Fall 2021 MATH 1101 Discrete Structures
Lecture 4-5 Algebra of Circuits. Boolean Algebras.

Part 1. Logic Gates and Circuits. Algebra of Circuits.

Part 2. Boolean Algebras. Definition, Examples, Properties.

Part 3. Sum-of-Product (SoP) Form for Boolean Expressions. Prime Implicant and Minimal SoP (MSoP).

Part 4. Karnaugh Maps Method to Find MSoP.

Part 5. Truth Tables and Boolean Functions.

Part 1. Logic Gates and Circuits. Logic Circuits as a Boolean Algebra.

Introduction.

In Lectures 1 and 2-3 we introduced and discussed two algebras:

(1) Algebra of sets (Lecture 1)

(2) Algebra of propositions (Lectures 2-3)

Below we remind definitions of these algebras.

(1). Algebra of sets:

Definition 1. Algebra of sets is a structure which contains:

a) A family, say $\mathcal{F}$, of subsets of Universal Set U, including the empty set, $\emptyset$, and universal set U itself;

and three operations

b) Binary operations - Union ($A \cup B$), Intersection ($A \cap B$),

c) Unary operation - Complement: A→ $A' = U - A$

Denotation for Algebra of Sets F = < F, $\emptyset$, U, $\cup$, $\cap$, $'$ > ■

Elements (sets) of the family $\mathcal{F}$ under the operations of union, intersection, and complement satisfy various laws (identities), which are listed in Table 1. In fact, we formally state this as:

Theorem 1. Algebra of sets satisfies the laws in Table1. ■

Table 1. Laws of the algebra of sets

Idempotent laws:	(1a) A∪A=A	(1b) A∩A=A
Associative laws:	(2a) (A∪B)∪C=A∪(B∪C)	(2b) (A∩B)∩C=A∩(B∩C)
Commutative laws:	(3a) A∪B=B∪A	(3b) A∩B=B∩A
Distributive laws:	(4a) A∪(B∩C)=(A∪B)∩(A∪C)	(4b) A∩(B∪C)=(A∩B)∪(A∩C)
Identity laws:	(5a) A∪∅=A	(5b) A∩**U**=A
Boundedness Laws (=Domination Laws)	(6a) A∪**U**=**U**	(6b) A∩∅=∅
Involution law:	(7) (A′)′=A	
Complement laws:	(8a) A∪A′=**U**	(8b) A∩A′=∅
	(9a) **U**′=∅	(9b) ∅′=**U**
DeMorgan's laws:	(10a) (A∪B) ′=A′∩B′	(10b) (A∩B) ′=A′∪B′

Duality

The identities in Table 1 are arranged in pairs, as, for example, (2a) and (2b). We now consider the principle behind this arrangement. Suppose E is an equation of set algebra. The dual E^* of E is the equation obtained by replacing each occurrence of ∪, ∩, **U** and ∅ in E by ∩, ∪, ∅, and **U**, respectively. For example, the dual of

$$(\mathbf{U}\cap A)\cup(B\cap A)=A \text{ is } (\emptyset\cup A)\cap(B\cup A)=A$$

Observe that the pairs of laws in Table 1 are duals of each other. It is a fact of set algebra, called the *principle of duality*, that if any equation E is an identity then its dual E^* is also an identity.

(2). Algebra of Propositions

Definition 2. Algebra of propositions is a structure which contains:

a) A family, say $\mathcal{P}$, of propositions including Tautology (which is always true, **T**) and Contradiction (which is always fail, **F**),

and three operations

b) two binary operations: $p \vee q - disjunction\ or$ "or" operation and $p \wedge q - conjuction\ or$ "and" operation;

c) one unary operation: negation, denoted as $(\neg p)$ or (p'). ■

Elements of the family $\mathcal{P}$ under the operations disjunction, conjunction, and negation, satisfy various laws (identities), which are listed in Table 2. In fact, we formally state this as:

Theorem 2. Propositions satisfy the laws of Table 2. ■

Table 2. Laws of the algebra of propositions

Idempotent laws:	(1a) p∨p≡p	(1b) p∧p≡p
Associative laws:	(2a) (p∨q)∨r≡p∨(q∨r)	(2b) (p∧q)∧r≡p∧(q∧r)
Commutative laws:	(3a) p∨q≡q∨p	(3b) p∧q≡q∧p
Distributive laws:	(4a) p∨(q∧r)≡(p∨q)∧(p∨r)	(4b) p∧(q∨r)≡(p∧q)∨(p∧r)
Identity laws:	(5a) p∨F≡p	(5b) p∧T≡p
Boundedness Laws (=Domination Laws)	(6a) p∨T≡T	(6b) p∧F≡F
Involution law:	(7) ¬¬p≡p	
Complement laws:	(8a) p∨¬p≡T (9a) ¬T≡F	(8b) p∧¬p≡F (9b) ¬F≡T
DeMorgan's laws:	(10a) ¬(p∨q)≡¬p∧¬q	(10b) ¬(p∧q)≡¬p∨¬q

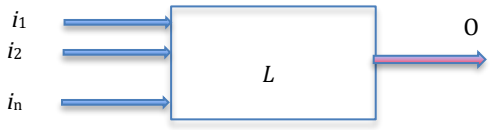
We observe that Tables 1 and 2 are completely similar.

Below we introduce the third algebra with similar properties.

Algebra of Circuits.

The aim of Part 1 is to introduce 3rd example of Algebras which is called **Algebra of Circuits**.

Logic circuits are structures which are built up from certain elementary circuits called *logic gates*. **Each logic circuit may be viewed as a machine L which contains one or more input devices (i_1, i_2, …, i_n) and exactly one output device (O).** Each input device sends a signal, specifically, a ***bit*, 0 or 1**, to the circuit L, and L processes the set of bits to yield an output bit.



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. First, we define the logic gates, and then we investigate the logic circuits.

Logic Gates

There are **three basic logic gates** (as *three basic logic operations* or as *three basic set operations*) which are described below. We adopt the convention that the lines entering the gate symbol from the left are input lines and the single line on the right is the output line.

(a) ***OR Gate:*** Figure 1(a) shows an OR gate with inputs A and B and output Y=A+B where "addition" is defined by the "truth table" in Figure 1(b). Thus, the output Y=0 only when inputs A=0 and B=0. Such an OR gate may, have more than two inputs. Figure 1(c) shows an OR gate with four inputs, A, B, C, D, and output Y=A+B+C+D. The output Y=0 if and only if all the inputs are 0.

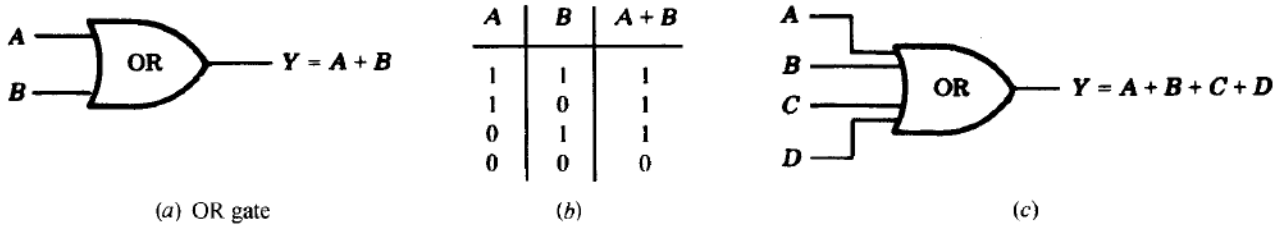


Figure 1

Suppose, for instance, the input data for the OR gate in Fig. 1(c) are the following 8-bit sequences: A=10000101, B=10100001, C=00100100, D=10010101

The OR gate only yields 0 when all input bits are 0. This occurs only in the 2nd, 5th, and 7th positions (reading from left to right). Thus, the output is the sequence Y=10110101.

(b) ***AND Gate:*** Figure 2(a) 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" in Fig. 2(b). Thus, the output Y=1 when inputs A=1 and B=1; otherwise Y=0. Such an AND gate may have more than two inputs. Figure 2(c) shows an AND gate with four inputs, A, B, C, D, and output Y =A·B·C·D. The output Y=1 if and only if all the inputs are 1.

Suppose, for instance, the input data for the AND gate in Figure 10(*c*) are the following 8-bit sequences: A=11100111, B=01111011, C=01110011, D=11101110

The AND gate only yields 1 when all input bits are 1. This occurs only in the 2nd, 3rd, and 7th positions. Thus, the output is the sequence Y=01100010.

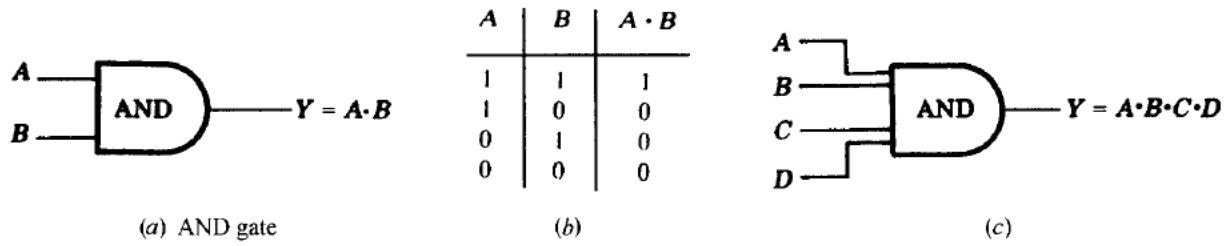
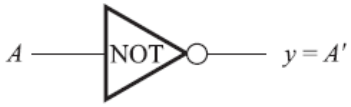


Figure 2

(c) ***NOT Gate:*** Figure 3(a) shows a NOT gate, also called an *inverter*, with input A and output Y=A′ where "inversion," denoted by the prime, is defined by the "truth table" in Figure 3(b). The value of the output Y=A′ is the opposite of the input A; that is, A′=1 when A=0 and A′=0 when A=1. We emphasize that a NOT gate can have only one input, whereas the OR and AND gates may have two or more inputs.



(a) NOT gate

A	A'
1	0
0	1

(b)

Figure 3

Suppose, for instance, a NOT gate is asked to process the following three sequences:

A_1=110001, A_2=10001111, A_3=101100111000

The NOT gate changes 0 to 1 and 1 to 0. Thus

A_1'=001110, A_2'=01110000, A_3'=010011000111

are the three corresponding outputs.

Logic Circuits

A **logic circuit L** is a well-formed structure whose elementary components are the above OR, AND, and NOT gates. Figure 4 is an example of a logic circuit with inputs A, B, and C and output Y. A dot indicates a place where the input line splits so that its bit signal is sent in more than one direction. Working from left to right, we express Y in terms of the inputs A, B, C as follows. The output of the AND gate is AB, which is then negated to yield (AB)′. The output of the lower OR gate is A′+C, which is then negated to yield (A′+C)′. The output of the OR gate on the right, with inputs (AB)′ and (A′+C)′, yields desired representation, that is, Y=(AB)′+(A′+C)′.

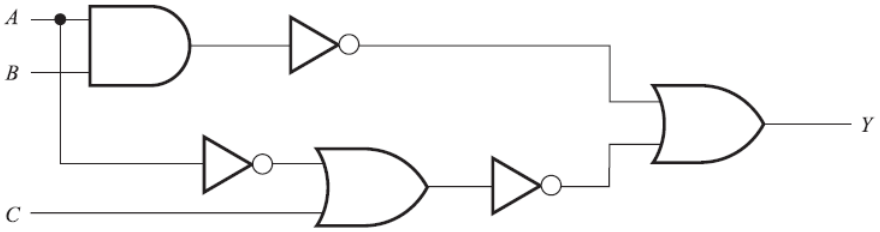


Figure 4

Logic Circuits as an Algebra

Observe that the truth tables for the OR, AND, and NOT gates are respectively identical to the truth tables for the propositions p∨q (disjunction, “p or q”), p∧q (conjunction, “p and q”), and ¬p (negation, “not p”). The only difference is that 1 and 0 are used instead of T and F. Thus, the logic circuits satisfy the same laws as do propositions and hence they form an algebra. We state this result formally.

Theorem 3. Logic circuits form an Algebra of Circuits with properties like the properties in Tables 1 or 2. ■

Exercise. Provide strong definition of Algebra of Circuits (like Algebra of Propositions).

AND-OR Circuits.

The following special form of logic circuit, called **AND-OR circuit**, will be very useful in future considerations. Such a circuit L has several inputs, where:

(1) Some of the inputs or their complements are fed into each AND gate.

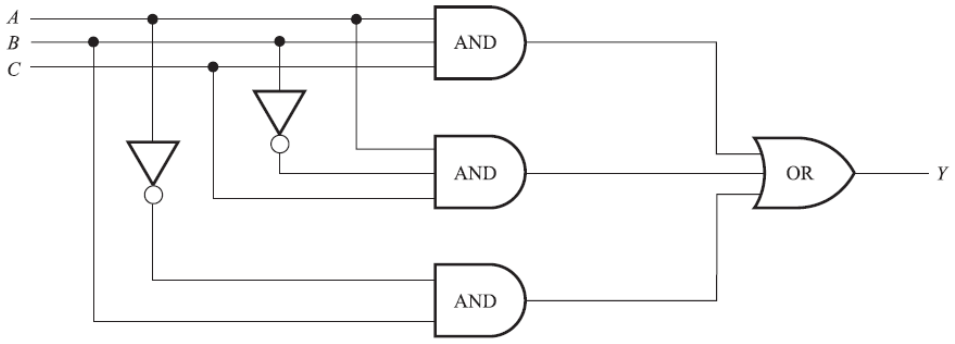
(2) The outputs of all the AND gates are fed into a single OR gate.

(3) The output of the OR gate is the output for the circuit L. The following illustrates this type of a logic circuit.

EXAMPLE 1. Figure 5 is a typical AND-OR circuit with three inputs, A, B, C and output Y. We can easily express Y as an expression in the inputs A, B, C as follows. First, we find the output of each AND gate:

(a) The inputs of the first AND gate are A, B, C; hence ABC is the output.
(b) The inputs of the second AND gate are A, B′, C; hence AB′C is the output.
(c) The inputs of the third AND gate are A′ and B; hence A′B is the output.

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.
Thus: Y=ABC+AB′C+A′B



■

Figure 5

NAND and NOR Gates

There are two additional gates which are equivalent to combinations of the above basic gates.

(a) A NAND gate, pictured in Fig.6(a), is equivalent to an AND gate followed by a NOT gate.

(b) A NOR gate, pictured in Fig.6(b), is equivalent to an OR gate followed by a NOT gate.

The truth tables for these gates (using two inputs A and B) appear in Fig. 6(c).

The NAND and NOR gates can have two or more inputs just like the corresponding AND and OR gates. Furthermore, the output of a NAND gate is 0 if and only if all the inputs are 1, and the 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*)

Figure 6

Observe that the only difference between the AND and NAND gates between the OR and NOR gates is that the NAND and NOR gates are each followed by a circle. Some texts also use such a small circle to indicate a complement before a gate. For example, the expressions corresponding to two logic circuits in Figure 7 are as follows:

(a) Y=(A′B)′, (b) Y=(A′+B′+C)′

(*a*)

(*b*)

Figure 7

EXAMPLE 2. Draw the logic circuit L with inputs A, B, and C and output Y which corresponds to the expression $Y=AB'C+ABC'+AB'C'$.

Solution.

These are sum-of-products expressions. Thus, L will be an AND-OR circuit which has an AND gate for each product and one OR gate for the sum. The required circuits are in Figure 8.

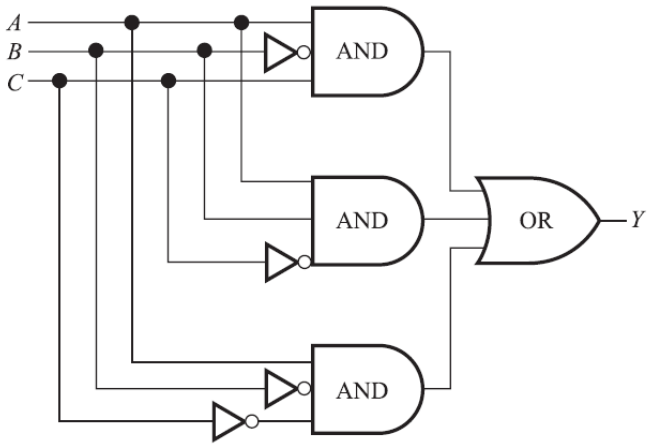


Figure 8

Part 2. Boolean Algebras. Definition, Examples, Properties.

Starting from this point we generalize our observations about three algebras above and introduce a general definition of so-called Boolean Algebra.

Based on general approach in Mathematics we try:

- **to choose some properties (as less as possible) in Table 1 (or 2) as axioms of a new structure and**
- **to prove other properties (like the remain properties of Tables 1 or 2) as theorems.**

BASIC DEFINITIONS.

Definition 3. Let B be a nonempty set with two binary operations + and $*$, a unary operation $'$, and two distinct elements **0** and **1.** Then B is called a *Boolean algebra* if the following axioms hold where a, b, c are any elements in B:

[$\mathbf{B_1}$] Commutative laws:

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

[$\mathbf{B_2}$] Distributive laws:

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

[$\mathbf{B_3}$] Identity laws:

(3a) $a+\mathbf{0}=a$ (3b) $a*\mathbf{1}=a$

[$\mathbf{B_4}$] Complement laws:

(4a) $a+a'=\mathbf{1}$ (4b) $a*a'=\mathbf{0}$ ■

We will sometimes designate a Boolean algebra by $\langle B, +, *, ', 0, 1\rangle$ when we want to emphasize its **six parts**. We say 0 is the *zero* element, l, is the *unit* element, and a' is the *complement* of a. We will usually drop the symbol $*$ and use juxtaposition instead. Then (2b) is written $a(b+c)=ab+ac$ which is the familiar algebraic identity of rings and fields. However, (2a) becomes $a+bc=(a+b)(a+c)$, which is certainly not a usual identity in algebra.

The operations +, $*$, and $'$ are called sum, product, and complement, respectively. We adopt the usual convention that, unless we are guided by parentheses, $'$ has precedence over $*$, and $*$ has

precedence over +.

For example, a+b∗c means a+(b∗c) and not (a+b)∗c; a∗b′ means a∗(b′) and not (a∗b)′ . Of course, when a+b∗c is written a+bc then the meaning is clear.

Examples of Boolean Algebras

(a) Algebra of Sets.
(b) Algebra of Propositions.
(c) Algebra of Circuits.
(d) Let **B**={0,1}, the set of *bits* (binary digits), with the binary operations of + and ∗ and the unary operation ′ defined by Table 3 below. Then **B** is a Boolean algebra.
(Note that ′ operation simply changes the bit, i.e., 1′=0 and 0′=1)

Table 3

+	1	0
1	1	1
0	1	0

∗	1	0
1	1	0
0	0	0

′	1	0
	0	1

(e) Let $\mathbf{B^n}=\mathbf{B}\times\mathbf{B}\times\cdots\times\mathbf{B}$ (n factors) where the operations of +, ∗, and ′ are defined componentwise using Table 3 above. For notational convenience, we write the elements of $\mathbf{B^n}$ as n-bit sequences without commas, e.g., x=110011 and y=111000 belong to $\mathbf{B^6}$. Hence x+y=111011, xy=110000, x′=001100. Then $\mathbf{B^n}$ is a Boolean algebra. Here 0=000⋯0 is the zero element, and 1=111⋯1 is the unit element. We note that $\mathbf{B^n}$ has 2^n elements. ■

Subalgebras, Isomorphic Boolean Algebras

Suppose C is a nonempty subset of a Boolean algebra B. We say C is a *subalgebra* of B if C itself is a Boolean algebra (with respect to the operations of B). We note that C is a subalgebra of B if and only if C is closed under the three operations of B, i.e., +, ∗, and ′.

Two Boolean algebras B and B′ are said to be *isomorphic* if there is a one-to-one correspondence f:B→B′ which preserves the three operations, i.e., such that, for any elements, a, b in B,

$$f(a+b)=f(a)+f(b), \qquad f(a*b)=f(a)*f(b) \qquad \text{and} \qquad f(a')=f(a)'$$

Duality

The *dual* of any statement in a Boolean algebra B is the statement obtained by interchanging the operations "+" and "∗" and interchanging their identity elements **0** and **1** in the original statement.

EXAMPLE 3.

The dual of the expression (1+a)∗(b+0)=b is (0∗a)+(b∗1)=b

Observe the symmetry in the axioms of a Boolean algebra B. That is, the dual of the set of axioms of B is the same as the original set of axioms. Accordingly, the principle of duality holds in B. ■

Theorem 4. (Principle of Duality): The dual of any theorem in a Boolean algebra is also a theorem.

Proof. (Exercise). ■

In other words, if any statement is a consequence of the axioms of a Boolean algebra, then the dual is also a consequence of those axioms since the dual statement can be proven by using the dual of each step of the proof of the original statement.

Basic Theorems

Using the axioms [$\mathbf{B_1}$] through [$\mathbf{B_4}$], we establish (prove) several very important properties of Boolean algebras. Some of these properties are already formalized in appropriate Tables for

particular algebras (Algebra of Sets, Table 1, Lecture 1, and Algebra of Propositions, Table 14 Lectures 2-3). Specifically, we prove the following theorem.

Theorem 5. Let **B** be a Boolean algebra and let a, b, c be any elements in B. Then the following statements are true:

. (i) Idempotent laws:
(B5a) a+a=a (B5b) aa=a

. (ii) Boundedness laws (=Domination Laws):
(B6a) a+1=1 (B6b) a0=0

. (iii) Absorption laws:
(B7a) a+(ab)=a (B7b) a(a+b)=a

. (iv) Associative laws:
(B8a) (a+b)+c=a+(b+c) (B8b) (ab)c=a(bc)

. (v) Uniqueness of Complement:
(B9) If a+x=1 and ax=0, then x=a′

. (vi) Involution law:
(B10) (a′)′=a

. (vii) (B11a) 0′=1 (B11b) 1′=0

. (viii) De Morgan's laws:
(B12a) (a+b)′=a′b′ (B12b) (ab)′=a′+b′.

Proof.

Proof method: we prove (BNa) (or (BNb)) and then use the duality principle to infer that BNb (BNa) is also true. In all considerations below we use juxtaposition instead of operation $*$.

Idempotent laws.

(B5b). a=(by [**B3**])=a1=(by [**B4**])=a(a+a′)=(by [**B2**])=(aa)+(aa′)=(by [**B4**])=(aa)+0=(by [**B3**])=aa

(B5a). Follows from (B5b) and duality.

Boundedness laws.

(B6b). a0=(by [**B3**])=(a0)+0=(by [**B4**])=(a0)+(aa′)=(by [**B2**])=a(0+a′)=(by [**B3**])=aa′=(by [**B4**])=0

(B6a). Follows from (B6b) and duality.

Absorption laws:

(B7b). a(a+b)=(by [**B3**])=(a+0)(a+b)=(by [**B2**])=a+(0b)=(by **(B6b)**)=a+0=a

(B7a). Follows from (B7b) and duality.

Associative laws:

(B8b). Let L=(ab)c and R=a(bc). We need to prove that L=R.

Step 1. We first prove that a+L=a+R. Using the absorption laws in the last two steps,

a+L=a+((ab)c)=(by [**B2**])=(a+(ab))(a+c)=(by **(B7a)**)=a(a+c)=(by **(B7b)**)=a

Also, using the absorption law in the last step,

a+R=a+(a(bc))=(by [**B2**])=(a+a)(a+(bc))=(by **(7a)**)=a(a+(bc))=(by **(7b)**)=a

Thus a+L=a+R.

Step 2. Next we show that a′+L=a′+R. We have,

a′+L=a′+((ab)c)=(by [**B2**])=(a′+(ab))(a′+c)=(by [**B2**])=((a′+a)(a′+b))(a′+c)=
=(by [**B4**])=(1(a′+b))(a′+c)=(by [**B3**])=(a′+b)(a′+c)=(by [**B2**])=a+(bc)

Also,

a′+R=a′+(a(bc))=(by [**B2**])=(a′+a)(a′+(bc))=(by [**B4**])=1(a+(bc))=(by [**B3**])=a+(bc)

Thus a′+L=a′+R. Consequently,

(ab)c=L=(by [**B3**])=0+L=(by [**B4**])=(aa′)+L=(by [**B2**])=(a+L)(a′+L)=as shown above
=(a+R)(a′+R)=(by [**B2**])=(aa′)+R=(by [**B4**])=0+R=(by [**B3**])=R=**a(bc),** as claimed.

(B8a). Follows from (B8b) and duality.

Uniqueness of Complement:

(B9). We have: a′=(by [**B3**])=a′+0=(by hypothesis)=a′+ax=(by [**B2**])=(a′+a)(a′+x)=
=(by [**B4**])=1(a′+x)=(by [**B3**])=a′+x=x+a′.

Also,

x=(by [**B3**])=x+0=(by [**B4**])=x+(aa′)=(by [**B2**])=(x+a)(x+a′)=
=(by hypothesis)=1(x+a′)=(by [**B3**])=x+a′

Right-hand sides of both chains are equal each other therefore the same is true for the left-hand sides. Hence, x=a′. Uniqueness of Complement is proved.

Involution law (=Double Complement Law):

(B10). By definition of complement, (by [**B4**]) a+a′=1 and aa′=0. By commutativity, (by [**B1**]) a′+a=1 and a′a=0. By Uniqueness of Complement, (by [**B9**]), a=(a′)′=a″, as claimed.

(B11). By boundedness law (by [**B6a]**), 0+1=1, and by identity axiom (by [**B3b]**), 0∗1=0. By uniqueness of complement, (by [**B9**]), 1 is the complement of 0, that is, 1=0′.

By duality, 0=1′.

Theorem 7. (De Morgan's laws**):**

(12a) (a+b)′=a′b′ (12b) (ab)′=a′+b′.

Proof. Exercise.

Theorem 8.

The following four statements are equivalent:
(*1*) a+b=b, (*2*) ab=a, (*3*) a′+b=1, (*4*) ab′=0
Proof. Exercise.

PART 3. SUM-of-PRODUCT (SoP) FORM FOR BOOLEAN EXPRESSIONS. PRIME IMPLICANT AND MINIMAL SoP (MSoP).

Goal of the Part 3 is to show that any expression in a Boolean Algebra (BA) can be reduced into a special form which is called **Sum-of-product (SoP)** form for the Boolean Expressions (**DNF – Disjunctive Normal Form** in other terminology).
An expression in BA is a formula over elements of BA which uses operations and parentheses.

SoP form for a Boolean expression.

Definition 4. Consider a set of variables, say x_1, x_2, ..., x_n. A *Boolean expression E* in these variables, sometimes written $E(x_1, x_2, ..., x_n)$, is any variable or any expression built up from the variables using the Boolean operations +, $*$, and $'$. ■

EXAMPLE 4.
$E_1=(x+y'z)'+(xyz'+x'y)'$, $E_2=((xy'z'+y)'+x'z)'$ are Boolean expressions in x, y, and z. ■

Definition 5. A *literal* is a variable or complemented variable, such as x, x′, y, y′, and so on. A *fundamental product* is a literal or a product of two or more literals in which no two literals involve the same variable. ■

EXAMPLE 5.
Products xz′, xy′z, x, y′, x′yz are fundamental products, but xyx′z and xyzy are not. Note that any product of literals can be reduced to either 0 or a fundamental product, e.g., xyx′z=0 since xx′=0 (complement law), and xyzy=xyz since yy=y (idempotent law). ■

Definition 6. A fundamental product P_1 is said to be *contained in* (or *included in*) another fundamental product P_2 if the literals of P_1 are also literals of P_2. ■

EXAMPLE 6.
x′z is contained in x′zy, but x′z is not contained in xy′z since x′ is not a literal of xy′z.

Observe that if P_1 is contained in P_2, say $P_2=P_1*Q$, then, by the absorption law, $P_1+P_2=P_1+P_1*Q=P_1*(1+Q)=P_1*1=P_1$. According to the last observation we have: x′z+x′yz=x′z

There is an obvious interpretation of the absorption law in terms of Algebra of Sets. Let x′=A, z=B, y=C, here A, B, C are subsets of universal set **U**. Then
$P_1=A\cap B$ and $P2= A\cap B\cap C \Rightarrow P_2\subset P_1 \Rightarrow P_1\cup P_2=P_1$ x′z+ x′zy= x′z ■

Definition 7: A Boolean expression E is called a *Sum-of-Products (*SoP or DNF*)* expression if E is a fundamental product or the sum of two or more fundamental products none of which is contained in another. ■

Definition 8: Let E be any Boolean expression. A *sum-of-products form* of E, SoP(E) or DNF(E), is an equivalent Boolean sum-of-products expression. ■

EXAMPLE 7. Let $E_1=xz'+y'z+xyz'$ and $E_2=xz'+x'yz'+xy'z$.
Although E_1 is a sum of products, it is not a SoP expression. Specifically, the product xz′ is contained in the product xyz′. However, by the absorption law, E_1 can be expressed as

$E_1=xz'+y'z+xyz'=xz'+xyz'+y'z=xz'+y'z$

This yields a SoP form for E_1. The second expression E_2 is already a SoP expression. ■

Question. How to find an SoP form for a given Boolean Expression?

Algorithm for Finding Sum-of-Products Forms.

Algorithm 1 below is a four-step algorithm which uses the Boolean algebra laws to transform any Boolean expression into an equivalent SoP expression.

Algorithm 1. ***Input*** is a Boolean expression E. ***Output*** is a SoP expression equivalent to E.

Step 1. Use DeMorgan's laws and involution to move the **complement** operation into any parenthesis until finally the complement operation only applies to variables. Then E will consist only sums and products of literals.

Step 2. Use the distributivity operation to next transform E into a sum of products.

Step 3. Use the commutative, idempotent and complement laws to transform each **product** in E into **0** or **fundamental product**.

Step 4. Use the absorption and identity laws to finally transform E into an **SoP form**. ■

EXAMPLE 8. Suppose Algorithm 1 is applied to the expression E=((xy)′z)′((x′+z)*(y′+z))′. That is, E is the input expression of Algorithm 1. We want to get SoP(E) which is equivalent to E.

Solution.

Step 1. Using De Morgan's laws and involution, we obtain

$$E = ((xy)''+z')((x'+z)'+(y'+z')') = (xy+z')(x''z'+y''z'')=(xy+z')(xz'+yz)$$

E now consists only of sums and products of literals.

Step 2. Using the distributive laws, we obtain

$$E=xyxz'+xyyz+xz'z'+yzz'$$

E now is a sum of products (but not SoP).

Step 3. Using the commutative, idempotent, and complement laws, we obtain

$$E=xyz'+xyz+xz'+0$$

Each term in E is a fundamental product or 0.

Step 4. The product xz′ is contained in xyz′; hence, by the absorption law,

$$xz'+xyz'=xz'$$

Thus, we may delete xyz′ from the sum. Also, by the identity law for 0, we may delete 0 from the sum. Accordingly,

$$E=xyz+xz'$$

E is now represented by a SoP expression. ■

Complete Sum-of-Products Forms (CSoP or CDNF)

Definition 9: A Boolean expression $E=E(x_1, x_2, ..., x_n)$ is said to be a *complete sum-of-products* expression (CSoP) if E is a SoP expression where each product P involves **all the n variables**. Such a fundamental product P which involves all the variables is called a *minterm*. ■

For instance, if n=3 there are 8 minterms:

xyz	xyz′	xy′z	xy′z′
x′yz	x′yz′	x′y′z	x′y′z′

Clear that there is a maximum of 2^n such minterms for n variables.

The following theorem applies.

Theorem 9: Every nonzero Boolean expression $E=E(x_1, x_2, ..., x_n)$ is equivalent to a complete sum-of-products expression and such a representation is unique. ■

The above unique representation of E is called the *complete sum-of-products form* of E, CSoP(E). Algorithm 1 tells us how to transform E into a SoP form. Algorithm 2 below transforms a SoP form into a CSoP form.

Algorithm 2. ***Input*** is a Boolean SoP expression $E=E(x_1, x_2, \ldots x_n)$. ***Output*** is a CSoP expression equivalent to E.

> ***Step 1.*** Find a summand (product) P in SoP expression E which does not involve the variable x_i and then multiply P by x_i+x_i', deleting any repeated products. (This is possible because $1=x_i+x_i'$, and P+P=P).
>
> ***Step 2.*** Repeat Step 1 until every product P in E is a minterm, that is product P involves all the variables. ■

EXAMPLE 9. Express E(x, y, z)=x(y′z)′ into its CSoP form.

(a) Apply Algorithm 1 to E to obtain a SoP expression:

E=x(y′z)′=x(y+z′)=xy+xz′

(b) Now apply Algorithm 2 to obtain the CSoP form:

E=xy+xz′=xy1+x1z′=xy(z+z′)+x(y+y′)z′=xyz+xyz′+xyz′+xy′z′=xyz+xyz′+xy′z′

Prime Implicants and Minimal SoP (MSoP) for Boolean Expressions.

There are many ways of representing the same Boolean expression E. Here we define and investigate a **minimal sum-of-products form for E (=MSoP)**.

We must also define and investigate prime implicants of E since the minimal sum-of-products involves such prime implicants.

Minimal Sum-of-Products (MSoP).

Consider a Boolean SoP expression E. Let E_L denote the number of literals in E (counted according to multiplicity), and let E_S denote the number of summands in E. For instance, if $E=xyz+x'y't+xy'z't+x'yzt$ then $E_L=3+3+4+4=14$ and $E_S=4$.

Definition 10: Suppose E and F are equivalent Boolean SoP expressions. We say E is *simpler* than F if:
(i) $E_L<F_L$ and $E_S\leq F_S$, or (ii) $E_L\leq F_L$ and $E_S<F_S$
We say E is *minimal* if there is no equivalent SoP, say F, which is simpler than E. ■

Note. There can be more than one equivalent minimal SoP expressions.

Prime Implicants

Definition 11: A fundamental product P is called a *prime implicant* of a Boolean expression E if P+E=E but no other fundamental product contained in P has this property ■

For instance, suppose $E=xy'+xyz'+x'yz'$. It is easy to check that: $xz'+E=E$ but $x+E\neq E$ and $z'+E\neq E$. Thus xz' is a prime implicant of E. Standard way to check that A+B=C+D (here A, B, C, D are Boolean Expressions) is to reduce LHS and RHS to the CSoP form and then compare results. Apply this method to our example.

$xz'+E=xz'+(xy'+xyz'+x'yz')=x1z'+x1y'+ xyz'+x'yz'=x(y+y')z'+ xy'(z+ z')+xyz'+x'yz'=$

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

$E=xy'+xyz'+x'yz'=xy'(z+z')+xyz'+x'yz'=xy'z+xy'z'+xyz'+x'yz'$.

Therefore, $xz'+E=E$. By the similar way we can check that $x+E\neq E$ and $z'+E\neq E$.

The following theorem applies.

Theorem 10. A minimal sum-of-products form for a Boolean expression E is a sum of prime implicants of E. ■

Question. How to find prime implicants?

Below we demonstrate Karnaugh Maps Method which is a Geometric Method to find prime implicants and a minimal SoP for a given Boolean expression E.

PART 4. KARNAUGH MAPS METHOD TO FIND MSoP.

Karnaugh maps, where minterms involving the same variables are represented by squares, are pictorial devices for finding prime implicants and minimal forms for Boolean expressions involving at most six variables.

We will only treat the cases of two, three, and four variables. In the context of Karnaugh maps, we will sometimes use the terms “squares” and “minterm” interchangeably.

Recall that a minterm is a fundamental product which involves all the variables, and that a CSoP expression is a sum of minterms.

First, we need to define the notion of adjacent products.

Definition 12: Two fundamental products P_1 and P_2 are said to be adjacent if P_1 and P_2 have the same variables and if they differ in exactly one literal. ■

Thus, there must be an uncomplemented variable in one product and complemented in the other. In particular, the sum of two such adjacent products will be a fundamental product with one less literal.

EXAMPLE 10. Find the sum of adjacent products P_1 and P_2.

Solution.

(a) Let $P_1=xyz'$ and $P_2=xy'z'$. Then $P_1+P_2=xyz'+xy'z'=xz'(y+y')=xz'(1)=xz'$

(b) Let $P_1=x'yzt$ and $P_2=x'yz't$. Then $P_1+P_2=x'yzt+x'yz't=x'yt(z+z')=x'yt(1)=x'yt$

(c) Let $P_1=x'yzt$ and $P_2=xyz't$. Here P_1 and P_2 are not adjacent since they differ in two literals.

(d) $P_1=xyz'$ and $P_2=xyzt$. Here P_1 and P_2 are not adjacent since they have different variables. Thus, they will not appear as squares in the same Karnaugh map. ■

Case of Two Variables.

The Karnaugh map corresponding to Boolean expressions E=E(x, y) with two variables x and y may be viewed as (2x2) table (matrix) as shown in Figure 9 below.

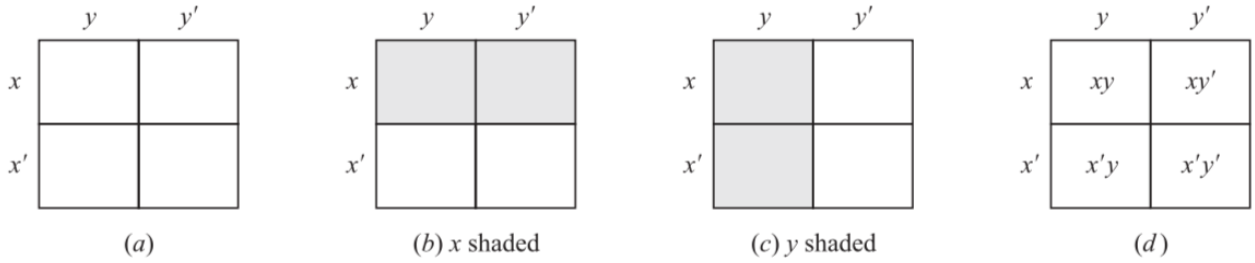


Figure 9

Here the 1st row represents the variable x (x-shaded, Figure 9b), the 2nd row – variable x′, the 1st column – variable y (y-shaded, Figure 9c), and the 2nd column – variable y′. Accordingly, the four possible minterms with two literals,

$$xy,\ xy',\ x'y,\ x'y'$$

are represented by the four squares (intersections of appropriate rows and columns) in the map (table), as labeled in Figure 9d.

Note that two such squares are adjacent, as defined above, if and only if the squares are geometrically adjacent (have a side in common).

Since any CSoP Boolean expression E(x, y) is a sum of minterms so it can be represented in the Karnaugh map by *placing checks in the appropriate squares*.

A prime implicant of E(x,y) is either *a pair* of checked adjacent squares in E or an *isolated* checked square (i.e., a checked square which is not adjacent to any other checked square of E(x, y)).

A minimal SoP form for E(x, y) is a sum of minimal number of prime implicants such that the sum covers all the checked squares in Karnaugh map (that is, covers E(x, y)).

EXAMPLE 11. Find the prime implicants and a minimal SoP form for each of the following CSoP Boolean expressions:

(a) $E_1=xy+xy'$; (b) $E_2=xy+x'y+x'y'$; (c) $E_3=xy+x'y'$

Solution. This can be solved by using Karnaugh maps as follows:

(a) Check the squares corresponding to xy and xy′ as in Figure 10(a). Note that E_1 consists of one prime implicant, the two adjacent squares designated by the loop in Figure 10(a). This pair of adjacent squares represents the variable x, **so x is a (the only) prime implicant of E_1.** Consequently, $E_1=x$ is its minimal sum.

(b) Check the squares corresponding to xy, x′y, and x′y′ as in Figure 10(b). Note that E_2 contains two pairs of adjacent squares (designated by the two loops) which include all the squares of E_2. The vertical pair represents y and the horizontal pair represents x′; hence y and x′ are the prime implicants of E_2. Thus $E_2=x'+y$ is its minimal sum.

(c) Check the squares corresponding to xy and x′y′ as in Figure 10(c). Note that E_3 consists of two isolated squares which represent xy and x′y′; hence xy and x′y′ are the prime implicants of E_3 and $E_3=xy+x'y'$ is its minimal sum. ■

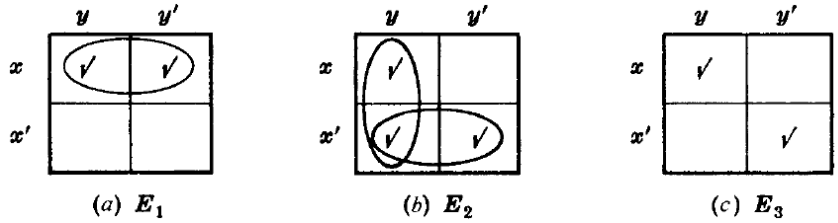


Figure 10

Case of Three Variables.

The Karnaugh map corresponding to Boolean expressions E = E(x,y,z) with three variables x, y, z is shown in Figure 11(a). Recall that there are exactly eight minterms with three variables:

xyz, xyz′, xy′z′, xy′z, x′yz, x′yz′, x′y′z′, x′y′z

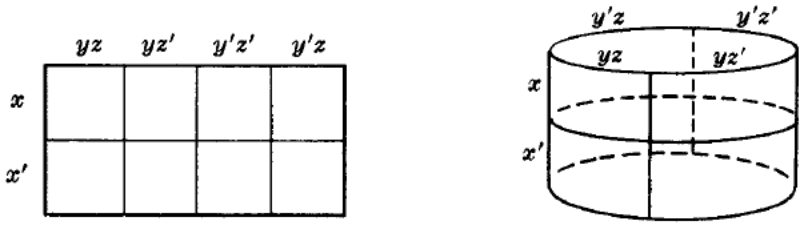


Figure 11

These minterms are listed so that they correspond to the eight squares in the Karnaugh map in the obvious way.

Furthermore, in order that every pair of adjacent products in Figure 11(a) are geometrically adjacent, the right and left edges of the map must be identified. This is equivalent to cutting out, bending, and gluing the map along the identified edges to obtain the cylinder pictured in Figure 11(b) where adjacent products are now represented by squares with one edge in common.

Viewing the Karnaugh map in Figure 11(a) as (2x4) Table, the areas represented by the variables x, y, and z are shown in Figure 12. Specifically,

x is represented by the 1st row, as shaded in Figure 12(a),
x′ - by the 2nd row,
y – by first 2 columns, as shaded in Figure 12(b),
y′ - by 3rd and 4th columns,
z – by 1st and 4th columns, as shaded in Figure 12(c),
z′ - by 2nd and 3rd columns.

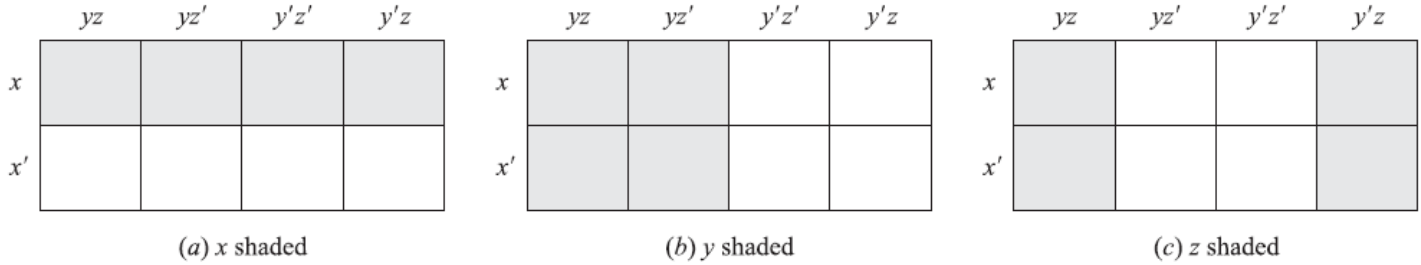


Figure 12

Definition 13: By a *basic rectangle* in the Karnaugh map with three variables, we mean:

- a square;
- two adjacent squares;
- four squares which form a one-by-four or a two-by-two rectangle. ■

These basic rectangles correspond to fundamental products of three, two, and one literal, respectively. Moreover, the fundamental product represented by a basic rectangle is the product of just those literals that appear in every square of the rectangle.

Suppose a CSoP Boolean expression E=E(x, y, z) is represented in the Karnaugh map by *placing checks in the appropriate squares*.

A *prime implicant* of a Boolean expression E geometrically is a *maximal basic rectangle* of E, i.e., a basic rectangle contained in E which and is not contained in any larger basic rectangle in E.

A MSoP form for E is the sum of prime implicants. Geometrically (Karnaugh maps method) MSoP is a *minimal cover* of E by maximal basic rectangles, that is, a covering by minimal number of maximal basic rectangles of E.

EXAMPLE 12. Find the prime implicants and a minimal sum-of-products form for each of the following complete sum-of-products Boolean expressions:

(a) E_1=xyz+xyz′+x′yz′+x′y′z

(b) E_2=xyz+xyz′+xy′z+x′yz+x′y′z

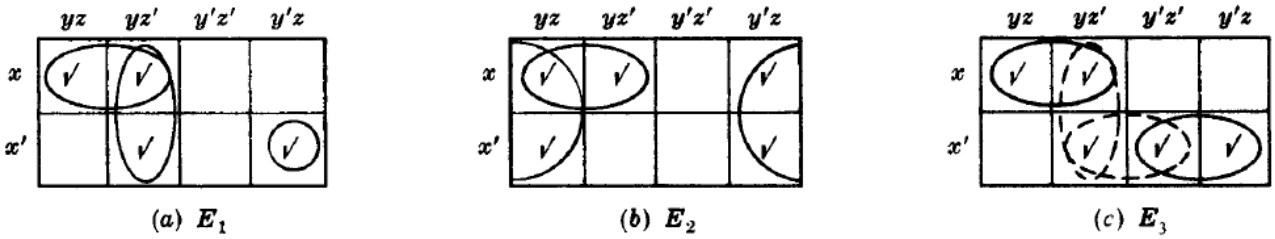
(c) E_3=xyz+xyz′+x′yz′+x′y′z′+x′y′z

Solution. This can be solved by using Karnaugh maps as follows:

. (a) Check the squares corresponding to the four summands as inFigure 13(a). Observe that E_1 has three prime implicants (maximal basic rectangles), which are circled; these are xy, yz′, and x′y′z. All three are needed to cover E_1; hence the minimal sum for E_1 is E_1= xy+yz′+x′y′z

. (b) Check the squares corresponding to the five summands as in Figure 13(b). Note that E_2 has two prime implicants, which are circled. One is the two adjacent squares which represents xy, and the other is the two-by-two square (spanning the identified edges) which represents z. Both are needed to cover E_2, so the minimal sum for E_2 is E_2=xy+z

. (c) Check the squares corresponding to the five summands as in Figure 13(c). As indicated by the loops, E_3 has four prime implicants, xy, yz′, x′z′, and x′y′. However, only one of the two dashed ones, i.e., one of yz′ or x′z′, is needed in a minimal cover of E_3. Thus, E_3 has two minimal sums: E_3=xy+yz′+x′y′, E_3=xy+x′z′+x′y′



(a) E_1 (b) E_2 (c) E_3

Figure 13

Case of Four Variables.

In Figure 14, we show the structure of a Karnaugh map to work with Boolean expressions E=E(x, y, z, t) of four variables x, y, z, t. Each of the 16 squares corresponds to one of the 16 minterms with

four variables,

$$xyzt, xyzt', xyz't', xyz't, ..., x'yz't$$

as indicated by the labels of the row and column of the square.

Observe that the top line (upper than the 1st row) and the left side (before than the 1st column) are labeled so that adjacent products differ in precisely one literal.

Again we must identify the left edge (=1st column) with the right edge (=4th column) as we did with three, variables; but we must also identify the top edge (=1st row) with the bottom edge (=4th row). These identifications give rise to a donut-shaped surface called a *torus*, and we may view our map as really being a torus.

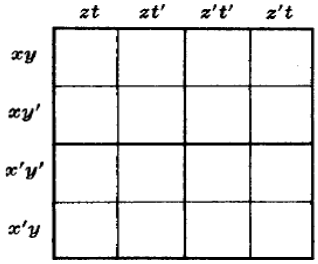


Figure 14

Definition 14: A *basic rectangle* in a four-variable Karnaugh map is a square, two adjacent squares, four squares which form a one-by-four or two-by-two rectangle, or eight squares which form a two-by-four rectangle. These rectangles correspond to fundamental products with four, three, two, and one literal, respectively. Again, maximal basic rectangles are the prime implicants.

The minimization technique for a Boolean expression E(x, y, z, t) is the same as before.

EXAMPLE 13. Find the fundamental product P represented by the basic rectangle in the Karnaugh maps shown in Figure 15 below.

Solution. In each case, we find the literals which appear in all the squares of the basic rectangle; P is the product of such literals.

(a) xy′, and z′ appear in both squares; hence P=xy′z′

(b) Only y and z appear in all four squares; hence P=yz

(c) Only t appears in all eight squares; hence P=t

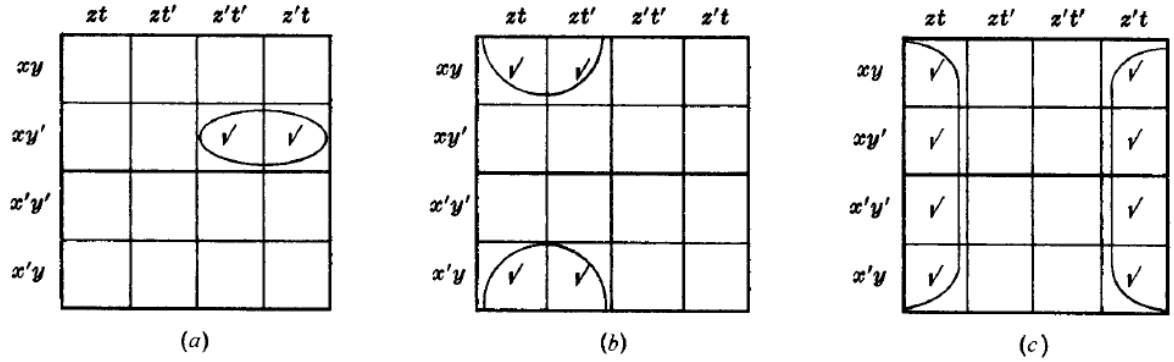


Figure 15

EXAMPLE 14. Use a Karnaugh map to find a minimal sum-of-products form for

$$E=xy'+xyz+x'y'z'+x'yzt'$$

Solution. Check all the squares representing each fundamental product. That is, check all four squares representing xy′, the two squares representing xyz, the two squares representing x′y′z′, and the one square representing x′yzt′, as in Figure 16 below. A minimal cover of the map consists of the three designated maximal basic rectangles.

The two-by-two squares represent the fundamental products xz and y′z′, and the two adjacent squares (on top and bottom) represents yzt′. Hence

$$E = xz + y'z' + yzt'$$

is a minimal sum for E.

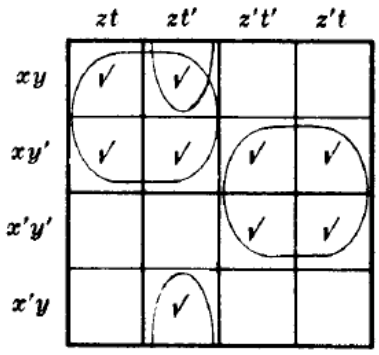


Figure 16

Part 5. Truth Tables, Boolean Functions.

In Part 5 we solve the following problems:

1) **Given AND-OR logic circuit L (equivalently, Boolean SoP expression Y) find its truth table.**
2) **Inverse Problem. Given truth table T(Y) (or T(L)) restore a Boolean expression (minimal or not).**

Consider a logic circuit L with n=3 input devices A, B, C, and output Y, say

$$Y=ABC+AB'C+A'B$$

Each assignment of a set of three bits to the inputs A, B, C yields an output bit for Y. All together there are $2^n=2^3=8$ possible ways to assign bits to the inputs as follows:

000, 001, 010, 011, 100, 101, 110, 111

The assumption is that the sequence of first bits is assigned to A, the sequence of second bits to B, and the sequence of third bits to C. Thus, the above set of inputs may be rewritten in the form

A = 00001111, B = 00110011, C = 01010101

We emphasize that these three 2^n= 8-bit sequences contain the eight possible combinations of the input bits.

Definition 15. The ***truth table*** **T = T (L)** of the above circuit L consists of the output sequence Y that corresponds to the input sequences A, B, C. This truth table T may be expressed using fractional or relational notation, that is, T may be written in the form

T(A, B, C) = Y or T(L) = [A, B, C; Y] ■

Last form for the truth table for L is essentially the same as the truth table for a proposition discussed earlier. The only difference is that here the values for A, B, C, and Y are written horizontally, whereas in previous considerations they are written vertically (see table below),

A	B	C	Y=T(A, B, C)
0	0	0	
0	0	1	
0	1	0	
0	1	1	

1	0	0	
1	0	1	
1	1	0	
1	1	1	

Consider a logic circuit L with n input devices. There are many ways to form n input sequences A_1, A_2, ..., A_n so that they contain the 2^n different possible combinations of the input bits. (Note that each sequence must contain 2^n bits.)

One assignment scheme is as follows:

A_1: Assign 2^{n-1} bits which are 0's followed by 2^{n-1} bits which are 1's.

A_2: Repeatedly assign 2^{n-2} bits which are 0's followed by 2^{n-2} bits which are 1's.

A_3: Repeatedly assign 2^{n-3} bits which are 0's followed by 2^{n-3} bits which are 1's.

And so on. The sequences obtained in this way will be called ***special sequences***. Replacing 0 by 1 and 1 by 0 in the special sequences yields the complements of the special sequences.

Remark: Assuming the input are the special sequences, we frequently do not need to distinguish between the truth table

$$T(L) = [A_1, A_2, ..., A_n; Y]$$

and the output Y itself.

EXAMPLE 15.

(a) Suppose a logic circuit L has n=4 input devices A, B, C, D. Find special sequences for input variables.

Solution. The $2^n=2^4=16$-bit special sequences for A, B, C, D follow:

A = 0000000011111111, C = 0011001100110011

B = 0000111100001111, D = 0101010101010101

That is:

(1) A begins with eight 0's followed by eight 1's. (Here $2^{n-1}=2^3=8$.)

(2) B begins with four 0's followed by four 1's, and so on. (Here $2^{n-2}=2^2=4$.)

(3) C begins with two 0's followed by two 1's, and so on. (Here $2^{n-3}=2^1=2$.)

(4) D begins with one 0 followed by one 1, and so on. (Here $2^{n-4}=2^0=1$.)

(b) Suppose a logic circuit L has n=3 input devices A, B, C. Find special sequences for the input variables and corresponding complements.

Solution. The $2^n=2^3=8$-bit special sequences for A, B, C and their complements A′, B′, C′ are as follows:

A=00001111, B=00110011, C=01010101

A′=11110000, B′=11001100, C′=10101010 ■

Algorithm 3 below contains a three-step for finding the truth table for a logic circuit L where the output Y is given by a Boolean sum-of-products expression in the inputs.

ALGORITHM 3. The input is a Boolean SoP expression $Y=Y(A_1, A_2, ..., A_n)$

Step 1. Write down the special sequences for the A_1, A_2, ... and their complements.

Step 2. Find each product appearing in Y. (Recall that a product $X_1X_2\ldots$ =1 in a position if and only if all the X_1, X_2, … have 1 in the position.)

Step 3. Find the sum Y of the products (Recall that a sum $X_1+X_2+\ldots=0$ in a position if and only if all the $X_1, X_2, \ldots$ have 0 in the position.)

EXAMPLE 16. Use the Algorithm 3 to find the truth table T=T(L) of the logic circuit L in Example 13, Figure 13 or, equivalently, of the Boolean SoP expression below

$$Y=ABC+AB'C+A'B$$

Solution.

(1) The special sequences and their complements appear in Example 2 (b).

(2) The products are as follows: ABC=00000001, AB′C= 00000100, A′B=00110000

(3) The sum is Y =00110101.

Accordingly,

$$T(00001111, 00110011, 01010101) = 00110101$$

or simply T(L)=00110101 where we assume the input consists of the special sequences. ■

Boolean Functions

Let E be a Boolean expression with n variables $x_1, x_2, ..., x_n$. The entire discussion above can also be applied to E where now the special sequences are assigned to the variables $x_1, x_2, ..., x_n$ instead of the input devices $A_1, A_2, ..., A_n$. The truth table T =T(E) of E is defined in the same way as the truth table T=T(L) for a logic circuit L.

For example, the Boolean expression

$$E=xyz+xy'z+x'y$$

which is analogous to the logic circuit L in Example 16, yields the truth table

$$T(00001111, 00110011, 01010101) =00110101$$

or simply T(E)=00110101, where we assume the input consists of the special sequences.

Remark: The truth table for a Boolean expression $E=E(x_1, x_2, ..., x_n)$ with n variables may also be viewed as a "Boolean" function from $\mathbf{B^n}$ into **B**. (The Boolean algebras $\mathbf{B^n}$ and **B**={0, 1} are defined by operations in Table 3.) That is, each element in $\mathbf{B^n}$ is a list of n bits which when assigned to the list of variables in E produces an element in **B**. The truth table T(E) of E is simply the graph of the function.

EXAMPLE 17.

(a) Consider Boolean expressions E=E(x, y, z) with three variables. Find truth tables for each of eight minterms.
Solution. The eight minterms (fundamental products involving all three variables) are as follows (Table 4):

Table 4 (Minterms in x, y, z)

xyz	xyz′	xy′z	xy′z′
x′yz	x′yz′	x′y′z	x′y′z′

The truth tables for these minterms (**using the special sequences for x, y, z**) are as follows:

Table 5 Truth tables for minterms from Table 4 using the special sequences for x, y, z

xyz=00000001	xyz′=00000010	xy′z=00000100	xy′z′=00001000
x′yz=00010000	x′yz′=00100000	x′y′z=01000000	x′y′z′=10000000

Observe that **each minterm assumes the value 1 in only one of the eight positions**.

(b) Consider the Boolean expression E=xyz′+x′yz+x′y′z. Find the truth table for E.

Solution. Note that E is a complete SoP expression containing three minterms. Accordingly, the truth table T=T(E) for E, using the special sequences for x, y, z, can be easily obtained from the sequences in part (a). Specifically, the truth table T (E) will contain exactly three 1's in the same positions as the 1's in the three minterms in E. Thus

T(00001111, 00110011, 01010101)=01010010 or simply T(E)=01010010 ■

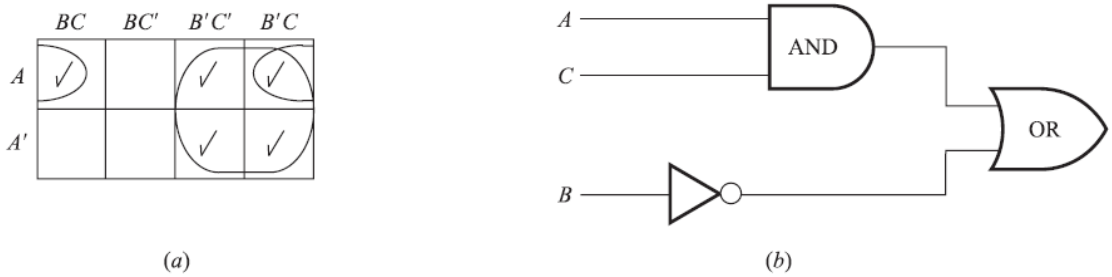
EXAMPLE 18. Design a three-input **minimal** AND-OR circuit L with the following truth table:

T =[A, B, C; L]=[00001111, 00110011, 01010101; 11001101]

Solution. From the truth table we can read off the complete SoP form for L:

L=A′B′C′+A′B′C+AB′C′+AB′C+ABC

The associated Karnaugh map is shown in Figure 17(a). Observe that L has two prime implicants, B′ and AC, in its minimal cover; hence L=B′+AC is a minimal sum for L. Figure 17(b) yields the corresponding minimal AND-OR circuit for L.



(a) (b)

Figure 17

■

EXAMPLE 19. Find the truth table T=T(E) for the Boolean expression

E=E(x, y, z)=(x′y)′yz′+x′(yz+z′)

Solution. First express E as a SoP:

E=(x+y′)yz′+x′yz+x′z′=xyz′+y′yz′+x′yz+x′z′=xyz′+x′yz+x′z′

Now express E as a CSoP:

E=xyz′+x′yz+x′z′(y+y′)=xyz′+x′yz+x′yz′+x′y′z′

Now use the truth table (Table 5) for the minterms from Example 17 to obtain T(E)=10110010. ■

EXAMPLE 20. Find the Boolean expression E=E(x, y, z) corresponding to the truth table:

(a) T (E) =01001001; (b) T (E) = 00010001.

Solution. Each digit 1 in T(E) corresponds to the minterm with the digit 1 in the same position (using the truth tables for the minterms from Example 17, Table 5). For example, the 1 in the second position corresponds to x′y′z whose truth table has a single 1 in the second position. Then E is the sum of these minterms. Thus:

(a) E=x′y′z+xy′z′+xyz; (b) E=x′yz+xyz

Again, we assume the input consists of the special sequences. ■

EXERCISES. SET 1 (Solved Problems).

Boolean Expressions.

1.1. Reduce the following Boolean products to either 0 or a fundamental product:

(a) xyx′z; (b) xyzy; (c) xyz′yx; (d) xyz′yx′z′

Solution. Use the commutative law $x*y=y*x$, the complement law $x*x'=0$, and the idempotent law $x*x=x$:

(a) $xyx'z=x'xyz=0yz=0$;
(b) $xyzy=xzyy=xzy$;
(c) $xyz'yx=xxyyz'=xyz'$;
(d) $xyz'yx'z'=xx'yyz'z'=0yz'=0$.

1.2. Express each Boolean expression E(x, y, z) as a SoP and then in its CSoP form:
(a) $E=x(xy'+x'y+y'z)$; (b) $E=z(x'+y)+y'$

Solution. First use Algorithm 1 to express E as SoP, and then use Algorithm 2 to express E as CSoP.

(a). First we apply Algorithm 1. We have $E=xxy'+xx'y+xy'z=xy'+xy'z$.

Then we continue (apply Algorithm 2) and obtain

$E=xy'+xy'z=xy'(z+z')+xy'z=xy'z+xy'z'+xy'z=xy'z+xy'z+xy'z'=xy'z+xy'z'$

(a). First we apply Algorithm 1. We have $E=x'z+yz+y'$

Then we continue (apply Algorithm 2) and obtain

$E=x'z+yz+y'=x'(y+y')z+(x+x')yz+(x+x')y'(z+z')=$

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

1.3. Express each set expression E(A, B, C) involving sets A, B, C as a union of intersections:
(a) $E=(A\cup B)'\cap(C'\cup B)$; (b) $E=(B\cap C)'\cap(A'\cup C)'$

Solution. Use Boolean notation, "′" for complement, "+" for union, and "∗" (or juxtaposition) for intersection, and then express E as a sum of products (union of intersections).

(a). $E=(A+B)'(C'+B)=A'B'(C'+B)=A'B'C'+A'B'B=A'B'C'$ or $E=A'\cap B'\cap C'$

(b). $E=(BC)'(A'+C)'=(B'+C')(AC')=AB'C'+AC'=(A\cap B'\cap C')\cup(A\cap C')$

Minimal Boolean Expressions, Karnaugh Maps.

1.4. For any Boolean SoP expression E, we let E_L denote the number of literals in E (counting multiplicity) and E_S denote the number of summands in E. Find E_L and E_S for each of the following:
(a) $E=xy'z+x'z'+yz'+x$ (b) $E=x'y'z+xyz+y+yz'+x'z$
(c) $E=xyt'+x'y'zt+xz't$ (d) $E==(xy'+z)'+xy'$

Solution. Simply add up the number of literals and the number of summands in each expression:

(a) $E_L=3+2+2+1=8$, $E_S=4$.

(b) $E_L=3+3+1+2+2=11$, $E_S=5$.

(c) $E_L=3+4+3=10$, $E_S=3$.

(d) Because E is not written as a sum of products, E_L and E_S are not defined.

1.5. Given that E and F are equivalent Boolean sum-of-products, define:
(a) E is simpler than F;
(b) E is minimal.

Solution.

(a) E is simpler than F if $E_L<F_L$ and $E_S\le F_S$, or if $E_L\le F_L$ and $E_S<F_S$.

(b) E is minimal if there is no equivalent sum-of-products expression which is simpler than E.

1.6. Find the fundamental product P represented by each basic rectangle in the Karnaugh map in Figure 18:

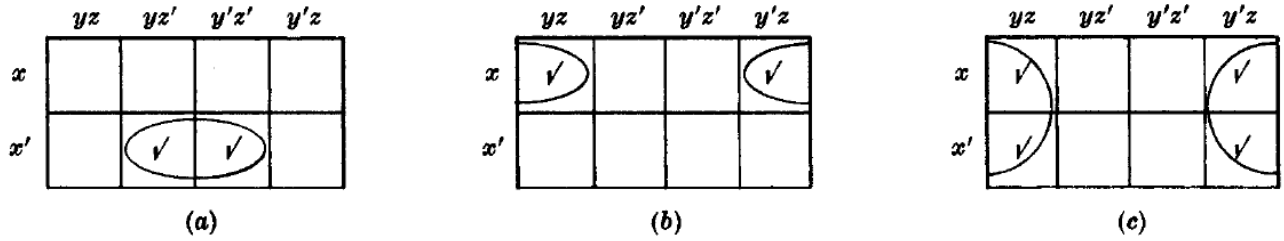


Figure 18

Solution.

In each case find those literals which appear in all the squares of the basic rectangle; then P is the product of such literals.

(a) x′ and z′ appear in both squares; hence P=x′z′.

(b) x and z appear in both squares; hence P=xz.

(c) Only z appears in all four squares; hence P=z.

1.7. Let R be a basic rectangle in a Karnaugh map for four variables x, y, z, t . State the number of literals in the fundamental product P corresponding to R in terms of the number of squares in R.

Solution.

P will have one, two, three, or four literals according as R has eight, four, two, or one squares.

1.8. Find the fundamental product P represented by each basic rectangle R in the Karnaugh map in Figure 19.

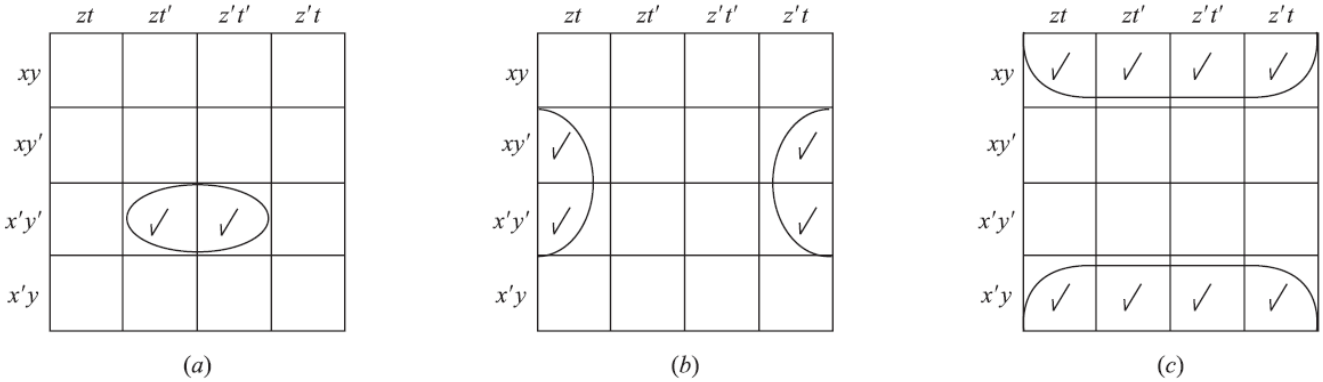


Figure 19.

Solution.

In each case find those literals which appear in all the squares of the basic rectangle; then P is the product of such literals. (Exercise 1.7 indicates the number of such literals in P).

(a) There are two squares in R, so P has three literals. Specifically, x′, y′, t′ appear in both squares; hence P=x′y′t′;

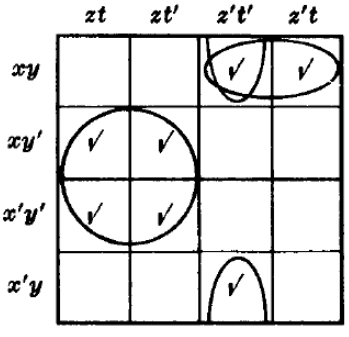
(b) There are four squares in R, so P has two literals. Specifically, only y′ and t appear in all four squares; hence P=y′t;

(c) There are eight squares in R, so P has only one literal. Specifically, only *y* appears in all eight squares; hence P=*y*.

1.9. Let E be the Boolean expression given in the Karnaugh map in Figure 20.

(a) Write E in its CSoP form. (b) Find a minimal form for E.

Figure 20.



Solution.

(a) List the seven fundamental products checked to obtain
E=xyz′t′+xyz′t+xy′zt+xy′zt′+x′y′zt+x′y′zt′+x′yz′t′

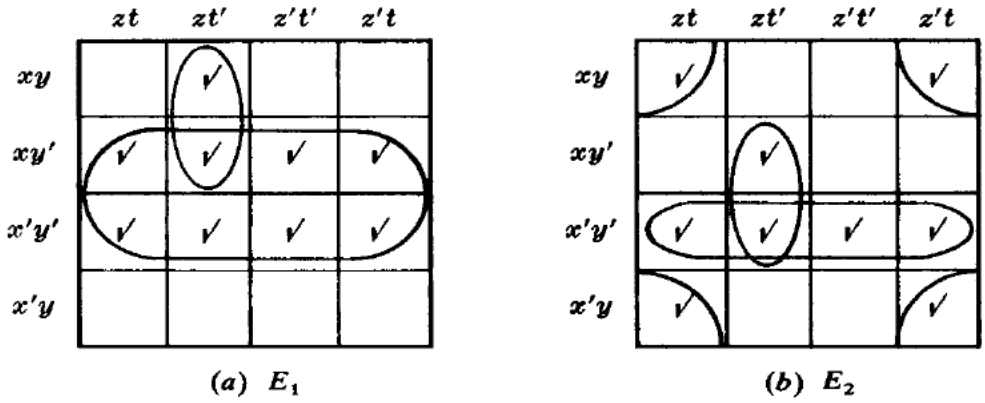
(b) The two-by-two maximal basic rectangle represents y′z since only y′ and z appear in all four squares. The horizontal pair of adjacent squares represents xyz′ and the adjacent squares overlapping the top and bottom edges represent yz′t′. As all three rectangles are needed for a minimal cover,

$$E=y'z+xyz'+yz't'$$

is the minimal sum for E.

1.10. Consider the Boolean expressions E_1 and E_2 in variables x, y, z, t which are given by the Karnaugh maps in Figure 21. Find a minimal sum for
(a) E_1; (b) E_2.

Figure 21.



Solution.

(a) Only *y* appears in all eight squares of the two-by-four maximal basic rectangle, and the designated pair of adjacent squares represents xzt′. As both rectangles are needed for a minimal cover,

$$E_1=y'+xzt'$$

is the minimal sum for E_1.

(b) The four corner squares form a two-by-two maximal basic rectangle which represents yt, since only y and t appear in all the four squares. The four-by-one maximal basic rectangle represents x′y′, and the two adjacent squares represent y′zt′. As all three rectangles are needed for a minimal cover,

$$E_2=yt+x'y'+y'zt'$$

is the minimal sum for E_2.

1.11. Consider the Boolean expressions E1 and E2 in variables x, y, z, t which are given by the Karnaugh maps in Figure 22. Find a minimal sum for
(a) E_1; (b) E_2.

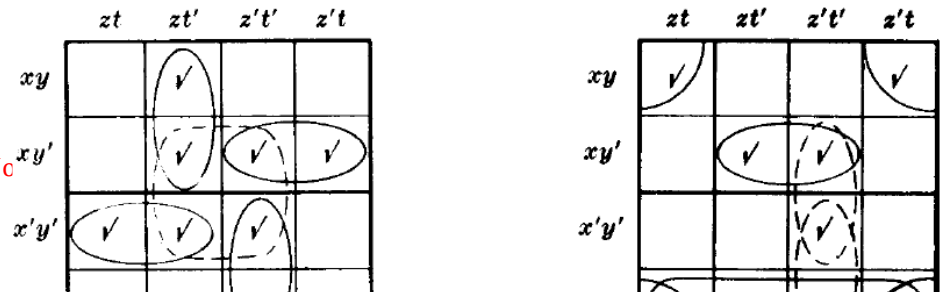


Figure 22.

Solution.

(a) There are five prime implicants, designated by the four loops and the dashed circle. However, the dashed circle is not needed to cover all the squares, whereas the four loops are required. Thus, the four loops give the minimal sum for E_1; that is,

$$E_1=xzt'+xy'z'+x'y'z+x'z't'$$

(b) There are five prime implicants, designated by the five loops of which two are dashed. Only one of the two dashed loops is needed to cover the square $x'y'z't'$. Thus, there are two minimal sums for E_2 as follows:

$$E_2=x'y+yt+xy't'+y'z't'=x'y+yt+xy't'+x'z't'$$

1.12. Use a Karnaugh map to find a minimal sum for:

(a) $E_1=x'yz+x'yz't+y'zt'+xyzt'+xy'z't'$

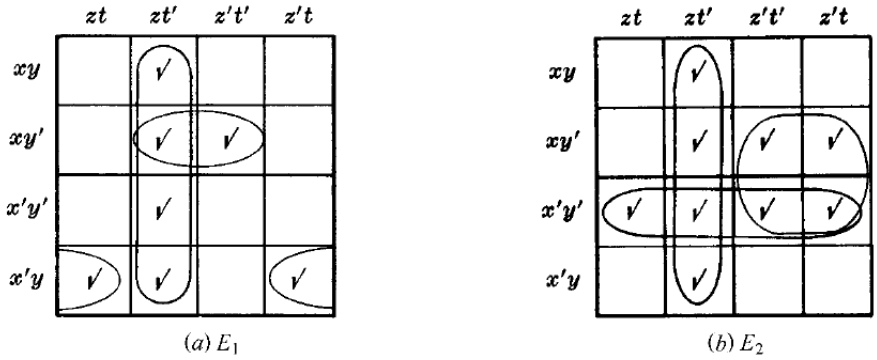
(b) $E_2=y't'+y'z't+x'y'zt+yzt'$

Solution.

(a) Check the two squares corresponding to each of $x'yz$ and $y'zt'$, and check the square corresponding to each of $x'yz't$, $xyzt'$, and $xy'z't'$. This gives the Karnaugh map in Figure 23(a). A minimal cover consists of the three designated loops. Thus, a minimal sum for E_1 follows:

$$E_1=zt'+xy't'+x'yt$$

Figure 23.



(a) Check the four squares corresponding to $y't'$, check the two squares corresponding to each of $y'z't$ and yzt', and check the square corresponding to $x'y'zt$. This gives the Karnaugh map in Figure 23(b). A minimal cover consists of the three designated loops. Thus, a minimal sum for E_2 follows:

$$E_2=zt'+y'z'+x'y'$$

Truth Tables and Boolean Expressions.

1.13. Find the output sequence Y for an AND gate with inputs A, B, C (or equivalently for Y=ABC) where:

(a) A=111001; B=100101; C=110011.

(b) A=11111100; B=10101010; C=00111100.

(c) A=00111111; B=11111100; C=11000011.

Solution.

The output Y=1 for an AND gate if and only if there are 1's in all the positions of the input sequences. Thus:

a) Only the first and last positions have 1's in all three sequences. Hence Y=100001.

(b) Only the third and fifth positions (reading from left to right) have 1's in all three sequences. Thus Y=0101000.

(c) No position has 1's in all three sequences. Thus Y=00000000.

1.14. Find the output sequence Y for an OR gate with inputs A, B, C (or equivalently for Y=A+B+C) where:

(a) A=100001; B=100100; C=110000.

(b) A=11000000; B=10101010; C=00000011.

(c) A=00111111; B=11111100; C=11000011.

Solution.

The output Y = 0 for an OR gate if and only if there are 0's in all the positions of the input sequences. Thus:

(a) Only the third and fifth positions have 0's in all three sequences. Hence, Y=110101.

(b) Only the fourth and sixth positions (reading from left to right) have 0's in all three sequences. Thus, Y=11101011.

(c) No position has 0's in all three sequences. Thus, Y=11111111.

1.15. Find the output sequence Y for a NOT gate with input A or, equivalently, for Y=A′, where:

(a) A=00111111; (b) A=11111100; (c) A=11000011.

Solution.

The NOT gate changes 0 to 1 and 1 to 0. Hence:

(a) A′=11000000; (b) A′=00000011; (c) A′=00111100.

1.16. Consider a logic circuit L with n=5 inputs A, B, C, D, E or, equivalently, consider a Boolean expression E with five variables x_1, x_2, x_3, x_4, x_5.

(a) Find the special sequences for the variables (inputs).

(b) How many different ways can we assign a bit (0 or 1) to each of the n=5 variables?

(c) What is the main property of the special sequences?

Solution.

(a) All sequences have length $2^n=2^5=32$. They will consist of alternating blocks of 0's and 1's where the lengths of the blocks are $2^{n-1}=2^4=16$ for x_1, $2^{n-2}=2^3=8$ for x_2, …, $2^{n-5}=2^0=1$ for x_5. Thus:

x_1=00000000000000001111111111111111

x_2=00000000111111110000000011111111

x_3=00001111000011110000111100001111

x_4=00110011001100110011001100110011

x_5=01010101010101010101010101010101

(b) There are two ways, 0 or 1, to assign a bit to each variable, and so there are $2^n=2^5=32$ ways of assigning a bit to each of the n=5 variables.

(c) The 32 positions in the special sequences give all the 32 possible combinations of bits for the five variables.

1.17. Find the truth table T=T(E) for the Boolean expression E=E(x, y, z) where:
(a) E=xz+x′y; (b) E=xy′z+xy+z′.

Solution.

The special sequences for the variables x, y, z and their complements follow:

x=00001111, y=00110011, z=01010101

x′=11110000, y′=11001100, z′=10101010

(a) Here xz=00000101 and x′y=00110000. Then E=xz+x′y=00110101. Thus

T(00001111, 00110011, 01010101)=00110101

or simply T(E)=00110101 where we assume the input consists of the special sequences.

(b) Here xy′z=00000100, xy=00000011, and z′=10101010. Then E=xy′z+xy+z′=10101111. Thus

T (00001111, 00110011, 01010101)=10101111

1.18. Find the truth table T=T(E) for the Boolean expression E=E(x, y, z) where:
(a) E=xyz′+x′yz; (b) E=xyz+xy′z+x′y′z.

Solution.

Here E is a CSoP expression which is the sum of minterms. Each minterm contains a single 1 in its truth table (see Example 17); hence the truth table of E will have 1's in the same positions as the 1's in the minterms in E. Thus:

(a) T(E)=00010010; (b) T(E)=01000101

1.19. Find the truth table T=T(E) for the Boolean expression:
E=E(x, y, z)=(x′y)′yz′+x′(yz+z′).

Solution.

First express E as a sum-of-products:

E=(x+y′)yz′+x′yz+x′z′=xyz′+y′yz+x′yz+x′z′=xyz′+x′yz+x′z′

Now express E as a complete sum-of-products:

E=xyz′+x′yz+x′z′=xyz′+x′yz+x′(y+y′)z′=xyz′+x′yz+x′yz′+x′y′z′

Use the truth tables for the minterms appearing in Table 5 Example 17 to obtain T(E)=10110010.

1.20. Find the Boolean expression E=E(x, y, z) corresponding to the truth table:
(a) T(E)=01001001; (b) T(E)=00010001

Solution.

Each 1 in T (E) corresponds to the minterm with the 1 in the same position (using the truth tables for the minterms appearing in Table 5 Example 17). For example, the 1 in the second position corresponds to x′y′z whose truth table has a single 1 in the second position. Then E is the sum of these minterms. Thus:

(a) E=x′y′z+xy′z′+xyz (b) E=x′yz+xyz

EXERCISES. SET 2 (Supplementary Problems).

Boolean Expressions. Prime Implicants.

2.1 Reduce the following Boolean products to either 0 or a fundamental product:

(a) xy′zxy′; (b) xyz′sy′ts (c) xy′xz′ty′ (d) xyz′ty′t

2.2 Express each Boolean expression E(x, y, z) as a SoP and then in its CSoP form:

(a) E=x(xy′+x′y+y′z); (b) E=(x+y′z)(y+z′) (c) E=(x′+y)′+y′z

(d) E=(x′y)′(x′+xyz′); (e) E=(x+y)′(xy′)′ (f) E=y(x+yz)′

2.3 Find a MSoP form for each of the Boolean expressions:

(a) E_1=xy′z′+x′y+x′y′z′+x′yz

(b) E_2=xy′+x′z′t+xyzt′+x′y′zt′

(c) E_3=xyzt+xyz′t′+xz′t′+x′y′z′+x′yz′t

Logic Gates. Truth Tables.

2.4 Express the output Y as a Boolean expression in the inputs A, B, C for the logic circuit in:

(a) Figure 24(a) (b) Figure 24(b)

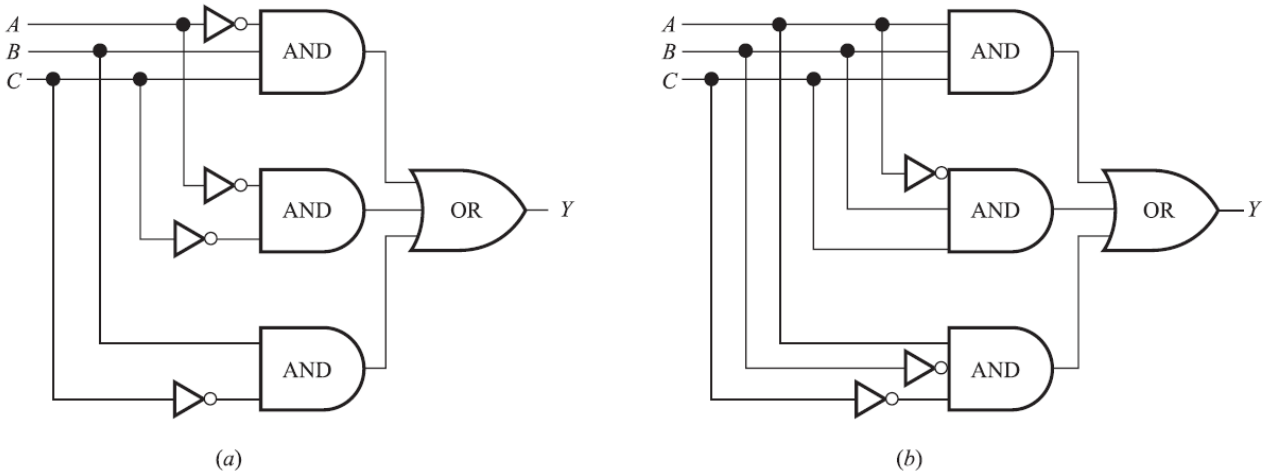


Figure 24.

2.5 Draw the logic circuit L with inputs A, B, C and output Y which corresponds to each Boolean expression:

(a) Y=AB′C+AC′+A′C (b) Y=A′BC+A′BC′+ABC′

2.6 Find the output sequence Y for an AND gate with inputs A, B, C (or equivalently for Y=ABC) where:

(a) A=110001; B=101101; C=110011.

(b) A=01111100; B=10111010; C=00111100.

(c) A=00111110; B=01111100; C=11110011.

2.7 Find the output sequence Y for an OR gate with inputs A, B, C (or equivalently for Y=A+B+C) where:

(a) A=100011; B=100101; C=100001.

(b) A=10000001; B=00100100; C=00000011.

(c) A=00111100; B=11110000; C=10000001.

2.8 Find the output sequence Y for a NOT gate with input A or, equivalently, for Y=A′, where:

(a) A=11100111; (b) A=10001000; (c) A=11111000.

2.9 Find the truth table T=T(E) for the Boolean expression E=E(x, y, z) where:

(a) E=xy+x′z; (b) E=xyz′+xy′z′+x′y′z′

2.10 Find the Boolean expression E=E(x, y, z) corresponding to the truth tables:

(a) T(E)=10001010; (b) T(E)=00010001; (c) T(E)=00110000.

2.11 Find all possible minimal sums for each Boolean expression E given by the Karnaugh maps in Figure 25.

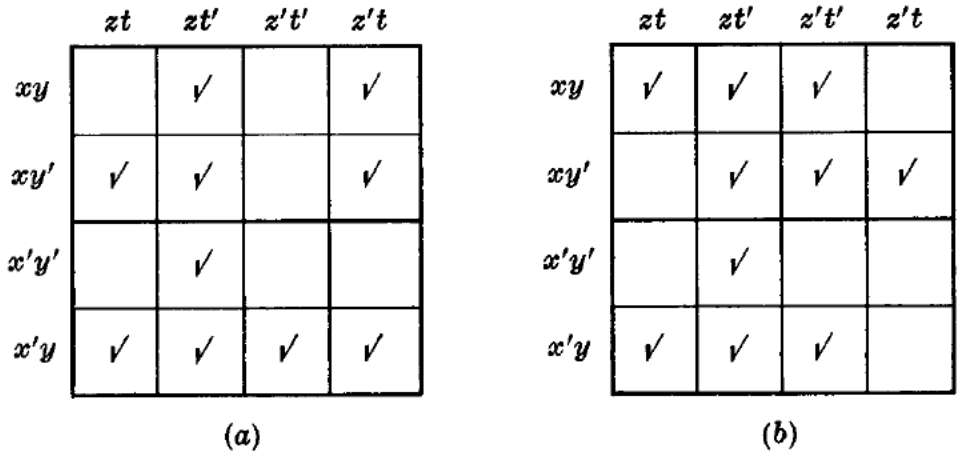


Figure 25

2.12 Use a Karnaugh map to find a minimal sum for the Boolean expression.

(a) E=y′z+y′z′t′+z′t (b) E=y′zt+xzt′+xy′z′

2.13 Use Karnaugh maps to redesign each circuit in Figure 26 so that it becomes a minimal AND-OR circuit.

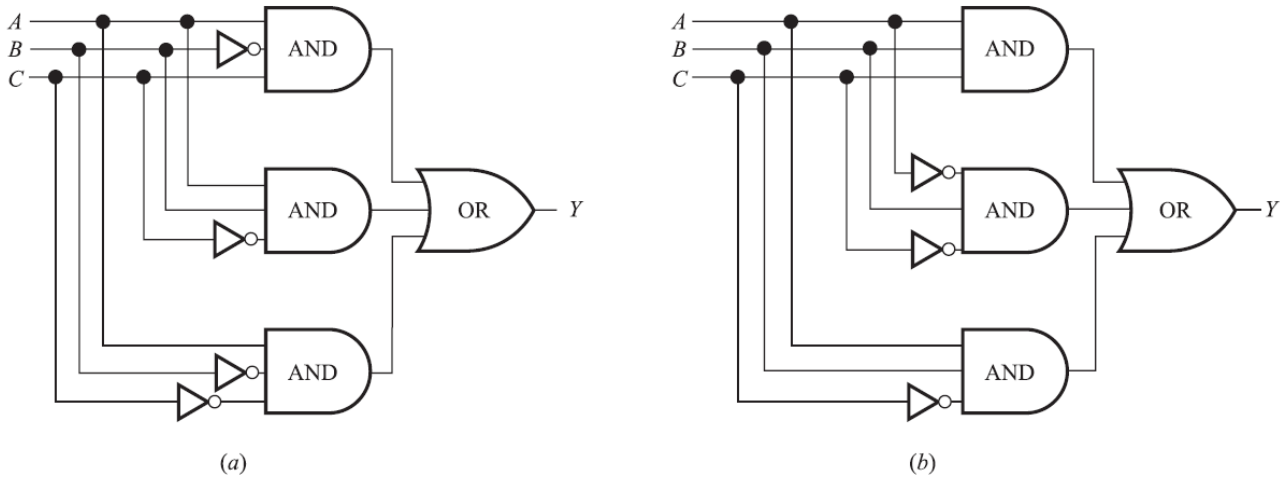


Figure 26

2.14 Suppose three switches A, B, C are connected to the same hall light. At any moment a switch may be "up" denoted by 1 or "down" denoted by 0. A change in any switch will change the parity (odd or even) of the number of 1's. The switches will be able to control the light if it associates, say, an odd parity with the light being "on" (represented by 1), and an even parity with the light being "off" (represented by 0).

(a) Show that the following truth table satisfies these conditions:

T(A,B,C)=T(00001111, 00110011, 01010101)=01101001

(b) Design a minimal AND-OR circuit L with the above truth table.