Boolean Algebra

John Winans

August 27, 2020

1 Basic Operations

We describe Boolean values as either *false* or *true*.[1]

In a system that represents information numerically using only binary digits:

- 0 = *false*
- 1 = *true*

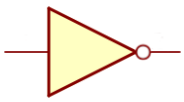
The following three basic Boolean operations represent the only operators we will use when reducing equations into their simplest form.

1.1 NOT

A function whose output is the opposite of its input.

$$Q = \overline{A} \tag{1}$$

A	Q
0	1
1	0



Depending on the context, any of the following forms might be seen used to express the **NOT** function applied to the variable a:

- $\neg a$
- $\overline{a}$
- a'
- `~a` (C bitwise operator)
- `!a` (C logical operator)

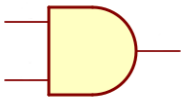
[1] In non-binary systems it might help to consider a more general definition where zero is considered *false* and anything that is not *false* is *true*.

1.2 AND

A function whose output is *true* if, and only if, all of its inputs are *true*.

$$Q = A \wedge B \tag{2}$$

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1



Depending on the context, any of the following forms might be seen used to express the **AND** function applied to the variables a and b

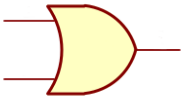
- $a \wedge b$
- $a \cdot b$
- ab
- `a & b` (C bitwise operator)
- `a && b` (C logical operator)

1.3 OR

A function whose output is *true* if any one, or both, of its inputs is/are *true*.

$$Q = A \vee B \tag{3}$$

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1



Depending on the context, any of the following forms might be seen used to express the **OR** function applied to the variables a and b

- $a \vee b$

- $a + b$
- `a | b` (C bitwise operator)
- `a || b` (C logical operator)

2 Composing New Functions

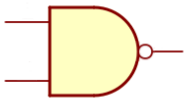
We can combine the above three functions to create new ones.

2.1 NAND

A function whose output is *false* if, and only if, all of its inputs are *true*.

$$Q = \overline{A \wedge B} \tag{4}$$

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Depending on the context, any of the following forms might be seen used to express the **NAND** function applied to the variables a and b

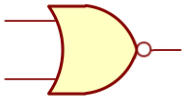
- $\overline{a \wedge b}$
- $\overline{ab}$
- $\overline{a \cdot b}$
- `~(a & b)` (C bitwise operators)
- `!(a && b)` (C logical operators)

2.2 NOR

A function whose output is *true* if, and only if, all of its inputs are *false*.

$$Q = \overline{A \vee B} \tag{5}$$

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0



Depending on the context, any of the following forms might be seen used to express the **NOR** function applied to the variables a and b

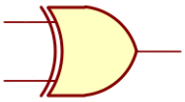
- $\overline{a \vee b}$
- $\overline{a + b}$
- `~(a | b)` (C bitwise operators)
- `!(a || b)` (C logical operators)

2.3 XOR (Exclusive OR)

A function whose output is *true* when an odd number of its inputs are *true*. We call this *odd parity*. In the special case (when there are only two inputs) the **XOR** function output is *true* when the two inputs are different and *false* when they are the same.

$$Q = A \oplus B \tag{6}$$
$$= \overline{A} \wedge B \vee A \wedge \overline{B} \tag{7}$$

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0



Depending on the context, any of the following forms might be seen used to express the **XOR** function applied to the variables a and b

- $a \oplus b$
- `a ^ b` (C bitwise operator)

3 Material Implication

A function with two inputs a and b whose output is *true* if either a is *false* or a is *true* when b is *true*.

$$Q = A \rightarrow B \tag{8}$$
$$= \overline{A} \vee B \tag{9}$$

A	B	Q
0	0	1
0	1	1
1	0	0
1	1	1

Depending on the context, any of the following forms might be seen used to express the **IMP** function applied to the variables a and b

- $a \rightarrow b$
- `a ? b : true` (C conditional/ternary operator)

4 All Possible Functions With Two Inputs

Consider the following:

- How many ways can we arrange 2 bits? (Answer: $2^2 = 4$)
- How many ways can we arrange 4 bits? (Answer: $2^4 = 16$)

Therefore:

1. There are exactly four possible ways to arrange two one-bit values.
2. There are exactly sixteen possible ways to arrange four one-bit values.

Conclusion: There are 16 possible Boolean functions that have two one-bit inputs a and b:

a	b	0	$a \wedge b$	$\overline{a \rightarrow b}$	a	$\overline{b \rightarrow a}$	b	$a \oplus b$	$a \vee b$	$\overline{a \vee b}$	$\overline{a \oplus b}$	$\overline{b}$	$b \rightarrow a$	$\overline{a}$	$a \rightarrow b$	$\overline{a \wedge b}$	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f

5 DeMorgan's Morgan's Laws

`https://en.m.wikipedia.org/wiki/De_Morgan's_laws`

De Morgan observed the following relationships:

$$\overline{a \wedge b} = \overline{a} \vee \overline{b} \tag{10}$$

$$\overline{a \vee b} = \overline{a} \wedge \overline{b} \tag{11}$$

Proof by truth table:

a	b	$\overline{a}$	$\overline{b}$	$a \wedge b$	$\overline{a \wedge b}$	$\overline{a} \vee \overline{b}$	$a \vee b$	$\overline{a \vee b}$	$\overline{a} \wedge \overline{b}$
0	0	1	1	0	1	1	0	1	1
0	1	1	0	0	1	1	1	0	0
1	0	0	1	0	1	1	1	0	0
1	1	0	0	1	0	0	1	0	0

6 Completeness

Completeness refers to the fact that the *basic operations* provide a sufficient basis to describe all other possible Boolean operations.

`https://en.wikipedia.org/wiki/Boolean_algebra#Completeness`

A proof showing that NAND is *complete* by showing a truth table for a NOT, AND and OR function that are constructed only using the NAND function. [2]

$$\overline{a} = \overline{a \wedge a} \tag{12}$$

$$a \wedge b = \overline{\overline{a \wedge b} \wedge \overline{a \wedge b}} \tag{13}$$

$$a \vee b = \overline{\overline{a \wedge a} \wedge \overline{b \wedge b}} \tag{14}$$

a	b	$\overline{a}$	$\overline{a \wedge a}$	$\overline{b \wedge b}$	$\overline{a \wedge a} \wedge \overline{b \wedge b}$	$\overline{\overline{a \wedge a} \wedge \overline{b \wedge b}}$	$a \vee b$	$\overline{a \wedge b}$	$\overline{\overline{a \wedge b} \wedge \overline{a \wedge b}}$	$a \wedge b$
0	0	1	1	1	1	0	0	1	0	0
0	1	1	1	0	0	1	1	1	0	0
1	0	0	0	1	0	1	1	1	0	0
1	1	0	0	0	0	1	1	0	1	1

7 Operator Precedence & Parenthesis

- The NOT operator has higher precedence than AND.
- The AND operator has higher precedence than OR.
- The OR operator has higher precedence than IMPLICATION.

Note that XOR is a function of one or more of these operators. Therefore its precedence is a matter of its implementation. Languages like C that include an explicit XOR operator put its precedence between that of the AND and OR operators.

Parenthesis can be used to manage the order of operations when the precedence rules would otherwise cause an undesirable result:

$$a \vee b \wedge c = a \vee (b \wedge c) \tag{15}$$

$$\neq (a \vee b) \wedge c \tag{16}$$

8 Laws of Boolean Algebra

Summarized from `https://en.wikipedia.org/wiki/Boolean_algebra`.

[2] This along with DeMorgan's laws and the above discussion on the composition of new functions demonstrates that the NAND function alone can be used to perform all 16 possible Boolean functions.

$a \wedge (b \wedge c) = (a \wedge b) \wedge c$	Associativity of $\wedge$	(17)
$a \vee (b \vee c) = (a \vee b) \vee c$	Associativity of $\vee$	(18)
$a \wedge b = b \wedge a$	Commutativity of $\wedge$	(19)
$a \vee b = b \vee a$	Commutativity of $\vee$	(20)
$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	Distributivity of $\wedge$ over $\vee$	(21)
$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$	Distributivity of $\vee$ over $\wedge$	(22)
$a \wedge 1 = a$	Identity for $\wedge$	(23)
$a \vee 0 = a$	Identity for $\vee$	(24)
$a \wedge 0 = 0$	Annihilator for $\wedge$	(25)
$a \vee 1 = 1$	Annihilator for $\vee$	(26)
$a \wedge a = a$	Idempotence $\wedge$	(27)
$a \vee a = a$	Idempotence $\vee$	(28)
$a \wedge (a \vee b) = a$	Absorption 1	(29)
$a \vee (a \wedge b) = a$	Absorption 2	(30)
$a \wedge \overline{a} = 0$	Complementation of $\wedge$	(31)
$a \vee \overline{a} = 1$	Complementation of $\vee$	(32)
$\overline{\overline{a}} = a$	Double negation	(33)
$\overline{a} \wedge \overline{b} = \overline{a \vee b}$	DeMorgan's 1	(34)
$\overline{a} \vee \overline{b} = \overline{a \wedge b}$	DeMorgan's 2	(35)

8.1 An Exercise

You should be able to show that each of the following equations are true by applying the above Boolean laws and by showing the truth tables for each case.

$$a \wedge b = \overline{\overline{a} \vee \overline{b}} \tag{36}$$

$$a \vee b = \overline{\overline{a} \wedge \overline{b}} \tag{37}$$

$$a \vee \overline{a} \wedge b = a \vee b \tag{38}$$

$$\overline{a} \vee a \wedge b = \overline{a} \vee b \tag{39}$$

$$(a \vee b) \wedge (c \vee d) = a \wedge c \vee a \wedge d \vee b \wedge c \vee b \wedge d \tag{40}$$

For example, a proof of Equation 39:

$f = \overline{a} \vee a \wedge b$	Given	(41)
$= \overline{a} \vee (a \wedge b)$	Show operator precedence	(42)
$= (\overline{a} \vee a) \wedge (\overline{a} \vee b)$	Distributivity of $\vee$ over $\wedge$	(43)
$= (a \vee \overline{a}) \wedge (\overline{a} \vee b)$	Commutativity of $\vee$	(44)
$= 1 \wedge (\overline{a} \vee b)$	Complementation of $\vee$	(45)
$= (\overline{a} \vee b) \wedge 1$	Commutativity of $\wedge$	(46)
$= \overline{a} \vee b$	Identity for $\wedge$	(47)

9 Minterm (lower-case m) (SOP form)

See also: `https://en.wikipedia.org/wiki/Canonical_normal_form`

Any Boolean function can be expressed in SOP form. [3]

In a minterm, each variable may appear only once as a or $\overline{a}$. For a three-input function, there are $2^3 = 8$ minterms:

$$m0 = \overline{a} \wedge \overline{b} \wedge \overline{c} \tag{48}$$
$$m1 = \overline{a} \wedge \overline{b} \wedge c \tag{49}$$
$$m2 = \overline{a} \wedge b \wedge \overline{c} \tag{50}$$
$$m3 = \overline{a} \wedge b \wedge c \tag{51}$$
$$m4 = a \wedge \overline{b} \wedge \overline{c} \tag{52}$$
$$m5 = a \wedge \overline{b} \wedge c \tag{53}$$
$$m6 = a \wedge b \wedge \overline{c} \tag{54}$$
$$m7 = a \wedge b \wedge c \tag{55}$$

Note that the appearance of the NOT-bars match the pattern of zeros when counting from zero to seven in binary.

Use of minterms in an equation allows a shorter notation as shown below:

$$f = m1 \vee m2 \vee m4 \vee m7 \tag{56}$$
$$= (\overline{a} \wedge \overline{b} \wedge c) \vee (\overline{a} \wedge b \wedge \overline{c}) \vee (a \wedge \overline{b} \wedge \overline{c}) \vee (a \wedge b \wedge c) \tag{57}$$

Parenthesis added for readability.

The long form shown in equation 57 is often called a SOP (Sum Of Products.)

	a	b	c	$(\overline{a} \wedge \overline{b} \wedge c)$	$(\overline{a} \wedge b \wedge \overline{c})$	$(a \wedge \overline{b} \wedge \overline{c})$	$(a \wedge b \wedge c)$	f
$m0$	0	0	0	0	0	0	0	0
$m1$	0	0	1	1	0	0	0	1
$m2$	0	1	0	0	1	0	0	1
$m3$	0	1	1	0	0	0	0	0
$m4$	1	0	0	0	0	1	0	1
$m5$	1	0	1	0	0	0	0	0
$m6$	1	1	0	0	0	0	0	0
$m7$	1	1	1	0	0	0	1	1

As can be seen in the truth table, the "mechanics" driving the SOP equation is based on identifying those specific function input patterns where the output is *true*. By using an OR gate to drive the output, it will set its output *true* any time one of the recognized input terms is *true*.

10 Maxterm (upper-case M) (POS form)

Same idea as the minterm, but inverted and use the $\vee$ instead of the $\wedge$ function:

[3] See: Peter J. Pahl, Rudolf Damrath, "Mathematical Foundations of Computational Engineering: A Handbook," 2001, Springer, page 15. `https://tinyurl.com/ujv8q7l`

Any Boolean function can be expressed in POS form.

$$M0 = a \vee b \vee c \tag{58}$$
$$M1 = a \vee b \vee \overline{c} \tag{59}$$
$$M2 = a \vee \overline{b} \vee c \tag{60}$$
$$M3 = a \vee \overline{b} \vee \overline{c} \tag{61}$$
$$M4 = \overline{a} \vee b \vee c \tag{62}$$
$$M5 = \overline{a} \vee b \vee \overline{c} \tag{63}$$
$$M6 = \overline{a} \vee \overline{b} \vee c \tag{64}$$
$$M7 = \overline{a} \vee \overline{b} \vee \overline{c} \tag{65}$$

Note that the appearance of the NOT-bars match the pattern of ones when counting from zero to seven in binary.

$$f = M0 \wedge M1 \wedge M2 \wedge M4 \tag{66}$$
$$= (a \vee b \vee c) \wedge (a \vee b \vee \overline{c}) \wedge (a \vee \overline{b} \vee c) \wedge (\overline{a} \vee b \vee c) \tag{67}$$

The long form shown in equation 67 is often called a POS (Product Of Sums.)

	a	b	c	$(a \vee b \vee c)$	$(a \vee b \vee \overline{c})$	$(a \vee \overline{b} \vee c)$	$(\overline{a} \vee b \vee c)$	f
$M0$	0	0	0	0	1	1	1	0
$M1$	0	0	1	1	0	1	1	0
$M2$	0	1	0	1	1	0	1	0
$M3$	0	1	1	1	1	1	1	1
$M4$	1	0	0	1	1	1	0	0
$M5$	1	0	1	1	1	1	1	1
$M6$	1	1	0	1	1	1	1	1
$M7$	1	1	1	1	1	1	1	1

As can be seen in the truth table, the "mechanics" driving the POS equation is based on identifying those specific function input patterns where the output is *false*. By using an AND to drive the output, it will set its output *false* any time one of the recognized input terms is *false*.

10.1 Relationship Between Minterms and Maxterms

The complement of a maxterm, such as $\overline{M5}$, is the respective minterm $m5$.

This can be verified with DeMorgan's as in:

$M5 = \overline{a} \vee b \vee \overline{c}$	Given	(68)
$m5 = a \wedge \overline{b} \wedge c$	Given	(69)
$\overline{M5} = \overline{\overline{a} \vee b \vee \overline{c}}$	Complement both sides	(70)
$\overline{M5} = \overline{\overline{a}} \wedge \overline{b} \wedge \overline{\overline{c}}$	DeMorgan 1	(71)
$\overline{M5} = a \wedge \overline{b} \wedge c$	Double negation	(72)
$\overline{M5} = m5$		(73)

Note that DeMorgan's also works with more than two inputs.

It turns out that converting a minterm to a maxterm works the same way. For example:

$$M5 = \overline{a} \vee b \vee \overline{c} \quad \text{Given} \tag{74}$$
$$m5 = a \wedge \overline{b} \wedge c \quad \text{Given} \tag{75}$$
$$\overline{m5} = \overline{a \wedge \overline{b} \wedge c} \quad \text{Complement both sides} \tag{76}$$
$$\overline{m5} = \overline{a} \vee \overline{\overline{b}} \vee \overline{c} \quad \text{DeMorgan 2} \tag{77}$$
$$\overline{m5} = \overline{a} \vee b \vee \overline{c} \quad \text{Double negation} \tag{78}$$
$$\overline{m5} = M5 \tag{79}$$

10.2 When to Apply the POS or SOP Form

If SOP and POS can be easily converted back and forth, it begs the question "Why concern ourselves with both forms?"

The answer lies in which of the two forms is easier to apply to a given situation. For example, the following truth table is best expressed in SOP form simply because there are fewer cases when the output is *true*.

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Therefore there are fewer terms expressed in the SOP form describing it:

$$f = m3 \vee m5 \vee m7 \tag{80}$$
$$= (\overline{a} \wedge b \wedge c) \vee (a \wedge \overline{b} \wedge c) \vee (a \wedge b \wedge c) \tag{81}$$

. . . than there are in the POS form:

$$f = M0 \wedge M1 \wedge M2 \wedge M4 \wedge M6 \tag{82}$$
$$= (a \vee b \vee c) \wedge (a \vee b \vee \overline{c}) \wedge (a \vee \overline{b} \vee c) \wedge (\overline{a} \vee b \vee c) \wedge (\overline{a} \vee \overline{b} \vee c) \tag{83}$$

The shorter function may be more desirable to work with.

11 Minimum SOP and POS Forms

Because it is easy to express any Boolean function in SOP or POS form, it is desirable to be able to then reduce them to their minimal set of operations.

~/NIU/courses/463/2020-fa/notes/BooleanAlgebra/BooleanAlgebra.tex
jwinans@niu.edu 2020-08-27 18:24:12 -0500 v2.0-614-gc072f26

To accomplish this, we apply the Boolean laws until the function becomes irreducible and has the fewest number of operators.

As an example we demonstrate a reduction of the following truth table using both the SOP and POS functions:

	a	b	c	f
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

SOP: (84)

$$f = m2 \vee m4 \vee m5 \vee m6 \vee m7 \quad \text{Given} \quad (85)$$

$$= (\overline{a} \wedge b \wedge \overline{c}) \vee (a \wedge \overline{b} \wedge \overline{c}) \vee (a \wedge \overline{b} \wedge c) \vee (a \wedge b \wedge \overline{c}) \vee (a \wedge b \wedge c) \quad \text{Expand the terms} \quad (86)$$

$$= (\overline{a} \wedge b \wedge \overline{c}) \vee (a \wedge \overline{b} \wedge (\overline{c} \vee c)) \vee (a \wedge b \wedge (\overline{c} \vee c)) \quad \text{Distributive of } \wedge \text{ over } \vee \quad (87)$$

$$= (\overline{a} \wedge b \wedge \overline{c}) \vee (a \wedge \overline{b} \wedge 1) \vee (a \wedge b \wedge 1) \quad \text{Complementation of } \vee \quad (88)$$

$$= (\overline{a} \wedge b \wedge \overline{c}) \vee (a \wedge \overline{b}) \vee (a \wedge b) \quad \text{Identity for } \wedge \quad (89)$$

$$= (\overline{a} \wedge b \wedge \overline{c}) \vee (a \wedge (\overline{b} \vee b)) \quad \text{Distributive of } \wedge \text{ over } \vee \quad (90)$$

$$= (\overline{a} \wedge b \wedge \overline{c}) \vee (a \wedge 1) \quad \text{Complementation of } \vee \quad (91)$$

$$= (\overline{a} \wedge b \wedge \overline{c}) \vee a \quad \text{Identity for } \wedge \quad (92)$$

$$= a \vee (\overline{a} \wedge b \wedge \overline{c}) \quad \text{Commutativity of } \vee \quad (93)$$

$$= a \vee (\overline{a} \wedge (b \wedge \overline{c})) \quad \text{Associativity of } \wedge \quad (94)$$

$$= (a \vee \overline{a}) \wedge (a \vee (b \wedge \overline{c})) \quad \text{Distributivity of } \vee \text{ over } \wedge \quad (95)$$

$$= 1 \wedge (a \vee (b \wedge \overline{c})) \quad \text{Complementation of } \vee \quad (96)$$

$$= a \vee (b \wedge \overline{c}) \quad \text{Identity for } \wedge \quad (97)$$

POS: (98)

$$f = M0 \wedge M1 \wedge M3 \quad \text{Given} \quad (99)$$

$$= (a \vee b \vee c) \wedge (a \vee b \vee \overline{c}) \wedge (a \vee \overline{b} \vee \overline{c}) \quad \text{Expand the terms} \quad (100)$$

$$= ((a \vee b) \vee (c \wedge \overline{c})) \wedge (a \vee \overline{b} \vee \overline{c}) \quad \text{Distributivity of } \vee \text{ over } \wedge \quad (101)$$

$$= ((a \vee b) \vee 0) \wedge (a \vee \overline{b} \vee \overline{c}) \quad \text{Complementation of } \wedge \quad (102)$$

$$= (a \vee b) \wedge (a \vee \overline{b} \vee \overline{c}) \quad \text{Identity for } \vee \quad (103)$$

$$= a \vee (b \wedge (\overline{b} \vee \overline{c})) \quad \text{Distributivity of } \vee \text{ over } \wedge \quad (104)$$

$$= a \vee ((b \wedge \overline{b}) \vee (b \wedge \overline{c})) \quad \text{Distributivity of } \wedge \text{ over } \vee \quad (105)$$

$$= a \vee (0 \vee (b \wedge \overline{c})) \quad \text{Complementation of } \wedge \quad (106)$$

$$= a \vee (b \wedge \overline{c}) \quad \text{Identity for } \vee \quad (107)$$

(108)

Ultimately, we see that both methods arrived at the same conclusion!