

Notes on Boolean Functions: A Study Guide and Exercises for CS 64

Robert R. Snapp
Department of Computer Science
University of Vermont
Burlington, VT 05405 U.S.A.
snapp@cs.uvm.edu

September 4, 2009

1 Enumerating the Boolean Functions

Our goal is to enumerate the functions of the form $f : \mathbb{B}^n \rightarrow \mathbb{B}$, where $\mathbb{B} \triangleq \{0, 1\}$. We begin with a useful counting trick, called *the multiplication principle*:

If a construction or process can be accomplished in exactly n stages such that the number of distinct ways of performing the i -th step, v_i , for $i = 1, \dots, n$, is independent of the outcomes of all previous states, then the number of distinct ways of performing this construction is given by

$$\prod_{i=1}^k v_i = v_1 \times v_2 \times \dots \times v_n. \quad (1)$$

The multiplication principle is easy to demonstrate for the case in which $n = 2$ by designing a table (or spreadsheet) with v_1 rows and v_2 columns. (Each row contributes to each way the first step can be performed, and each column to each way that the second step can be performed.) Since each cell represents a distinct construction, there are exactly $v_1 \times v_2$ distinct constructions. Likewise, the case $n = 3$ is demonstrated using a three-dimensional table with v_1 rows, v_2 columns, and v_3 slices, which results in $v_1 \times v_2 \times v_3$ distinct cells. Later in the semester we will learn a proof method called *mathematical induction*, which can be used to rigorously extend the multiplication principle to every positive integer n .

Example 1.1. *In order to count the number of permutations that exist for the four symbols A, B, C, D, note that the number of ways of choosing the first symbol (stage 1) is $v_1 = 4$. Once the first symbol has been chosen, exactly three choices remain for the second symbol (stage 2), so $v_2 = 3$. Likewise, independent of the previous choices, $v_3 = 2$ and $v_4 = 1$, as only once unused symbol remains for stage 4. Thus the number of permutations equals $4 \times 3 \times 2 \times 1 = 4!$.*

Before we enumerate the number of Boolean functions, $f : \mathbb{B}^n \rightarrow \mathbb{B}$, we first enumerate the number of different input patterns that reside in the domain \mathbb{B}^n . If we represent the image of the function, according to custom, as

$$f(x_1, x_2, \dots, x_n)$$

we see that each argument $x_i \in \{0, 1\}$ can assume one of two possible values. Thus $v_i = 2$ for $i = 1, \dots, n$, and thus from Eqn. (1) there are 2^n binary patterns (or Boolean vectors) in the function domain \mathbb{B}^n . These domains are readily graphed for dimensions one through four in Fig. 1.

In the case of $n = 1$, the number of Boolean functions $f : \mathbb{B} \rightarrow \mathbb{B}$ equals the number of different ways that the output values 0 and 1 can be assigned to the input vertices 0 and 1. The following table suggests that this can be done in four ways

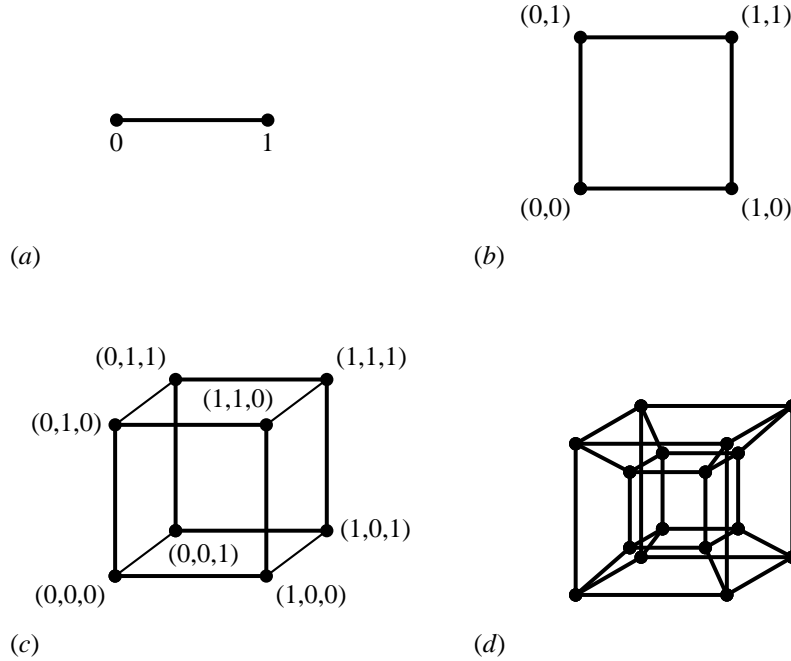


Figure 1: Graphs of the Boolean domains \mathbb{B}^n for (a) $n = 1$, (b) $n = 2$, (c) $n = 3$, and (d) $n = 4$ (with vertex labels omitted). In each graph the edges connect the vertices that differ in exactly one bit.

x	$f_1(x) = 0$	$f_2(x) = x$	$f_3(x) = \neg x$	$f_4(x) = 1$
0	0	0	1	1
1	0	1	0	1

Each of the column of zeros and ones beneath each column to the right of the double vertical bars designates a *truth table* for the function in its heading. Here, f_1 and f_4 correspond to constant functions, as their ranges consist of a single value. The function $f_2(x) = x$ is called the *identity function*.

The most useful of the four is $f_3(x) = \neg x$, which goes by many different names, e.g., the *complement*, the *negation*, the *inverse*, or the NOT of x . The symbol \neg should be regarded as a unary operator, which is immediately applied to the next symbol or expression enclosed in parentheses, just like the negative sign in arithmetic. Some authors prefer to denote this operator by a tilde \sim (as in our textbooks **I** and **II**). Another useful notational convention is to use a horizontal upper bar. Thus, $\neg x = \sim x = \bar{x} = \text{NOT}(x)$ are all equivalent notations for the negation of x .

That the total number of Boolean functions of a single argument equals four follows directly from the multiplication principle: each Boolean function on \mathbb{B} assigns a range value of 0 or 1 to each of the two points in Fig. 1(a). Thus the number of such functions is $2 \times 2 = 4$. Similarly, we expect that there are $2 \times 2 \times 2 \times 2 = 2^4 = 16$ Boolean function of the form $f : \mathbb{B}^2 \rightarrow \mathbb{B}$, as each such function must assign a range value of 0 or 1 to each of the four points in Fig. 1(b). Likewise, the number of distinguishable Boolean functions $f : \mathbb{B}^n \rightarrow \mathbb{B}$ is 2^{2^n} . How many distinguishable functions of the form $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ are there?

2 Binary Boolean Operators

In the previous section, we introduced a unary operator, $\neg x$ to represent a frequently used Boolean function of a single argument. We shall now describe each of the 16 Boolean functions of two arguments, $f : \mathbb{B}^2 \rightarrow \mathbb{B}$, and represent the functionality of each by a distinct binary operator in analogy with those used in ordinary arithmetic, e.g., $x + y$, $x - y$, $x \times y$, and $x \div y$. Thus, for each function $f_i(x, y)$, for $i = 1, \dots, 16$, we will define a binary operator \circ , such that $x \circ y = f_i(x, y)$, where \circ represents a place holder for the i -th binary operator.

Of these 16 operations, the *conjunction of x and y* (denoted by $x \wedge y$) and *disjunction of x and y* (denoted by $x \vee y$) are the most widely known. Each is defined by a column in the following truth table.

x	y	$x \wedge y$	$x \vee y$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

The conjunction operator is frequently called the “and,” and the disjunction is frequently called the “or;” the rationale for this terminology will become clearer when we study propositional logic. Thus we may write $\text{AND}(x, y) = x \wedge y$ and $\text{OR}(x, y) = x \vee y$.

A second pair of operations can be defined by computing the negation of each. Thus, we define,

$$x \overline{\wedge} y = \text{NAND}(x, y) = \neg \text{AND}(x, y) = \neg(x \wedge y)$$

$$x \overline{\vee} y = \text{NOR}(x, y) = \neg \text{OR}(x, y) = \neg(x \vee y).$$

And consequently, their truth tables are

x	y	$x \overline{\wedge} y$	$x \overline{\vee} y$
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

Another important function is *exclusive disjunction*, or *exclusive or*, denoted by the binary operator $x \oplus y$ as follows.

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

We also define $\text{XOR}(x, y) = x \oplus y$. The above is also called a “parity” function, as it returns 1 if and only if an odd number of its input bits equal 1.)

The *equivalence* function is denoted a by the binary operations $x \equiv y$ or $x \Leftrightarrow y$, and has the truth table

x	y	$x \equiv y$
0	0	1
0	1	0
1	0	0
1	1	1

Note that the equivalence is the negation of the exclusive or; thus we may write

$$\begin{aligned} x \equiv y &= \neg(x \oplus y) \\ &= \overline{x \oplus y}. \end{aligned}$$

There are two noteworthy subtleties in the above notation. Firstly, the parentheses in the expression $\neg(x \oplus y)$ are necessary as by conventions, negation has a higher degree of precedence than any binary Boolean operator. Secondly, although parentheses do not explicitly appear in the alternate expression $\overline{x \oplus y}$, they are implicitly assume to exist around any expression that lies beneath an extended negation bar.

Another important Boolean functions is *implication*, which is represented by the following truth table,

x	y	$x \Rightarrow y$
0	0	1
0	1	1
1	0	0
1	1	1

Note that $x \Rightarrow y = \bar{x} \vee y$. We will also understand why this is called implication when we study propositional logic.

Following Knuth [2008], we represent every Boolean function of two variable arguments as a binary operation in Table 1. In this table, each vertical truth table that appears above is represented as a horizontal row: simply take the vertical pattern of zeros and ones that appear above under each operator, and transpose the pattern horizontally from left to right.

Truth Table	Alternate Forms	Operator symbol \circ	Description
0000	0	\perp	contradiction, constant 0
0001	$x \wedge y$	\wedge	conjunction, AND
0010	$x \wedge \bar{y}$	\nRightarrow	nonimplication
0011	x	L	left projection
0100	$\bar{x} \wedge y$	\Leftarrow	converse nonimplication
0101	y	R	right projection
0110	$x \oplus y, (x \wedge \bar{y}) \vee (\bar{x} \wedge y)$	\oplus	exclusive disjunction, XOR
0111	$x \vee y$	\vee	disjunction, OR
1000	$\bar{x} \vee \bar{y}, \bar{x} \wedge \bar{y}$	$\bar{\vee}$	nondisjunction, NOR
1001	$x \equiv y, x \Leftrightarrow y, (\bar{x} \wedge \bar{y}) \vee (x \wedge y)$	\equiv	equivalence, if and only if, iff
1010	$\bar{y}, \neg y$	\bar{R}	right complementation
1011	$x \vee y, x \Leftarrow y$	\Leftarrow	converse implication
1100	$\bar{x}, \neg x$	\bar{L}	left complementation
1101	$\bar{x} \vee y, x \Rightarrow y$	\Rightarrow	implication
1110	$\bar{x} \wedge \bar{y}, \bar{x} \vee \bar{y}$	$\bar{\wedge}$	nonconjunction, NAND
1111	1	\top	affirmation, constant 1

Table 1: An exhaustive construction of the sixteen possible Boolean functions of two arguments, $f : \mathbb{B}^2 \rightarrow \mathbb{B}$.

3 Boolean Algebra

Two Boolean functions, f and g are said to be *equal*, $f = g$, if and only if their truth tables are identical over the full set of their arguments. The following identities appear as Theorem 2 in Bender and Williamson [2005] (I:BF1.)

IDENTITY	DESCRIPTION
$(x \wedge y) \wedge z = x \wedge (y \wedge z)$	associativity of \wedge
$(x \vee y) \vee z = x \vee (y \vee z)$	associativity of \vee
$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	distributivity of \wedge over \vee
$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$	distributivity of \vee over \wedge
$x \wedge x = x, \quad x \vee x = x,$	idempotency of \wedge and \vee
$\neg(\neg x) = x$	double negation
$\neg(x \wedge y) = \neg x \vee \neg y, \quad \neg(x \vee y) = \neg x \wedge \neg y$	De Morgan's rules
$x \wedge y = y \wedge x$	commutivity of \wedge
$x \vee y = y \vee x$	commutivity of \vee
$x \vee (x \wedge y) = x, \quad x \wedge (x \vee y) = x$	absorption
$x \wedge 0 = 0, \quad x \wedge 1 = x$	bound rules for \wedge
$x \vee 0 = x, \quad x \vee 1 = 1$	bound rules for \vee
$x \wedge \neg x = 0, \quad x \vee \neg x = 1$	negation rules.

Table 2: Frequently used identities of Boolean operators. See Bender and Williamson [2005]

Each of these identities is proved by systematically constructing truth tables for the left and right sides, and validating their equality row by row. For example, to prove that $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ is correct, we construct the following

table.

x	y	z	$x \wedge y$	$y \wedge z$	$(x \wedge y) \wedge z$	$x \wedge (y \wedge z)$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

The important columns are the last two, which clearly agree for every one of the $2^3 = 8$ possible input patterns. The auxiliary truth tables for $x \wedge y$ and $y \wedge z$ were simply introduced to help derive the last two columns.

Many of these identities generalize to more variables. For example, De Morgan's laws can be generalized to n arguments:

$$\neg(x_1 \wedge x_2 \wedge \cdots \wedge x_n) = \bar{x}_1 \vee \bar{x}_2 \vee \cdots \vee \bar{x}_n \quad (2)$$

$$\neg(x_1 \vee x_2 \vee \cdots \vee x_n) = \bar{x}_1 \wedge \bar{x}_2 \wedge \cdots \wedge \bar{x}_n. \quad (3)$$

General formulas such as these that depend on an arbitrary number of arguments can be usually proved by mathematical induction, which we will study later on in the course.

4 Representation Techniques

Note that every Boolean function of two variables can be expressed using a composition of \wedge , \vee , and \neg operators. (See the second column of Table 1 for hints of this fact.) From De Morgan's laws, either of the following sets will suffice: (i) \neg and \wedge , or (ii) \neg and \vee . Each of these sets of operators is called a universal basis of the binary Boolean operators. It turns out that all sixteen functions can be expressed using only *one* kind of operator, either (iii) $\bar{}$ or (iv) $\bar{\vee}$. To show this first represent the \wedge , \vee , and \neg operators using only $\bar{}$ or $\bar{\vee}$ operators. For example, observe that

$$\begin{aligned} x \wedge y &= (x \bar{y}) \bar{(x \bar{y})}, \\ x \vee y &= (x \bar{x}) \bar{(y \bar{y})}, \quad \text{and} \\ \neg x &= x \bar{x}. \end{aligned}$$

The remaining 14 binary operators can be implemented using only the $\bar{}$ operator by applying the "Alternate Forms" that appear in Table 1. Alternate forms for the constant functions can be obtained from the negation rules in the previous section.

Any Boolean expression that consists of a single Boolean variable (e.g., x), or its negation (\bar{x}) is called a *literal*. Using literals one can demonstrate that it is possible to represent any truth table, hence, any Boolean function, as a disjunction of conjuncted literals. We will show this by construction, first using an example, which we will subsequently generalize.

Example 4.1. *Given the truth table*

x	y	z	$f(x, y, z)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

We first identify every row in that table for which $f = 1$. For each such row, we construct a conjunction of literals choosing one from the x variable set $\{x, \bar{x}\}$, one from the y variable set, $\{y, \bar{y}\}$, one from the z variable set, $\{z, \bar{z}\}$, etc., so that the resulting conjunction evaluates to 1 for that row. Thus for the third row above we obtain $\bar{x} \wedge y \wedge \bar{z}$, and for the fifth row, $x \wedge \bar{y} \wedge \bar{z}$. The function is then obtained by constructing the disjunction of all such terms:

$$f(x, y, z) = (\bar{x} \wedge y \wedge \bar{z}) \vee (x \wedge \bar{y} \wedge \bar{z}). \quad (4)$$

The above expression is said to be a *disjunctive normal form* (or DNF) as it consists of a disjunction of conjunctions of literals. Each conjunction in the above, e.g., $\bar{x} \wedge y \wedge \bar{z}$ is called a *minterm* or *implicant*. If every variable is represented in every minterm, as they are above, the the DNF is said to be a *full disjunction normal form*. This method clearly generalizes to any finite truth table.

It is also possible to represent any Boolean function as a conjunction of disjunctions of literals. Such a form is called a *conjunctive normal form* (or CNF). We derive the CNF representation by negating the DNF of the complementary Boolean function. Thus for the truth table in Example 4.1, we first compute $\neg f(x, y, z)$, as follows:

x	y	z	$f(x, y, z)$	$\neg f(x, y, z)$
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Then we represent $\neg f(x, y, z)$ as a DNF:

$$\neg f(x, y, z) = (\bar{x} \wedge \bar{y} \wedge \bar{z}) \vee (\bar{x} \wedge \bar{y} \wedge z) \vee (\bar{x} \wedge y \wedge z) \vee (x \wedge \bar{y} \wedge z) \vee (x \wedge y \wedge \bar{z}) \vee (x \wedge y \wedge z).$$

We then obtain the equivalent CNF by negating both sides of the above, and then by applying successive applications of De Morgan's laws, Eqns. (2) and (3):

$$\begin{aligned} f(x, y, z) &= \neg(\neg f(x, y, z)) \\ &= \neg((\bar{x} \wedge \bar{y} \wedge \bar{z}) \vee (\bar{x} \wedge \bar{y} \wedge z) \vee (\bar{x} \wedge y \wedge z) \vee (x \wedge \bar{y} \wedge z) \vee (x \wedge y \wedge \bar{z}) \vee (x \wedge y \wedge z)) \\ &= \neg(\bar{x} \wedge \bar{y} \wedge \bar{z}) \wedge \neg(\bar{x} \wedge \bar{y} \wedge z) \wedge \neg(\bar{x} \wedge y \wedge z) \wedge \neg(x \wedge \bar{y} \wedge z) \wedge \neg(x \wedge y \wedge \bar{z}) \wedge \neg(x \wedge y \wedge z) \\ &= (x \vee y \vee z) \wedge (x \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z}). \end{aligned} \quad (5)$$

An alternate derivation of the CNF starts with the DNF representation, Eqn. (4), and applies the distributive laws of \wedge and \vee to reorder these operations. Each "factor" in the CNF, such as $(x \vee y \vee z)$ is called a *minclause*.

Note that we have demonstrated quite a profound result:

Theorem 4.1. Every Boolean function of the form $f : \mathbb{B}^n \rightarrow \mathbb{B}$, for integer $n \geq 0$ can be expressed algebraically using only the operators \wedge , \vee , \neg , parentheses, and the function arguments, x_1, \dots, x_n .

In other words, the unary and binary functions that we introduced in Sections 1 and 2, form the basic building blocks that we need to construct any function of the form $f : \mathbb{B}^n \rightarrow \mathbb{B}$ for any finite n . For the electrical engineer, this means that every Boolean circuit can be built using only AND gates, OR gates, and inverters (NOT); or even simpler, using just NAND gates, or just NOR gates. (Explain.) (See also Theorem 1 in Section I:BF1 in Bender and Williamson [2005].)

5 Exercises

1. Use truth tables to show that

$$(a) \quad x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

$$(b) \quad x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$

(c) $\neg(x \wedge y) = \neg x \vee \neg y$

(d) $\neg(x \vee y) = \neg x \wedge \neg y$

(e) $x \vee (x \wedge y) = x \wedge (x \vee y) = x$

2. Simplify the following Boolean expressions

(a) $(a \vee b) \wedge (a \vee \bar{b})$

(b) $(a \wedge b) \vee (a \wedge \bar{b})$

3. Simplify $\neg(\neg(\neg x \vee y) \wedge \neg(x \vee \neg y))$.

4. Prove or refute the following identities:

(a) $x \wedge (y \oplus z) = (x \wedge y) \oplus (x \wedge z)$

(b) $x \vee (y \oplus z) = (x \vee y) \oplus (x \vee z)$

5. Using only the \wedge , \vee and \neg operators, construct a Boolean function of three arguments $f(a, b, c)$ that returns 1 if and only if the majority of its arguments equal 1.

6. Using any operators that you please, construct a Boolean function of three arguments $\Pi(a, b, c)$ that returns 1 if and only if the number of arguments that equal 1 is odd. (This would be called the 3-parity function.) Can you generalize your solution to obtain an expression for the n -parity function?

7. Show that every Boolean function of two arguments, $f : \mathbb{B}^2 \rightarrow \mathbb{B}$, can be expressed entirely in terms of one or more $\bar{\vee}$ operators.

8. Derive an expression for the number of distinct Boolean functions that exist of the form $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$, in terms of the input and output dimensions, n and m .

9. Use truth tables to verify that both the DNF representation of $f(x, y, z)$, Eqn. (4), and the CNF representation, Eqn. (5), correctly reflect the truth table shown in Example 4.1.

10. Consider the following Boolean function of three arguments:

x	y	z	$f(x, y, z)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

(a) Derive an expression for $f(x, y, z)$ as a disjunctive normal form (DNF).

(b) Derive an expression for $f(x, y, z)$ as a conjunctive normal form (CNF).

References

E. A. Bender and S. G. Williamson. *A Short Course in Discrete Mathematics*. Dover, Mineola, NY, 2005.

D. E. Knuth. *The Art of Computer Programming*, volume 4, fascicle 0 (Introduction to Combinatorial Algorithms and Boolean Functions). Addison-Wesley, Upper Saddle River, NJ, 2008.