean expression for the table. ✍

This kind of problem is important, because in applications you may need to find a Boolean expression that models real-world data that is expressed as a truth table.

The answers to the previous two examples, $(X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge \neg Y)$ and $(X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge Y \wedge Z) \vee (\neg X \wedge Y \wedge \neg Z)$, have a special form. They are made up of parenthesized expressions (called **clauses**) joined by $\vee$, whereas each clause consists of variables (or negations of variables) joined by $\wedge$, Expressions like this are said to have **disjunctive normal form (DNF)**, and we call them **DNF expressions**. For instance, $(X \wedge Z) \vee (\neg X \wedge Y \wedge Z)$ is DNF, but $(X \vee Z) \wedge (\neg X \vee Y \vee Z)$ is not. Also $X \vee (\neg Y \wedge Z) \vee (X \wedge Z)$ is DNF (the leading $X$ is the only variable in its clause, so parentheses are not needed). Likewise, $X \vee Y \vee Z$ is DNF (three one-variable clauses), and so is $X \wedge Y \wedge Z$ (a single 3-variable clause).

In finding a Boolean expression for a truth table (as in Examples 4.1 and 4.2) we arrive at DNF expressions in which each clause contains all variables. We say such an expression is **full-DNF**.

The exercises below ask you to find DNF expressions for truth tables. They should convince you that every Boolean expression is equal to a (possibly different) DNF expression. For example, $X \wedge (Y \vee Z)$ is not DNF, but by the distributive law (Equation (3.4) on page 60), $X \wedge (Y \vee Z) = (X \wedge Y) \vee (X \wedge Z)$, which *is* DNF. Alternatively you could write the truth table for $X \wedge (Y \vee Z)$, then find its DNF as in the examples above.

## Exercises for Section 4.1

Find a DNF Boolean expression for each of the truth tables.

**1.**

| $X$ | $Y$ | ? |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

**2.**

| $X$ | $Y$ | ? |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

**3.**

| $X$ | $Y$ | ? |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

**4.**

| $X$ | $Y$ | ? |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

**5.**

| $X$ | $Y$ | $Z$ | ? |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |

**6.**

| $X$ | $Y$ | $Z$ | ? |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |

**7.**

| $X$ | $Y$ | $Z$ | ? |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |

**8.**

| $X$ | $Y$ | $Z$ | ? |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |

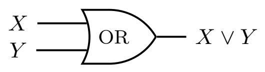
**9.** In addition to disjunctive normal form, there is **conjunctive normal form** (CNF). In CNF, the clauses are joined by $\wedge$, and each clause consists of variables (or their negations) joined by $\vee$. For example, $(X \vee Y \vee \neg Z) \wedge (X \vee \neg Y \vee Z) \wedge (X \vee Z)$ is CNF. Explain how any Boolean expression equals some CNF expression. (Hint: By DeMorgan's law, the negation of DNF is CNF. Also, consider how the truth tables for a Boolean expression and its negation are related.)

## 4.2 Logic Circuits

There is a striking connection between logic and computer circuitry. At the most basic level, a computer's memory can be viewed as a very long sequence or array of Boolean variables (called **bits**), each of which holds a value of 0 or 1. Internally, each 0 may be represented by a low (or zero) voltage, and each 1 by a high voltage. As the computer runs, the variables may change values according to instructions from the central processing unit. At any point in time, the value a particular bit may depend on the values of other bits. The value of that bit, then, can be expressed as a Boolean expression (or function) involving the other bits. Thus Boolean expressions (and hence logic operators) are at the heart of computation.

Certain physical electronic components called *logic gates* emulate the logical operators $\vee$, $\wedge$ and $\neg$. Each gate has a visual schematic description. The so-called **OR gate** is represented as follows. You can think of it as having two input wires labeled $X$ and $Y$ on the left, and one output wire on the right. Each input can carry a value of 0 or 1 (you can think of this as a voltage), and the output is $X \vee Y$. indexOR gate

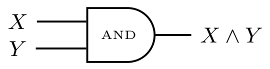
OR gate:



| input | | output |
|---|---|---|
| $X$ | $Y$ | $X \vee Y$ |
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

Likewise, there is an AND gate whose input mirrors the $\wedge$ operator. indexAND gate

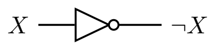
AND gate:



| input | | output |
|---|---|---|
| $X$ | $Y$ | $X \wedge Y$ |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

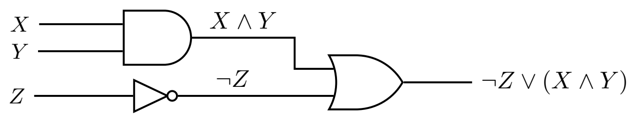
The NOT gate toggles its input, so the output is the opposite value of the input. Its schematic is as follows. indexNOT gate

NOT gate:



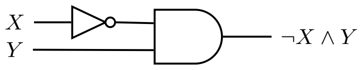
| input | output |
|---|---|
| $X$ | $\neg X$ |
| 1 | 0 |
| 0 | 1 |

These three gates encode the operators $\wedge$, $\vee$ and $\neg$, so any Boolean expression can be built from them. For instance, here is a circuit for $\neg Z \vee (X \wedge Y)$.

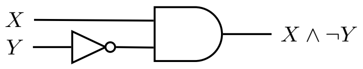


**Example 4.3.** Design a circuit with inputs $X, Y$, and output $(\neg X \wedge Y) \vee (X \wedge \neg Y)$.

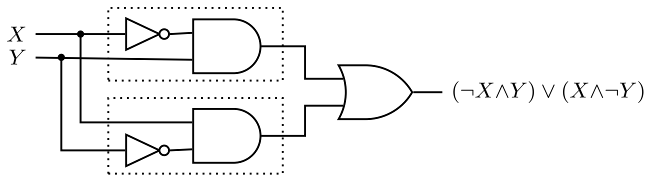
**Solution.** We will build this in pieces. Begin with the circuit below, which negates $X$, then feeds the negation into an AND gate with $Y$. The output is $\neg X \wedge Y$.



Similarly, the output of the circuit below is $X \wedge \neg Y$.



Now hook these together by connecting their $X$ inputs and $Y$ inputs as shown below. Also, feed the two outputs into an OR gate. The final output is $(\neg X \wedge Y) \vee (X \wedge \neg Y)$.

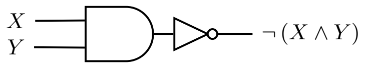


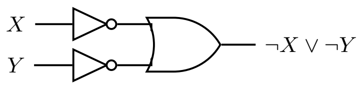
| input | | output |
|---|---|---|
| $X$ | $Y$ | $X \oplus Y$ |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

The resulting circuit is sometimes called the "*exclusive or*," or **XOR**. Its output is 1 if and only if exactly one (but not both) of the two inputs is 1. Exclusive or is sometimes denoted as $X \oplus Y$. ✍

Example 4.3 highlights a convention that we will adopt in diagraming circuits. A solid dot on two wires ┰ means that they are connected. But a crossing without a dot, like this ┼, means that the wires are not connected at the crossing. (Think of one wire as crossing over top the other.)

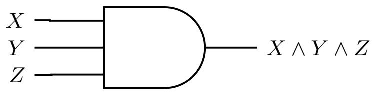
Take note that it is possible for different circuits to describe the same Boolean function. For example, given any input, the two circuit below give identical outputs. The reason is that the first circuit models $\neg(X \wedge Y)$, and the second models $\neg X \vee \neg Y$, and these two expressions are equal by DeMorgan's law.

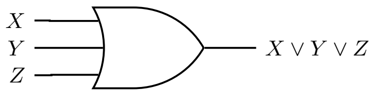




You might regard the circuit on the left as the simpler one, because it uses fewer gates. In the next section explores how a complicated circuit can often be replaced with a much simpler one that gives identical output.

It is possible for AND and OR gates to have more than two inputs, as indicated below. Such gates will be used in the next example.

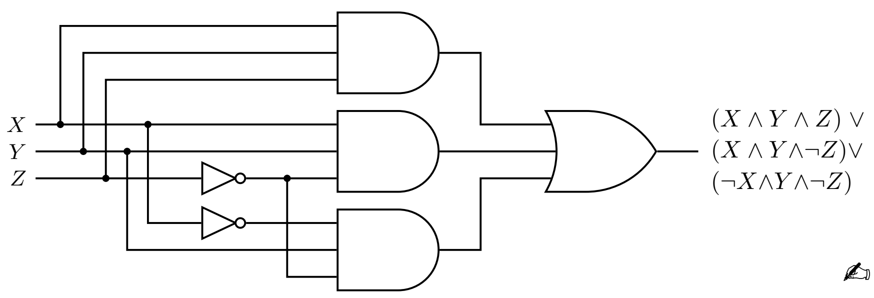




**Example 4.4.** Design a logic circuit that yields the outputs described in this table.

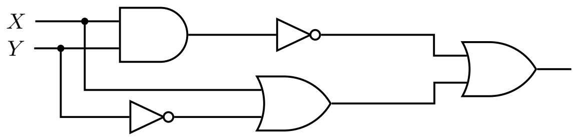
| $X$ | $Y$ | $Z$ | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 |

**Solution.** As explained in the previous section, we can find a Boolean expression for this table, namely $(X \wedge Y \wedge Z) \vee (X \wedge Y \wedge \neg Z) \vee (\neg X \wedge Y \wedge \neg Z)$, which equals 1 on exactly on the table's first, second and sixth line. Here is a corresponding circuit.

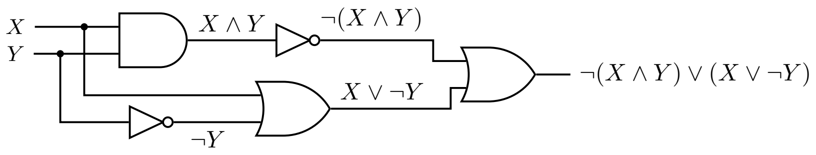

$X$
$Y$
$Z$
$(X \wedge Y \wedge Z) \vee$
$(X \wedge Y \wedge \neg Z) \vee$
$(\neg X \wedge Y \wedge \neg Z)$

✍

**Example 4.5.** Find a Boolean expression for the output of the following circuit.

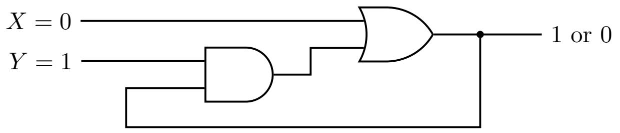


**Solution.** Working from left to right, label the outputs of each gate with its output, as indicated below.



On the right we arrive at the output expression $\neg(X \wedge Y) \vee (X \vee \neg Y)$. ✍

Though any Boolean expression can be modeled with a logic circuit, not just any connection of gates results in a meaningful circuit. Consider the example below. Given the input $X = 0$ and $Y = 1$, either output 0 or 1 fulfills the circuit.



Therefore this circuit does not have a well-defined output for every input. It is not a circuit for any Boolean expression. If you tried to write one, it would have some kind of meaningless feedback loop such as $X \vee (Y \wedge (X \vee (Y \wedge (X \vee (Y \wedge \cdots$

To avoid this kind of ambiguous situation we adhere to two rules in designing circuits. No output of any gate can eventually feed back into that gate's input (as happened in the above circuit), and no two separate input wires can be connected.

It should be mentioned that present-day integrated circuits are not built up from individual, distinct logic gates, and therefore our treatment above is somewhat of a simplification. However, understanding individual logic gates—as outlined in this section—remains a crucial step towards a deeper understanding of computer circuitry.

## Exercises for Section 4.2

**A.** Design a logic circuit whose output matches the given Boolean expression

**1.** $\neg X \wedge (X \vee Y)$

**2.** $\neg\big((X \wedge Y) \vee (\neg X \wedge \neg Y)\big)$

**3.** $\neg(X \wedge (Y \vee Z))$

**4.** $\neg(X \vee Z) \wedge (\neg Y) \wedge (Y \vee \neg Z)$

**5.** $\neg X \vee \neg(Y \vee \neg Z)$

**6.** $\neg(X \wedge Y \wedge Z) \vee (\neg X \vee Y)$

**B.** Design a logic circuit whose output matches the given table.

**7.**

| $X$ | $Y$ | |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

**8.**

| $X$ | $Y$ | |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

**9.**

| $X$ | $Y$ | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

**10.**

| $X$ | $Y$ | |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

**11.**

| $X$ | $Y$ | $Z$ | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |

**12.**

| $X$ | $Y$ | $Z$ | |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |

**13.**

| $X$ | $Y$ | $Z$ | |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |

**14.**

| $X$ | $Y$ | $Z$ | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |

## 4.3 Simplifying Boolean Expressions and Circuits

In Section 4.1 we learned how to find disjunctive normal form (DNF) Boolean expressions for truth tables. For example, for the following truth table would yield the disjunctive normal form $(X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge Y)$ for the third column.

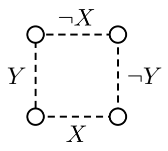
| $X$ | $Y$ | ? |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

But notice that this is also the truth table for $X \vee Y$. We've completed it with $(X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge Y)$, but the much simpler expression $X \vee Y$ would suffice. In fact, the table tells us that $(X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge Y) = X \vee Y$, that is, that the two expressions are logically equivalent, or *equal*.

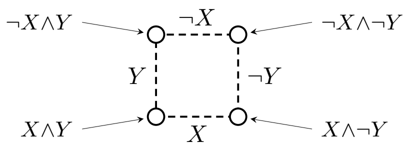
We say that $(X\wedge Y)\vee(X\wedge\neg Y)\vee(\neg X\wedge Y)$ *simplifies* to $X\vee Y$. You already know how to simplify certain *algebraic* expressions. For instance, $(x+y)-(y+z) = x-z$. Now you will learn how to simplify DNF *Boolean expressions*.

First, recall that in a DNF expression like $(X\wedge Y)\vee(X\wedge\neg Y)\vee(\neg X\wedge Y)$, the terms $(X\wedge Y)$, $(X\wedge\neg Y)$ and $(\neg X\wedge Y)$ are called the *clauses* of the expression. So a clause consists of variables (or their negations) joined by $\wedge$ (that is, "ANDed together"). A DNF expression is a collection of clauses joined by $\vee$ (that is, "ORed together").

Let's begin with DNF expressions that have just two variables, say $X$ and $Y$. A simple diagram systematizes the simplification of such expressions. Consider the following square with dashed edges and circles at its corners. The edges are labeled with the two variables, and their negations, so that each variable and its negation appear at opposite edges.



Each corner corresponds to a possible 2-variable clause, as indicated below. For example, edges $\neg X$ and $Y$ meet at the upper-left corner, so that corner corresponds to $(\neg X \wedge Y)$. In this way, the edges correspond to possible 1-variable clauses, and the corners correspond to possible 2-variable clauses.

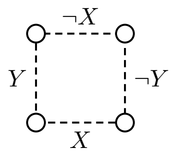


The next two examples illustrate how this diagram can help simplify Boolean expressions. After the examples, we will carefully formalize the technique.

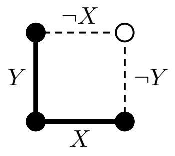
The first example shows how to simplify $(X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge Y)$, which we already happen to know (from the previous page) simplifies to $X \vee Y$.

**Example 4.6.** Simplify $(X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge Y)$.

**Solution.** Begin by drawing a dashed square with edges labeled $X, \neg X, Y, \neg Y$, such that each pair of opposite edges is labeled by a variable and its negation.



Next, darken each corner corresponding to a clause in $(X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge Y)$. For example, $(X \wedge Y)$ is a clause, so darken the corner where edges $X$ and $Y$ meet. And $(X \wedge \neg Y)$ is a clause, so darken the corner where edges $X$ and $\neg Y$ meet. Finally, $(\neg X \wedge Y)$ is a clause, so darken the corner where edges $\neg X$ and $Y$ meet.

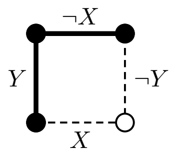


Next, make solid any edge that joins two solid dots. Edges $X$ and $Y$ are bold, and this indicates that $\boxed{\text{the expression simplifies to } X \vee Y.}$

This works because an edge label equals 1 if and only if a clause at its endpoints equals 1. For instance, $X = 1$ if and only $X \wedge Y = 1$ or $X \wedge \neg Y = 1$ (check this). Thus, if either of the solid edges equals 1 (that is, if $X \vee Y = 1$), then at least one of darkened clauses equals 1. In other words, $(X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge Y) = 1$ if and only if $X \vee Y = 1$. ✍

**Example 4.7.** Simplify $(X \wedge Y) \vee (\neg X \wedge Y) \vee (\neg X \wedge \neg Y)$.

**Solution.** Fill in the corners corresponding to the three clauses to get this square.

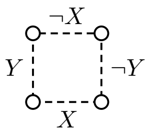


Edges $\neg X$ and $Y$ join solid dots, to the expression simplifies to $\neg X \vee Y$. (You can confirm this by showing that $(X \wedge Y) \vee (\neg X \wedge Y) \vee (\neg X \wedge \neg Y)$ and $\neg X \vee Y$ have the same truth table.) ✍

One final comment before summarizing our technique. For expressions such as $(X \wedge Y) \vee (\neg X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge \neg Y)$, all four corners are darkened. Note that this expression equals 1. If all corners are darkened, the simplification is 1.

**Fact 4.1.** How to simplify a DNF expression that has two variables $X$ and $Y$.

1. Draw a dashed square, labeled as shown below.



   Darken the corners corresponding to the expression's 2-variable clauses. Make solid all edges (if any) corresponding to the expression's 1-variable clauses, and darken their endpoints. Finally, make solid the remaining edges (if any) that join two darkened corners.

2. If all four corners are darkened, then the expression simplifies to 1. If no corners are darkened, then the expression simplifies to 0. Otherwise, OR together the following expressions:
   (a) the clauses corresponding to filled darkened *not* on a solid edge.
   (b) the labels of any solid edges.

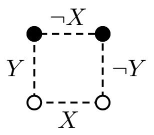
   The result is the simplified expression.

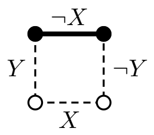
Step 2 mentions a situation in which no corners are darkened. How is this possible? If the expression has a clause like $(X \wedge \neg X)$, containing both a variable and its negation, then no corner meets *both* edges $X$ and $\neg X$. In this rare situation, there is no corner to darken for the clause. Such a clause equals 0. If all clauses are like this, no corner is darkened, and the entire expression equals 0.

Another rare but possible scenario concerns clauses like $(X \wedge X \wedge Y)$ with a repeated term. Because $X \wedge X = X$, the extra $X$ is superfluous. For this clause, darken the corner for $X \wedge Y$, etc.

**Example 4.8.** Simplify $(\neg X \wedge Y) \vee (\neg X \wedge \neg Y)$.

**Solution.** Darken the corners for the two clauses to get the square below (left). Then make solid all edges joining darkened corners. There is only one, labeled $\neg X$. We get the square below (right). This completes Step 1.

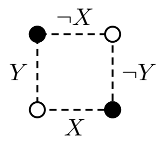




Step 2(b) of Fact 4.1 says to OR together the labels of solid edges. There is only one solid edge, labeled $\neg X$, so the simplified expression is $\neg X$. ✍

**Example 4.9.** Simplify $(\neg X \wedge Y) \vee (X \wedge \neg Y)$.

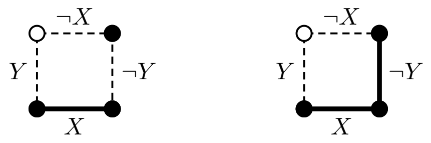
**Solution.** Step 1 of Fact 4.1 says to darken the corners corresponding to the two clauses, and make solid the edge connecting them. This yields the following square.



No edge joins two darkened corners, so there are no solid edges. Step 2(a) says to OR together the clauses corresponding to the darkened corners. This results in $(\neg X \wedge Y) \vee (X \wedge \neg Y)$. Note that there is no simplification in this case. ✍

**Example 4.10.** Simplify $X \vee (\neg X \wedge \neg Y)$.

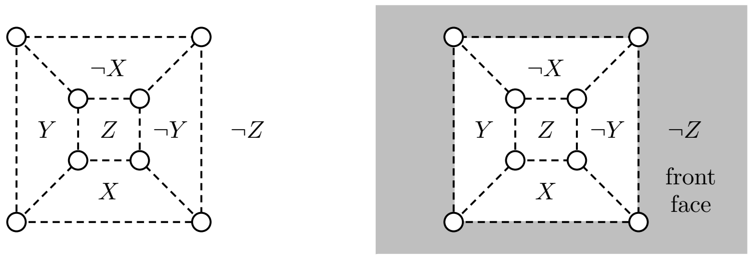
**Solution.** There are two clauses, the 1-variable clause $X$ and the 2-variable clause $(\neg X \wedge \neg Y)$. Step 1 of Fact 4.1 says to darken the corner for $(\neg X \wedge \neg Y)$, then make solid the edge $X$ and darken its endpoints. This results in the diagram below (left). Continuing Step 1, solid in the edge $\neg Y$ that joins darkened corners (below, right).



Step 2(a) of Fact 4.1 says to OR together the clauses corresponding to the solid edges. This results in the simplification $X \vee \neg Y$. ✍

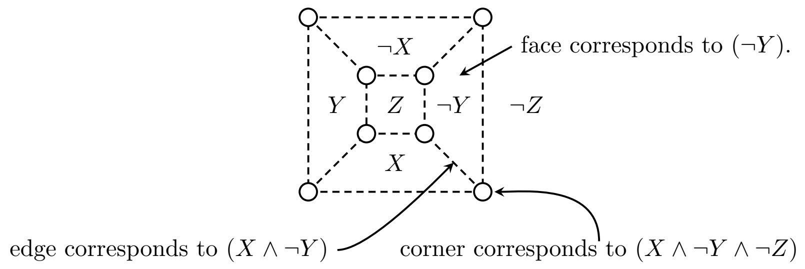
You may find that you can sometimes reason out a simplification without doing a diagram. That is good.

Next we consider simplifying DNF expressions with *three* variables, say $X, Y, Z$. The diagram we will need for this is not a *square*, but a *cube*. Label the six *faces* of a cube with $X$,$\neg X$, $Y$, $\neg Y$, $Z$ and $\neg Z$ as shown below (left). Notice that each variable and its negation are on opposite faces.



In this view, the "front" face (the one bounded by the largest face) overlaps the other five faces. It's best to regard this front face as a region *outside* the largest square of the cube, as shown shaded above (right). This front face is labeled $\neg Z$.

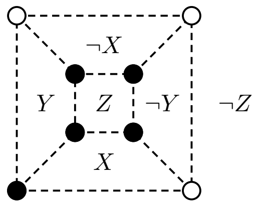
Notice how each corner (circle) touches three faces and thus corresponds to the 3-variable clause formed by the three face labels. Similarly, any (dashed) edge touches two labeled faces, so it corresponds to a 2-variable clause formed by the two face labels. In turn, each face represents a 1-variable clause, as indicated below.


$\neg X$
face corresponds to $(\neg Y)$.
$Y$
$Z$
$\neg Y$
$\neg Z$
$X$
edge corresponds to $(X \wedge \neg Y)$
corner corresponds to $(X \wedge \neg Y \wedge \neg Z)$

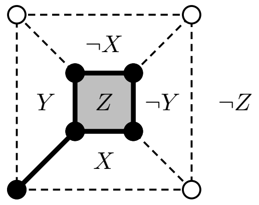
The next example show how to use this cube diagram to simplify a 3-variable DNF expression. After the example, we will state the procedure in detail.

**Example 4.11.** Simplify the following expression:
$(X \wedge Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge Y \wedge Z) \vee (X \wedge Y \wedge \neg Z)$.

**Solution.** Draw the cube, and darken the corners corresponding to the clauses in the above expression. For example, the first clause is $(X \wedge Y \wedge Z)$, so darken the circle that touches the faces labeled $X, Y$ and $Z$. The second clause is $(X \wedge \neg Y \wedge Z)$, so darken the circle that touches faces labeled $X, \neg Y, Z$, and so on.



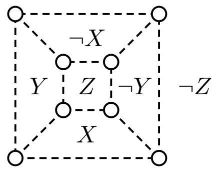
Now shade any face that has four darkened corners, and also make solid any edge that has both endpoints darkened.



Face $Z$ is shaded, and it corresponds to the clause $Z$. And there is one solid edge not on a shaded face, and it corresponds to the clause $(X \wedge Y)$. The simplified expression is the OR of these two clauses, namely $Z \vee (X \wedge Y)$. ✍

**Fact 4.2.** How to simplify a DNF expression that has three variables $X, Y, Z$.

1. Draw a dashed cube with faces labeled as indicated. Darken all corners corresponding to the expression's 3-variable clauses. Make solid all edges corresponding to the 2-variable clauses (if any) and darken their endpoints. Shade all faces corresponding to 1-variable clauses (if any), darken their corners, and make their edges solid.

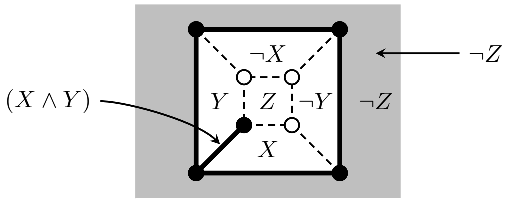


   Make solid any remaining dashed edges have both endpoints darkened. Shade any remaining unshaded faces that have all corners darkened.

2. If all eight corners are darkened, then the expression equals 1.
   If no corner is darkened, then the expression equals 0.
   Otherwise, the simplification is the OR of the following expressions:
   (a) all 1-variable clauses (if any) corresponding to shaded faces,
   (b) all 2-variable clauses (if any) corresponding to solid edges that are *not* edges of a shaded face,
   (c) all 3-variable clauses (if any) corresponding to darkened circles that are *not* endpoints of solid edges.

**Example 4.12.** Simplify the following expression:
$(X \wedge Y \wedge \neg Z) \vee (X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge Y \wedge \neg Z) \vee (X \wedge Y \wedge Z)$.

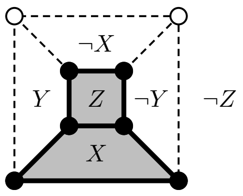
**Solution.** All clauses have three variables. Darken the corners corresponding to the clauses, as shown below. The outside face $\neg Z$ has all corners darkened, so shade it and make its edges solid. The edge between faces $X$ and $Y$ has both endpoints darkened, so make it solid.



Face $\neg Z$ is shaded, and it corresponds to the clause $\neg Z$. And there is one solid edge between unshaded faces $X$ and $Y$, corresponding to the clause $(X \wedge Y)$. The simplified expression is the OR of these clauses, namely $(X \wedge Y) \vee \neg Z$. ✍

**Example 4.13.** Simplify the following expression:
$(X \wedge Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge Y \wedge Z) \vee (X \wedge Y \wedge \neg Z) \vee (X \wedge \neg Y \wedge \neg Z)$.

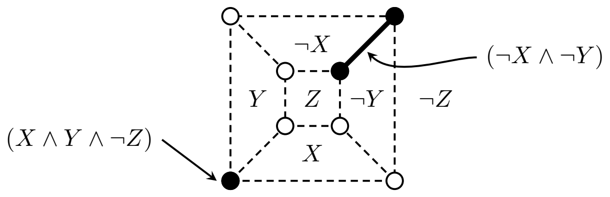
**Solution.** Draw the cube diagram and darken the corners for the six clauses. Faces $X$ and $Z$ have all corners darkened, so shade them and make their edges solid.



The simplified expression is $X \vee Z$. ✍

**Example 4.14.** Simplify $(X \wedge Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge \neg Z)$.

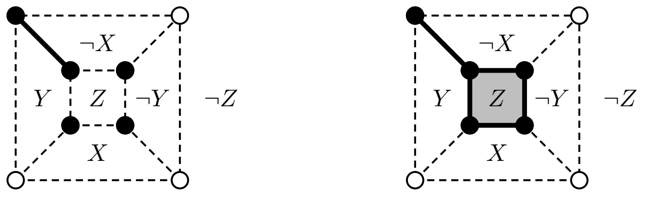
**Solution.** Darken the corners corresponding to the three clauses. No face has all corners darkened, so there is no face to shade. There *is* an edge joining darkened corners, so make it solid.



The simplified expression is $(\neg X \wedge \neg Y) \vee (X \wedge Y \wedge \neg Z)$. ✍

**Example 4.15.** Simplify $(X \wedge Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge Y)$.

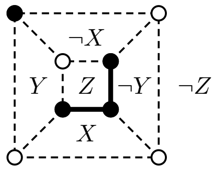
**Solution.** Draw the cube and darken the corners corresponding to the first three clauses. The last clause has just two variables. It corresponds to the edge between faces $\neg X$ and $Y$, so make that edge bold and darken its endpoints. See below (left).



The face $Z$ has all four corners darkened, so shade it in and make its edges bold (above, right). From this we read off the simplified expression as $Z \vee (X \wedge \neg Y)$. ✍

**Example 4.16.** Simplify $(\neg X \wedge Y \wedge \neg Z) \vee (X \wedge Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z)$.

**Solution.** The filled-in cube is as follows.



The simplified expression is $(\neg X \wedge Y \wedge \neg Z) \vee (X \wedge Z) \vee (\neg Y \wedge Z)$. ✍

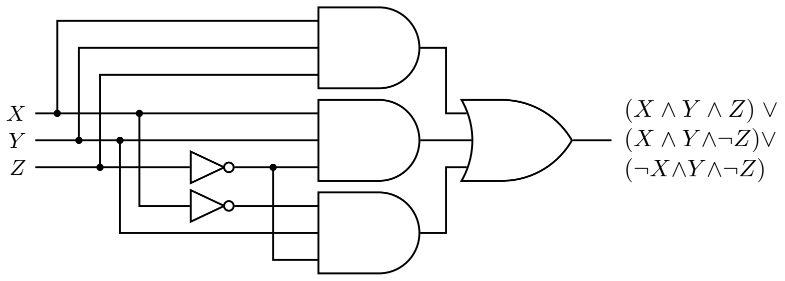
As you might expect, simplifying DNF expressions with *four* variables would involve diagrams resembling four-dimensional cubes. We will not pursue this, though we *will* discuss four-dimensional cubes in Chapter 16.

Simplifying DNF expressions has important applications in circuit design. If a circuit corresponds to a Boolean expression that can be simplified, then the circuit can be simplified too. The simpler circuit is then more economical, efficeint and robust. The next example, which is a continuation of Example 4.4, illustrates this. Example 4.4 gave a truth table and asked for a circuit whose output matched the table. Now we repeat Example 4.4, but then go on to *simplify* the circuit.

**Example 4.17.** Find the simplest circuit that has the following outputs.

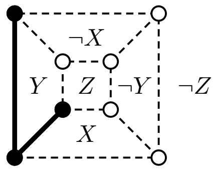
| $X$ | $Y$ | $Z$ | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 |

**Solution.** In Example 4.4, we found a DNF expression for this table, namely $(X \wedge Y \wedge Z) \vee (X \wedge Y \wedge \neg Z) \vee (\neg X \wedge Y \wedge \neg Z)$. Here is the corresponding circuit.

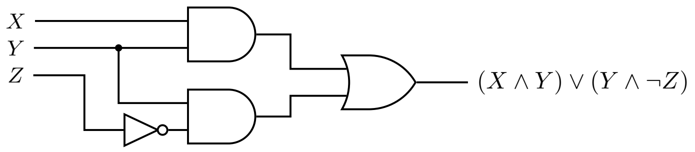


$X$
$Y$
$Z$
$(X \wedge Y \wedge Z) \vee$
$(X \wedge Y \wedge \neg Z) \vee$
$(\neg X \wedge Y \wedge \neg Z)$

To simplify the output expression $(X \wedge Y \wedge Z) \vee (X \wedge Y \wedge \neg Z) \vee (\neg X \wedge Y \wedge \neg Z)$, we draw the following diagram.



This reveals that $(X \wedge Y \wedge Z) \vee (\neg X \wedge Y \wedge \neg Z) \vee (X \wedge Y \wedge \neg Z) = (X \wedge Y) \vee (Y \wedge \neg Z)$. This simplified expression yields the simplified circuit shown below.



This simplified circuit has only four gates, wheres the original circuit had six. ✍

The simplification techniques (Facts 4.1 and 4.2) described here are adapted from a technique known as *Karnaugh maps*, which can handle up to five or so variables. Unfortunately, the general problem of simplifying boolean expressions (even full-DNF) with more than seven variables is known to be extremely difficult.

## Exercises for Section 4.3

In exercises 1-10, simplify the DNF expression, if possible.

**1.** $(X \wedge Y) \vee (\neg X \wedge \neg Y)$ **2.** $(\neg X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge \neg Y)$

**3.** $(X \wedge Y \wedge Z) \vee (\neg X \wedge Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge Y \wedge \neg Z)$

**4.** $(X \wedge Y \wedge Z) \vee (X \wedge Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge \neg Z)$

**5.** $(X \wedge Y \wedge Z) \vee (X \wedge Y \wedge \neg Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z)$

**6.** $(\neg X \wedge Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (X \wedge \neg Y \wedge \neg Z)$

**7.** $(X \wedge Y \wedge Z) \vee (X \wedge Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge \neg Z)$

**8.** $(X \wedge Y \wedge Z) \vee (\neg X \wedge Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (X \wedge Y \wedge \neg Z) \vee (\neg X \wedge Y \wedge \neg Z) \vee (X \wedge \neg Y \wedge \neg Z)$

**9.** $(X \wedge Y \wedge Z) \vee (\neg X \wedge Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge Y \wedge \neg Z) \vee (X \wedge \neg Y \wedge \neg Z)$

**10.** $(X \wedge Y \wedge Z) \vee (\neg X \wedge Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge Y \wedge \neg Z) \vee$
$(X \wedge \neg Y \wedge \neg Z) \vee (X \wedge Y \wedge \neg Z)$

**11.** Draw the simplest circuit whose output is given by the expression in Exercise 5 above.

**12.** Draw the simplest circuit whose output is given by the expression in Exercise 7 above.

**13.** Draw the simplest circuit whose output is given by the expression in Exercise 9 above.

## 4.4 Case Study: Binary Addition Circuits

This section introduces the idea that logic circuits can perform numeric calculations. But to keep the discussion simple, we will consider only addition of binary integers. However, the ideas and approaches introduced here can be extended to any numeric operation. Indeed, the topic of this section—if taken further—is the basis for the internal workings of any computer. (If you are so inclined, you can explore this in a more advanced course.)

Recall (from Section 1.5, if necessary), that a binary integer can be represented as a sting of 0's and 1's, like 1011010. Each digit therefore has only two possible values, 0 or 1, so we can think of an $n$-digit binary number as $n$ different inputs to a logic circuit. Below we will design a circuit whose inputs are two binary numbers, and whose output is their sum.

To begin, think of the simple process of adding two 1-digit binary numbers. The four possibilities are shown below.

$$\begin{array}{r}0\\+\ 0\\\hline 0\end{array}\qquad\begin{array}{r}1\\+\ 0\\\hline 1\end{array}\qquad\begin{array}{r}0\\+\ 1\\\hline 1\end{array}\qquad\begin{array}{r}1\\+\ \ 1\\\hline 1\,0\end{array}$$

The first three sums have 1-digit answers, but the last one, which expresses *one plus one equals two*, has a 2-digit answer. To be totally uniform, lets agree to pad the first three answers with an extra 0 on the right, like this.

$$\begin{array}{r}0\\+\ \ 0\\\hline 0\,0\end{array}\qquad\begin{array}{r}1\\+\ \ 0\\\hline 0\,1\end{array}\qquad\begin{array}{r}0\\+\ \ 1\\\hline 0\,1\end{array}\qquad\begin{array}{r}1\\+\ \ 1\\\hline 1\,0\end{array}$$

So we are adding binary digits $X$ and $Y$ get a 2-digit binary number $CS$, as follows.

$$\begin{array}{r}X\\+\ \ Y\\\hline C\ S\end{array}$$

We call $S$ the **sum bit** and $C$ the **carry bit**, for reasons that will soon be apparent.

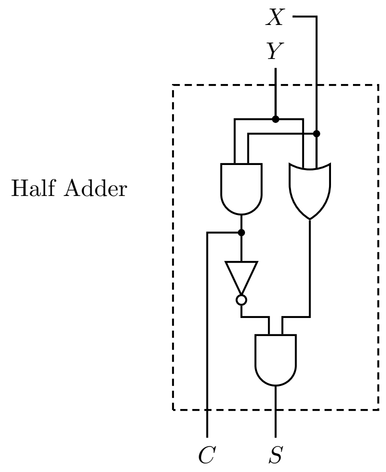
Notice that $S = 1$ precisely when exactly one of $X$ or $Y$ equals 1. That is, $S = 1$ when $X$ or $Y$ is 1, but they are not both 1. In symbols,

$$S = (X \vee Y) \wedge \neg(X \wedge Y).$$

Further, $C = 1$ exactly when both $X$ and $Y$ are 1, which is to say

$$C = X \wedge Y.$$

Figure 4.1 shows the circuit corresponding to these Boolean expressions. It is oriented vertically to echo the process of column addition. There are two inputs $X$ and $Y$ at the top, and two outputs $C$ and $S$ at the bottom, and their relationship is $CS = X + Y$. Thus the circuit indeed adds two 1-digit binary numbers. This circuit is called a **half adder**. (It is our first example of a circuit with *two* outputs.)



$$\begin{array}{rr} & X \\ + & Y \\ \hline & C\ S \end{array} \begin{array}{l} \left.\vphantom{\begin{array}{c}X\\Y\end{array}}\right\} \text{input} \\ \longleftarrow \text{output} \end{array}$$

$$S = (X \vee Y) \wedge \neg(X \wedge Y)$$
$$C = X \wedge Y$$

Fig. 4.1 The **half adder** adds two 1-digit binary numbers $X$ and $Y$, yielding a 2-digit number $CS$.

Before expanding this to a circuit that adds multi-digit binary numbers, it is helpful to review the process of adding two binary numbers by hand. As an example, suppose we wish to perform the following addition.

$$\begin{array}{rr} & 1\,0\,1\,1\,1\,1 \\ + & 1\,0\,1\,0 \\ \hline \end{array}$$

First, add the right-most digits (the digits in the one's place). We get $1 + 0 = 1$.

$$\begin{array}{rr} & 1\,0\,1\,1\,1\,1 \\ + & 1\,0\,1\,0 \\ \hline & 1 \end{array}$$

Next, add the digits in the two's place. We get $1 + 1 = 10$, but we carry the 1.

$$\begin{array}{rr} & {\scriptstyle 1}\phantom{\,1\,1} \\ & 1\,0\,1\,1\,1\,1 \\ + & 1\,0\,1\,0 \\ \hline & 0\,1 \end{array}$$

Now add the digits in the four's place. We get $1 + 1 + 0 = 10$, and we carry the 1.

$$\begin{array}{rr} & {\scriptstyle 1\,1}\phantom{\,1\,1} \\ & 1\,0\,1\,1\,1\,1 \\ + & 1\,0\,1\,0 \\ \hline & 0\,0\,1 \end{array}$$

Now add the digits in the eight's place. We get $1 + 1 + 1 = 11$, and we carry again.

$$\begin{array}{rr} & {\scriptstyle 1\,1\,1}\phantom{\,1\,1} \\ & 1\,0\,1\,1\,1\,1 \\ + & 1\,0\,1\,0 \\ \hline & 1\,0\,0\,1 \end{array}$$

Continuing in this fashion yields the final sum.

$$\begin{array}{r} {\scriptstyle 1\,1\,1\phantom{\,1\,1\,1}} \\ 1\,0\,1\,1\,1\,1 \\ +\quad 1\,0\,1\,0 \\ \hline 1\,1\,1\,0\,0\,1 \end{array}$$

We have performed the addition $47 + 10 = 57$, in binary.

In column-adding binary numbers, as above, whenever a column has a cary digit, *three* 1-digit numbers in that column must be added. A typical column is as follows.

$$\begin{array}{r} C \\ X \\ +\quad Y \\ \hline C'\ S \end{array}$$

Here $C$ is the carry digit, which is added to the two digits below it. This results in a 2-digit binary number $C'S$, where $C'$ a new carry digit for the next column.

Consequently, a key ingredient of a circuit that adds binary numbers is a circuit that adds *three* 1-digit binary numbers. We turn out attention to this now.

Here are the eight possible ways that three 1-digit binary numbers can add.

$$\begin{array}{r} 0 \\ 0 \\ +\quad 0 \\ \hline 0\,0 \end{array} \quad \begin{array}{r} 1 \\ 0 \\ +\quad 0 \\ \hline 0\,1 \end{array} \quad \begin{array}{r} 0 \\ 1 \\ +\quad 0 \\ \hline 0\,1 \end{array} \quad \begin{array}{r} 0 \\ 0 \\ +\quad 1 \\ \hline 0\,1 \end{array} \quad \begin{array}{r} 0 \\ 1 \\ +\quad 1 \\ \hline 1\,0 \end{array} \quad \begin{array}{r} 1 \\ 0 \\ +\quad 1 \\ \hline 1\,0 \end{array} \quad \begin{array}{r} 1 \\ 1 \\ +\quad 0 \\ \hline 1\,0 \end{array} \quad \begin{array}{r} 1 \\ 1 \\ +\quad 1 \\ \hline 1\,1 \end{array}$$

The circuit in Figure 4.2, called a **full adder**, performs such additions. It adds $X$ and $Y$ with a half adder, then adds $C$ to their sum with a second half adder. indexfull adder

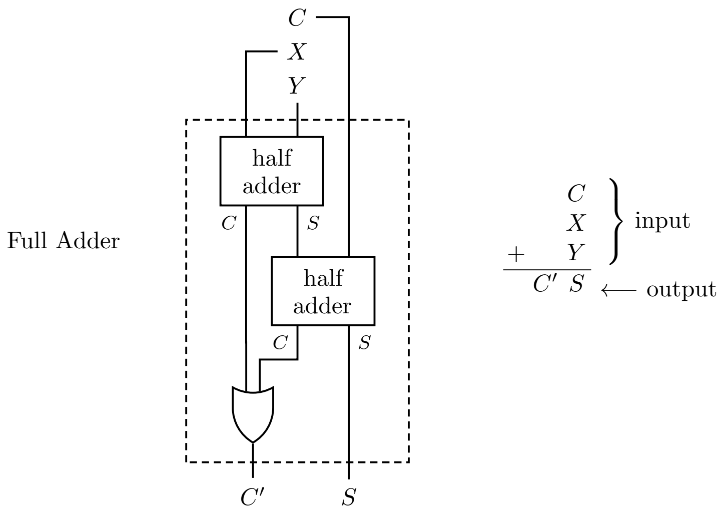


Fig. 4.2 The **full adder** adds three 1-digit binary numbers $C, X, Y$, yielding a 2-digit number $C'S$.

Notice that the only way that $C + X + Y$ can produce a carry is if the sum

$X+Y$ does, or if the sum $C+(X+Y)$ does. Therefore, in the full adder, the carry digits of the two half adders feed into an OR gate to produce a carry digit $C'$. You can check that the full adder gives the correct output for all eight 0-1 combinations for $C$, $X$ and $Y$.

Now that we have developed half and full adders, we can string them together to design a circuit that mimics column addition of binary numbers.

Denote an arbitrary binary number as a sequence like $X_5X_4X_3X_2X_1X_0$, with subscripts increasing right-to-left. Then for any subscript $i$, the digit $X_i$ (which is either 0 or 1) is the digit in the $2^i$'s place.

With this convention, the column-addition of two typical binary numbers is organized as follows.

$$\begin{array}{cccccc} & X_4 & X_3 & X_2 & X_1 & X_0 \\ + & Y_4 & Y_3 & Y_2 & Y_1 & Y_0 \\ \hline S_5 & S_4 & S_3 & S_2 & S_1 & S_0 \end{array}$$

The usual hand computation (adding column by column, and carrying) of such a sum is mirrored—and automated—by the circuit shown in Figure 4.3. The pattern could be extended indefinitely to the right, to accommodate the addition of arbitrarily large numbers.

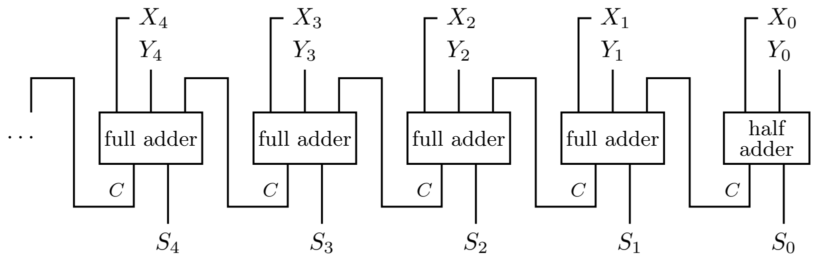


Fig. 4.3 A binary addition circuit.

As mentioned before, the discussion here is intended only to provide a taste of the subject of numeric computation. Not only have we omitted other operations like division or multiplication, but we have considered only addition of *positive integers*. A more complete treatment would include computer implementation of negative and rational numbers.

## Solutions for Chapter 4

### Section 4.1

**1.** $(X \wedge \neg Y) \vee (\neg X \wedge Y)$

**3.** $(X \wedge Y) \vee (\neg X \wedge Y)$

**5.** $(X \wedge \neg Y \wedge Z) \vee (X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge \neg Z)$

**7.** $(X \wedge Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z)$

**9.** Consider a Boolean expression $f(X, Y, Z, \ldots)$ with some number of variables. In Section 4.1 we saw how any Boolean expression, has a disjunctive normal form. So put the Boolean expression $\neg f(X, Y, Z, \ldots)$ into disjunctive normal form. For example, perhaps

$$\neg f(X, Y, Z) = (X \wedge \neg Y \wedge Z) \vee (X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z)$$

By DeMorgan's law, the negation of this is

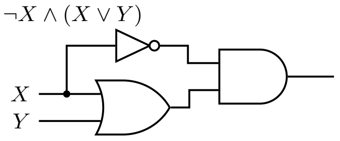
$$\begin{aligned} f(X, Y, Z) = \neg\Big(&(X \wedge \neg Y \wedge Z) \vee (X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z)\Big) \\ &\neg(X \wedge \neg Y \wedge Z) \wedge \neg(X \wedge \neg Y \wedge \neg Z) \wedge \neg(\neg X \wedge \neg Y \wedge Z) \\ &(\neg X \vee Y \vee \neg Z) \wedge (\neg X \vee Y \vee Z) \wedge \neg(X \vee Y \vee \neg Z), \end{aligned}$$

which is conjunctive normal form.

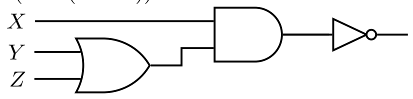
### Section 4.2

**A.** Design a logic circuit whose output matches the given Boolean expression

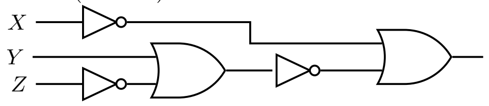
**1.** $\neg X \wedge (X \vee Y)$



**3.** $\neg(X \wedge (Y \vee Z))$



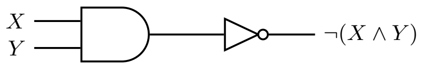
**5.** $\neg X \vee \neg(Y \vee \neg Z)$



**B.** Design a logic circuit whose output matches the given table.

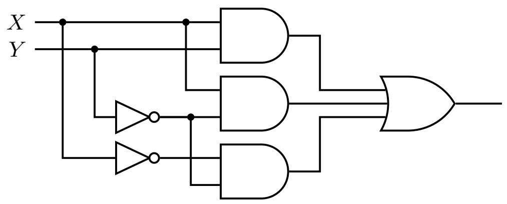
**7.** The full-DNF expression for this table is $(X \wedge \neg Y) \vee (\neg X \wedge Y) \vee (\neg X \wedge \neg Y)$. But note that this is the table for $\neg(X \wedge Y)$, so a simpler circuit is shown below.

| $X$ | $Y$ | |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |



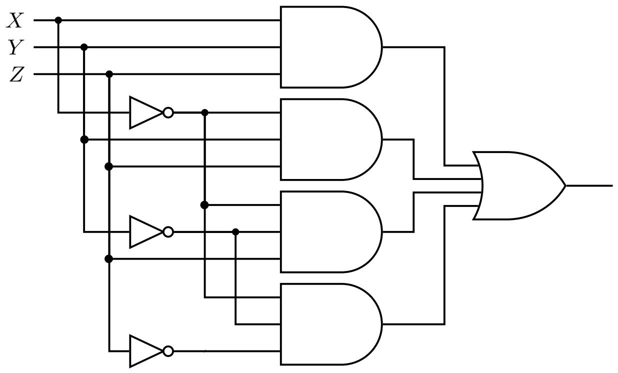
**9.** The full-DNF expression for this table is $(X \wedge Y) \vee (X \wedge \neg Y) \vee (\neg X \wedge \neg Y)$. The corresponding circuit is shown below.

| $X$ | $Y$ | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |



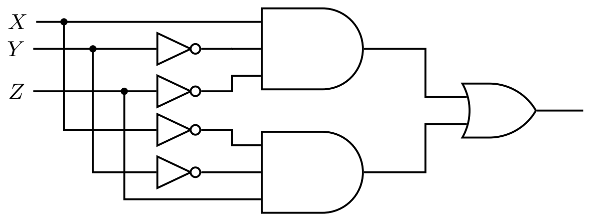
**11.** The full-DNF is $(X \wedge Y \wedge Z) \vee (\neg X \wedge Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge \neg Z)$. The corresponding circuit is shown below.

| $X$ | $Y$ | $Z$ | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |



**13.** The full-DNF expression for this table is $(X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z)$. The corresponding circuit is shown below.

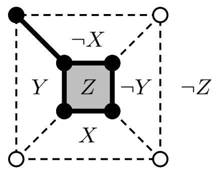
| $X$ | $Y$ | $Z$ | |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |



## Section 4.3

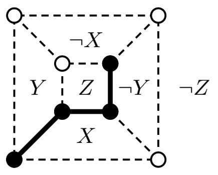
**1.** $(X \wedge Y) \vee (\neg X \wedge \neg Y)$ No simplification. (Square has no solid edges.)

**3.** $(X \wedge Y \wedge Z) \vee (\neg X \wedge Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge Y \wedge \neg Z)$



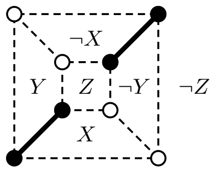
Simplification: $Z \vee (\neg X \wedge Y)$

**5.** $(X \wedge Y \wedge Z) \vee (X \wedge Y \wedge \neg Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge Z)$



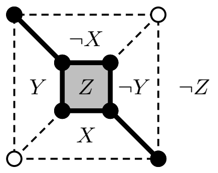
Simplification: $(X \wedge Y) \vee (X \wedge Z) \vee (\neg Y \wedge Z)$

**7.** $(X \wedge Y \wedge Z) \vee (X \wedge Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (\neg X \wedge \neg Y \wedge \neg Z)$



Simplification: $(X \wedge Y) \vee (\neg X \wedge \neg Y)$

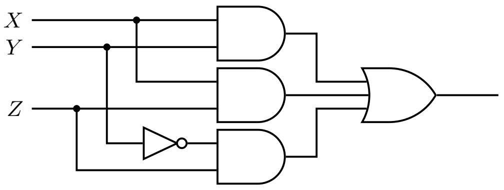
**9.** $(X\wedge Y\wedge Z)\vee(\neg X\wedge Y\wedge Z)\vee(X\wedge\neg Y\wedge Z)\vee(\neg X\wedge\neg Y\wedge Z)\vee(\neg X\wedge Y\wedge\neg Z)\vee(X\wedge\neg Y\wedge\neg Z)$



Simplification: $Z \vee (X \wedge \neg Y) \vee (\neg X \wedge Y)$

**11.** Draw the simplest circuit whose output is given by the expression in Exercise 5 above.

The simplified expression in Exercise 5 is $(X \wedge Y) \vee (X \wedge Z) \vee (\neg Y \wedge Z)$. The corresponding circuit is shown below.



**13.** Draw the simplest circuit whose output is given by the expression in Exercise 9 above.

The simplified expression in Exercise 9 is $Z\vee(X\wedge\neg Y)\vee(\neg X\wedge Y)$. The corresponding circuit is shown below.