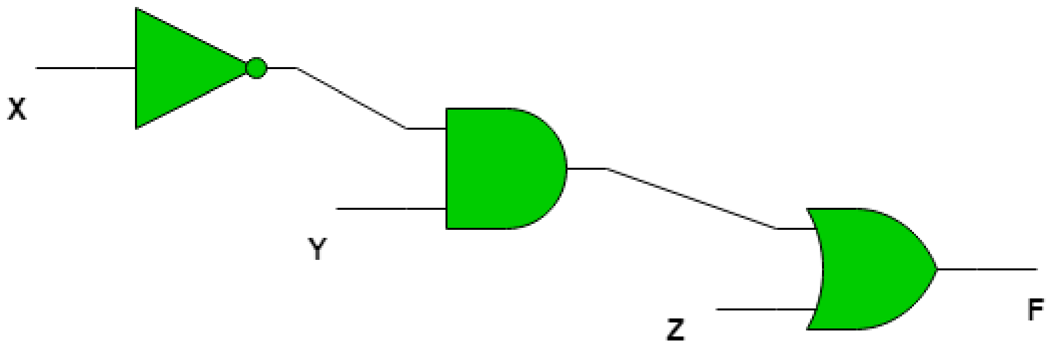
Boolean algebra

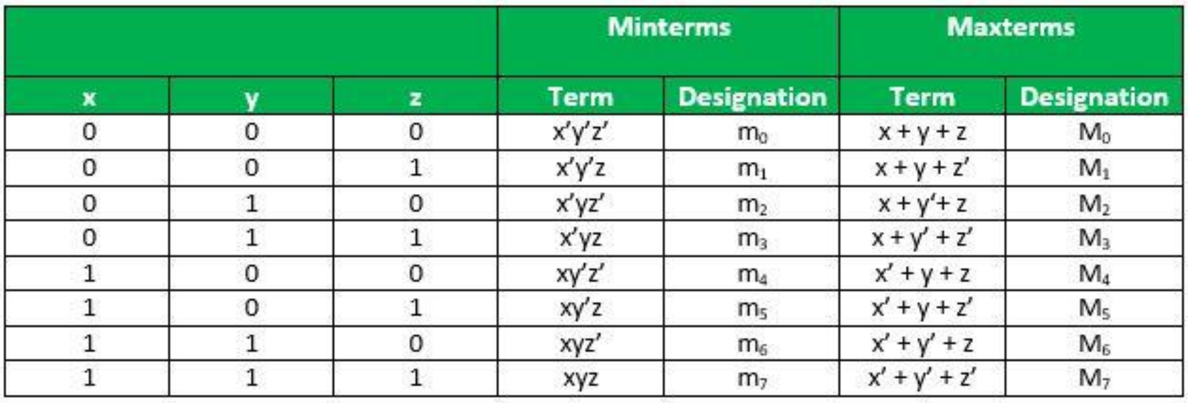
In mathematics and mathematical logic, **Boolean algebra** is a branch of algebra. It differs from elementary algebra in two ways. First, the values of the variables are the truth values *true* and *false*, usually denoted 1 and 0, whereas in elementary algebra the values of the variables are numbers. Second, Boolean algebra uses logical operators such as conjunction (*and*) denoted as $\wedge$, disjunction (*or*) denoted as $\vee$, and the negation (*not*) denoted as $\neg$. Elementary algebra, on the other hand, uses arithmetic operators such as addition, multiplication, subtraction, and division. Boolean algebra is therefore a formal way of describing logical operations, in the same way that elementary algebra describes numerical operations.

A **Boolean function** is described by an algebraic expression consisting of binary variables, the constants 0 and 1, and the logic operation symbols +,., For a given set of values of the binary variables involved, the boolean function can have a value of 0 or 1. For example, the boolean function $F=x^{/}y+z$ is defined in terms of three binary variables x,y,z. The function is equal to 1 if x=0 and y=1 simultaneously or z=1. Every boolean function can be expressed by an algebraic expression, such as one mentioned above, or in terms of a Truth Table. A function may be expressed through several algebraic expressions, on account of them being logically equivalent, but there is only one unique truth table for every function. A Boolean function can be transformed from an algebraic expression into a circuit diagram composed of logic gates connected in a particular structure. Circuit diagram for F –



Canonical and Standard Forms – Any binary variable can take one of two forms, x or $x^{/}$. A boolean function can be expressed in terms of binary variables. If all the binary variables are combined together using the AND operation, then there are a total of 2^ncombinations since each variable can take two forms. Each of the combinations is called a **minterm** or **standard product**.

In a similar way, if the variables are combined together with OR operation, then the term obtained is called a **maxterm** or **standard sum**.



			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	x′y′z′	m_0	x + y + z	M_0
0	0	1	x′y′z	m_1	x + y + z′	M_1
0	1	0	x′yz′	m_2	x + y′+ z	M_2
0	1	1	x′yz	m_3	x + y′ + z′	M_3
1	0	0	xy′z′	m_4	x′ + y + z	M_4
1	0	1	xy′z	m_5	x′ + y + z′	M_5
1	1	0	xyz′	m_6	x′ + y′ + z	M_6
1	1	1	xyz	m_7	x′ + y′ + z′	M_7

Relation between Minterms and Maxterms – Each minterm is the complement of it’s corresponding maxterm.

Constructing Boolean Functions – Now that we know what minterms and maxterms are, we can use them to construct boolean expressions. “A Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.” For example, consider two functions f1 and f2 with the following truth tables –

x	y	z	f_1	f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Standard Forms – Canonical forms are basic forms obtained from the truth table of the function. These forms are usually not used to represent the function as they are cumbersome to write and it is preferable to represent the function in the least number of literals possible. There are two types of standard forms –

1. **Sum of Products(SOP)-** A boolean expression involving AND terms with one or more literals each, OR'ed together.
2. **Product of Sums(POS)** A boolean expression involving OR terms with one or more literals each, AND'ed together, e.g.

SOP-

$$x' + xy + yz'$$

POS-

$$(x').(x + y).(y + z')$$

<u>Simplifications of Boolean Functions</u>

Boolean functions can be simplified using various techniques such as Boolean algebra, Karnaugh maps, and Quine-McCluskey algorithm. In Boolean algebra, we use the laws and rules of Boolean algebra to simplify expressions. We can convert any Boolean expression into a sum-of-products (SOP) form or a product-of-sums (POS) form. We can also relate a Boolean expression to a truth table and use a Karnaugh map to simplify Boolean expressions.
Karnaugh maps are grid-like representations of truth tables that are used to simplify Boolean algebra expressions. They provide grouping together Boolean expressions with common factors and eliminate unwanted variables from the expression.
Quine-McCluskey algorithm is another method used for simplifying Boolean functions. It is an iterative procedure that uses the concept of prime implicants to simplify Boolean expressions.

Laws of Boolean Algebra:

The basic laws of Boolean algebra—the commutative laws for addition and multiplication, the associative laws for addition and multiplication, and the distributive law—are the same as in ordinary algebra. Each of the laws is illustrated with two or three variables, but the number of variables is not limited to this.

Commutative Laws:

The commutative law of addition for two variables is written as A + B = B + A

The commutative law of multiplication for two variables is AB = BA

Associative Laws:

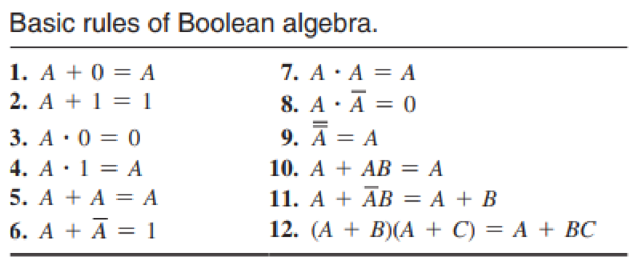
The associative law of addition is written as follows for three variables: A + (B + C) = (A + B) + C

The associative law of multiplication is written as follows for three variables: A(BC) = (AB)C

Distributive Law:

The distributive law is written for three variables as follows: A(B + C) = AB + AC

Rules of Boolean Algebra



Basic rules of Boolean algebra.

1. $A + 0 = A$	**7.** $A \cdot A = A$
2. $A + 1 = 1$	**8.** $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	**9.** $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	**10.** $A + AB = A$
5. $A + A = A$	**11.** $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	**12.** $(A + B)(A + C) = A + BC$

DeMorgan’s Theorems

DeMorgan, a mathematician who knew Boole, proposed two theorems that are an important part of Boolean algebra. In practical terms, DeMorgan’s theorems provide mathematical verification of the equivalency of the NAND and negative-OR gates and the equivalency of the NOR and negative-AND gates.

DeMorgan’s first theorem is stated as follows: The complement of a product of variables is equal to the sum of the complements of the variables.

The complement of two or more ANDed variables is equivalent to the OR of the complements of the individual variables. The formula for expressing this theorem for two variables is

$$\overline{XY} = \overline{X} + \overline{Y}$$

DeMorgan’s second theorem is stated as follows:

The complement of a sum of variables is equal to the product of the complements of the variables.

Stated another way,

The complement of two or more ORed variables is equivalent to the AND of the complements of the individual variables. The formula for expressing this theorem for two variables is

$$\overline{X + Y} = \overline{X}\,\overline{Y}$$

The Karnaugh Map

A Karnaugh map is similar to a truth table because it presents all of the possible values of input variables and the resulting output for each value. Instead of being organized into columns and rows like a truth table, the Karnaugh map is an array of cells in which each cell represents a binary value of the input variables. The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells. Karnaugh maps can be used for expressions with two, three, four, and five variables.

The number of cells in a Karnaugh map, as well as the number of rows in a truth table, is equal to the total number of possible input variable combinations. For three variables, the number of cells is 23 = 8. For four variables, the number of cells is 24 = 16.

In many digital circuits and practical problems we need to find expression with minimum variables. We can minimize Boolean expressions of 3, 4 variables very easily using K-map without using any Boolean algebra theorems. K-map can take two forms Sum of Product (SOP) and Product of Sum (POS) according to the need of problem. K-map is table like representation but it gives more information than TRUTH TABLE. We fill grid of K-map with 0's and 1's then solve it by making groups.

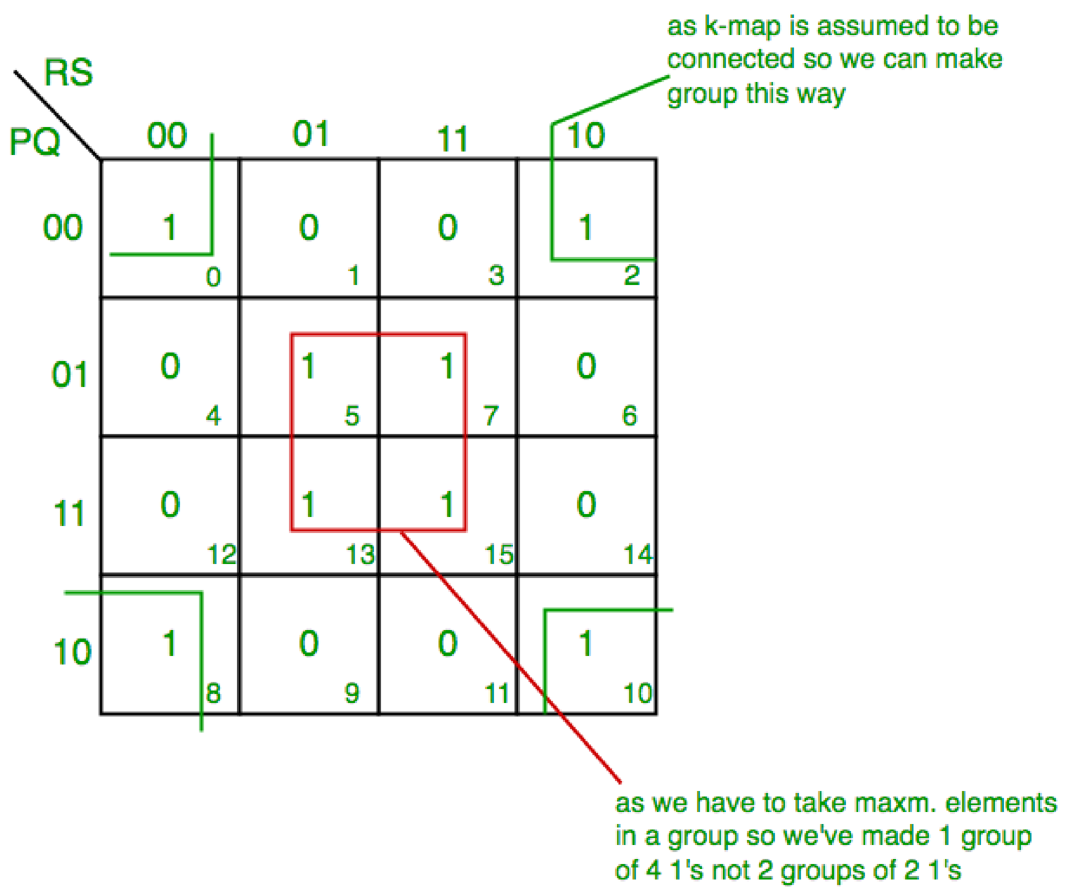
Steps to solve expression using K-map-

1. Select K-map according to the number of variables.
2. Identify minterms or maxterms as given in problem.
3. For SOP put 1's in blocks of K-map respective to the minterms (0's elsewhere).
4. For POS put 0's in blocks of K-map respective to the maxterms(1's elsewhere).
5. Make rectangular groups containing total terms in power of two like 2,4,8 ..(except 1) and try to cover as many elements as you can in one group.
6. From the groups made in step 5 find the product terms and sum them up for SOP form.

K-map for 4 variables –

K-map 4 variable SOP form

F(P,Q,R,S)=?(0,2,5,7,8,10,13,15)


as k-map is assumed to be connected so we can make group this way
RS
PQ
00
01
11
10
as we have to take maxm. elements in a group so we've made 1 group of 4 1's not 2 groups of 2 1's

	00	01	11	10
00	1 (0)	0 (1)	0 (3)	1 (2)
01	0 (4)	1 (5)	1 (7)	0 (6)
11	0 (12)	1 (13)	1 (15)	0 (14)
10	1 (8)	0 (9)	0 (11)	1 (10)

From **red** group we get product term—
QS

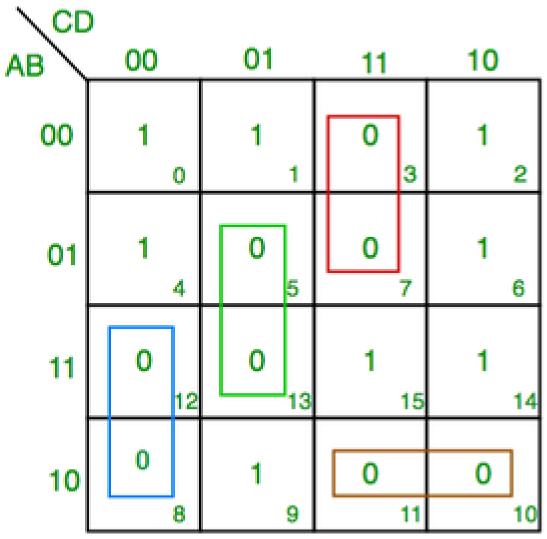
From **green** group we get product term—
Q’S’

Summing these product terms we get- **Final expression (QS+Q’S’)**

K-map of 4 variables –

K-map 4 variable POS form

F(A,B,C,D)=?(3,5,7,8,10,11,12,13)



From **green** group we find terms
C’ D B

Taking their complement and summing them

(C+D’+B’)

From **red** group we find terms
C D A’

Taking their complement and summing them

(C’+D’+A)

From **blue** group we find terms
A C’ D’

Taking their complement and summing them

(A’+C+D)

From **brown** group we find terms
A B’ C

Taking their complement and summing them

(A’+B+C’)

Finally we express these as product –

(C+D’+B’).(C’+D’+A).(A’+C+D).(A’+B+C’)

Quine McCluskey Method

Boolean function simplification is one of the basics of Digital Electronics. The quine-McCluskey method also called the **tabulation method** is a very useful and convenient method for simplification of the Boolean functions for a large number of variables (greater than 4). This method is useful over K-map when the number of variables is larger for which K-map formation is difficult. This method uses prime implicants for simplification.

Quine McCluskey Method (QMC):

- Quine McCluskey method also known as the tabulation method is used to minimize the Boolean functions.
- It simplifies boolean expression into the simplified form using prime implicants.
- This method is convenient to simplify boolean expressions with more than 4 input variables.
- It uses an automatic simplification routine.

Terminologies:

Implicant: Implicant is defined as a group of 1’s(for minterm).
Prime implicant: It is the largest possible group of 1’s(for minterm).

Essential Prime implicant: Essential prime implicants are groups that cover at least one minterm which cannot be covered by other applicants.

Note: This method uses decimal to binary representation.

Steps for Quine McCluskey Method:

1. Arrange the given minterms according to the number of ones present in their binary representation in ascending order.
2. Take the minterms from the continuous group if there is only a one-bit change to make their pair.
3. Place the '-' symbol where there is a bit change accordingly and keep the remaining bits the same.
4. Repeat steps 2 to 3 until we get all prime implicants (when all the bits present in the table are different).
5. Make a prime implicant table that consists of the prime implicants (obtained minterms) as rows and the given minterms (given in problem) as columns.
6. Place '1' in the minterms (cell) which are covered by each prime implicant.
7. Observe the table, if the minterm is covered by only one prime implicant then it is an essential to prime implicant.
8. Add the essential prime implicants to the simplified boolean function.

Example: Simplify using tabulation method: F(A,B,C,D,E,F,G) = ∑m(20,28,52,60)
Solution: Convert the given minterms in their binary representation and arrange them according to number of one's present in the binary representation.

TABLE-1

Group	Minterms	A	B	C	D	E	F	G
0	20	0	0	1	0	1	0	0
1	28	0	0	1	1	1	0	0
	52	0	1	1	0	1	0	0
2	60	0	1	1	1	1	0	0

As 20 has 2 1s in its representation it is kept in one group(0). Similarly, 28 and 52 contain 3 1s in their representation so it is kept in the next group(1). 60 in the next group(2).

Now, for table-2 take minterms from successive groups(simultaneous group only) which have an only a 1-bit difference in their representation and form their pair by merging them and making a group of the pairs which are from the same groups that are merged (for example 20 is from group 0 and 28 is from group 1 so it is added to the group 0. 20 belongs to group 0 in table 1 and 52 belongs to group 1 in table 1 so its kept in the same group in table 2. Similarly, make all the possible pairs with the help of the above table and mark – where it is a bit different.

TABLE-2								
Group	**Pair**	**A**	**B**	**C**	**D**	**E**	**F**	**G**
0	(20,28)	0	0	1	–	1	0	0
	(20,52)	0	–	1	0	1	0	0
1	(28,60)	0	–	1	1	1	0	0
	(52,60)	0	1	1	–	1	0	0

For table 3 repeat the same step by taking pairs of successive groups merging them where there is only a 1-bit difference and keeping them in groups according to the groups from where they are merged and placed – in bit difference.

TABLE-3								
Group	**Quad**	**A**	**B**	**C**	**D**	**E**	**F**	**G**
0	(20,28,52,60)	0	–	1	–	1	0	0

After table 3 the process is stopped as there is no 1-bit difference in the remaining group minterms in the simultaneous groups of table 3.

Now, the remaining quads present in table 3 represent the prime implicants for the given boolean function. So, we construct prime implicants table which contains the obtained prime implicants as rows and the given minterms as columns. Place 1 in the corresponding place which the minterm can represent. Add the minterm to the simplified boolean expression if the given minterm is only covered by this prime implicant.

A’CEF’G’ is obtained from table 3 as A, F, G contains 0 so A’F’G’, C, and E contain 1 so CE.

Prime Implicants Table				
Minterms ⇢ **Prime Implicants ⇣**	**20**	**28**	**52**	**60**
A’CEF’G'(20,28,52,60)	1	1	1	1
Simplified Boolean Function = A’CEF’G’				

A’CEF’G’ is in simplified function as it is the only prime implicant that covers all minterms.