Discrete Mathematics (ITPC-309)

Boolean Algebra

Dr. Sanga Chaki

Department of Information Technology

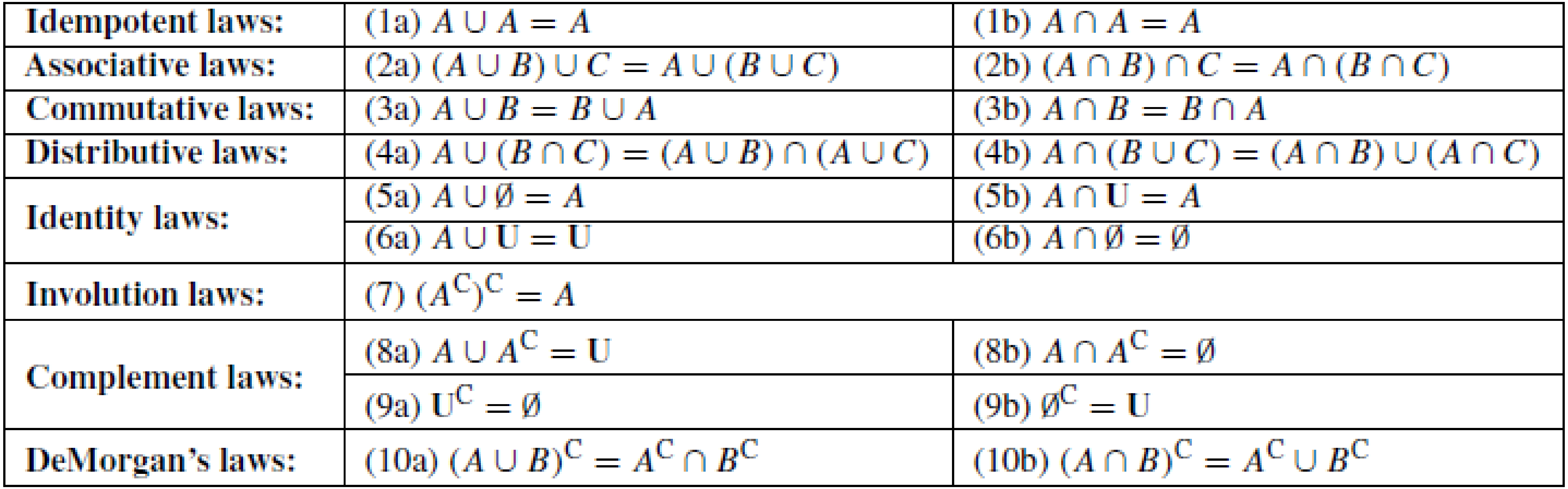
Dr. B. R. Ambedkar National Institute of Technology, Jalandhar

Contents

1. Boolean Algebra
2. Binary Operations,
3. Lattices and Boolean Algebra,
4. Boolean Expressions,
5. Logic Gates and Circuits

Introduction

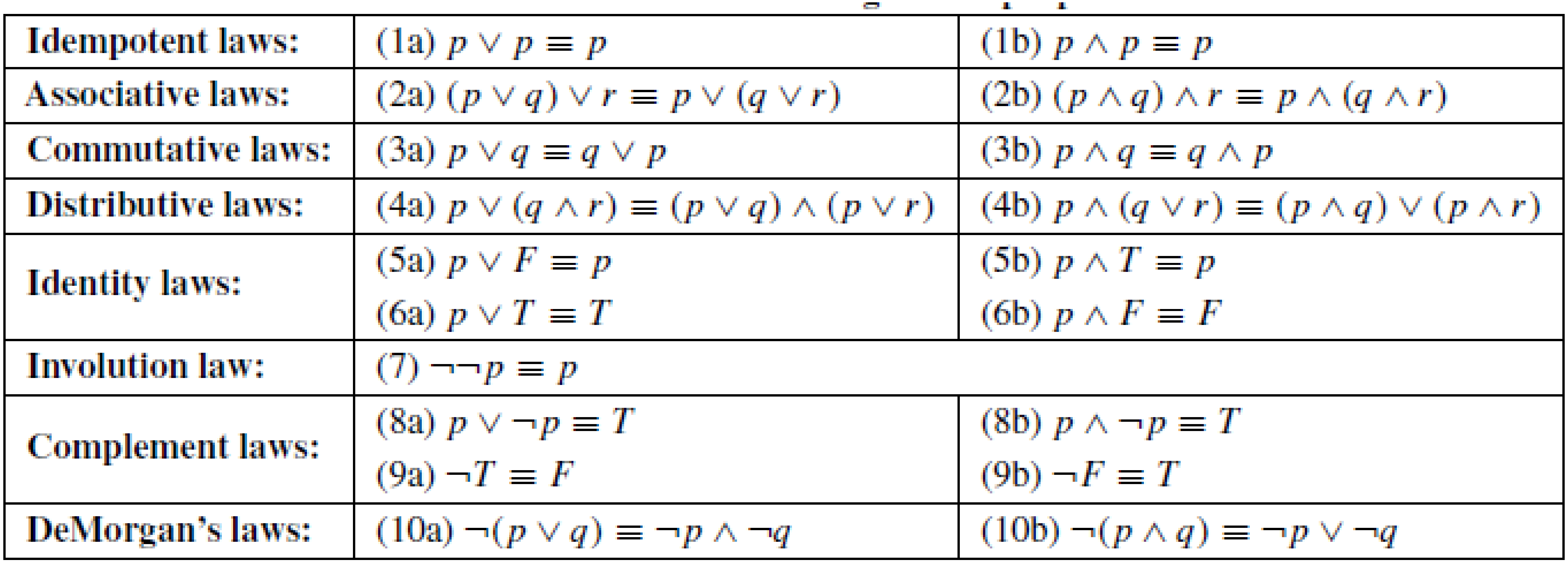
1. Both sets and propositions satisfy similar laws
2. These laws are used to define an abstract mathematical structure called a Boolean algebra, which is named after the mathematician George Boole
3. Laws of the algebra of sets



Idempotent laws:	(1a) $A \cup A = A$	(1b) $A \cap A = A$
Associative laws:	(2a) $(A \cup B) \cup C = A \cup (B \cup C)$	(2b) $(A \cap B) \cap C = A \cap (B \cap C)$
Commutative laws:	(3a) $A \cup B = B \cup A$	(3b) $A \cap B = B \cap A$
Distributive laws:	(4a) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	(4b) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
Identity laws:	(5a) $A \cup \emptyset = A$	(5b) $A \cap \mathrm{U} = A$
	(6a) $A \cup \mathrm{U} = \mathrm{U}$	(6b) $A \cap \emptyset = \emptyset$
Involution laws:	(7) $(A^{\mathrm{C}})^{\mathrm{C}} = A$	
Complement laws:	(8a) $A \cup A^{\mathrm{C}} = \mathrm{U}$	(8b) $A \cap A^{\mathrm{C}} = \emptyset$
	(9a) $\mathrm{U}^{\mathrm{C}} = \emptyset$	(9b) $\emptyset^{\mathrm{C}} = \mathrm{U}$
DeMorgan's laws:	(10a) $(A \cup B)^{\mathrm{C}} = A^{\mathrm{C}} \cap B^{\mathrm{C}}$	(10b) $(A \cap B)^{\mathrm{C}} = A^{\mathrm{C}} \cup B^{\mathrm{C}}$

Introduction

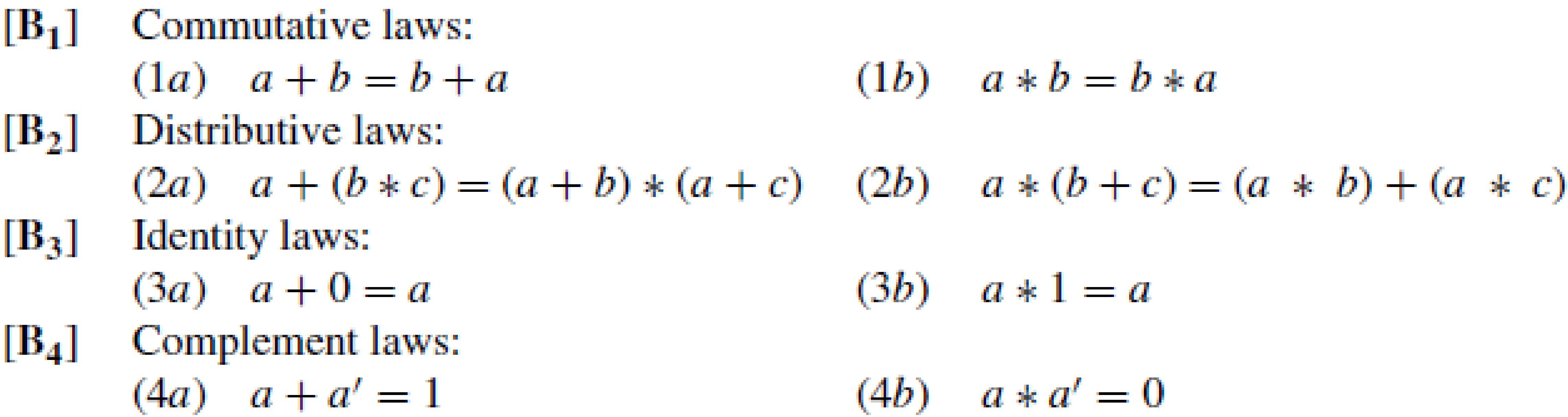
1. Both sets and propositions satisfy similar laws
2. These laws are used to define an abstract mathematical structure called a Boolean algebra, which is named after the mathematician George Boole
3. Laws of the algebra of propositions



Idempotent laws:	(1a) $p \vee p \equiv p$	(1b) $p \wedge p \equiv p$
Associative laws:	(2a) $(p \vee q) \vee r \equiv p \vee (q \vee r)$	(2b) $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Commutative laws:	(3a) $p \vee q \equiv q \vee p$	(3b) $p \wedge q \equiv q \wedge p$
Distributive laws:	(4a) $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	(4b) $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Identity laws:	(5a) $p \vee F \equiv p$ (6a) $p \vee T \equiv T$	(5b) $p \wedge T \equiv p$ (6b) $p \wedge F \equiv F$
Involution law:	(7) $\neg\neg p \equiv p$	
Complement laws:	(8a) $p \vee \neg p \equiv T$ (9a) $\neg T \equiv F$	(8b) $p \wedge \neg p \equiv T$ (9b) $\neg F \equiv T$
DeMorgan's laws:	(10a) $\neg(p \vee q) \equiv \neg p \wedge \neg q$	(10b) $\neg(p \wedge q) \equiv \neg p \vee \neg q$

Basic Definitions

1. Let B be a nonempty set with two binary operations + and ∗, a unary operation ‘, and two distinct elements 0 and 1.
2. Then B is called a Boolean algebra if the following axioms hold where a, b, c are any elements in B:



[B$_1$] Commutative laws:

(1a) $a + b = b + a$ (1b) $a * b = b * a$

[B$_2$] Distributive laws:

(2a) $a + (b * c) = (a + b) * (a + c)$ (2b) $a * (b + c) = (a * b) + (a * c)$

[B$_3$] Identity laws:

(3a) $a + 0 = a$ (3b) $a * 1 = a$

[B$_4$] Complement laws:

(4a) $a + a' = 1$ (4b) $a * a' = 0$

Basic Definitions

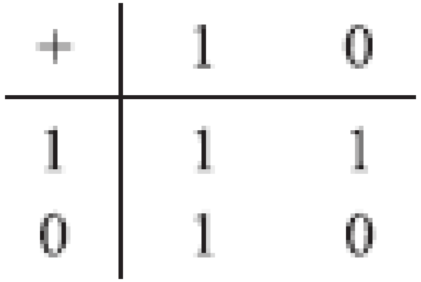
1. We will sometimes designate a Boolean algebra by **<B,+, ∗,’ , 0, 1>** to emphasize its six parts.
2. We say 0 is the **zero element**
3. 1 is the **unit element**
4. a’ is the **complement** of a.
5. We will usually drop the symbol ∗ and use juxtaposition instead.
   - Then a*(b + c) = a*b + a*c is written a(b + c) = ab + ac

Basic Definitions

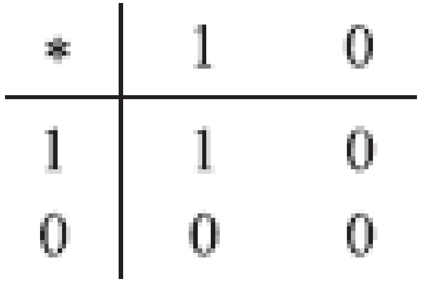
1. The operations +, ∗, and ‘ are called sum, product, and complement,
2. Unless guided by parentheses, ‘ has precedence over ∗, and ∗ has precedence over +.
3. For example, a + b ∗ c means a + (b ∗ c) and not (a + b) ∗ c;
4. a ∗ b’ means a ∗ (b’) and not (a ∗ b)’

Examples:

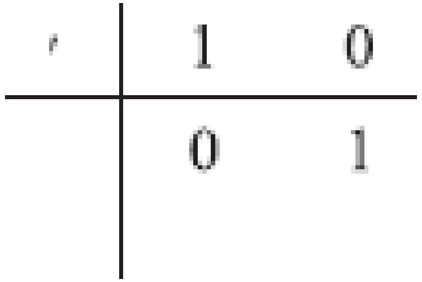
1. Let B = {0, 1}, the set of bits (binary digits), with the binary operations of + and $*$ and the unary operation ‘ defined in the below figure.



+	1	0
1	1	1
0	1	0



$*$	1	0
1	1	0
0	0	0



′	1	0
	0	1

2. Then B is a Boolean algebra – check if the four laws are followed.

Examples:

1. Let D_{70} = {1, 2, 5, 7, 10, 14, 35, 70}, the divisors of 70.
2. Define +, ∗, and ‘ on D_{70} by following

$$a + b = \mathrm{lcm}(a, b), \quad a * b = \gcd(a, b), \quad a' = \frac{70}{a}$$

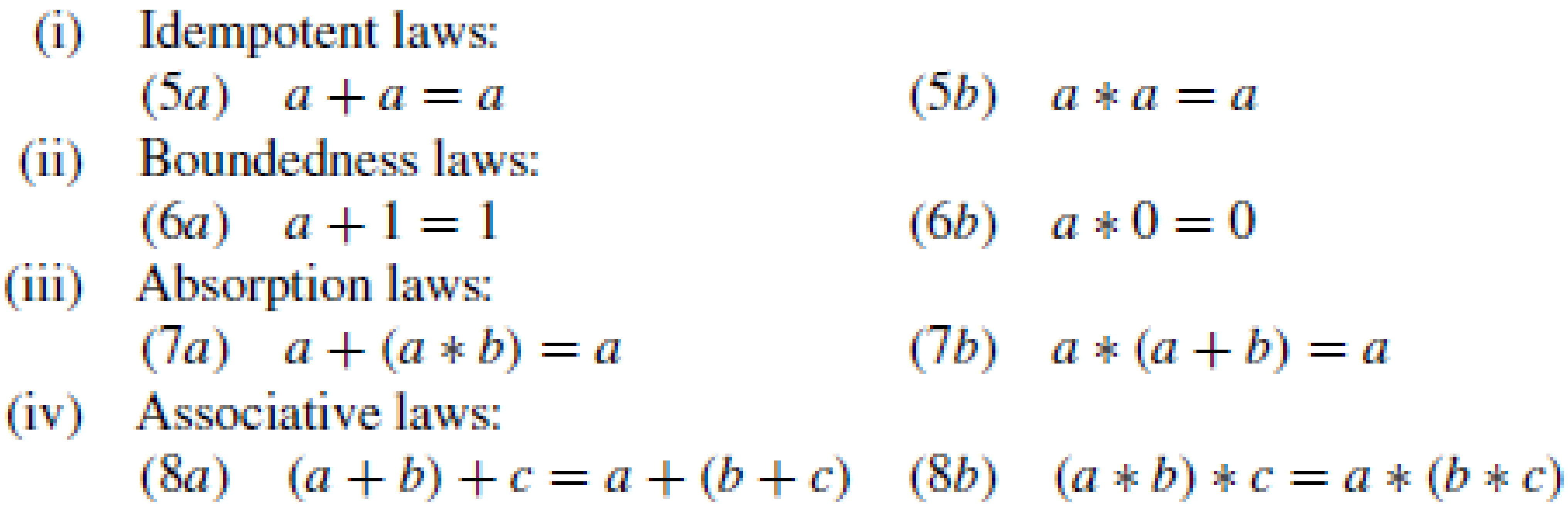
3. Then D_{70} is a Boolean algebra with 1 as the zero element and 70 the unit element.
4. Check if all the properties are satisfied.

Duality

1. The dual of any statement in a Boolean algebra B is the statement obtained by
   - interchanging the operations + and $\ast$, and
   - interchanging their identity elements 0 and 1 in the original statement.
2. For example,
   1. the dual of $(1 + a) \ast (b + 0) = b$ is
   2. $(0 \ast a) + (b \ast 1) = b$
3. Principle of Duality: The dual of any theorem in a Boolean algebra is also a theorem.

Basic Theorems

1. Let a, b, c be any elements in a Boolean algebra B.
2. Then, the following theorems hold:



(i) Idempotent laws:

(5a) $a + a = a$ (5b) $a * a = a$

(ii) Boundedness laws:

(6a) $a + 1 = 1$ (6b) $a * 0 = 0$

(iii) Absorption laws:

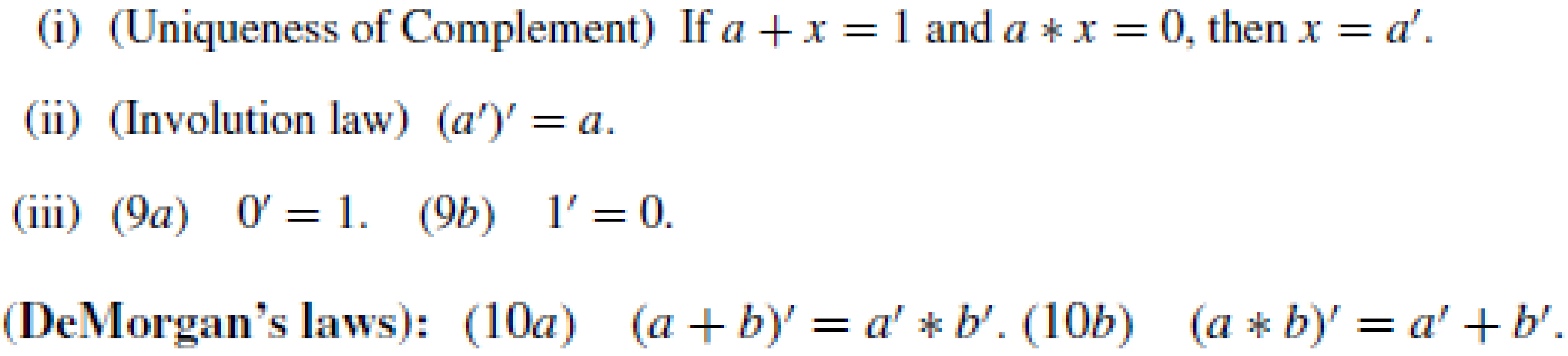
(7a) $a + (a * b) = a$ (7b) $a * (a + b) = a$

(iv) Associative laws:

(8a) $(a + b) + c = a + (b + c)$ (8b) $(a * b) * c = a * (b * c)$

Basic Theorems

1. Let a, b, c be any elements in a Boolean algebra B.
2. Then, the following theorems hold:



(i) (Uniqueness of Complement) If $a + x = 1$ and $a * x = 0$, then $x = a'$.

(ii) (Involution law) $(a')' = a$.

(iii) (9a) $0' = 1$. (9b) $1' = 0$.

(DeMorgan's laws): (10a) $(a + b)' = a' * b'$. (10b) $(a * b)' = a' + b'$.

Boolean Algebras as Lattices

1. Every Boolean algebra B satisfies the **associative, commutative, and absorption** laws and hence **is a lattice where + and $*$ are the join and meet** operations, respectively
2. With respect to this lattice,
   a) a+1 = 1 implies a ≤ 1
   b) And a $*$ 0 = 0 implies 0 ≤ a, for any element a ∈ B.
   c) Thus B is a bounded lattice.
3. Axioms [B2] – **distributive law** and [B4] – **complement law** show that B is also distributive and complemented.
4. So**, a Boolean algebra B is a bounded, distributive and complemented lattice**.

Boolean expression

1. Consider a set of variables (or letters or symbols), say $x_1, x_2, \ldots, x_n$.
2. A **Boolean expression E** in these variables, written $E(x_1, \ldots, x_n)$, is any variable or any expression built up from the variables using the Boolean operations +, ∗, and ‘.
3. Eg: Following are Boolean expressions in x, y, and z.

$$E_1 = (x + y'z)' + (xyz' + x'y)' \quad \text{and} \quad E_2 = ((xy'z' + y)' + x'z)'$$

Boolean expression

1. A **literal** is a variable or complemented variable, such as x, x’, y, y’ etc
2. A **fundamental product** is a literal or a product of two or more literals in which no two literals involve the same variable.
3. Eg. xz’ , xy’z, x, y’, x’yz are fundamental products,
4. But xyx’z and xyzy are not.
5. Any product of literals can be reduced to either 0 or a fundamental product,
6. e.g., xyx’z = 0 since xx’= 0 (complement law),
7. xyzy = xyz since yy = y (idempotent law).

Boolean expression

1. A fundamental product P1 is said to be contained in (or included in) another fundamental product P2 if the literals of P1 are also literals of P2.
2. Eg, x’z is contained in x’yz
3. But x’z is not contained in xy’z since x’ is not a literal of xy’z.
4. Note: If P1 is contained in P2, say P2 = P1 ∗ Q, then, by the absorption law,
   - P1 + P2 = P1 + P1 ∗ Q = P1
5. Thus, for instance, x’z + x’yz = x’z(1+y) = x’z

Sum-of-products Form For Boolean Algebras

1. A Boolean expression E is called a **sum-of-products expression** if E is a fundamental product or the sum of two or more fundamental products none of which is contained in another.
2. Let E be any Boolean expression. A sum-of-products form of E is an equivalent Boolean sum-of-products expression.
3. Example: **E1 = xz’ + y’z + xyz’**
4. E1 is a sum of products, but it is not a sum-of-products expression, because the product xz’ is contained in the product xyz’.
5. We can convert it into a sum-of-products expression:

$$E_1 = xz' + y'z + xyz' = xz' + xyz' + y'z = \underline{xz' + y'z}$$

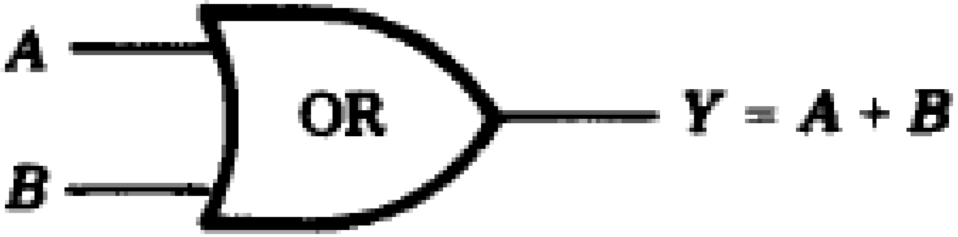
6. The underlined expression is the sum of products form for E1

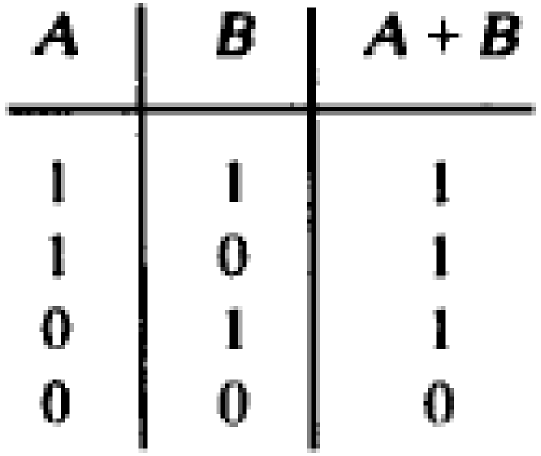
Logic Gates and Circuits

1. **Logic circuits** (also called logic networks) are structures which are built from certain elementary circuits called **logic gates**.
2. Each logic circuit may be viewed as a machine L which contains one or more **input devices** and **exactly one output** device.
3. Each input device in L sends a signal, a bit (binary digit), 0 or 1 to the circuit L
4. L processes the set of bits to yield an output bit.
5. Accordingly, an n-bit sequence may be assigned to each input device, and L processes the input sequences one bit at a time to produce an n-bit output sequence.
6. First we define the logic gates, and then we investigate the logic circuits.

Logic Gates

1. There are three basic logic gates which are described below.
2. Convention: lines entering the gate symbol from the left are input lines and the single line on the right is the output line.
3. **OR – Gate**:



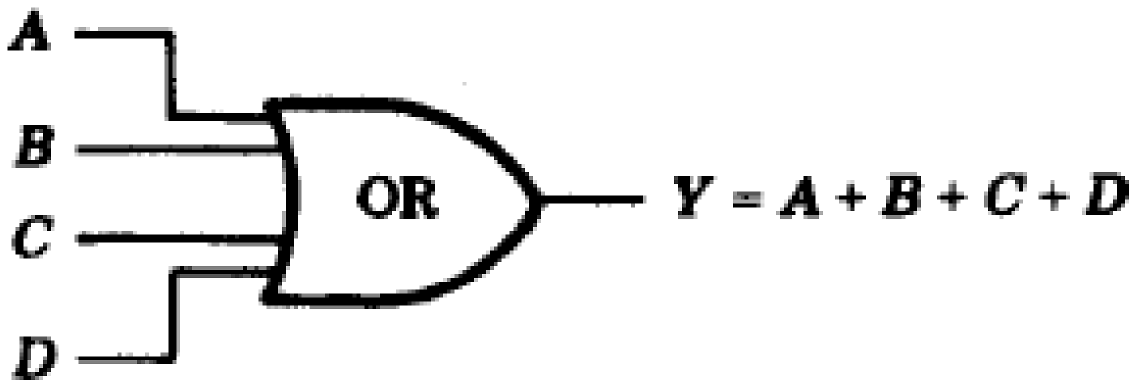


A	B	$A + B$
1	1	1
1	0	1
0	1	1
0	0	0

4. Inputs A and B and output Y = A + B where “addition” is defined by the “truth table”

Logic Gates

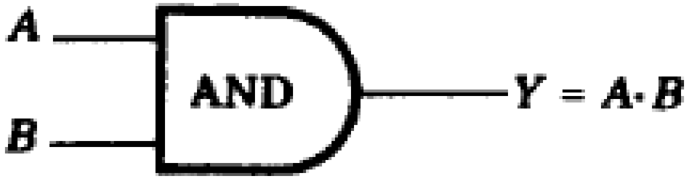
1. Such an OR gate may, have more than two inputs

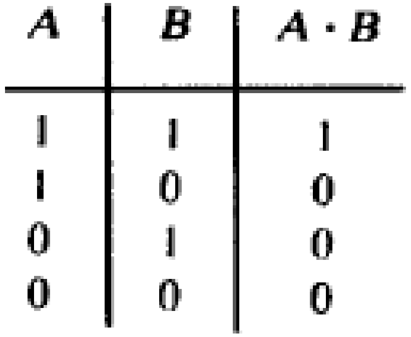


2. The output Y = 0 if and only if all the inputs are 0

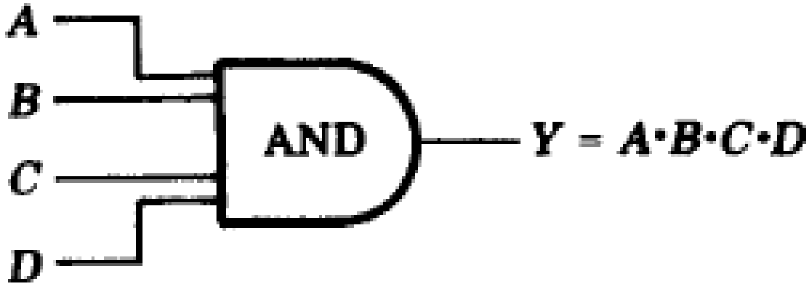
Logic Gates

1. **AND – Gate**:





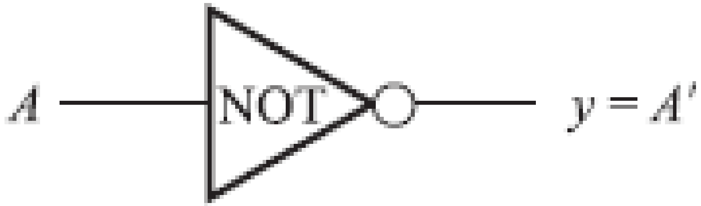
A	B	$A \cdot B$
1	1	1
1	0	0
0	1	0
0	0	0



2. Above figure shows an AND gate with inputs A and B and output Y = A · B (or simply Y = AB) where “multiplication” is defined by the “truth table”
3. Thus the output Y = 1 when inputs A = 1 and B = 1; otherwise Y = 0

Logic Gates

1. **NOT – Gate**:

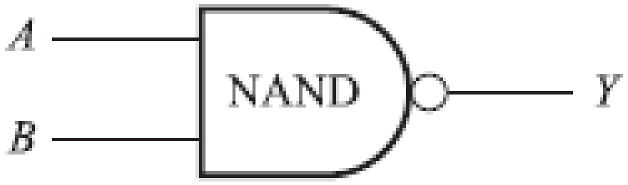


A	A'
1	0
0	1

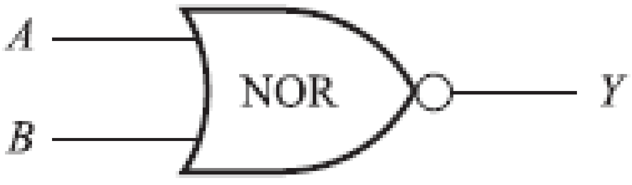
2. Figure shows a NOT gate, also called an inverter, with input A and output Y = A’
3. Where “inversion,” denoted by the prime, is defined by the “truth table”
4. A NOT gate can have only one input

Logic Gates

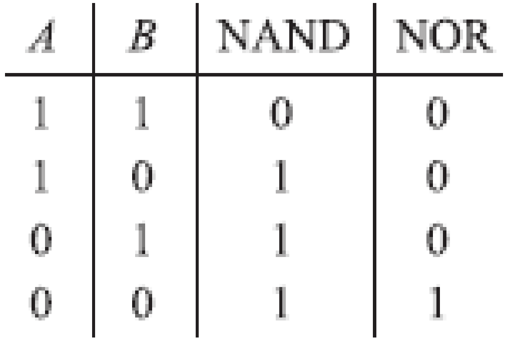
1. NAND – Gate: Equivalent to an AND gate followed by a NOT gate
2. NOR – Gate: equivalent to an OR gate followed by a NOT gate
3. Output of a NAND gate is 0 if and only if all the inputs are 1,
4. Output of a NOR gate is 1 if and only if all the inputs are 0.



(*a*) NAND gate



(*b*) NOR gate

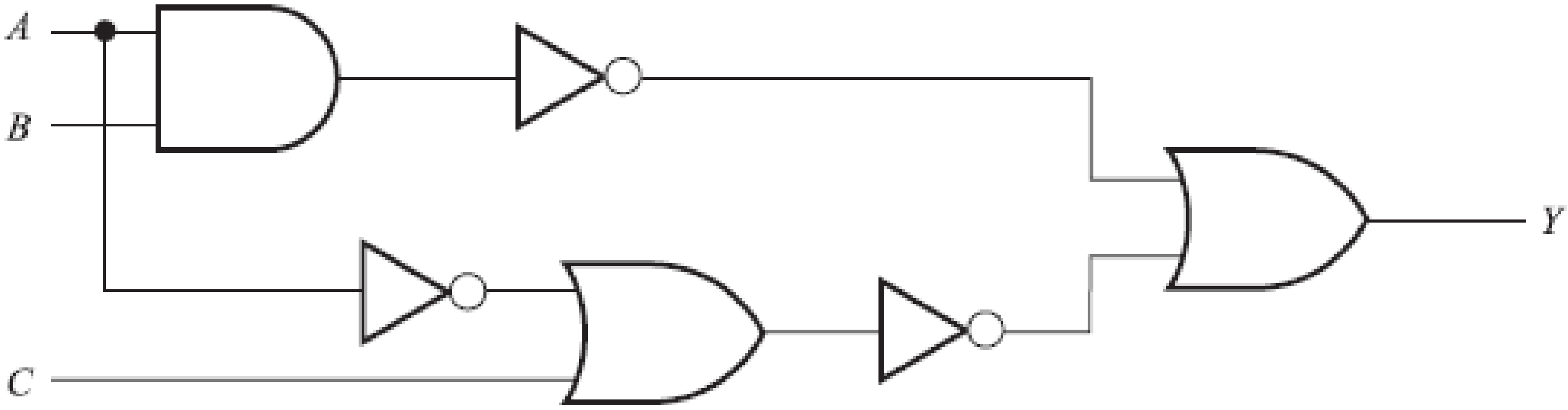


A	*B*	NAND	NOR
1	1	0	0
1	0	1	0
0	1	1	0
0	0	1	1

(*c*)

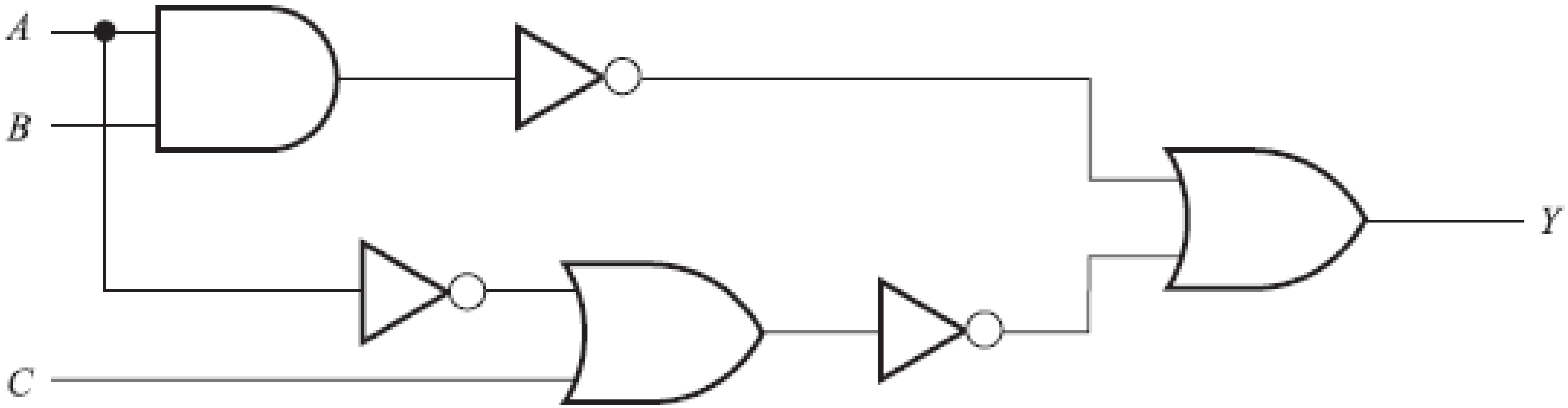
Logic Circuit

1. A logic circuit L is a well-formed structure whose elementary components are the above OR, AND, and NOT gates.
2. Below figure is an example of a logic circuit with inputs A, B, C and output Y .
3. A dot indicates a place where the input line splits so that its bit signal is sent in more than one direction.
4. What is Y?



Logic Gates

1. Working from left to right, we express Y in terms of the inputs A, B, C as follows.
2. The output of the AND gate is A · B, which is then negated to yield (A · B)’ .
3. The output of the lower OR gate is A’ + C, which is then negated to yield (A’ + C)’.
4. The output of the OR gate on the right, with inputs (A · B)’ and (A’ + C)’, gives us our desired representation, Y = (A · B)’ + (A’ + C)’

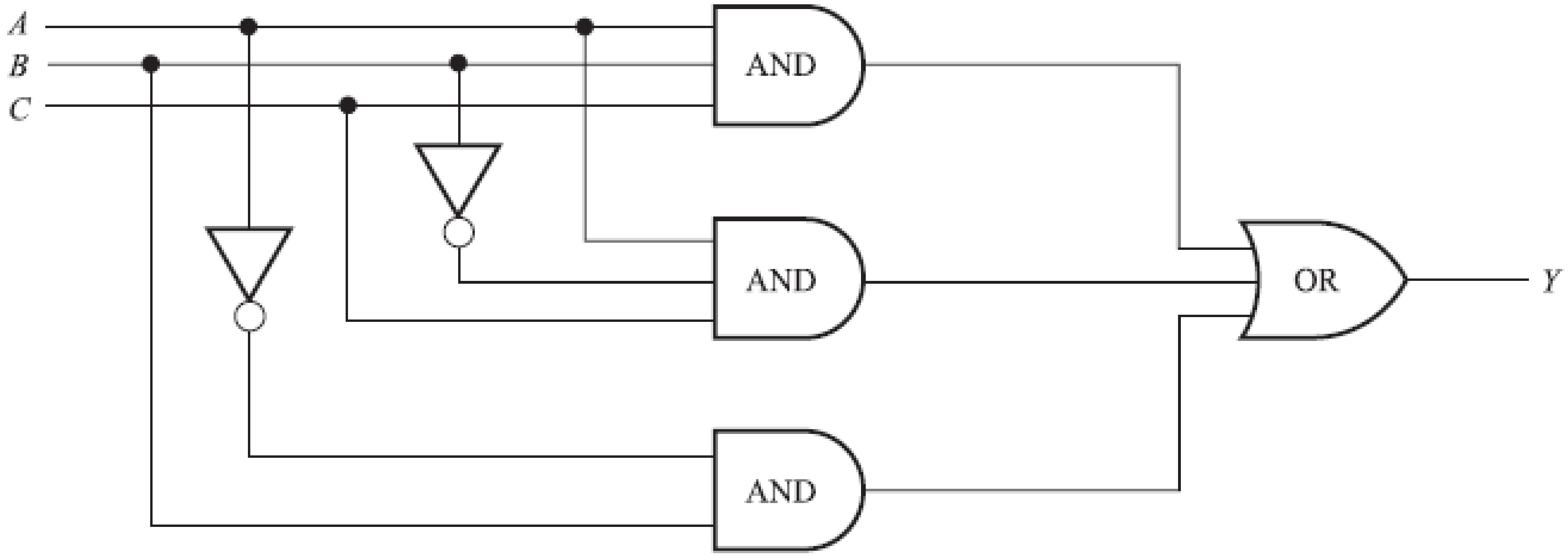


Logic Circuits as a Boolean Algebra

1. The truth tables for the OR, AND, and NOT gates are respectively identical to the truth tables for the propositions p V q (disjunction, “p or q”), p Λ q (conjunction, “p and q”), and ¬p (negation, “not p”)
2. Difference is that 1 and 0 are used instead of T and F.
3. Thus the **logic circuits satisfy the same laws as do propositions and hence they form a Boolean algebra**
4. Accordingly, all terms used with Boolean algebras, such as, complements, literals, fundamental products etc may also be used with our logic circuits

Example:

1. In the following AND-OR circuit with three inputs, A, B, C and output Y, what is the value of Y in terms of A, B and C, put in the form of a Boolean Expression?

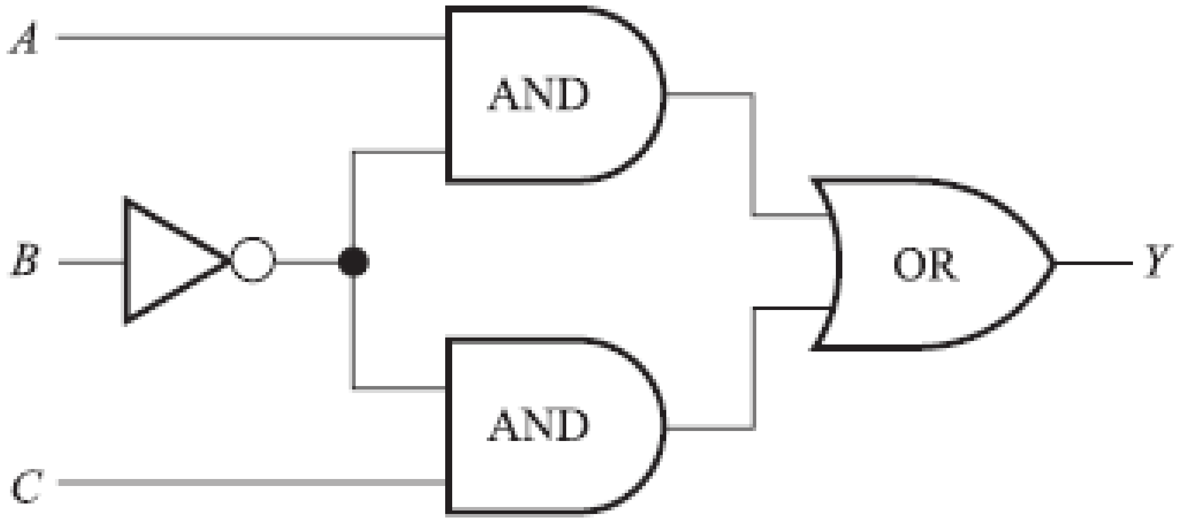


2. First find the output of each AND gate. Then the sum of the outputs of the AND gates is the output of the OR gate, which is the output Y of the circuit.

$$Y = A \cdot B \cdot C + A \cdot B' \cdot C + A' \cdot B$$

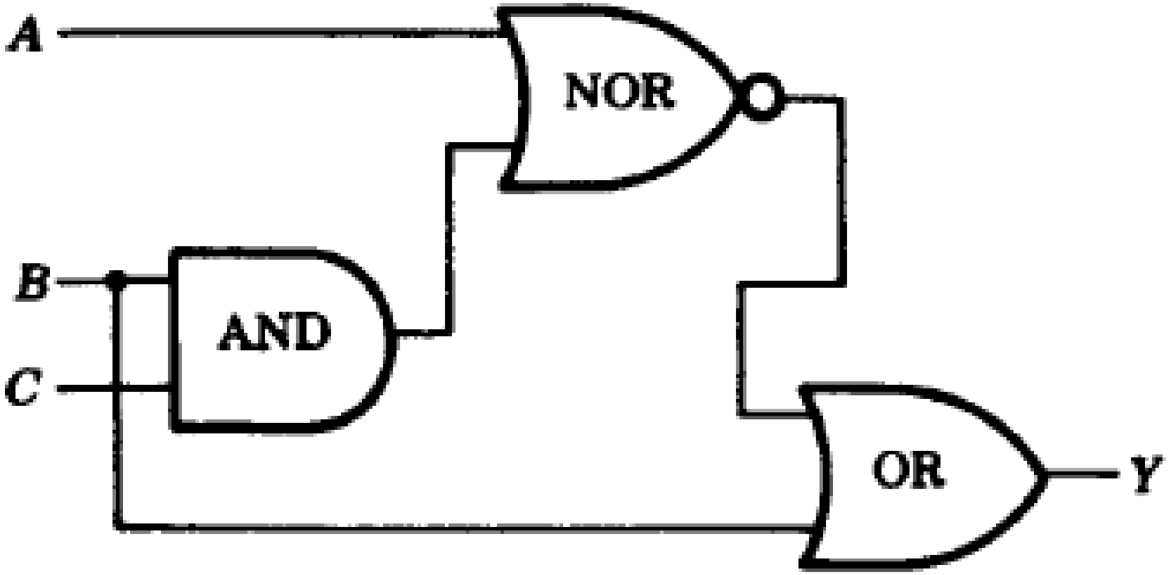
Example:

1. What is the output?
2. Output of first AND = AB’
3. Output of second AND = B’C
4. Y = AB’ + B’C



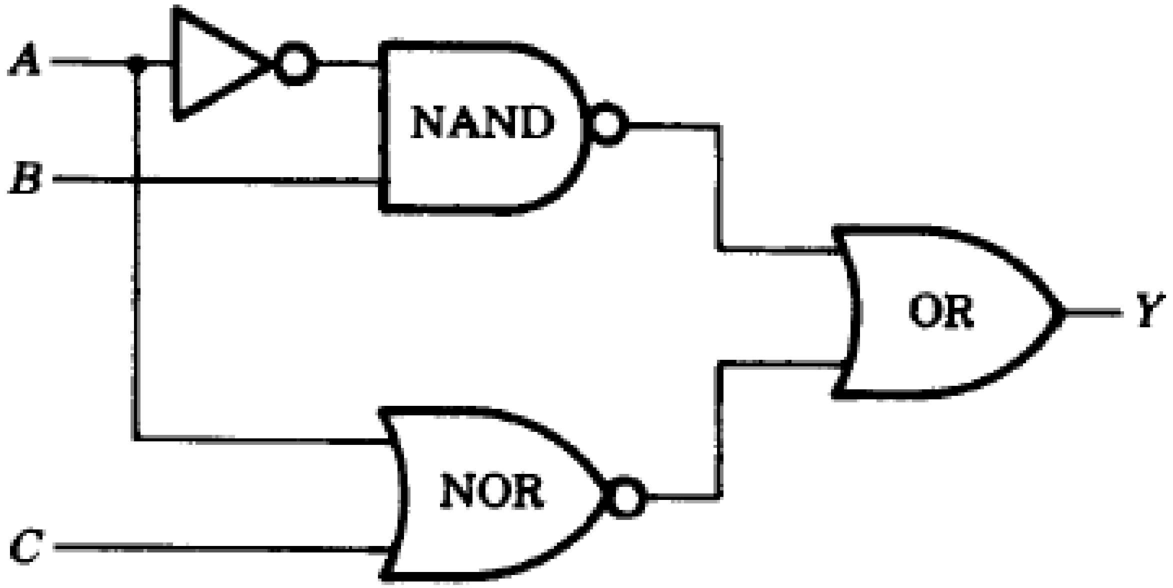
Example:

1. What is the output?
2. Output of AND = BC
3. Output of NOR = (A + BC)’
4. Y = AB’ + B’C
5. Input OR gate = (A + BC)’ and B
6. Thus, Y = (A + BC)’ + B



Example:

1. Try this: What is Y?


A
B
C
NAND
NOR
OR
Y