Application of Boolean Logic

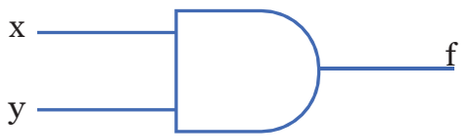
Boolean Expression through Logic gates

We know that a Boolean function is an algebraic expression formed with Boolean variables, operator OR, AND and NOT, parenthesis and equal to sign. Any Boolean function can be transformed in a straight forward manner from an algebraic expression into logic circuit diagram. This logic circuit diagram can be composed of basic gates AND, OR and NOT or advance gates, NAND and NOR, etc.

Logic circuits that perform logical operations such as AND, OR and NOT are called gates. A gate is block of hardware that produces a logic 0 or a logic 1 output, in response to the binary signal(s) applied to it as input.

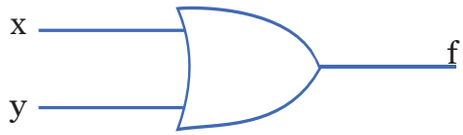
Let's look at the symbols used to represent digital logic gates

AND $f(x, y) = x.y$



For logical multiplication operation

OR $f(x, y) = x+y$

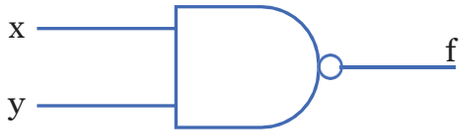


For logical addition operation

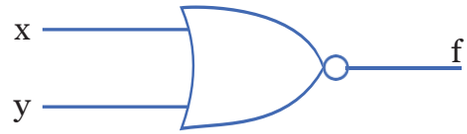
NOT $f(x) = x'$

For Complementation

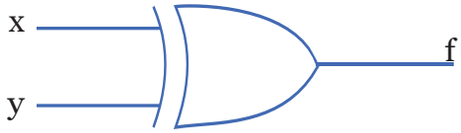
NAND $f(x,y) = (x.y)'$



NOR $f(x,y) = (x+y)'$



XOR $f(x,y) = x \oplus y = x'y + xy'$

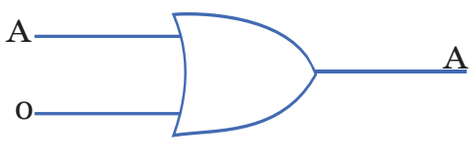


All the gates, except NOT, have two inputs represented on LHS and one output - on the RHS. Gates can have more than two inputs also. As the logic circuit is a way of representing Boolean expression, let's represent, axioms and theorems in this form.

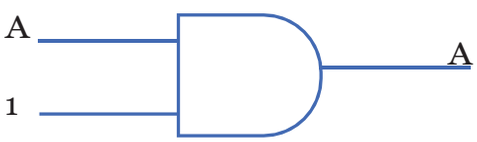
When a Boolean Function is implemented with logic gates, each literal in the function becomes an input to a gate. That's why we minimize the function - to reduce the number of inputs to gate also to reduce the total number of gates in the circuit. Also, let's assume that literal in both the forms- normal and complemented is available. This will help us in drawing neat circuit diagrams.

1. Identity

i) $A + 0 = A$



ii) $A \cdot 1 = A$



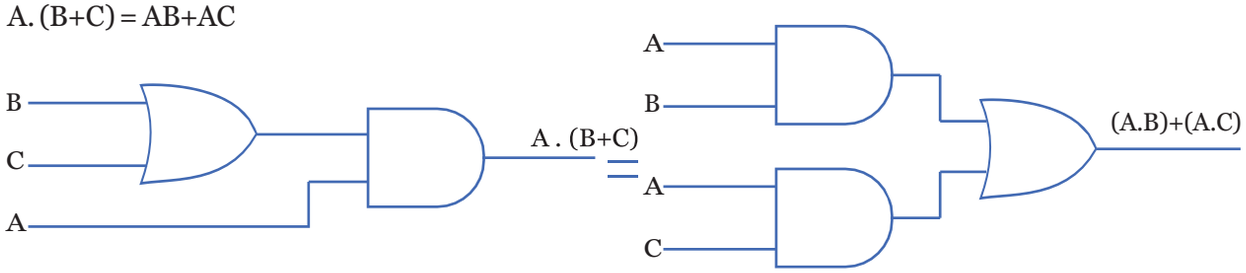
2. **Commutative**

i) A+B = B+A

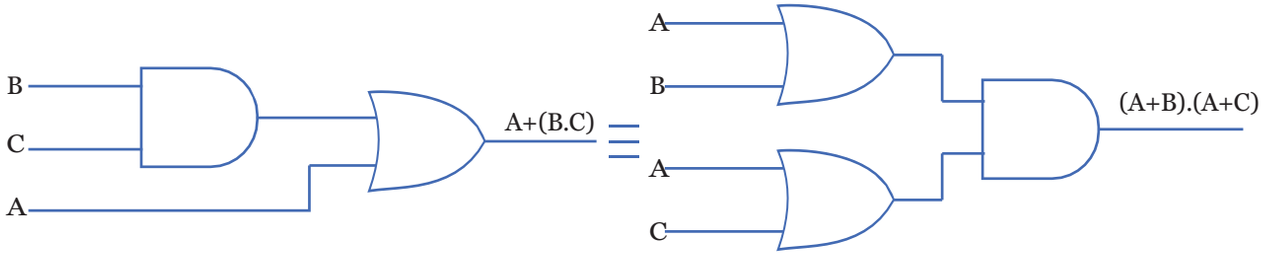
ii) A . B = B . A

3. **Distributive**

i) A. (B+C) = AB+AC

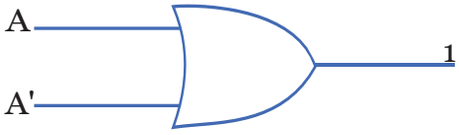


ii) A+ (B.C) = (A+B). (A+C)

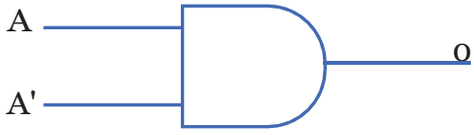


4. **Complement**

i) A+A' = 1

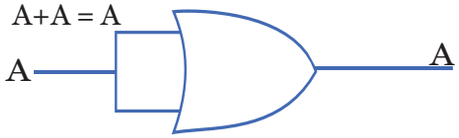


ii) A.A' = 0

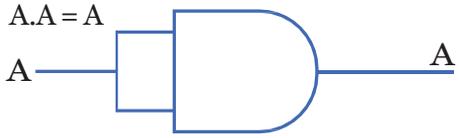


5. **Indempotency**

i) A+A = A

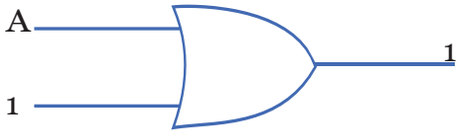


ii) A.A = A

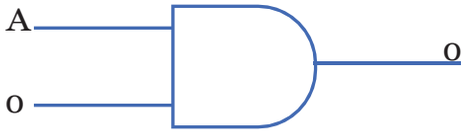


6. **Null Element**

i) A+1 = 1



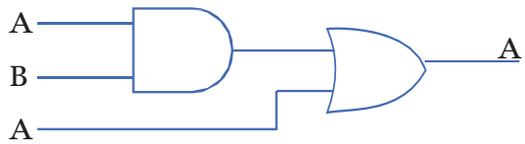
ii) A.0 = 0



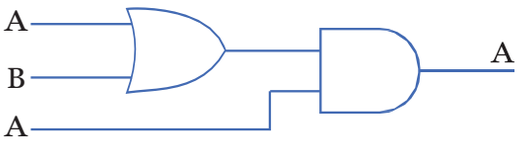
7. **Involution (A')' = A**

8. **Absorption**

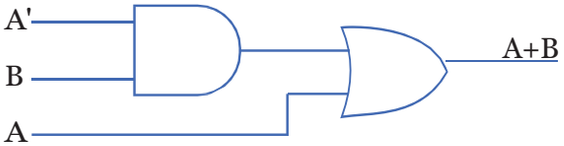
First law: i) A+ (AB) = A



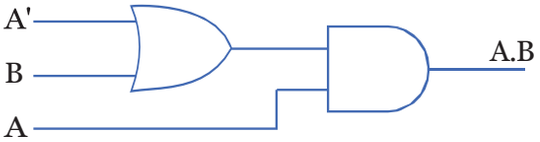
ii) A.(A+B) = A



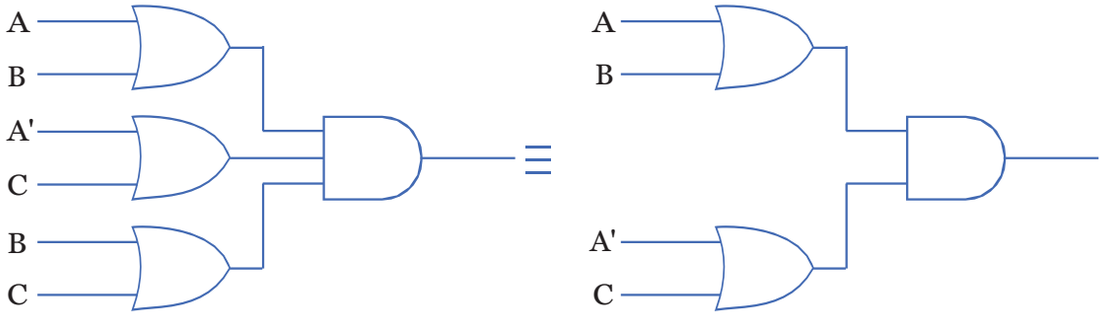
Second law: i) A+A'B = A+B



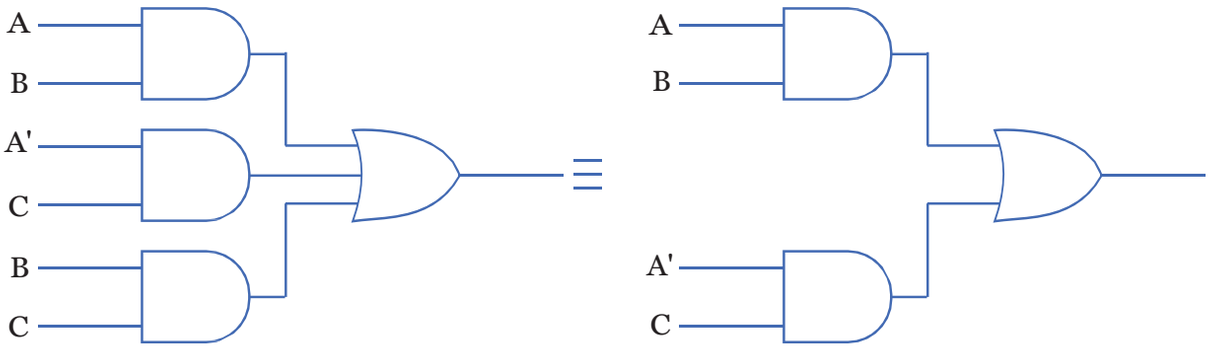
ii) A.(A'+B) = A.B



Third law: i) (A + B) (A' + C) (B + C) = (A + B) (A' + C)

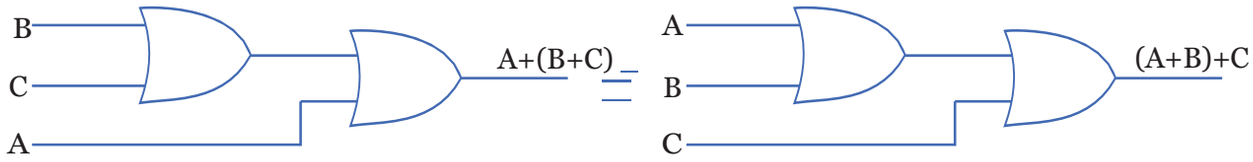


ii) (A . B) + (A' . C) + (B . C) = A . B + A' . C

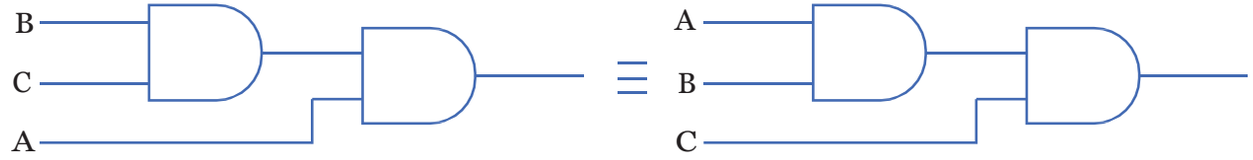


9. **Associative**

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



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



10. **DeMorgan's Theorem**

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

(NOR) **(Negative AND)**

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

(NAND) **(Negative OR)**

We have already verified these theorems using truth table and algebraic method in previous chapter.

Any Boolean function can be associated with a logic circuit, in which the inputs and outputs, represent the statement of Boolean algebra. Whatever is the complexity of function, they all can be constructed using three basic gates. For implementation of a Boolean function in logic circuit form:

The function will either be in SOP form or POS form. The SOP (Sum-of- Product) form is implemented in the following manner:

i) Each AND term is represented by AND gate (one gate for each AND term)

ii) Output of each AND gate is connected as input to single OR gate.

POS (Product-of-Sums) is implemented as

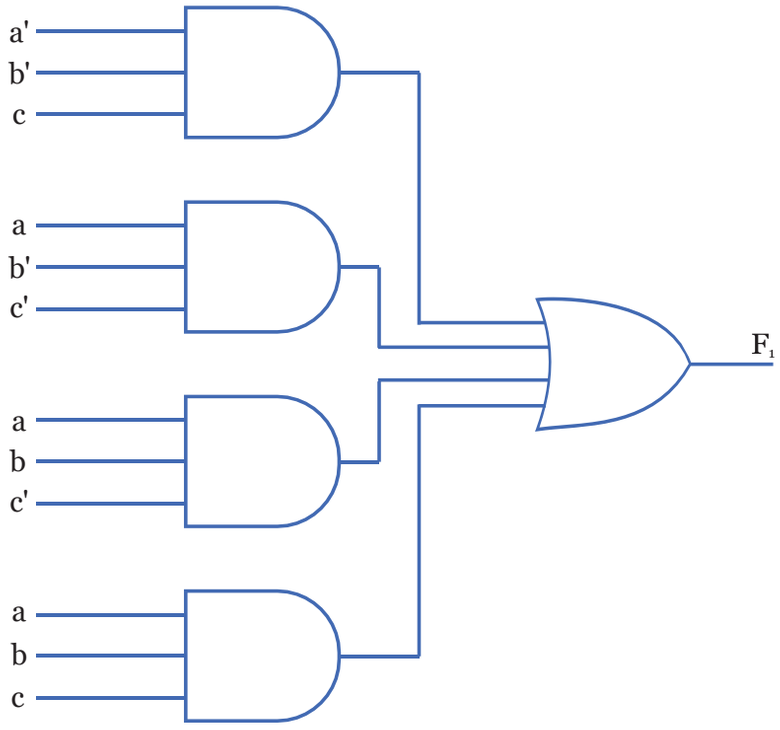
i) Each OR term is represented using an OR gate.

ii) Output of all the OR gate(s) are connected as input to single AND gate.

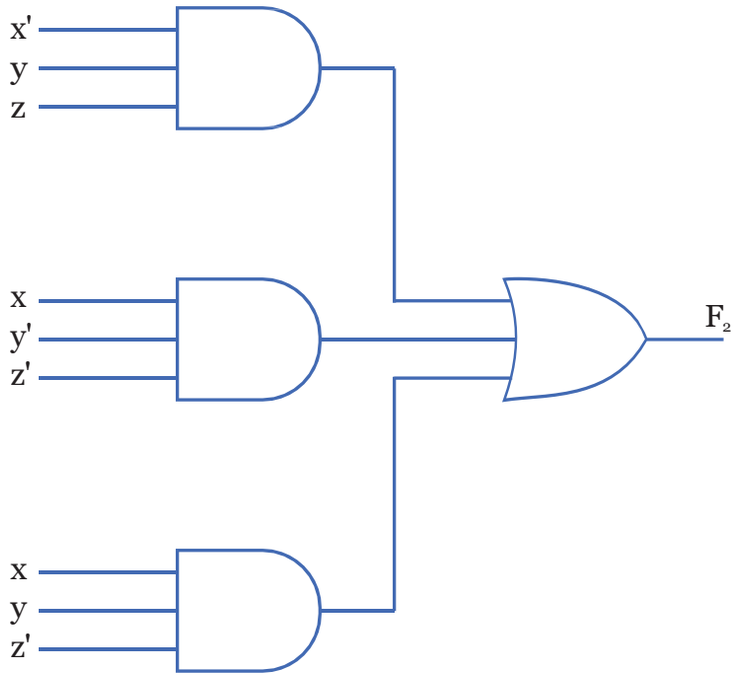
In both the representations, *it is assumed that both normal and complemented inputs, for all the variables, are available. So inverters are not needed.* Also in both the types of representation, two levels of gates are used. In SOP form AND gates are connected to OR gate and in POS form OR gates are connected to AND gates. Thus the implementation is known as two-level implementation.

Let's illustrate some Boolean functions from previous chapter in circuit form.

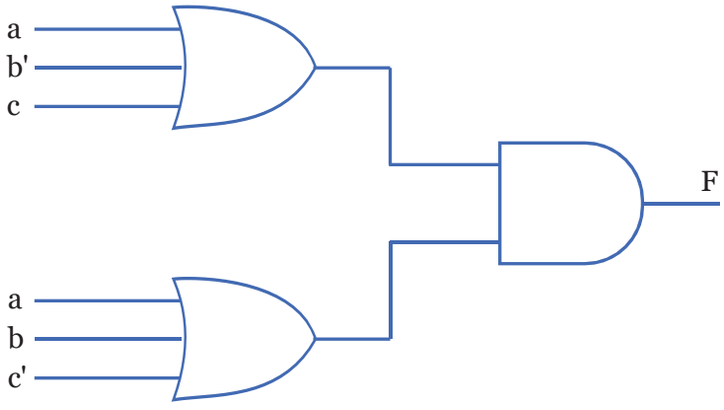
$F_1 = a'b'c + ab'c' + abc' + abc$



$F_2 = x'yz + xy'z' + xyz'$



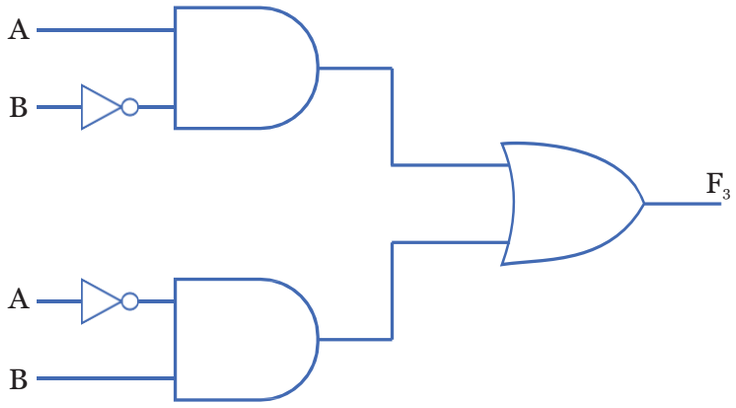
$F_3 (a,b,c) = (a+b'+c) . (a+b+c')$



How to get a Boolean expression for a Logic Circuit?

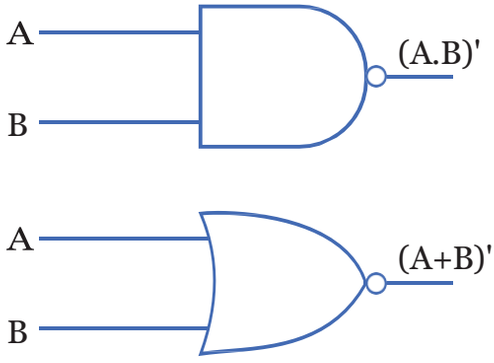
To derive the Boolean expression for a given logic circuit begin at the left most inputs and work towards the final output, by writing the expression for each gate.

For example:



- The expression for the left-most first gate will be B'
- This output along with A becomes input to AND gate so its output will be A.B'
- Similarly, working on 2nd set of gate from Left most side we will have A' and then A'.B
- Output of these two AND gates is input to OR gate, therefore the expression for the OR gate will be A.B'+ A'.B which is the final output expression for the entire circuit

The circuits made, till now have one type of gate(s) AND / OR at first level and another type of gate OR / AND at second level. As these are simple circuits, but in a complex circuit we might have a function which uses both type of gate(s) at first level. Building a really big complex digital system using many types of gates will make the job tedious. We need to have a gate, which can replace the function of all other gates, hence can be used to implement any digital circuit, such gates are also known as universal gates. NAND and NOR are such universal gates. NAND for 2 input is (A.B)' and NOR for 2 input is (A+B)'



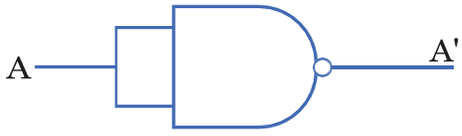
An SOP expression, represented by AND gate(s) at first level and OR gate at second level, can naturally be implemented by only NAND gate(s). Similarly a POS expression, which uses OR gates at first level and AND gate at second level can naturally be implemented through NOR gate(s) only.

Now let's see how each logic gate can be created with universal gate, so that entire circuit can be implemented through single type of gate.

Realization of all functions using NAND gate

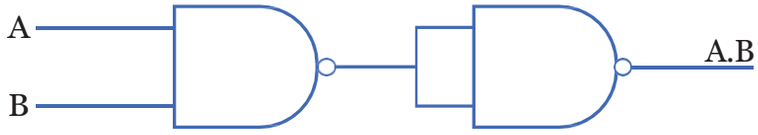
NOT A' = (A.A)'

i.e. A nand A



AND A.B = ((A.B)')'

i.e. (A nand B) nand (A nand B)



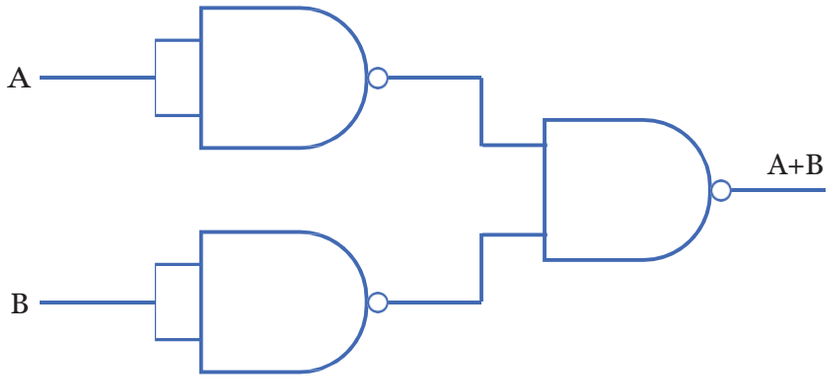
OR = A+B

= ((A+B)')'

= (A'.B')'

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

i.e. (A nand A) nand (B nand B)

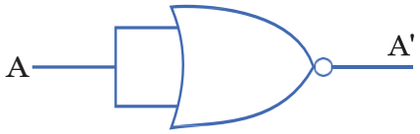


Realization of logic functions using NOR gate

NOT = (A)'

= (A+A)'

i.e. A NOR A



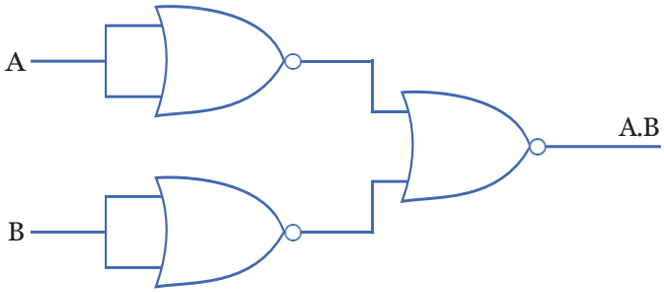
AND = A.B

= ((A.B)')'

= (A'+B')'

= ((A nor A) + (B nor B))'

i.e. (A nor A) nor (B nor B)

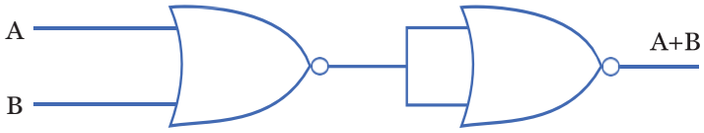


OR = A+B

= ((A+B)')'

= (A nor B)'

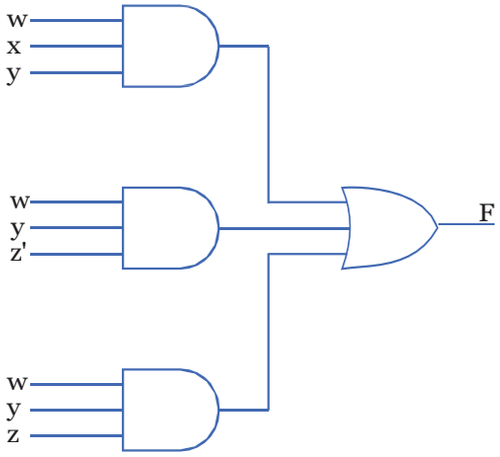
i.e. (A nor B) nor (A nor B)



Let's take some example of implementing complete circuit through universal gate

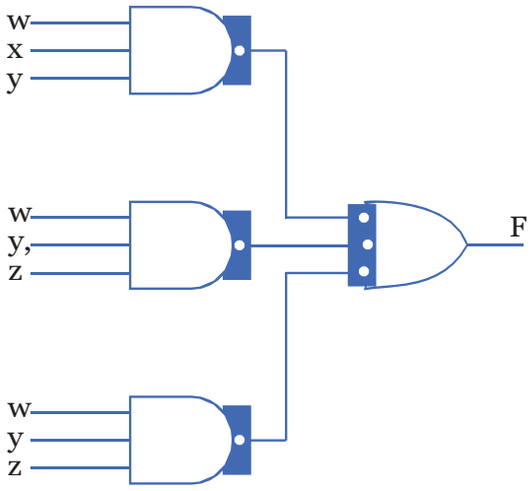
F (w,x,y,z) = wxy + wyz' + wyz

This is an SOP expression and will be represented by AND OR gate

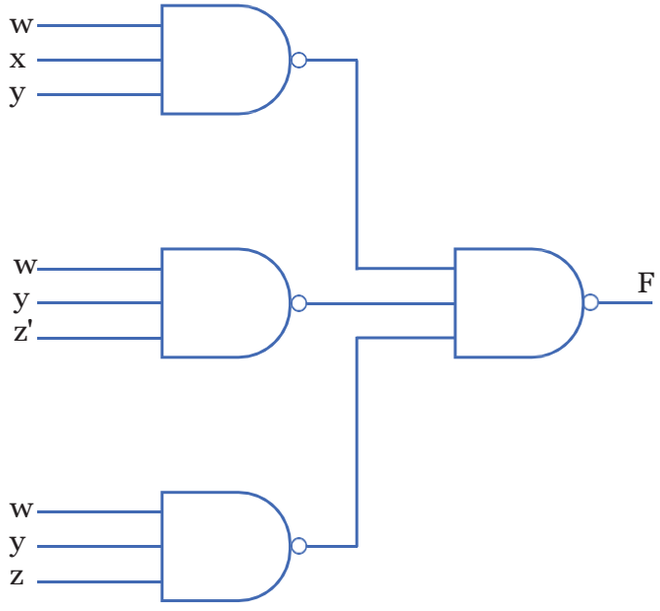


Let's apply De Morgan's theorem to implement the above circuit with minimum number of NAND gates, because if each gate is replaced by universal gate using above solution we will end up using a large number of gates and increase the size of circuit. You may try this for a Boolean function, which can then be compared by the solution given below.

If we introduce bubbles at AND gate(s) output and OR gate(s) input, the resultant will remain same as A = (A')'. Applying this, the above circuit will become



All the gates at first level are NAND gates. If you look at the second level gate it is (A'+B'+C') assuming A, B & C are output from 1st, 2nd, and 3rd gate respectively. (A'+B'+C') = (A.B.C)' using DeMorgan's Theorem. So now the complete circuit may be represented using NAND gate as:

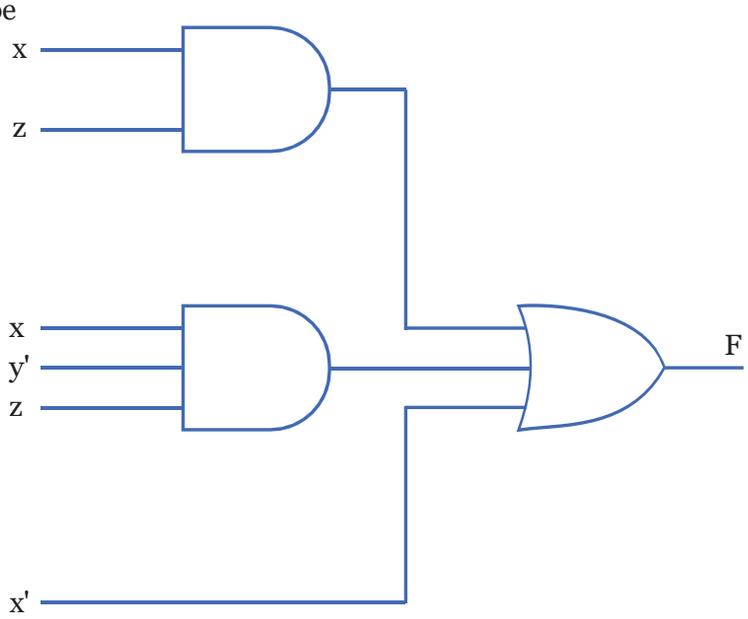


So the trick is replace AND gate at 1st level and OR gate at 2nd level by NAND gate.

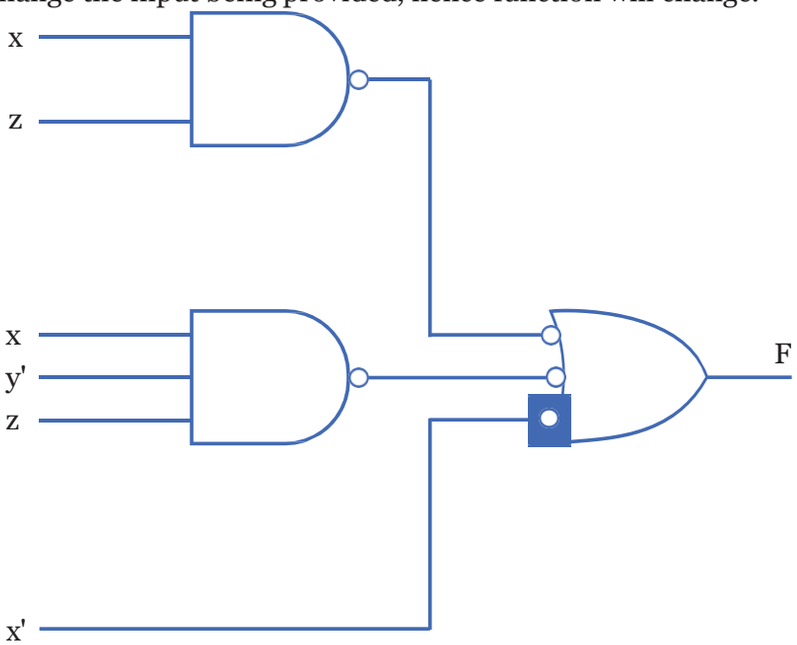
Let's take another kind of example

$F(x,y,z) = xz + xy'z + x'$

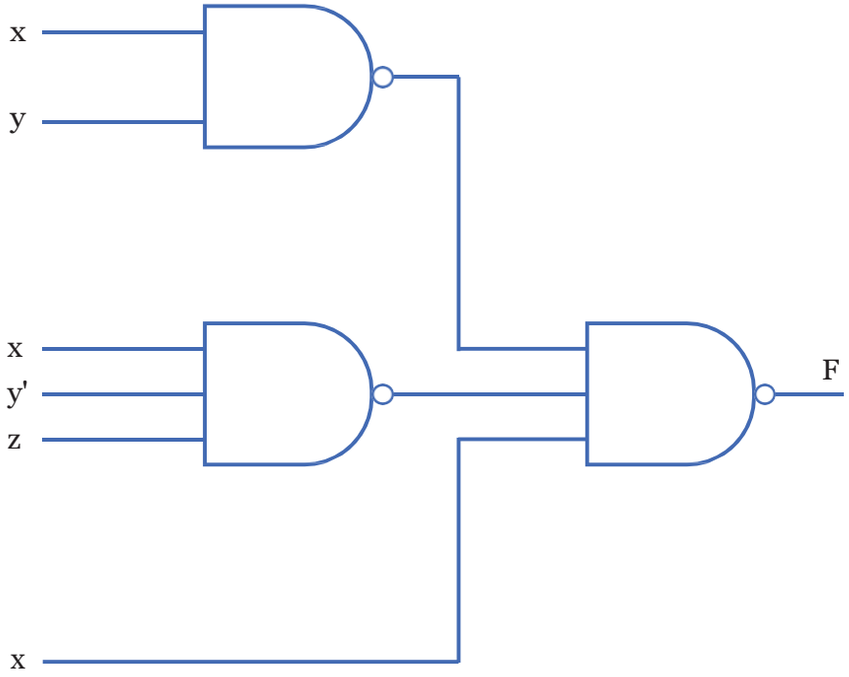
Its **AND → OR** diagram will be



If we replace all gates through NAND only, then the single input going to second level gate (OR) will get complemented, which will change the input being provided, hence function will change.



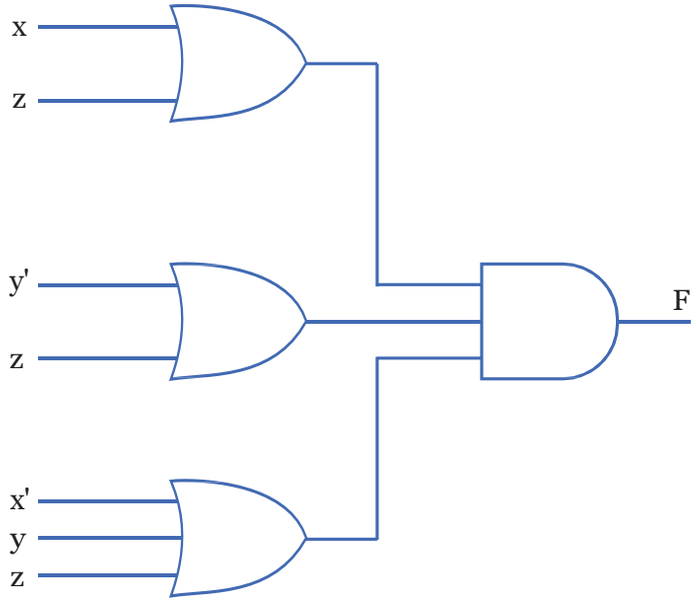
This can be handled by sending an inverted input to 2nd level gate whenever there is single input which passes directly to second level gate. So the correct solution will be



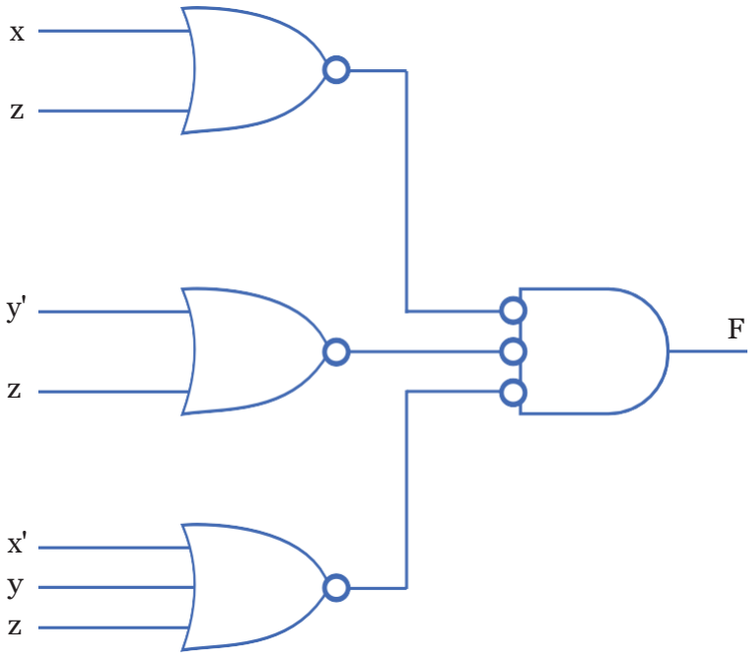
Now let's implement a complete circuit using NOR gate(s)

F (x,y,z) = (x+z) (y'+z) (x'+y+z)

Its natural representation is **OR → AND**



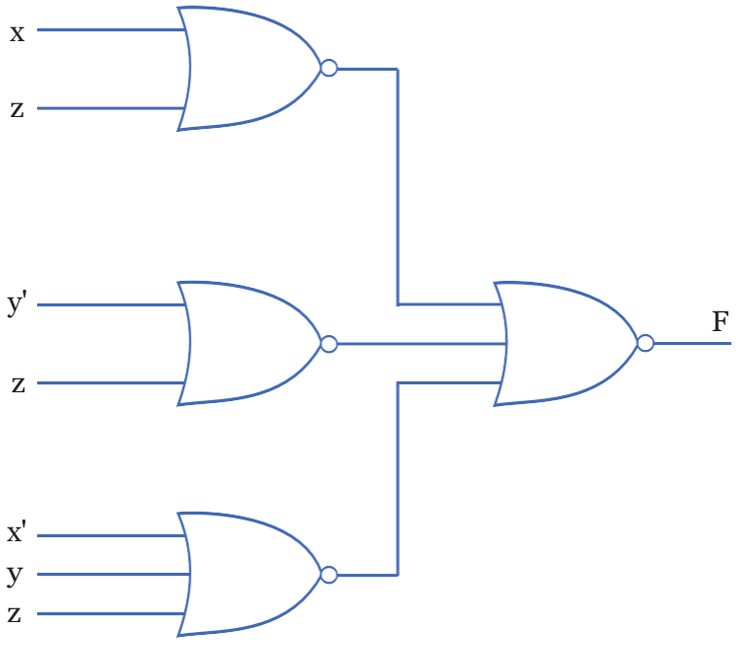
Again using the same concept, we have



bubbles to be highligted

Which is OK as the output of 1st level gate is complemented and it is again complemented before providing to next level gate.

The second level gate is equivalent to NOR gate as per DeMorgan's theorem. So the circuit implementation using NOR gate only will be



In this implementation also, if an input is directly moving to second level gate in normal OR→AND representation then, in NOR implementation the input will be complemented before moving to 2nd level gate.

Now we know SOP expression can be implemented using AND → OR circuit or universal NAND gate. POS expression can be implemented using OR → AND or universal NOR gate. Representing POS using only NAND gates will require conversion of POS to SOP form and then implement same using only NAND only gates. We have already learnt the way of converting expression from one form to another.

Let's take an example:

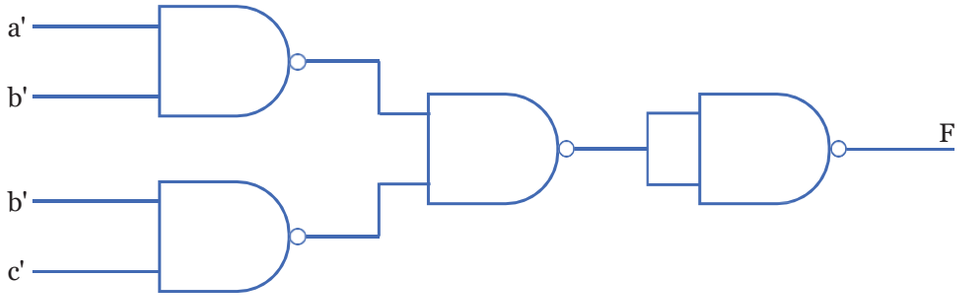
F (a,b,c) = (a+b).(b+c)

= [((a+b) . (b+c))']'

= [(a+b)' + (b+c)']'

= [(a'.b') + (b'.c')]'

So the circuit will be



For representation of SOP through only NOR gates, we can proceed in similar manner.

One of the most recent use of Boolaen logic is, Internet search engine and Database search. You have applied operator AND, OR and NOT in SQL queries and already know it's usage. Let's learn the usage of Boolean operators for searching Internet.

We know, Internet is a vast computer database and the proper use of Boolean operators - AND, OR and NOT would increase the effectiveness of web search. These operators can act as effective filters for finding just the information one need from Internet. So most of the search engine support some form of Boolean query using Boolean operators (check the help section of your favorite search engine supported by it). An engine is a search program, that allows us to search its data base.

These operators can be used in two ways:

i) You can use the words - AND, OR, NOT

ii) You can use symbols (math equivalents) - **+** for AND, **-** for NOT, OR is the default setting in many search engine.

Note: When you are using symbols, don't add space between the symbols and your query.

OR operator is used to broaden the search. It's interpreted as "at least one is required, more than one or all can be returned". So when search strings (keyword) are joined using OR, the resultant will include - any, some or all of the keywords used in the statement. The operator will ensure that you do not miss anything valuable.

AND operator is used to narrow the search. It is interpreted as "all is required ". So when search string i.e. keywords are joined using AND, the resultant will include the documents containing all the keywords.

NOT operator will again narrow down the search, this time by excluding the search string i.e. keyword.

LET'S REVISE

- Gate is an electronic system that performs a logical operation on a set of input signal(s). They are the building blocks of IC.
- An SOP expression when implemented as circuit - takes the output of one or more AND gates and OR's them together to create the final output.
- An POS expression when implemented as circuit - takes the output of one or more OR gates and AND's them together to create the final output.
- Universal gates are the ones which can be used for implementing any gate like AND, OR and NOT, or any combination of these basic gates; NAND and NOR gates are universal gates.
- Implementation of a SOP expression using NAND gates only
  1) All 1st level AND gates can be replaced by one NAND gate each.
  2) The output of all 1st level NAND gate is fed into another NAND gate.

  This will realize the SOP expression
  3) If there is any single literal in expression, feed its complement directly to 2nd level NAND gate.

  Similarly POS using NOR gate can be implemented by replacing NAND by NOR gate.
- Implementation of POS / SOP expression using NAND / NOR gates only.
  1) All literals in the first level gate will be fed in their complemented form.
  2) Add an extra NAND / NOR gate after 2nd level gate to get the resultant output.