Boolean Algebra and Digital Logic

All modern digital computers are dependent on circuits that implement Boolean functions. We shall discuss two classes of such circuits: Combinational and Sequential. The difference between the two types of circuits is that sequential circuits contain elements that implement memory, whereas combinational circuits do not have memory.

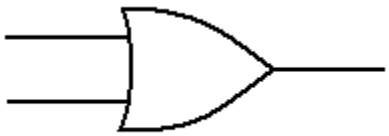
The first part on this chapter focuses on Boolean functions of Boolean variables and the simple combinational circuits that implement those functions. We then move on to digital components – circuits that implement more complex but very useful functions. We finish this section with a discussion of sequential logic.

Boolean variables are variables that can take two possible values: 0 and 1, FALSE and TRUE, etc. Boolean functions are functions that can have only Boolean values. The term Boolean function commonly refers to a Boolean function of Boolean variables; that is the sense in which we shall use the term. The hypothetical integer equality function IsEqual may be regarded as a Boolean function of integer variables; IsEqual(A, B) = TRUE if and only if A = B. Any reference to a Boolean function such as this will be explicit.

Consider a Boolean function of N variables. It is easily shown (by induction) that there are exactly 2^N distinct combinations of N variables. For small numbers of variables, this allows a representation of functions that is unique to Boolean variables: specification of the function value for every possible value of the input variables.

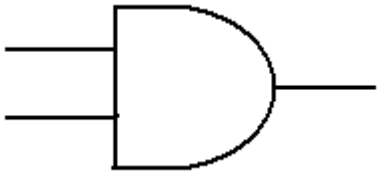
Before we discuss Boolean algebra in detail, we shall first describe the four basic gates used to implement the Boolean functions: NOT, AND, OR, and XOR. We then shall define a few "derived gates"; i.e., gates that can be viewed as combinations of these four.

The first gate to be discussed is the OR gate. The truth table for a two-input OR gate is shown below. In general, if any input to an OR gate is 1, the output is 1.



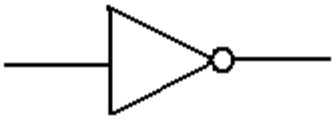
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

The next gate to be discussed is the AND gate. The truth table for a two-input AND gate is shown now. In general, if any input to an AND gate is 0, the output is 0.



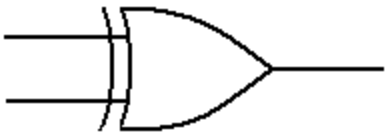
A	B	A • B
0	0	0
0	1	0
1	0	0
1	1	1

The third of the four basic gates is the single input NOT gate. Note that there are two ways of denoting the NOT function. NOT(A) is denoted as either A' or $\overline{A}$. We use $\overline{A}$.



A	$\overline{A}$
0	1
1	0

The last of the gates to be discussed at this first stage is not strictly a basic gate. We include it at this level because it is extremely convenient. This is the two-input Exclusive OR (XOR) gate, the function of which is shown in the following truth table.



A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

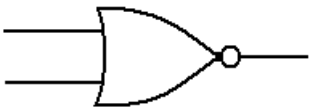
We note immediately an interesting and very useful connection between the XOR function and the NOT function. For any A, A ⊕ 0 = A, and A ⊕ 1 = $\overline{A}$. The proof is by truth table.

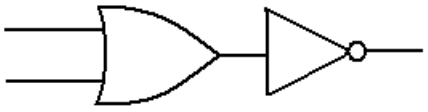
A	B	A ⊕ B	Result
0	0	0	A
1	0	1	$\overline{A}$
0	1	1	$\overline{A}$
1	1	0	$\overline{A}$

This result is extremely useful when designing a ripple carry adder/subtractor.

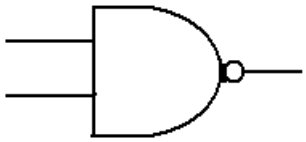
"Derived Gates"

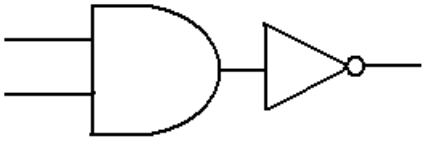
We now show 2 gates that may be considered as derived from the above: NAND and NOR.





The NOR gate is the same as an OR gate followed by a NOT gate.





The NAND gate is the same as an AND gate followed by a NOT gate.

Electrical engineers may validly object that these two gates are not "derived" in the sense that they are more basic than the gates previously discussed. In fact, NAND gates are often used to make AND gates. As always, we are interested in the non-engineering approach and stay with our view: NOT, AND, and OR are basic. We shall ignore the XNOR gate and, if needed, implement its functionality by an XOR gate followed by a NOT gate.

Truth Tables

A truth table for a function of N Boolean variables depends on the fact that there are only 2^N different combinations of the values of these N Boolean variables. For small values of N, this allows one to list every possible value of the function.

Consider a Boolean function of two Boolean variables X and Y. The only possibilities for the values of the variables are:

X = 0 and Y = 0
X = 0 and Y = 1
X = 1 and Y = 0
X = 1 and Y = 1

Similarly, there are eight possible combinations of the three variables X, Y, and Z, beginning with X = 0, Y = 0, Z = 0 and going through X = 1, Y = 1, Z = 1. Here they are.

X = 0, Y = 0, Z = 0 X = 0, Y = 0, Z = 1 X = 0, Y = 1, Z = 0 X = 0, Y = 1, Z = 1
X = 1, Y = 0, Z = 0 X = 1, Y = 0, Z = 1 X = 1, Y = 1, Z = 0 X = 1, Y = 1, Z = 1

As we shall see, we prefer truth tables for functions of not too many variables.

X	**Y**	**F(X, Y)**
0	0	1
0	1	0
1	0	0
1	1	1

The figure at left is a truth table for a two-variable function. Note that we have four rows in the truth table, corresponding to the four possible combinations of values for X and Y. Note also the standard order in which the values are written: 00, 01, 10, and 11. Other orders can be used when needed (it is done below), but one must list all combinations.

For another example of truth tables, we consider the figure at the right, which shows two Boolean functions of three Boolean variables. Truth tables can be used to define more than one function at a time, although they become hard to read if either the number of variables or the number of functions is too large. Here we use the standard shorthand of F1 for F1(X, Y, Z) and F2 for F2(X, Y, Z). Also note the standard ordering of the rows, beginning with 0 0 0 and ending with 1 1 1. This causes less confusion than other ordering schemes, which may be used when there is a good reason for them.

X	Y	Z	F1	F2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

As an example of a truth table in which non-standard ordering might be useful, consider the following table for two variables. As expected, it has four rows.

X	Y	X • Y	X + Y
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	1

A truth table in this non-standard ordering would be used to prove the standard Boolean axioms:

X • 0 = 0 for all X X + 0 = X for all X
X • 1 = X for all X X + 1 = 1 for all X

Labeling Rows in Truth Tables

We now discuss a notation that is commonly used to identify rows in truth tables. The exact identity of the rows is given by the values for each of the variables, but we find it convenient to label the rows with the integer equivalent of the binary values. We noted above that for N variables, the truth table has 2^N rows. These are conventionally numbered from 0 through $2^N - 1$ inclusive to give us a handy way to reference the rows. Thus a two variable truth table would have four rows numbered 0, 1, 2, and 3. Here is a truth-table with labeled rows.

Row	A	B	G(A, B)
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

0 = 0•2 + 0•1
1 = 0•2 + 1•1
2 = 1•2 + 0•1
3 = 1•2 + 1•1

We can see that G(A, B) = A ⊕ B, but this value has nothing to do with the row numberings, which are just the decimal equivalents of the values in the A & B columns as binary.

A three variable truth table would have eight rows, numbered 0, 1, 2, 3, 4, 5, 6, and 7. Here is a three variable truth table for a function F(X, Y, Z) with the rows numbered.

Row Number	X	Y	Z	F(X, Y, Z)
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Note that the row numbers correspond to the decimal value of the three bit binary, thus

0 = 0•4 + 0•2 + 0•1
1 = 0•4 + 0•2 + 1•1
2 = 0•4 + 1•2 + 0•1
3 = 0•4 + 1•2 + 1•1
4 = 1•4 + 0•2 + 0•1
5 = 1•4 + 0•2 + 1•1
6 = 1•4 + 1•2 + 0•1
7 = 1•4 + 1•2 + 1•1

Truth tables are purely Boolean tables in which decimal numbers, such as the row numbers above do not really play a part. However, we find that the ability to label a row with a decimal number to be very convenient and so we use this. The row numberings can be quite important for the standard algebraic forms used in representing Boolean functions.

Question: Where to Put the Ones and Zeroes

Every truth table corresponds to a Boolean expression. For some truth tables, we begin with a Boolean expression and evaluate that expression in order to find where to place the 0's and 1's. For other tables, we just place a bunch of 0's and 1's and then ask what Boolean expression we have created. The truth table just above was devised by selecting an interesting pattern of 0's and 1's. The author of these notes had no particular pattern in mind when creating it. Other truth tables are more deliberately generated.

Let's consider the construction of a truth table for the Boolean expression.

$$F(X, Y, Z) = X \bullet Y + Y \bullet Z + X \bullet \overline{Y} \bullet Z$$

Let's evaluate this function for all eight possible values of X, Y, Z.

X = 0 Y = 0 Z = 0 F(X, Y, Z) = 0•0 + 0•0 + 0•1•0 = 0 + 0 + 0 = 0
X = 0 Y = 0 Z = 1 F(X, Y, Z) = 0•0 + 0•1 + 0•1•1 = 0 + 0 + 0 = 0
X = 0 Y = 1 Z = 0 F(X, Y, Z) = 0•1 + 1•0 + 0•0•0 = 0 + 0 + 0 = 0
X = 0 Y = 1 Z = 1 F(X, Y, Z) = 0•1 + 1•1 + 0•0•1 = 0 + 1 + 0 = 1
X = 1 Y = 0 Z = 0 F(X, Y, Z) = 1•0 + 0•0 + 1•1•0 = 0 + 0 + 0 = 0
X = 1 Y = 0 Z = 1 F(X, Y, Z) = 1•0 + 0•1 + 1•1•1 = 0 + 0 + 1 = 1
X = 1 Y = 1 Z = 0 F(X, Y, Z) = 1•1 + 1•0 + 1•0•0 = 1 + 0 + 0 = 1
X = 1 Y = 1 Z = 1 F(X, Y, Z) = 1•1 + 1•1 + 1•0•1 = 1 + 1 + 0 = 1

From the above, we create the truth table for the function. Here it is.

X	Y	Z	F(X, Y, Z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A bit later we shall study how to derive Boolean expressions from a truth table. Truth tables used as examples for this part of the course do not appear to be associated with a specific Boolean function. Often the truth tables are associated with well-known functions, but the point is to derive that function starting only with 0's and 1's.

Consider the truth table given below, with no explanation of the method used to generate the values of F1 and F2 for each row.

Row	X	Y	Z	F1	F2
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

Figure: Our Sample Functions F1 and F2

Students occasionally ask how the author knew where to place the 0's and 1's in the above table. There are two answers to this, both equally valid. We reiterate the statement that a Boolean function is completely specified by its truth table. Thus, one can just make an arbitrary list of 2^N 0's and 1's and then decide what function of N Boolean variables has been represented. In that view, the function F2 is that function specified by the sequence (0, 0, 0, 1, 0, 1, 1, 1) and nothing more. We can use methods described below to assign it a functional representation. Note that F2 is 1 if and only if two of X, Y, and Z are 1. Given this, we can give a functional description of the function as F2 = X•Y + X•Z + Y•Z.

As the student might suspect, neither the pattern of 0's and 1's for F1 nor that for F2 were arbitrarily selected. The real answer is that the instructor derived the truth table from a set of known Boolean expressions, one for F1 and one for F2. The student is invited to compute the value of F2 = X•Y + X•Z + Y•Z for all possible values of X, Y, and Z; this will verify the numbers as shown in the truth table.

We have noted that a truth table of two variables has four rows (numbered 0, 1, 2, and 3) and that a truth table of three variables has eight rows (numbered 0 through 7). We now prove that a truth table of N variables has 2^N rows, numbered 0 through $2^N - 1$. Here is an inductive proof, beginning with the case of one variable.

1. Base case: a function of one variable X requires 2 rows,
   one row for X = 0 and one row for X = 1.
2. If a function of N Boolean variables $X_1, X_2, \ldots, X_N$ requires 2^N rows, then
   the function of (N + 1) variables $X_1, X_2, \ldots, X_N, X_{N+1}$ would require
   2^N rows for $X_1, X_2, \ldots, X_N$ when $X_{N+1} = 0$
   2^N rows for $X_1, X_2, \ldots, X_N$ when $X_{N+1} = 1$
3. $2^N + 2^N = 2^{N+1}$, so the function of (N + 1) variables required 2^{N+1} rows.

While we are at it, we show that the number of Boolean functions of N Boolean variables is 2^R where $R = 2^N$, thus the number is 2^{2^N}. The argument is quite simple. We have shown that the number of rows in a truth table is given by $R = 2^N$. The value in the first row could be a 0 or 1; thus two choices. Each of the $R = 2^N$ rows could have two choices, thus the total number of functions is 2^R where $R = 2^N$.

For N = 1, R = 2, and $2^2 = 4$. A truth table for the function F(X) would have two rows, one for X = 0 and one for X = 1. There are four functions of a single Boolean variable.

$F_1(X) = 0$, $F_2(X) = 1$, $F_3(X) = X$, and $F_4(X) = \overline{X}$.

We might mention that purists for binary labeling might write this list as

$F_0(X) = 0$, $F_1(X) = 1$, $F_2(X) = X$, and $F_3(X) = \overline{X}$,

but who cares about such precision?

It might be interesting to give a table of the number of rows in a truth table and number of possible Boolean functions for N variables. The number of rows grows quickly, but the number of functions grows at an astonishing rate.

N	$R = 2^N$	2^R
1	2	4
2	4	16
3	8	256
4	16	65 536
5	32	4 294 967 296
6	64	$2^{64} \approx 1.845 \bullet 10^{19}$

Note on computation: $\log 2 = 0.30103$, so $2^{64} = (10^{0.30103})^{64} = 10^{19.266}$.
$\log 1.845 = 0.266$, so $10^{0.266} \approx 1.845$ and $10^{19.266} \approx 1.845 \bullet 10^{19}$

The number of Boolean functions of N Boolean variables is somewhat of interest. More to interest in this course is the number of rows in any possible truth-table representation of a function of N Boolean variables. For N = 2, 3, and 4, we have $2^N = 4$, 8, and 16 respectively, so that truth tables for 2, 3, and 4 variables are manageable. Truth tables for five variables are a bit unwieldy and truth tables for more than five variables are almost useless.

Don't-Cares in Truth Tables

Sometimes we use a short-hand notation when the truth table gets a bit long. Consider the following truth table that has F(X, Y, Z) = 0 whenever X = 0.

X	Y	Z	F(X, Y, Z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

The table at right is the abbreviated form of the table above. What it says is that when X = 0, the value of F(X, Y, Z) is 0 without regard to the values of Y and Z. Thus the value "d" or "don't-care" for Y and Z in the truth table at right. What this says is that when X = 0, Y can be either 0 or 1 and Z can be either 0 or 1 and the value of F(X, Y, Z) will still be 0. We use the "d" or "don't-care" conditions considerable when we use flip-flops in sequential circuits – a topic studied much later.

X	Y	Z	F(X, Y, Z)
0	d	d	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Evaluation of Boolean Expressions

Here is another topic that this instructor normally forgets to mention, as it is so natural to one who has been in the "business" for many years. The question to be addressed now is:

"What are the rules for evaluating Boolean expressions?"

Operator Precedence

The main question to be addressed is the relative precedence of the basic Boolean operators: AND, OR, and NOT. This author is not aware of the relative precedence of the XOR operator and prefers to use parentheses around an XOR expression when there is any doubt.

The relative precedence of the operators is:

1) NOT do this first
2) AND
3) OR do this last

Consider the Boolean expression A•B + C•D, often written as AB + CD. Without the precedence rules, there are two valid interpretations: either (A•B) + (C•D) or A•(B + C)•D. The precedence rules for the operators indicate that the first is the correct interpretation; in this Boolean algebra follows standard algebra as taught in high-school. Consider now the expression $\overline{A} \bullet B + C \bullet \overline{D}$; according to our rules, this is read as $\left(\left(\overline{A}\right) \bullet B\right) + \left(C \bullet \left(\overline{D}\right)\right)$.

Note that parentheses and explicit extension of the NOT over-bar can override the precedence rules, so that A•(B + C)•D is read as the logical AND of three terms: A, (B + C), and D. Note also that the two expressions $\overline{A \bullet B}$ and $\overline{A} \bullet \overline{B}$ are different. The first expression, better written as $\overline{(A \bullet B)}$, refers to the logical NOT of the logical AND of A and B; in a language such as LISP it would be written as NOT (AND A B). The second expression, due to the precedence rules, refers to the logical AND of the logical NOT of A and the logical NOT of B; in LISP this might be written as AND((NOT A) (NOT B)). LISP expressions are easy for a computer to parse, but quite annoying for a human to read. As one wag put it, the term "LISP" stands for "**L**ots of **I**nsipid **S**tupid **P**arentheses".

Evaluation of Boolean expressions implies giving values to the variables and following the precedence rules in applying the logical operators. Let A = 1, B = 0, C = 1, and D = 1.

$A \bullet B + C \bullet D = 1 \bullet 0 + 1 \bullet 1 = 0 + 1 = 1$

$A \bullet (B + C) \bullet D = 1 \bullet (0 + 1) \bullet 1 = 1 \bullet 1 \bullet 1 = 1$

$\overline{A} \bullet B + C \bullet \overline{D} = \overline{1} \bullet 0 + 1 \bullet \overline{1} = 0 \bullet 0 + 1 \bullet 0 = 0 + 0 = 0$

$\overline{A \bullet B} = \overline{1 \bullet 0} = \overline{0} = 1$

$\overline{A} \bullet \overline{B} = \overline{1} \bullet \overline{0} = 0 \bullet 1 = 0$

Also

$A \bullet (B + C \bullet D) = 1 \bullet (0 + 1 \bullet 1) = 1 \bullet (0 + 1) = 1 \bullet 1 = 1$

$(A \bullet B + C) \bullet D = (1 \bullet 0 + 1) \bullet 1 = (0 + 1) \bullet 1 = 1$

Properties of Boolean Algebra

We now study briefly the formal postulates and theorems of Boolean algebra, together known as the basic properties of Boolean algebra.

Most of these properties are quite similar to the similar properties seen in standard algebra. We focus on those properties that are somewhat different. In order to emphasize the similarities and differences, we state four basic postulates with no introductory explanation.

For all X	0•X	=	0	OK
For all X	1•X	=	X	OK
For all X	0 + X	=	X	OK
For all X	1 + X	=	1	What?

The first three postulates are what we would expect from standard algebra; they look very familiar from our high school courses. It is the last postulate that looks very strange. We can use a truth table approach to prove the last two postulates at the same time.

W	X	W + X
0	0	0
0	1	1
1	0	1
1	1	1

To understand the last two, recall the truth table for the OR function. The first two lines show 0 + X for both values of X. We see that we do have 0 + X = X. The next two lines show 1 + X for both values of X. We see that 1 + X = 1 for both values of X. As X can take on only the two values 0 and 1, we have shown 1 + X = 1 for all X.

Here we note one possible use of truth tables as proof devices for Boolean functions of a small number of Boolean variables. To show that two functions are equal, we compute the value of each for all possible combinations of the variables. If all values are equal for each possible combination of the variables, then the functions are equal.

The **principle of duality** is another property that is unique to Boolean algebra – there is nothing like it in standard algebra. This principle says that if a statement is true in Boolean algebra, so is its dual. The dual of a statement is obtained by changing AND to OR, OR to AND, 0s to 1s, and 1s to 0s. In the above, we can arrange the postulates as duals.

Postulate		Dual		
0•X	= 0	1 + X	= 1	
1•X	= X	0 + X	= X	
0 + X	= X	1•X	= X	These last two statements just repeat
1 + X	= 1	0•X	= 0	the first two if one reads correctly.

Both standard algebra and Boolean algebra have **distributive postulates**. Standard algebra has one distributive postulate; Boolean algebra must have two distributive postulates as a result of the principle of duality.

In standard algebra, we have the equality A•(B + C) = A•B + A•C for all values of A, B, and C. This is the distributive postulate as seen in high school; we know it and expect it.

In Boolean algebra we have the distributive postulate A•(B + C) = A•B + A•C, which looks familiar. The principle of duality states that if A•(B + C) = A•B + A•C is true then the dual statement A + B•C = (A + B)•(A + C) must also be true. We prove the second statement using a method unique to Boolean algebra. This method depends on the fact that there are only two possible values for A: A = 0 and A = 1. We consider both cases.

If A = 1, the statement becomes 1 + B•C = (1 + B)•(1 + C), or 1 = 1•1, obviously true.
If A = 0, the statement becomes 0 + B•C = (0 + B)•(0 + C), or B•C = B•C.

A Formal Statement of the Postulates and Theorems of Boolean Algebra

Definition: A Boolean algebra is a closed algebraic system containing a set K of two or more elements and three operators, two binary and one unary. The binary operators are denoted "+" for OR and "•" for AND. The unary operator is NOT, denoted by a over-bar placed over the variable. The system is **closed**, so that for all X and Y in K, we have X + Y in K, X • Y in K and NOT(X) in K. The set K must contain two constants 0 and 1 (FALSE and TRUE), which obey the postulates below. In normal use, we say K = {0, 1} – set notation.

The postulates of Boolean algebra are:

P1	Existence of 0 and 1	a) X + 0 = X for all X
		b) X • 1 = X for all X
P2	Commutativity	a) X + Y = Y + X for all X and Y
		b) X • Y = Y • X for all X and Y
P3	Associativity	a) X + (Y + Z) = (X + Y) + Z, for all X, Y, and Z
		b) X • (Y • Z) = (X • Y) • Z, for all X, Y, and Z
P4	Distributivity	a) X • (Y + Z) = (X • Y) + (X • Z), for all X, Y, Z
		b) X + (Y • Z) = (X + Y) • (X + Z), for all X, Y, Z
P5	Complement	a) $X + \overline{X} = 1$, for all X
		b) $X \bullet \overline{X} = 0$, for all X

The theorems of Boolean algebra are:

T1	Idempotency	a) X + X = X, for all X
		b) X • X = X, for all X
T2	1 and 0 Properties	a) X + 1 = 1, for all X
		b) X • 0 = 0, for all X
T3	Absorption	a) X + (X • Y) = X, for all X and Y
		b) X • (X + Y) = X, for all X and Y
T4	Absorption	a) $X + \overline{X} \bullet Y = X + Y$, for all X and Y
		b) $X \bullet (\overline{X} + Y) = X \bullet Y$, for all X and Y
T5	DeMorgan's Laws	a) $\overline{(X + Y)} = \overline{X} \bullet \overline{Y}$
		b) $\overline{(X \bullet Y)} = \overline{X} + \overline{Y}$
T6	Consensus	a) $X \bullet Y + \overline{X} \bullet Z + Y \bullet Z = X \bullet Y + \overline{X} \bullet Z$
		b) $(X + Y) \bullet (\overline{X} + Z) \bullet (Y + Z) = (X + Y) \bullet (\overline{X} + Z)$

A Few Proofs

At this point in the course, we note two facts:

1) That some of the theorems above look a bit strange.
2) That we might need more work on proof of Boolean theorems.

Towards that end, let's look at a few variants of the Absorption Theorem. We prove that

$$X + (X \bullet Y) = X, \text{ for all X and Y}$$

The first proof will use a truth table. Remember that the truth table will have four rows, corresponding to the four possible combinations of values for X and Y:

X = 0 and Y = 0, X = 0 and Y = 1, X = 1 and Y = 0, X = 1 and Y = 1.

X	Y	X • Y	X + (X • Y)
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Having computed the value of X + (X • Y) for all possible combinations of X and Y, we note that in each case we have X + (X • Y) = X. Thus, we conclude that the theorem is true.

Here is another proof method much favored by this author. It depends on the fact that there are only two possible values of X: X = 0 and X = 1. A theorem involving the variable X is true if and only if it is true for both X = 0 and X = 1. Consider the above theorem, with the claim that X + (X • Y) = X. We look at the cases X = 0 and X = 1, computing both the LHS (Left Hand Side of the Equality) and RHS (Right Hand Side of the Equality).

If X = 0, then X + (X • Y) = 0 + (0 • Y) = 0 + 0 = 0, and LHS = RHS.
If X = 1, then X + (X • Y) = 1 + (1 • Y) = 1 + Y = 1, and LHS = RHS.

Consider now a trivial variant of the absorption theorem.

$$\overline{X} + (\overline{X} \bullet Y) = \overline{X}, \text{ for all X and Y}$$

There are two ways to prove this variant of the theorem., given that the above standard statement of the absorption theorem is true. An experienced mathematician would claim that the proof is obvious. We shall make it obvious.

The absorption theorem, as proved, is X + (X • Y) = X, for all X and Y. We restate the theorem, substituting the variable W for X, getting W + (W • Y) = W, for all X and Y. Now, we let W = $\overline{X}$, and the result is indeed obvious: $\overline{X} + (\overline{X} \bullet Y) = \overline{X}$.

We close this section with another proof of this variant of the absorption theorem, that $X + (\overline{X} \bullet Y) = X + Y$ for all X and Y. The theorem can also be proved using the two case method.

X	Y	$\overline{X}$ • Y	X + ($\overline{X}$ • Y)	X + Y
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

We now consider the formal use of these basic properties to prove a simplification. By simplification we prove that $A \oplus B + A \bullet B = A + B$.
By definition of the exclusive OR (XOR) function we have $A \oplus B = \overline{A} \bullet B + A \bullet \overline{B}$, so we must prove that $\overline{A} \bullet B + A \bullet \overline{B} + A \bullet B = A + B$.

$\overline{A} \bullet B + A \bullet \overline{B} + A \bullet B$	$= \overline{A} \bullet B + A \bullet \overline{B} + A \bullet B + A \bullet B$	Idempotency theorem
	$= \overline{A} \bullet B + A \bullet B + A \bullet \overline{B} + A \bullet B$	Commutativity postulate
	$= (\overline{A} + A) \bullet B + A \bullet (\overline{B} + B)$	Distributivity postulate
	$= 1 \bullet B + A \bullet 1$	Complement postulate
	$= B + A$	Identity Postulate
	$= A + B$	Commutativity postulate

Just for fun, we give a truth-table proof of the above identity. The truth table for two variables has four rows, so the proof is short. Let $F(A, B) = \overline{A} \bullet B + A \bullet \overline{B} + A \bullet B$.

A	B	$\overline{A} \bullet B$	$A \bullet \overline{B}$	$A \bullet B$	F(A, B)	A + B
0	0	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	1	1

This is a complete proof using the truth table. We have used the truth table to compute F(A, B) and show it to be identical to A + B; thus $A \oplus B + A \bullet B = A + B$.

For another example, we try a standard proof of the consensus theorem. As in many standard algebra proofs, we only simplify by first making the expression more complex.
The Consensus Theorem: $X \bullet Y + \overline{X} \bullet Z + Y \bullet Z = X \bullet Y + \overline{X} \bullet Z$

$$
\begin{aligned}
X \bullet Y + \overline{X} \bullet Z + Y \bullet Z &= X \bullet Y \bullet (\overline{Z} + Z) + \overline{X} \bullet (\overline{Y} + Y) \bullet Z + (\overline{X} + X) \bullet Y \bullet Z \\
&= \overline{X} \bullet \overline{Y} \bullet Z + \overline{X} \bullet Y \bullet Z + \overline{X} \bullet Y \bullet Z + X \bullet Y \bullet Z + X \bullet Y \bullet \overline{Z} + X \bullet Y \bullet Z \\
&= \overline{X} \bullet \overline{Y} \bullet Z + \overline{X} \bullet Y \bullet Z + X \bullet Y \bullet \overline{Z} + X \bullet Y \bullet Z \\
&= \overline{X} \bullet (\overline{Y} + Y) \bullet Z + X \bullet Y \bullet (\overline{Z} + Z) \\
&= \overline{X} \bullet Z + X \bullet Y
\end{aligned}
$$

As an alternate proof, we consider two possible values of Z.

If Z = 0, the LHS becomes $X \bullet Y + \overline{X} \bullet Z + Y \bullet Z = X \bullet Y + \overline{X} \bullet 0 + Y \bullet 0 = X \bullet Y$
the RHS becomes $\overline{X} \bullet Z + X \bullet Y = \overline{X} \bullet 0 + X \bullet Y = X \bullet Y$

If Z = 1 the LHS becomes $X \bullet Y + \overline{X} \bullet Z + Y \bullet Z = X \bullet Y + \overline{X} \bullet 1 + Y \bullet 1 = \overline{X} + Y + X \bullet Y$
by the absorption theorem $Y + X \bullet Y = Y$, so the LHS = $\overline{X} + Y$
the RHS becomes $\overline{X} \bullet Z + X \bullet Y = \overline{X} \bullet 1 + X \bullet Y = \overline{X} + X \bullet Y$
by the absorption theorem $\overline{X} + X \bullet Y = \overline{X} + Y$, so the RHS = $\overline{X} + Y$.

As the LHS = RHS for each of Z = 0 and Z = 1, the theorem is proved.

We now consider one of the more interesting consequences of the principle of duality. Both standard algebra and Boolean algebra have **distributive postulates**. Standard algebra has one distributive postulate; Boolean algebra must have two distributive postulates as a result of the principle of duality.

In standard algebra, we have the equality A•(B + C) = A•B + A•C for all values of A, B, and C. This is the distributive postulate as seen in high school; we know it and expect it. Just for fun, we provide a simple proof of this almost-obvious postulate.

Let A = 0 A•(B + C) = 0•(B + C) = 0
A•B + A•C = 0•B + 0•C = 0 + 0 = 0
Let A = 1 A•(B + C) = 1•(B + C) = B + C
A•B + A•C = 1•B + 1•C = B + C

The dual of the postulate A•(B + C) = (A•B) + (A•C) is
A + B•C = (A + B)•(A + C)

The principle of duality states that if A•(B + C) = A•B + A•C is true then the dual statement A + B•C = (A + B)•(A + C) must also be true. We prove the second statement using a method unique to Boolean algebra. This method depends on the fact that there are only two possible values for A: A = 0 and A = 1. We consider both cases using a proof technique much favored by this instructor: consider both possibilities for one variable.

If A = 1, the statement becomes 1 + B•C = (1 + B)•(1 + C), or 1 = 1•1, obviously true.
If A = 0, the statement becomes 0 + B•C = (0 + B)•(0 + C), or B•C = B•C.

Just for fun, we offer a truth-table proof of the second distributive postulate.

A	B	C	B•C	**A + B•C**		(A + B)	(A + C)	**(A + B)•(A + C)**
0	0	0	0	**0**		0	0	**0**
0	0	1	0	**0**		0	1	**0**
0	1	0	0	**0**		1	0	**0**
0	1	1	1	**1**		1	1	**1**
1	0	0	0	**1**		1	1	**1**
1	0	1	0	**1**		1	1	**1**
1	1	0	0	**1**		1	1	**1**
1	1	1	1	**1**		1	1	**1**

Figure: A + B•C = (A + B)•(A + C)

Note that to complete the proof, one must construct the truth table, showing columns for each of the two functions A + B•C and (A + B)•(A + C), then note that the contents of the two columns are identical for each row.

Canonical Forms: Sum of Products (SOP) and Product of Sums (POS)

We conclude our discussion of Boolean algebra by giving a formal definition to two notations commonly used to represent Boolean functions:

SOP **S**um **o**f **P**roducts
POS **P**roduct **o**f **S**ums

The canonical forms are exactly equivalent to truth-tables. For example, consider the XOR function, shown in the following truth table.

A	B	F(A, B) = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

We have mechanical ways to produce the canonical forms for any given truth table. Using the methods (which will be discussed below), we produce two equivalent expressions.

$F(A, B) = \overline{A} \bullet B + A \bullet \overline{B}$

$F(A, B) = (A + B) \bullet (\overline{A} + \overline{B})$

We begin by assuming the definition of a **variable**. A **literal** is either a variable in its true form or its complemented form, examples are A, $\overline{B}$, and D. A **product term** is the logical AND of one or more literals or the logical OR of one or more literals. According to the strict definition, the single literal $\overline{B}$ is both a product term and sum term.

Before proceeding with the discussion, we mention one condition that is so obvious that this instructor has ignored it for years. This holds true for both sum terms and product terms. A given variable may appear only once in each term; specifically a variable and its negation may not appear in the same term. Whenever a variable appears twice in the same term, that term may be immediately simplified.

$A \bullet A$ simplifies to A $A + A$ simplifies to A
$A \bullet \overline{A}$ simplifies to 0 $\overline{A} + A$ simplifies to 1.

A **sum of products (SOP)** is the logical OR of one or more product terms.
A **product of sums (POS)** is the logical AND of one or more sum terms.

Sample SOP: $A + B \bullet C$ $A \bullet B + A \bullet C + B \bullet C$ $A \bullet B + A \bullet C + A \bullet B \bullet C$
Sample POS: $A \bullet (B + C)$ $(A + B) \bullet (A + C) \bullet (B + C)$ $(A + B) \bullet (A + C) \bullet (A + B + C)$

The student will note a few oddities in the above definition. These are present in order to avoid special cases in some of the more general theorems. The first oddity is the mention of the logical AND of one term and the logical OR of one term; each of these operators is a binary operator and requires two input variables. What we are saying here is that if we take a single literal as either a sum term or a product term, our notation is facilitated. Consider the expression (A + B + C), which is either a POS or SOP expression depending on its use. As a POS expression, it is a single sum term comprising three literals. As an SOP expression, it is the logical OR of three product terms, each of which comprising a single literal. For cases such as these the only rule is to have a consistent interpretation.

We now consider the concept of normal and canonical forms. These forms apply to both Sum of Products and Produce of Sums expressions, so we may have quite a variety of expression types including the following.

1. Boolean Expression.
2. Sum of Products, but not normal.
3. Product of Sums, but not normal.
4. Normal Sum of Products.
5. Normal Product of Sums.
6. Canonical Sum of Products.
7. Canonical Product of Sums.

Note that any Boolean expression can be converted to canonical form, either SOP or POS, but that does not imply that all Boolean expressions are in a specified form.

In order to define the concept of a normal form, we must consider the idea of inclusion. A product term X is **included** in another product term Y if every literal that is in X is also in Y. A sum term X is **included** in another sum term Y if every literal that is in X is also in Y. For inclusion, both terms must be product terms or both must be sum terms. Consider the SOP formula A•B + A•C + A•B•C. Note that the first term is included in the third term as is the second term. The third term A•B•C contains the first term A•B and the term A•C.

Consider the POS formula (A + B)•(A + C)•(A + B + C). Again, the first term (A + B) is included in the third term (A + B + C), as is the second term.

An extreme form of inclusion is observed when the expression has identical terms. Examples of this would be the SOP expression A•B + A•C + A•B and the POS expression (A + B)•(A + C)•(A + C). Each of these has duplicate terms, so that the inclusion is 2-way. The basic idea is that an expression with included (or duplicate) terms is not written in the simplest possible form. The idea of simplifying such expressions arises from the theorems of Boolean algebra, specifically the following two.

T1	Idempotency	a) X + X = X, for all X b) X • X = X, for all X
T3	Absorption	a) X + (X • Y) = X, for all X and Y b) X • (X + Y) = X, for all X and Y

Rule: Let X and Y be terms (either both sum terms or both product terms) in an expression. If term X is included in term Y (X $\subseteq$ Y), then term Y can be removed from the expression.

As a direct consequence of these theorems, we can perform the following simplifications.

A•B + A•C + A•B	= A•B + A•C	
(A + B)•(A + C)•(A + C)	= (A + B)•(A + C)	
A•B + A•C + A•B•C	= A•B + A•C	A•B $\subseteq$ A•B•C A•C $\subseteq$ A•B•C
(A + B)•(A + C)•(A + B + C)	= (A + B)•(A + C)	(A + B) $\subseteq$ (A + B + C) (A + C) $\subseteq$ (A + B + C)

We now consider these two formulae with slight alterations: $A \bullet B + A \bullet C + \overline{A} \bullet B \bullet C$ and $(A + B) \bullet (A + C) \bullet (\overline{A} + B + C)$. Since the literal must be included exactly, neither of formulae in this second set contains included terms. To be more specific, the term $A \bullet B$ is not included in the term $\overline{A} \bullet B \bullet C$, because the first term has the literal A while the second term has the literal $\overline{A}$. While the two literals A and $\overline{A}$ are based on the same variable, they are not the same literal, so there is no inclusion in either of these two expressions.

We now can produce the definitions of normal forms. A formula is in a normal form only if it contains no included terms; thus, a **normal SOP form** is a SOP form with no included terms and a **normal POS form** is a POS form with no included terms.

Sample Normal SOP:	$A + B \bullet C$	$A \bullet B + A \bullet C + B \bullet C$	$\overline{A} \bullet B + A \bullet C + B \bullet \overline{C}$
Sample Normal POS:	$A \bullet (B + C)$	$(A + B) \bullet (A + C) \bullet (B + C)$	$(\overline{A} + B) \bullet (A + \overline{C})$

We now can define the canonical forms. A normal form over a number of variables is in canonical form if every term contains each variable in either the true or complemented form. A **canonical SOP form** is a normal SOP form in which every product term contains a literal for every variable. A **canonical POS form** is a normal POS form in which every sum term in which every sum term contains a literal for every variable.

For example, consider a canonical expression (either SOP or POS) for a Boolean function of three variables: F(X, Y, Z). The three Boolean variables are X, Y, and Z. Thus, we have six possible literals based on these variables: X, $\overline{X}$, Y, $\overline{Y}$, Z, and $\overline{Z}$. If the expression is canonical, then every term contains exactly one of X or $\overline{X}$, exactly one of Y or $\overline{Y}$, and exactly one of Z or $\overline{Z}$; exactly three literals, one for each of the three variables.

Note that all canonical forms are also normal forms. Canonical forms correspond directly to truth tables and can be considered as one-to-one translations of truth tables. Here are the rules for converting truth tables to canonical forms.

To produce the Sum of Products representation from a truth table, follow this rule.

a) Generate a product term for each row where the value of the function is 1.
b) The variable is complemented if its value in the row is 0, otherwise it is not.

To produce the Product of Sums representation from a truth table, follow this rule.

a) Generate a sum term for each row where the value of the function is 0.
b) The variable is complemented if its value in the row is 1, otherwise it is not.

Note that the value of a Boolean function is either 0 or 1, so that every row of the truth table will either generate a product term for the SOP expression or a sum term for the POS. Part a) of each rule identifies the rows for which a term is required and part b) shows the method for generating that term. Put another way, every row in the truth table has either a 0 or a 1 under each variable; we must decide which of the two literals for that variable belong in the term.

As an example of conversion from a truth table to the normal forms, consider the two Boolean functions F1 and F2, each of three Boolean variables, denoted A, B, and C. Note that a truth table for a three-variable function requires $2^3 = 8$ rows.

Row	A	B	C	F1	F2
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

Recall that the truth table forms a complete specification of both F1 and F2. We may elect to represent each of F1 and F2 in either normal or canonical form, but that is not required. There are two ways to represent the canonical forms. We first present a pair of forms that this author calls the Σ–list and Π–list. The Σ–list is used to represent canonical SOP and the Π–list is used to represent canonical POS forms.

We should note the following concerning the names used here.

1) What this author calls a Σ–list is more commonly called a **minterm list**.
2) What this author calls a Π–list is more commonly called a **maxterm list**.

The origin of the names "minterm" and "maxterm" become obvious from the following observations. A **minterm** is the logical AND of one or more literals. If any of these literals evaluates to 0 (False), the minterm itself evaluates to 0, the minimum of the values. The same holds for a **maxterm**, if any term in the OR of one or more literals evaluates to 1 (True), then the maxterm itself evaluates to 1, the maximum of the values.

To generate the Σ–list, we just list the rows of the truth table for which the function has a value of 1. In the truth table, we have the following.

F1 has value 1 for rows 1, 2, 4, and 7; so $F1 = \Sigma(1, 2, 4, 7)$.
F2 has value 1 for rows 3, 5, 6, and 7; so $F2 = \Sigma(3, 5, 6, 7)$.

To generate the Π–list, we just list the rows of the truth table for which the function has a value of 0. In the truth table, we have the following.

F1 has value 1 for rows 0, 3, 5, and 6; so $F1 = \Pi(0, 3, 5, 6)$.
F2 has value 1 for rows 0, 1, 2, and 4; so $F2 = \Pi(0, 1, 2, 4)$.

The student should note that, for each of F1 and F2, any term is either in the Σ–list or the Π–list. This reflects the fact that for each row in the truth table a Boolean function can have only one of two values: 0 (placing the row in the Π–list) or 1 (placing it in the Σ–list).

Note that conversion directly between the Σ–list and Π–list forms is easy if one knows how many variables are involved. Here we have 3 variables, with 8 rows numbered 0 through 7. Thus, if F1 = Σ(1, 2, 4, 7), then F1 = Π(0, 3, 5, 6) because the numbers 0, 3, 5, and 6 are the only numbers in the range from 0 to 7 inclusive that are not in the Σ–list. The conversion rule works both ways. If F2 = Π(0, 1, 2, 4), then F2 = Σ(3, 5, 6, 7) because the numbers 3, 5, 6, and 7 are the only numbers in the range from 0 to 7 inclusive not in the Π–list.

We now address the generation of the canonical SOP form of F1 from the truth table. The rule is to generate a product term for each row for which the function value is 1. Reading from the top of the truth table, the first row of interest is row 1.

A	B	C	F1
0	0	1	1

We have A = 0 for this row, so the corresponding literal in the product term is $\overline{A}$.
We have B = 0 for this row, so the corresponding literal in the product term is $\overline{B}$.
We have C = 1 for this row, so the corresponding literal in the product term is C.
The product term generated for this row in the truth table is $\overline{A} \bullet \overline{B} \bullet C$.

We now address the generation of the canonical POS form of F1 from the truth table. The rule is to generate a sum term for each row for which the function value is 0. Reading from the top of the truth table, the first row of interest is row 0.

A	B	C	F1
0	0	0	0

We have A = 0 for this row, so the corresponding literal in the sum term is A.
We have B = 0 for this row, so the corresponding literal in the sum term is B.
We have C = 0 for this row, so the corresponding literal in the sum term is C.
The sum term generated for this row in the truth table is A + B + C.

Consider another row in the generation of the POS form of F1.

A	B	C	F1
0	1	1	0

We have A = 0 for this row, so the corresponding literal in the sum term is A.
We have B = 1 for this row, so the corresponding literal in the sum term is $\overline{B}$.
We have C = 1 for this row, so the corresponding literal in the sum term is $\overline{C}$.
The sum term generated for this row in the truth table is A + $\overline{B}$ + $\overline{C}$.

Thus we have the following representation as Sum of Products

F1 = $\overline{A} \bullet \overline{B} \bullet C + \overline{A} \bullet B \bullet \overline{C} + A \bullet \overline{B} \bullet \overline{C} + A \bullet B \bullet C$

F2 = $\overline{A} \bullet B \bullet C + A \bullet \overline{B} \bullet C + A \bullet B \bullet \overline{C} + A \bullet B \bullet C$

We have the following representation as Product of Sums

F1 = $(A + B + C) \bullet (A + \overline{B} + \overline{C}) \bullet (\overline{A} + B + \overline{C}) \bullet (\overline{A} + \overline{B} + C)$

F2 = $(A + B + C) \bullet (A + B + \overline{C}) \bullet (A + \overline{B} + C) \bullet (\overline{A} + B + C)$

The choice of representations depends on the number of terms. For N Boolean variables Let C_{SOP} be the number of terms in the Sum of Products representation. Let C_{POS} be the number of terms in the Product of Sums representation. Then $C_{SOP} + C_{POS} = 2^N$. In the above example $C_{SOP} = 4$ and $C_{POS} = 4$, so either choice is equally good. However, if the Sum of Products representation has 6 terms, then the Product of Sums representation has only 2 terms, which might recommend it.

Non-Canonical Forms

The basic definition of a canonical form (both SOP and POS) is that every term contain each variable, either in the negated or non-negated state. In the example above, note that every product term in the SOP form contains all three variables A, B, and C in some form. The same holds for the POS forms.

It is often the case that a non-canonical form is simpler. For example, one can easily show that F2(A, B, C) = A•B + A•C + B•C. Note that in this simplified form, not all variables appear in each of the terms. Thus, this is a normal SOP form, but not a canonical form.

The simplification can be done by one of three ways: algebraic methods, Karnaugh Maps (K-Maps), or the Quine-McCluskey procedure. Here we present a simplification of F2(A, B, C) by the algebraic method with notes to the appropriate postulates and theorems.

The idempotency theorem, states that for any Boolean expression X, we have X + X = X. Thus X = X + X = X + X + X and A•B•C = A•B•C + A•B•C + A•B•C, as A•B•C is a valid Boolean expression for any values of A, B, and C.

F2	= A'•B•C	+ A•B'•C	+ A•B•C'	+ A•B•C	Original form
	= A'•B•C	+ A•B'•C	+ A•B•C'	+ A•B•C + A•B•C + A•B•C	Idempotency
	= A'•B•C	+ A•B•C	+ A•B'•C	+ A•B•C + A•B•C' + A•B•C	Commutativity
	= (A' +A)•B•C		+ A•(B' + B)•C	+ A•B•(C' + C)	Distributivity
	= 1•B•C	+ A•1•C	+ A•B•1		
	= B•C	+ A•C	+ A•B		
	= A•B	+ A•C	+ B•C		Commutativity

The main reason to simplify Boolean expressions is to reduce the number of logic gates required to implement the expressions. This was very important in the early days of computer engineering when the gates themselves were costly and prone to failure. It is less of a concern these days.

More On Conversion Between Truth Tables and Boolean Expressions

We now give a brief discussion on the methods used by this author to derive Boolean expressions from truth tables and truth tables from canonical form Boolean expressions. We shall do this be example. Consider the truth table expression for our function F1.

The truth table for F1 is shown below.

Row	A	B	C	F1
0	0	0	0	0
1	**0**	**0**	**1**	**1**
2	**0**	**1**	**0**	**1**
3	0	1	1	0
4	**1**	**0**	**0**	**1**
5	1	0	1	0
6	1	1	0	0
7	**1**	**1**	**1**	**1**

The rows for which the function F1 has value 1 are shown in bold font. As noted above, we can write F1 in the Σ-list form as F1 = Σ(1, 2, 4, 7). To convert this form to canonical SOP, we just replace the decimal numbers by binary; thus F1 = Σ(001, 010, 100, 111). To work from the truth tables, we just note the values of A, B, and C in the selected rows.

First write the Boolean expression in a form that cannot be correct, writing one identical Boolean product term for each of the four rows for which F1 = 1.

```
F1(A, B, C) = A•B•C + A•B•C + A•B•C + A•B•C
```

Then write under each term the 0's and 1's for the corresponding row.

```
F1(A, B, C) = A•B•C + A•B•C + A•B•C + A•B•C
              0 0 1   0 1 0   1 0 0   1 1 1
```

Wherever one sees a 0, complement the corresponding variable, to get.

```
F1(A, B, C) = A'•B'•C + A'•B•C' + A•B'•C' + A•B•C
```

To produce the truth-table from the canonical form SOP expression, just write a 0 under every complemented variable and a 1 under each variable that is not complemented.

```
F2(A, B, C) = A'•B•C + A•B'•C + A•B•C' + A•B•C
              0 1 1   1 0 1    1 1 0   1 1 1
```

This function can be written as F2 = Σ(011, 101, 110, 111) in binary or F2 = Σ(3, 5, 6, 7). To create the truth table for this, just make 8 rows and fill rows 3, 5, 6, and 7 with 1's and the other rows with 0's.

It is also possible to generate the canonical POS expressions using a similar procedure. Consider again the truth table for F1, this time with the rows with 0 values highlighted.

Row	A	B	C	F1
0	**0**	**0**	**0**	**0**
1	0	0	1	1
2	0	1	0	1
3	**0**	**1**	**1**	**0**
4	1	0	0	1
5	**1**	**0**	**1**	**0**
6	**1**	**1**	**0**	**0**
7	1	1	1	1

The rows for which the function F1 has value 0 are shown in bold font. As noted above, we can write F1 in the Π-list form as F1 = Π(0, 3, 5, 6). To convert this form to canonical POS, we just replace the decimal numbers by binary; thus F1 = Π(000, 011, 101, 110). To work from the truth tables, we just note the values of A, B, and C in the selected rows.

Again we write a Boolean expression with one sum term for each of the four rows for which the function has value 0. At the start, this is not correct.

F1 = (A + B + C) • (A + B + C) • (A + B + C) • (A + B + C)

Now write the expression with 0's and 1's for the corresponding row numbers.

```
F1 = (A + B + C) • (A + B + C) • (A + B + C) • (A + B + C)
       0   0   0     0   1   1     1   0   1     1   1   0
```

Wherever one sees a 1, complement the corresponding variable, to get.

F1 = (A + B + C)•(A + B′ + C′)•(A′ + B + C′)•(A′ + B′ + C)

To produce the truth-table from the canonical form POS expression, just write a 1 under every complemented variable and a 0 under each variable that is not complemented.

```
F2 = (A + B + C)•(A + B + C′)•(A + B′ + C)•(A′ + B + C)
       0   0   0   0   0   1    0   1    0   1   0    0
```

This function can be written as F2 = Π(000, 001, 010, 100) in binary or F2 = Π(0, 1, 2, 4). To create the truth table for this, just make 8 rows and fill rows 0, 1, 2, and 4 with 0's and the other rows with 1's.

We now solve a number of problems as taken from another textbook. The question for each of these is what form, if any, are these expressions in.

F(X, Y, Z) = X•Y' + Y•Z' + Z•Y'
This is a normal SOP expression. The last product term could easily be written as Y'•Z and normally would be so written; however, the order of literals is not important in determining the form of an expression. The two terms Y'•Z and Y•Z' both contain the variables Y and Z, but the literals are different so we have no included terms.

F(A, B, C, D) = (A + B' + C)•(A' + C' + D)•(A' + C')
This is obviously a POS expression. It is also obvious that it is not a canonical form. The first term by itself shows that the form is not canonical; it lacks a literal for the variable D. We now note the second and third terms and see that the third term is included in the second term, so the form is not normal. **The answer is POS form, not a normal form**.

As an exercise, we convert the expression (A + B' + C)•(A' + C' + D)•(A' + C') into a normal form. We use a form of the absorption theorem X•(X + Y) = X for any X and Y. We see the truth of this theorem by considering two cases X = 0 and X = 1. For X = 0, the identity becomes 0•(0 + Y) = 0 and for X = 1 it becomes 1•(1 + Y) = 1, both true. We now consider the above with X = A' + C' and Y = D; thus (A' + C' + D)•(A' + C') = (A' + C') and we obtain F(A, B, C, D) = (A + B' + C)•(A' + C').

F(P, Q, R) = P•Q' + Q•R'•(P + Q') + (R' + Q')
This is not either a POS form or a SOP form, although many students wish to label it a SOP form as it can be easily converted to SOP.

Again as an exercise, we convert the expression P•Q' + Q•R'•(P + Q') + (R' + Q') to a normal form.

F(P, Q, R) = P•Q' + Q•R'•(P + Q') + (R' + Q')
= P•Q' + P•Q•R' + Q•Q'•R' + R' + Q'
= P•Q' + P•Q•R' + R' + Q' as Q•Q'•R' = 0 for any value of Q
= P•Q' + Q' + P•Q•R' + R'
= (P + 1)•Q' + (P•Q + 1)•R'
= Q' + R' as 1 + X = 1 for any literal X.

F(A, B, C) = (A + B + C')•(A' + B' + C')•(A' + B + C')
This is a canonical POS expression. We note that it can be simplified , using the observation that (X + Y)•(X + Y') = X for any X and Y (for Y = 0, the left hand side of the identity is (X + 0)•(X + 1) = X; for Y = 1 it is (X + 1)•(X + 0) = X). To simplify, let X = A' + C' and Y = B. So (A' + B' + C')•(A' + B + C') = (A' + C') and F = (A + B + C')•(A' + C').

As another example, we consider the problem of conversion to canonical forms. This involves use of the Boolean identities X•1 = X for all X and (X' + X) = 1 for all X; thus, for all X and Y we have Y = Y•1 = Y•(X' + X) = Y•X' + Y•X

Problem:
Express F(A, B, C) = (A + B')•C + A'•C in canonical SOP and canonical POS.

Solution:
The expression, as written, is not in a normal form. The first step is to convert it to either normal SOP or normal POS. Most people are more comfortable with conversion into SOP, so that is the approach we shall take. The first two steps are simple

F(A, B, C) = (A + B')•C' + A'•C
= A•C' + B'•C' + A'•C this is Normal SOP
= A'•C + A•C' + B'•C' rearranged for simpler representation.

We note that there is no theoretical justification for the latter rearrangement. In practice, it is found that a systematic way of writing terms in a Boolean expression leads to less confusion. This convention is to place A' first, then A, then B', etc.

We then apply the rule Y = Y•1 = Y•(X' + X) three times to get
F(A, B, C) = A'•(B' + B)•C + A•(B' + B)•C' + (A' + A)•B'•C'
= A'•B'•C + A'•B•C + A•B'•C' + A•B•C' + A'•B'•C' + A•B'•C'

This may appear to be in Canonical SOP, but it is not. To see why, rearrange the terms.
F(A, B, C) = A'•B'•C' + A'•B'•C + A'•B•C + A•B'•C' + A•B'•C' + A•B•C'

By converting to a standard order, we find a repeated term. This can be removed to produce the Canonical SOP representation of the function.
F(A, B, C) = A'•B'•C' + A'•B'•C + A'•B•C + A•B'•C' + A•B•C'

The conversion to Canonical POS is facilitated by a trick based on the rule for producing a Canonical SOP form from a truth table. Here we replace a negated variable with a 0 and a variable in the true form by a 1 (this is the SOP rule) to get the following list.

000 001 011 100 110

To convert to Canonical POS, note that the above list has 5 of the eight three-bit unsigned integers. We note the three missing numbers 010, 101, and 111. We then apply the POS rule that 1 stands for a negated variable and 0 for a variable in the true form.

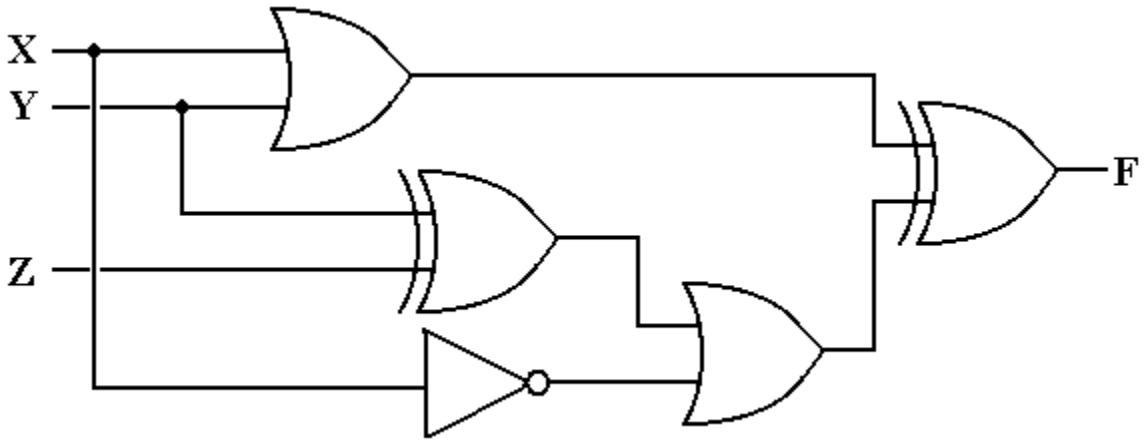
010 101 111
(A + B' + C)(A' + B + C')(A' + B'+C')

Thus F(A, B, C) = (A + B' + C)•(A' + B + C')•(A' + B'+C') – Canonical POS.

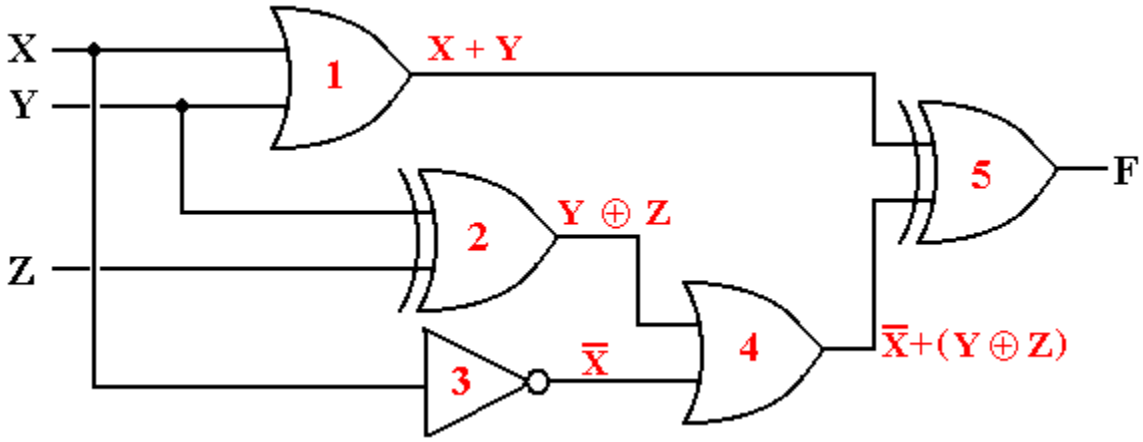
Circuits and Truth Tables

We now address an obvious problem – how to relate circuits to Boolean expressions. The best way to do this is to work an example. Here it is.

What is the truth table and the Boolean expressions that describe the following circuit.



The method to get the answer is to label each gate and determine the output of each. The following diagram shows the gates as labeled and the output of each gate.



The outputs of each gate are as follows:

The output of gate 1 is (X + Y),
The output of gate 2 is (Y ⊕ Z),
The output of gate 3 is X',
The output of gate 4 is X' + (Y ⊕ Z), and
The output of gate 5 is (X + Y) ⊕ [X' + (Y ⊕ Z)]

We now produce the truth table for the function.

X	Y	Z	**X + Y**	(Y ⊕ Z)	X'	**X' + (Y ⊕ Z)**	(X + Y) ⊕ [X'+(Y ⊕ Z)]
0	0	0	**0**	0	1	**1**	1
0	0	10	**0**	1	1	**1**	1
0	1	0	**1**	1	1	**1**	0
0	1	1	**1**	0	1	**1**	0
1	0	0	**1**	0	0	**0**	1
1	0	1	**1**	1	0	**1**	0
1	1	0	**1**	1	0	**1**	0
1	1	1	**1**	0	0	**0**	1

The above is the truth table for the function realized by the figure on the previous page.

Lets give a simpler representation.

X	Y	Z	F(X, Y, Z)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

We now produce both the SOP and POS representations of this function. For the SOP, we look at the four 1's of the function; for the POS, we look at the four zeroes.

SOP

$$F(X, Y, Z) = X'\bullet Y'\bullet Z' + X'\bullet Y'\bullet Z + X\bullet Y'\bullet Z' + X\bullet Y\bullet Z$$
$$0\,0\,0 \qquad 0\,0\,1 \qquad 1\,0\,0 \qquad 1\,1\,1$$

POS

$$F(X, Y, Z) = (X + Y' + Z) \bullet (X + Y' + Z') \bullet (X' + Y + Z') \bullet (X' + Y' + Z)$$
$$0\,1\,0 \qquad 0\,1\,1 \qquad 1\,0\,1 \qquad 1\,1\,0$$

To simplify in SOP, we write the function in a slightly more complex form.

$$F(X, Y, Z) = X'\bullet Y'\bullet Z' + X'\bullet Y'\bullet Z + X'\bullet Y'\bullet Z' + X\bullet Y'\bullet Z' + X\bullet Y\bullet Z$$
$$= X'\bullet Y'\bullet(Z' + Z) + (X + X')\bullet Y'\bullet Z' + X\bullet Y\bullet Z$$
$$= X'\bullet Y' + Y'\bullet Z' + X\bullet Y\bullet Z$$

To simplify in POS, we again write the function in a slightly bizarre form.

$$F(X, Y, Z) = (X + Y' + Z) \bullet (X + Y' + Z') \bullet (X' + Y + Z')$$
$$\bullet (X' + Y' + Z) \bullet (X + Y' + Z)$$
$$= (X + Y') \bullet (X' + Y + Z')\bullet (Y' + Z)$$

The Secret Revealed

These notes have focused on two apparently arbitrary Boolean functions, called F1 and F2. The reason for not identifying these functions was to encourage the student to focus on the manipulation of truth tables and Boolean expressions independently of the meaning or significance of those expressions. We now reveal the secret identity of F1 and F2. The function F1 is the sum from a full adder and the function F2 is the carry out of a full adder. The following is a simple circuit implementing the Sum (F1) and Carry (F2) of the full adder. In binary arithmetic, note that 1 + 1 = 0 with a carry of 1.

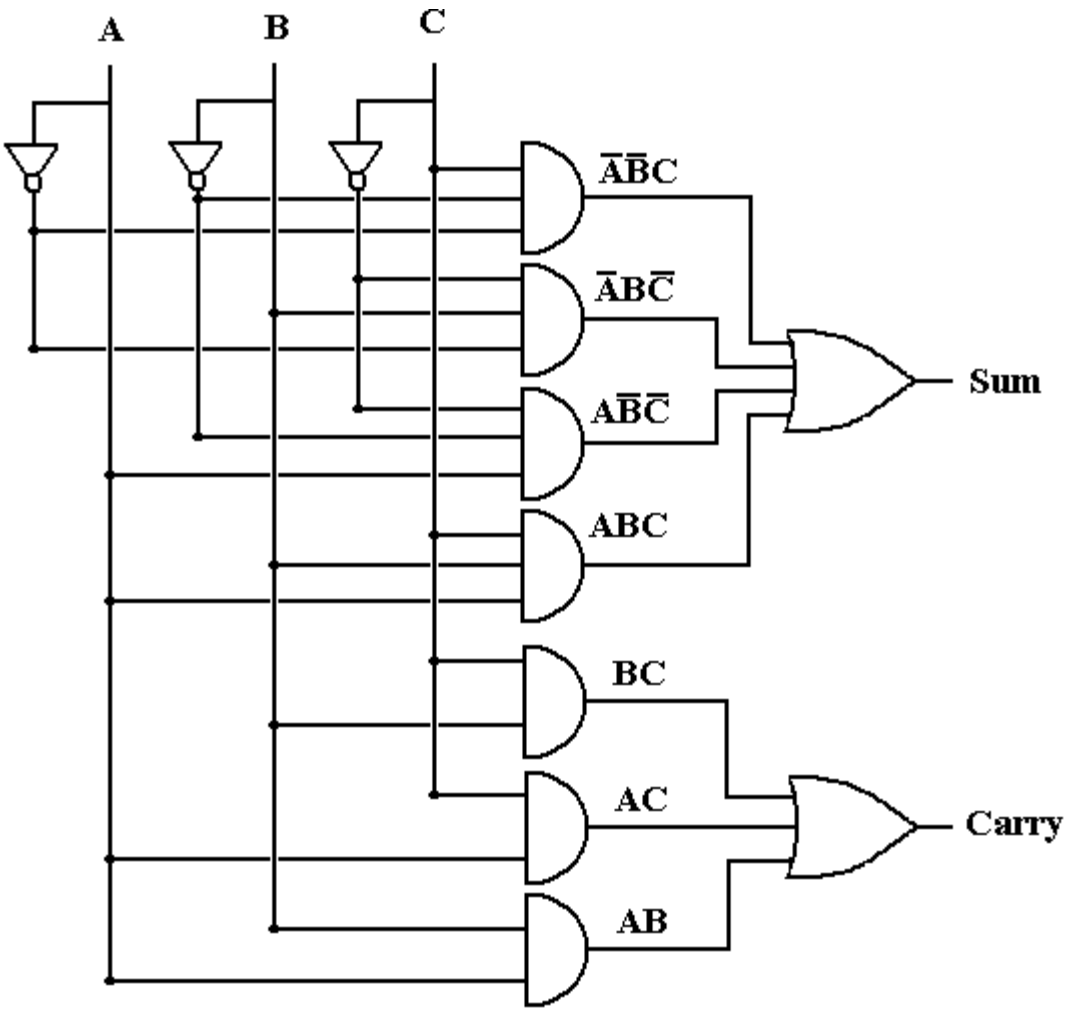


Figure: A Basic Implementation of a Full Adder.