Boolean algebra

The logical symbol 0 and 1 are used for representing the digital input or output. The symbols "1" and "0" can also be used for a permanently open and closed digital circuit. The digital circuit can be made up of several logic gates. To perform the logical operation with minimum logic gates, a set of rules were invented, known as the **Laws of Boolean Algebra**. These rules are used to reduce the number of logic gates for performing logic operations.

The Boolean algebra is mainly used for simplifying and analyzing the complex Boolean expression. It is also known as **Binary algebra** because we only use binary numbers in this. **George Boole** developed the binary algebra in **1854**.

Rules in Boolean algebra

1. Only two values(1 for high and 0 for low) are possible for the variable used in Boolean algebra.
2. The overbar(-) is used for representing the complement variable. So, the complement of variable C is represented as .
3. The plus(+) operator is used to represent the ORing of the variables.
4. The dot(.) operator is used to represent the ANDing of the variables.

Properties of Boolean algebra

These are the following properties of Boolean algebra:

Annulment Law

When the variable is AND with 0, it will give the result 0, and when the variable is OR with 1, it will give the result 1, i.e.,

B.0 = 0

B+1 = 1

Identity Law

When the variable is AND with 1 and OR with 0, the variable remains the same, i.e.,

B.1 = B

B+0 = B

Idempotent Law

When the variable is AND and OR with itself, the variable remains same or unchanged, i.e.,

B.B = B

B+B = B

Complement Law

When the variable is AND and OR with its complement, it will give the result 0 and 1 respectively.

B.B' = 0

B+B' = 1

Double Negation Law

This law states that, when the variable comes with two negations, the symbol gets removed and the original variable is obtained.

((A)')' = A

Commutative Law

This law states that no matter in which order we use the variables. It means that the order of variables doesn't matter in this law.

A.B = B.A

A+B = B+A

Associative Law

This law states that the operation can be performed in any order when the variables priority is of same as '*' and '/'.

(A.B).C = A.(B.C)

(A+B)+C = A+(B+C)

Distributive Law

This law allows us to open up of brackets. Simply, we can open the brackets in the Boolean expressions.

A+(B.C) = (A+B).(A+C

A.(B+C) = (A.B)+(A.C)

Absorption Law

This law allows us for absorbing the similar variables.

B+(B.A) = B

B.(B+A) = B

De Morgan Law

The operation of an OR and AND logic circuit will remain same if we invert all the inputs, change operators from AND to OR and OR to AND, and invert the output.

(A.B)' = A'+B'

(A+B)' = A'.B'

Boolean Functions

The binary variables and logic operations are used in Boolean algebra. The algebraic expression is known as **Boolean Expression**, is used to describe the **Boolean Function**. The Boolean expression consists of the constant value 1 and 0, logical operation symbols, and binary variables.

Example 1: F=xy' z+p

We defined the Boolean function **F=xy' z+p** in terms of four binary variables x, y, z, and p. This function will be equal to 1 when x=1, y=0, z=1 or z=1.

Example 2:

$$\underbrace{F(A,B,C,D)}_{Boolean\ Function} = \underbrace{A + \overline{BC} + ACD}_{Boolean\ Expression} \qquad Equation\ No.1$$

The output Y is represented on the left side of the equation. So,

$$Y = A + BC + ACD$$

Apart from the algebraic expression, the Boolean function can also be described in terms of the truth table. We can represent a function using multiple algebraic expressions. They are their logically equivalents. But for every function, we have only one unique truth table.

In truth table representation, we represent all the possible combinations of inputs and their result. We can convert the switching equations into truth tables.

Example: F(A,B,C,D)=A+BC'+D

The output will be high when A=1 or BC'=1 or D=1 or all are set to 1. The truth table of the above example is given below. The 2^n is the number of rows in the truth table. The n defines the number of input variables. So the possible input combinations are $2^3=8$.

Inputs				Output
A	**B**	**C**	**D**	**F**
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Methods of simplifying the Boolean function

There are two methods which are used for simplifying Boolean function. These functions are as follows:

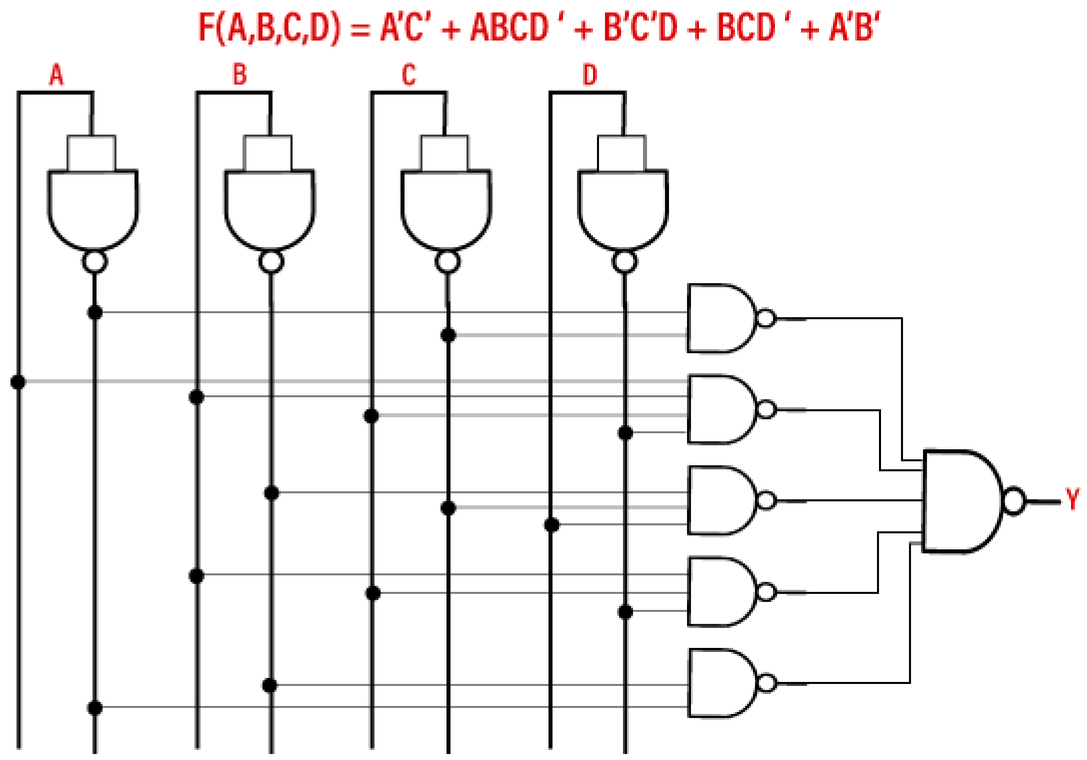
Karnaugh-map or K-map

De-Morgan's law is very helpful for manipulating logical expressions. The logic gates can also realize the logical expression. The k-map method is used to reduce the logic gates for a minimum possible value required for the realization of a logical expression. The K-map method will be done in two different ways, which we will discuss later in the **Simplification of Boolean expression** section.

NAND gates realization

Apart from the K-map, we can also use the NAND gate for simplifying the Boolean functions. Let's see an example:

Example 1: F(A,B,C,D)=A' C'+ABCD'+B' C' D+BCD'+A'B'



Laws and Rules of Boolean algebra

In simplification of the Boolean expression, the laws and rules of the Boolean algebra play an important role. Before understanding these laws and rules of Boolean algebra, understand the Boolean operations addition and multiplication concept.

Boolean Addition

The addition operation of Boolean algebra is similar to the OR operation. In digital circuits, the OR operation is used to calculate the sum term, without using AND operation. A + B, A + B', A + B + C', and A' + B + + D' are some of the examples of 'sum term'. The value of the sum term is true when one or more than one literals are true and false when all the literals are false.

Boolean Multiplication

The multiplication operation of Boolean algebra is similar to the AND operation. In digital circuits, the AND operation calculates the product, without using OR operation. AB, AB, ABC, and ABCD are some of the examples of the product term. The value of the product term is true when all the literals are true and false when any one of the literal is false.

Laws of Boolean algebra

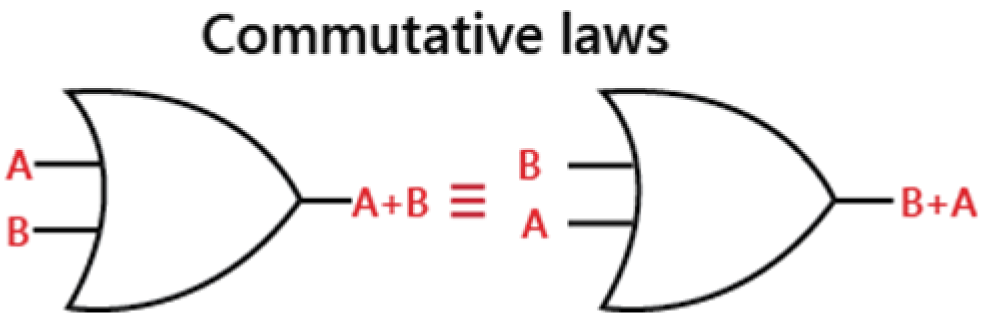
There are the following laws of Boolean algebra:

Commutative Law

This law states that no matter in which order we use the variables. It means that the order of variables doesn't matter. In Boolean algebra, the OR and the addition operations are similar. In the below diagram, the OR gate display that the order of the input variables does not matter at all.

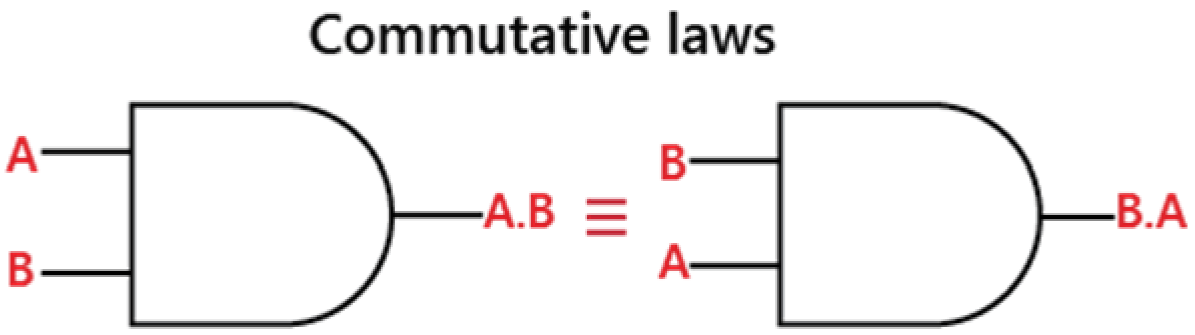
For two variables, the commutative law of addition is written as:

```
A+B = B+A
```



For two variables, the commutative law of multiplication is written as:

A.B = B.A

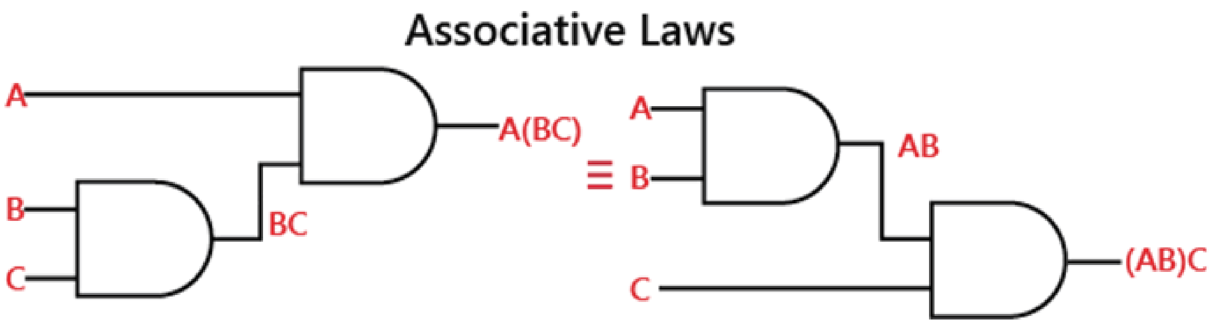


Associative Law

This law states that the operation can be performed in any order when the variables priority is same. As '*' and '/' have same priority. In the below diagram, the associative law is applied to the 2-input OR gate.

For three variables, the associative law of addition is written as:

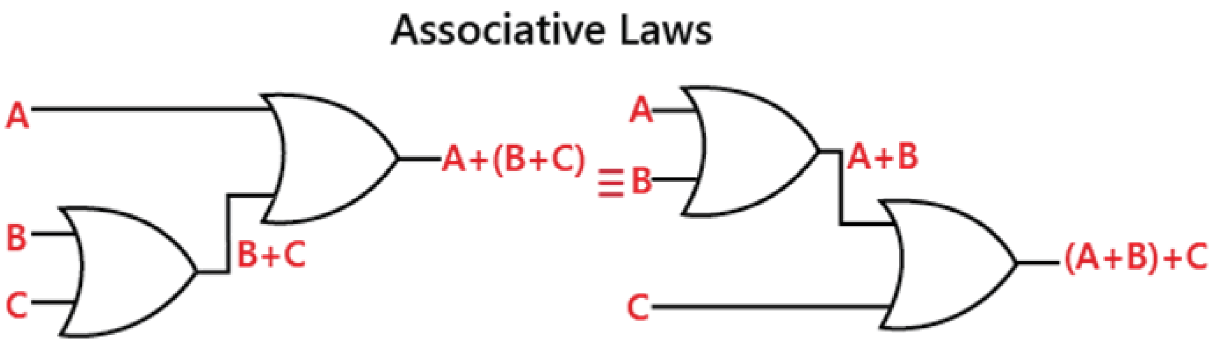
A + (B + C) = (A + B) + C



For three variables, the associative law of multiplication is written as:

A(BC) = (AB)C

According to this law, no matter in what order the variables are grouped when ANDing more than two variables. In the below diagram, the associative law is applied to 2-input AND gate.

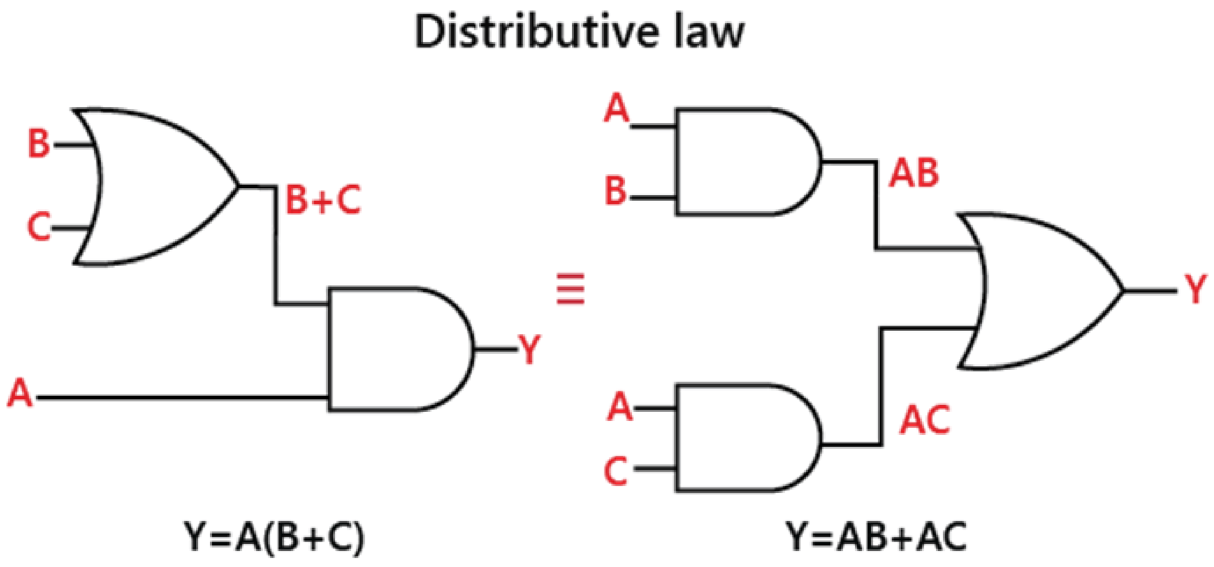


Distributive Law:

According to this law, if we perform the OR operation of two or more variables and then perform the AND operation of the result with a single variable, then the result will be similar to performing the AND operation of that single variable with each two or more variable and then perform the OR operation of that product. This law explains the process of factoring.

For three variables, the distributive law is written as:

$A(B + C) = AB + AC$



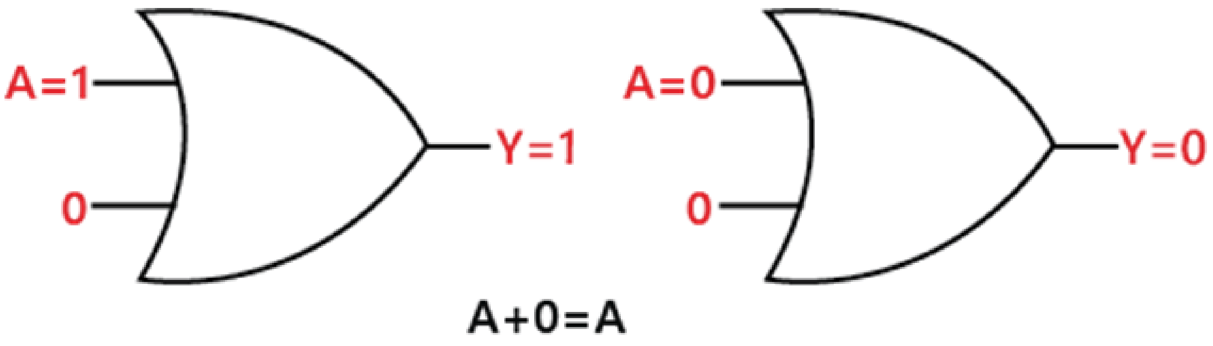
Rules of Boolean algebra

There are the following rules of Boolean algebra, which are mostly used in manipulating and simplifying Boolean expressions. These rules plays an important role in simplifying boolean expressions.

1.	A+0=A	**7.**	A.A=A
2.	A+1=1	**8.**	A.A'=0
3.	A.0=0	**9.**	A''=A
4.	A.1=A	**10.**	A+AB=A
5.	A+A=A	**11.**	A+A'B=A+B
6.	A+A'=1	**12.**	(A+B)(A+C)=A+BC

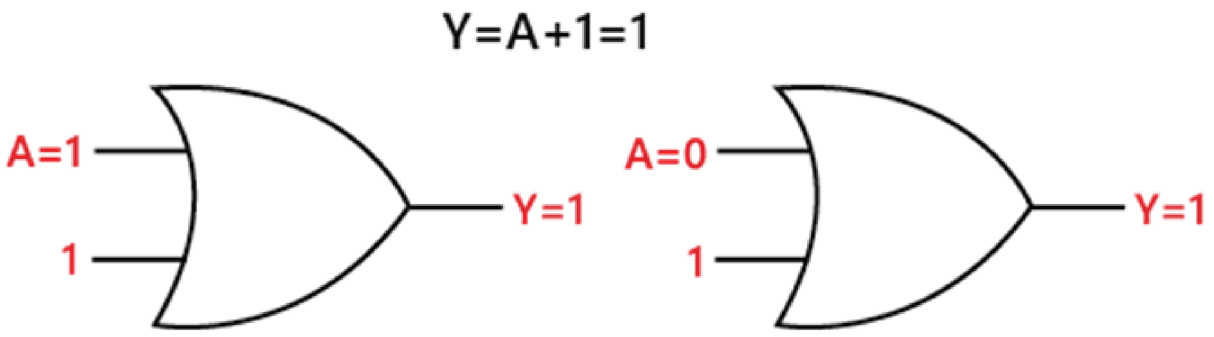
Rule 1: A + 0 = A

Let's suppose; we have an input variable A whose value is either 0 or 1. When we perform OR operation with 0, the result will be the same as the input variable. So, if the variable value is 1, then the result will be 1, and if the variable value is 0, then the result will be 0. Diagrammatically, this rule can be defined as:



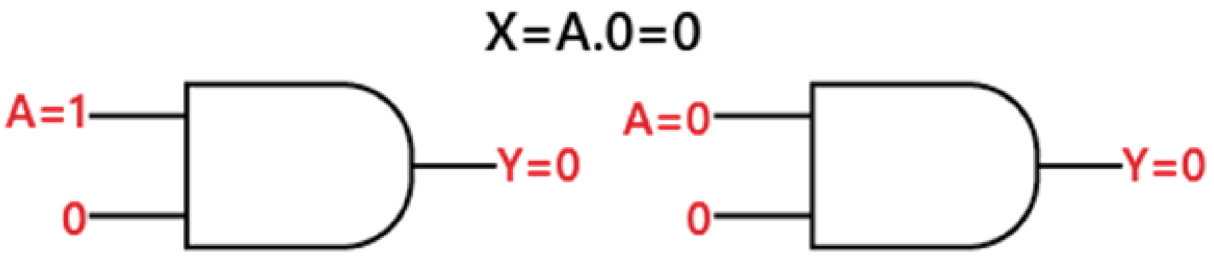
Rule 2: (A + 1) = 1

Let's suppose; we have an input variable A whose value is either 0 or 1. When we perform OR operation with 1, the result will always be 1. So, if the variable value is either 1 or 0, then the result will always be 1. Diagrammatically, this rule can be defined as:



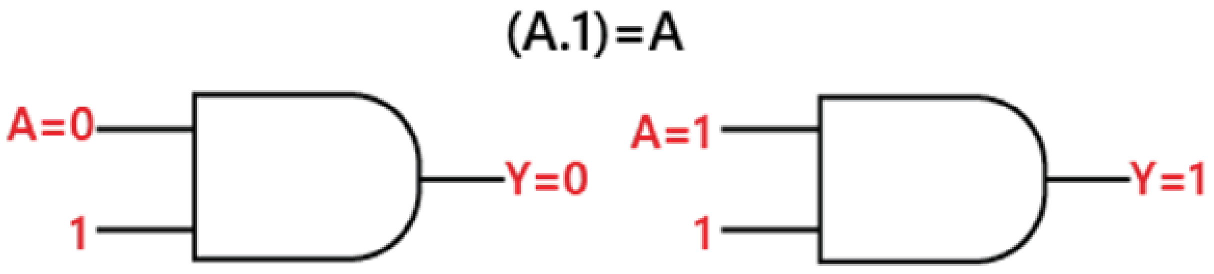
Rule 3: (A.0) = 0

Let's suppose; we have an input variable A whose value is either 0 or 1. When we perform the AND operation with 0, the result will always be 0. This rule states that an input variable ANDed with 0 is equal to 0 always. Diagrammatically, this rule can be defined as:



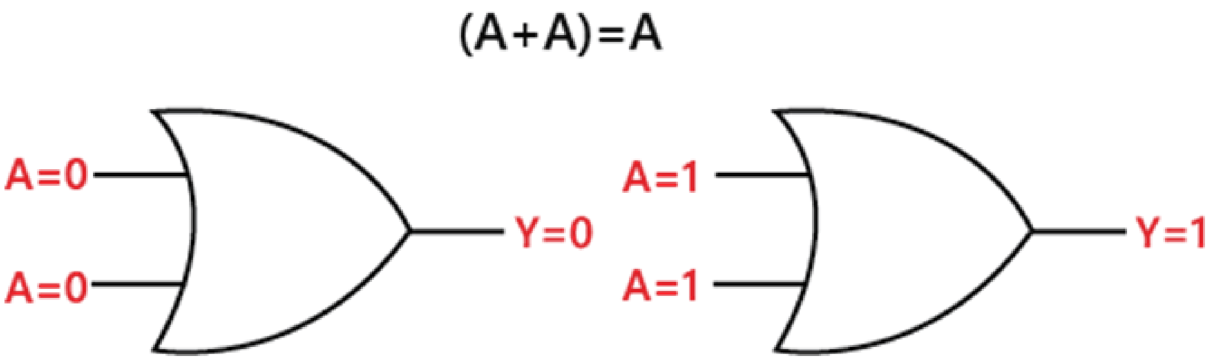
Rule 4: (A.1) = A

Let's suppose; we have an input variable A whose value is either 0 or 1. When we perform the AND operation with 1, the result will always be equal to the input variable. This rule states that an input variable ANDed with 1 is equal to the input variable always. Diagrammatically, this rule can be defined as:



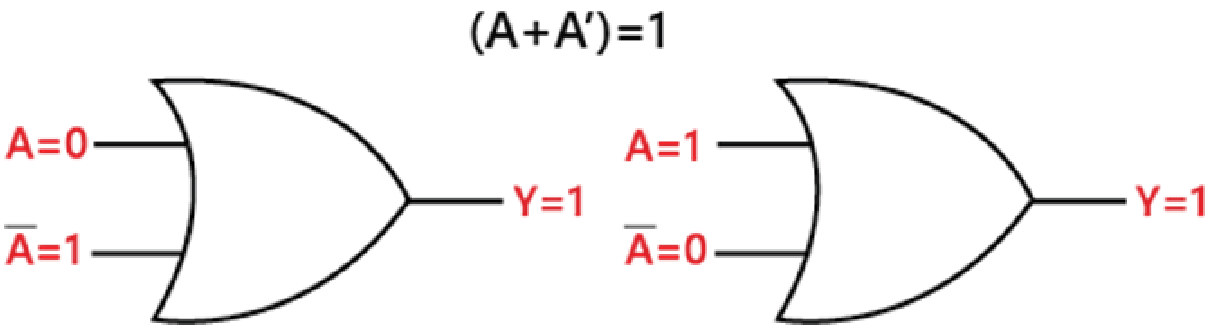
Rule 5: (A + A) = A

Let's suppose; we have an input variable A whose value is either 0 or 1. When we perform the OR operation with the same variable, the result will always be equal to the input variable. This rule states an input variable ORed with itself is equal to the input variable always. Diagrammatically, this rule can be defined as:



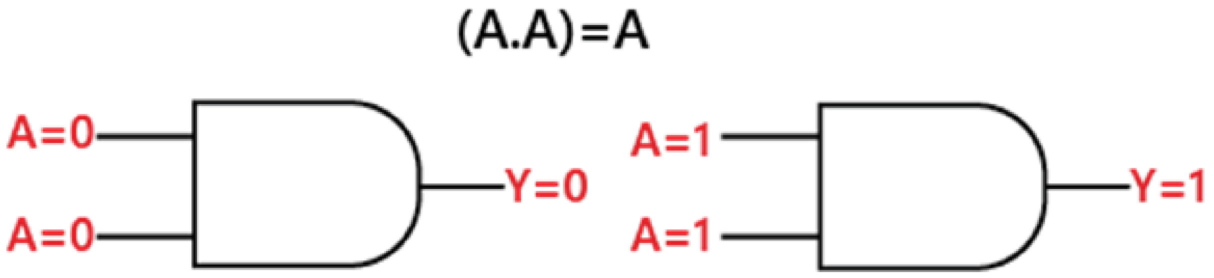
Rule 6: (A + A') = 1

Let's suppose; we have an input variable A whose value is either 0 or 1. When we perform the OR operation with the complement of that variable, the result will always be equal to 1. This rule states that a variable ORed with its complement is equal to 1 always. Diagrammatically, this rule can be defined as:



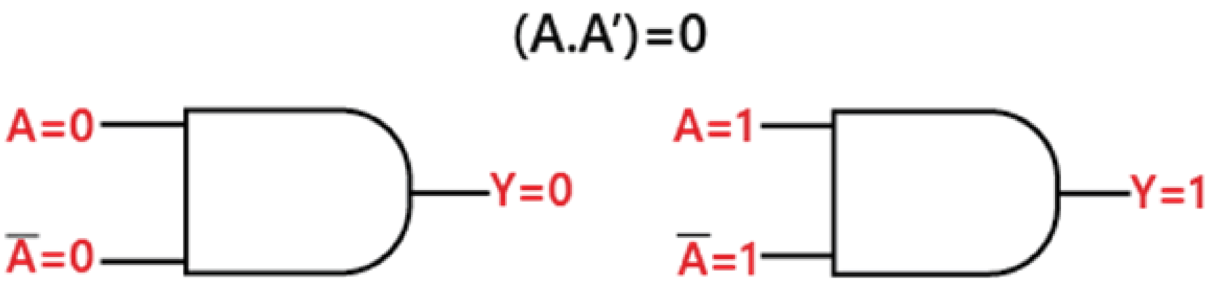
Rule 7: (A.A) = A

Let's suppose; we have an input variable A whose value is either 0 or 1. When we perform the AND operation with the same variable, the result will always be equal to that variable only. This rule states that a variable ANDed with itself is equal to the input variable always. Diagrammatically, this rule can be defined as:



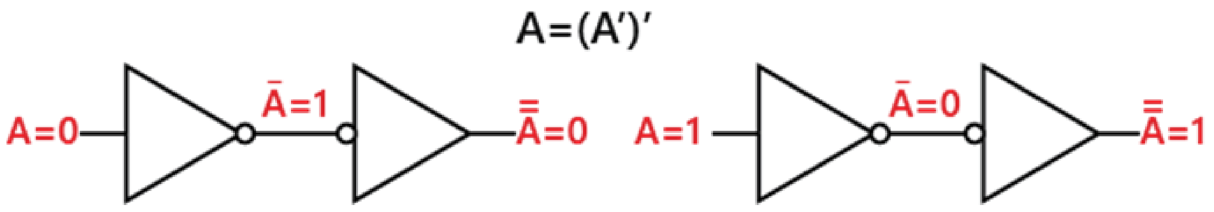
Rule 8: (A.A') = 0

Let's suppose; we have an input variable A whose value is either 0 or 1. When we perform the AND operation with the complement of that variable, the result will always be equal to 0. This rule states that a variable ANDed with its complement is equal to 0 always. Diagrammatically, this rule can be defined as:



Rule 9: A = (A')'

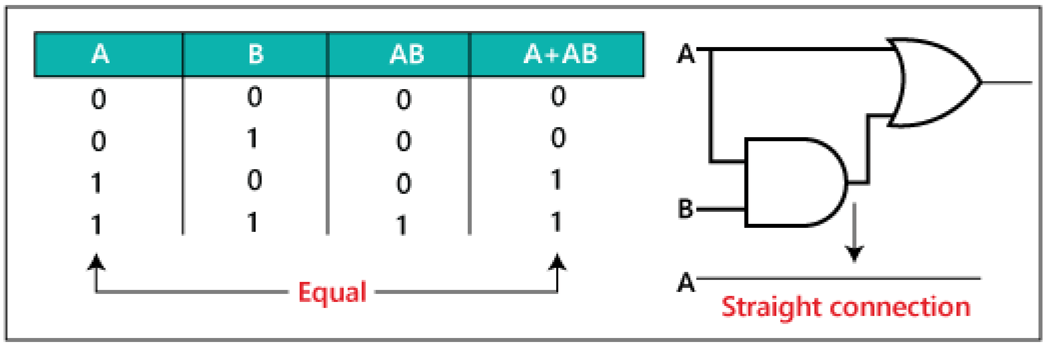
This rule states that if we perform the double complement of the variable, the result will be the same as the original variable. So, when we perform the complement of variable A, then the result will be A'. Further if we again perform the complement of A', we will get A, that is the original variable.



Rule 10: (A + AB) = A

We can prove this rule by using the rule 2, rule 4, and the distributive law as:

A + AB = A(1 + B) Factoring (distributive law)
A + AB = A.1 Rule 2: (1 + B)= 1
A + AB = A Rule 4: A .1 = A

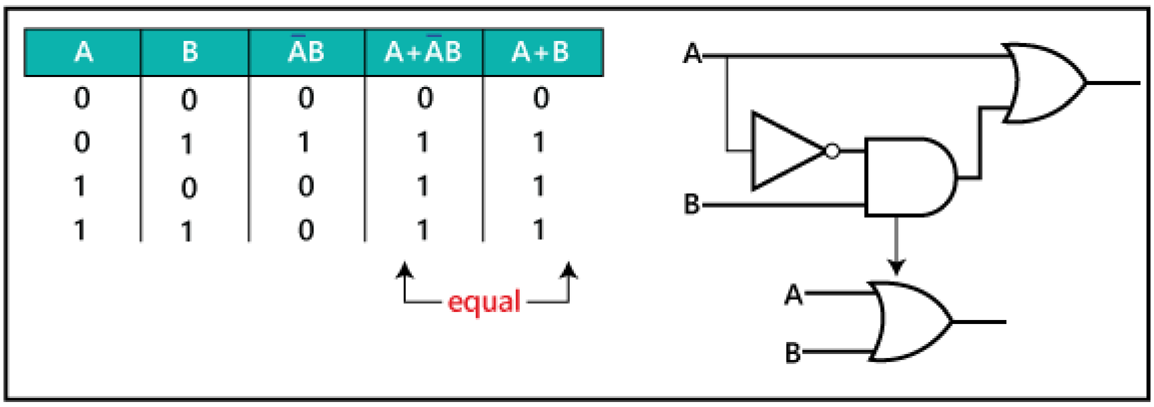


A	B	AB	A+AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Rule 11: A + AB = A + B

We can prove this rule by using the above rules as:

A + AB = (A + AB)+ AB	Rule 10: A = A + AB
A+AB= (AA + AB)+ AB	Rule 7: A = AA
A+AB=AA +AB +AA +AB	Rule 8: adding AA = 0
A+AB= (A + A)(A + B)	Factoring
A+AB= 1.(A + B)	Rule 6: A + A = 1
A+AB=A + B	Rule 4: drop the 1

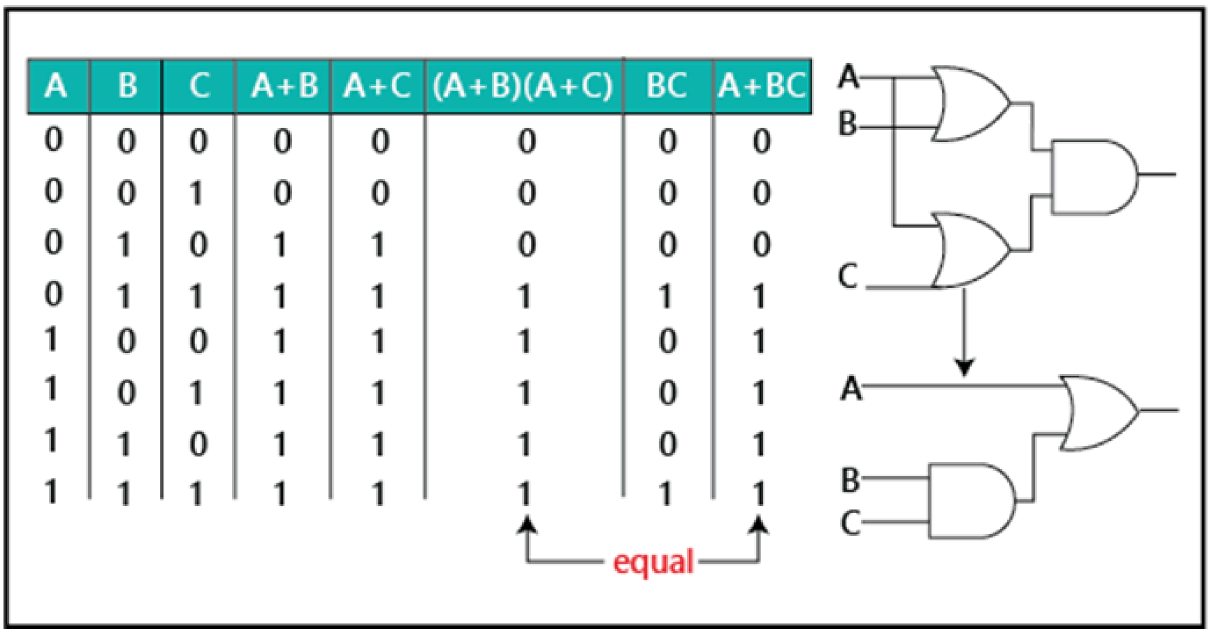


A	B	$\bar{A}B$	$A+\bar{A}B$	A+B
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

Rule 12: (A + B)(A + C) = A + BC

We can prove this rule by using the above rules as:

(A + B)(A + C)= AA + AC + AB + BC	Distributive law
(A + B)(A + C)= A + AC + AB + BC	Rule 7: AA = A
(A + B)(A + C)= A(1 + C)+ AB + BC	Rule 2: 1 + C = 1
(A + B)(A + C)= A.1 + AB + BC	Factoring (distributive law)
(A + B)(A + C)= A(1 + B)+ BC	Rule 2: 1 + B = 1
(A + B)(A + C)= A.1 + BC	Rule 4: A .1 = A
(A + B)(A + C)= A + BC	



A	B	C	A+B	A+C	(A+B)(A+C)	BC	A+BC
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	1	1	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Next Topic Logic Gates