Elements of Boolean Algebra

Intended as a Resource for Electrical Engineers and Other Practitioners of the Boolean Arts, 2nd ed.

James M. Cargal

What is Known

It is hoped that you have been exposed to Boolean Algebra already. It is assumed that you have been exposed to elementary set theory – to unions, intersections, complements, and Venn diagrams (although I will not employ them here). You should have facility with truth tables. In general we want algebraic arguments, but truth tables are always a fallback; they will work as a last resort or as confirmation. Here we will use truth tables primarily for definitions. But we will get surprising mileage out of them.

What is Boolean Algebra?

Boolean algebra concerns an arithmetic with just two values. These values are often denoted by T and F, where T stands for "true" and F stands for "false". Usually we will use 1 in place of T and 0 in place of F. However it is very useful to remember the logical connotations of true and false.

If we have an arithmetic of just 0s and 1s our addition and multiplication tables would be as follows:

+	1	0
1	?	1
0	1	0

and

·	1	0
1	1	0
0	0	0

That is everything is as in ordinary arithmetic except for 1 + 1. Ordinarily the answer is 2, but we only have 0 and 1 as possible answers. So which do we take? The proper answer is: either one. If we have 1 + 1 = 1 then we essentially are working in a Boolean Algebra, but if on the other hand we take 1 + 1 = 0 we are working in a Boolean Ring. Both perspectives are useful and eventually we will combine them, meaning we will have two types of addition operating simultaneously with different notation for the two types of addition.

Boolean Functions

A Boolean function is a function on Boolean values that yields a Boolean value. In other words, the input consists of 0s and 1s and so does the output. [I know that people tend to write 0's instead of 0s, but the apostrophe is meaningless here.] For a generic function I will use the symbol □. For example, a binary function might be written as follows: z = □(x, y). However, for binary functions I will use the usual convention of writing z = x □ y. Binary functions will in fact be our main focus. Two of the most important functions will be an “and” function and an “or” function. It can become extremely awkward to speak about, say, the and and the or function. When speaking in the text about these functions I will use a convention of uppercase type in a different font. For example I could write: The AND and OR functions are commutative.

We are keenly interested in how many Boolean functions there are, and this is a fairly easy question to answer. Consider a Boolean function of order n, meaning it has n input variables; we could denote a generic case as $f(x_1, x_2, x_3, \ldots x_n)$. Since each input variable can take on only 2 values, there are 2^n possible different inputs. And for each input there are 2 possible outputs, which means that there are $2^{\left(2^n\right)}$ such functions. We are interested in all such functions but primarily binary functions of which there are $2^{(2^2)} = 2^4 = 16$; these we will study in great detail. One reason we are interested in binary functions is because of disjunctive normal form.

Boolean Algebra and Disjunctive Normal Form

Boolean algebra is often built upon three operators (functions) which we will formally define later: AND, OR, and NOT. Typically AND is denoted by $\wedge$ and OR is denoted by $\vee$; and the NOT operator which is denoted by $\neg$. For example if we wanted to write "(P and Q) or NOT P", we would write it as $(P \wedge Q) \vee (\neg P)$.

Disjunctive normal form uses these three operators. Again, we will formally define these operators later, so if necessary you can come back and read this section then. However, the point of disjunctive normal form is that we can use it to express any Boolean function. Suppose, for example, that F is a Boolean function of four Boolean variables. We can write F in terms of the three operations, $\wedge \vee \neg$, if we simply list every combination of the four variables, say, A, B, C, and D, that make F true. Now there are $2^4 = 16$ cases that might come up, but in this example there are three ways in which F can be true. Suppose F is true if and only if:

1. A is true, B is true, C is true, and D is false, or
2. A and B are true and C and D are false, or
3. A is false, B is true, C is true, D is false.

This can be written as: :

$$F(A,B,C,D) = (A \wedge B \wedge C \wedge (\neg D)) \vee (A \wedge B \wedge (\neg C) \wedge (\neg D)) \vee ((\neg A) \wedge B \wedge C \wedge (\neg D)).$$

It may be easier to see in the notation that we will use instead of the usual $\wedge \vee \neg$. (This is a rude fact of Boolean algebra, there are many notational conventions. It is important to know them all and to be able to switch between them automatically.) We will denote the operator OR by +, rather than by ∨. That is, X OR Y will be written X + Y. Similarly, AND will be represented by multiplication, rather than ∧. That is, X AND Y will be written X·Y or just XY. NOT X will be written with an over bar as $\overline{X}$ rather than ¬X. Using this notation we can rewrite our expression as $F(A,B,C,D) = ABC\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}BC\overline{D}$. In principle any Boolean function can be expressed in this way. Not only can we express any Boolean function using just these three operators but Boolean expressions are not unique. The laws that I am about to show you are redundant in that all of the laws can be derived from a smaller subset of the laws. However, it is useful to list the laws as follows:

Associativity

$$(A+B)+C = A+(B+C)$$
$$(AB)C = A(BC)$$

Commutivity

$$A+B = B+A$$
$$AB = BA$$

Distribution

$$A(B+C) = AB+AC$$
$$A+(BC) = (A+B)(A+C)$$

De Morgan's Laws

$$\overline{A+B} = \overline{A}\,\overline{B}$$
$$\overline{AB} = \overline{A}+\overline{B}$$

Negation

$$\overline{1} = 0$$
$$\overline{0} = 1$$

Identity

$$X \cdot 1 = X$$
$$X + 0 = X$$

Double Negation

$$\overline{\overline{X}} = X$$

Complementarity

$$X + \overline{X} = 1$$
$$X \cdot \overline{X} = 0$$

Absorption

$X+X=X$

$X\cdot X=X$

$X+1=1$

$X\cdot 0=0$

Duality

In Boolean Algebra, not only does multiplication distribute over addition but addition distributes over multiplication. I find this fact and the DeMorgan laws to be most useful. These laws also show a very important characteristic – duality. In all but one case, which is self-dual, you have two laws that are duals of one another. To get the dual of a formula one simply exchanges the AND and OR symbols and also the 1s and 0s. When we prove a theorem in Boolean algebra, the dual of the theorem is itself a theorem which follows from the dual of the proof which is itself a proof. In particular the dual of a true statement is true, and that fact is very useful.

Do not confuse duality with negation. Given the expression $A + B$, its negation is $\overline{A + B}$ which by the De Morgan Laws is $\overline{A}\cdot\overline{B}$. However, the dual is just AB.

Stone's Theorem

The reader might notice something odd. The laws just given for Boolean algebra also hold for sets. If we take the laws above and replace + (also denoted $\vee$) by $\cup$ and $\cdot$ (also denoted $\wedge$) by $\cap$ and 1 by U (for the universal set) and 0 by $\varnothing$, then we have the laws of set theory. A theorem by Marshall Stone says that Boolean algebra and set theory are in fact different views of the same thing. In fact these very same laws show up in still other guises.

Boolean Real Functions

I want to show you something that is not generally mentioned in the literature. Modern programming languages usually have a Boolean data type where in particular 1 + 1 = 1 (perhaps expressed in another form). However, you can create such a function in ordinary algebra as follows:

$$X\ OR\ Y\ =\ X - XY + Y$$
$$X\ AND\ Y\ =\ XY$$
$$NOT\ X\ =\ 1 - X$$

For example, in the first case, X OR Y, if either X or Y is equal to 1 then that is the output. Otherwise the function yields 0. It follows from using disjunctive normal form that any logical operator can be encoded with such a function. I call these "Boolean real functions" and I find them very useful. Later when we cover all of the basic truth tables I will provide these with the denotation: BRF.

An Example of A Truth Table

The truth table for OR is as follows:

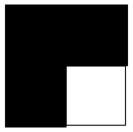
	1	0
1	1	1
0	1	0

This says that 0 + 0 is 0 (or $0 \vee 0 = 0$ in the older notation) and all the other cases give you 1. This is precisely how we use OR in real life; p OR q is true if and only if either p or q is true. However, in this essay we are going to depart from the usual truth tables and use iconic truth tables. For example, the truth table for OR will be given in the form:

Name: OR

Symbol: $p + q$, $p \vee q$

BRF: $p + q - p \cdot q$



Here OR is a name for the function. Not all of the functions have names. + and $\vee$ are common symbols for this function. Not all of the functions have symbols. $p + q - p \cdot q$ is a corresponding Boolean Real Function (see above, if you missed this topic). We always assume that the variable on the left is p and q is the variable on the right. There is always a Boolean Real Function, and there is always an icon.

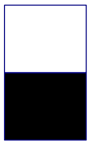
The Unary Function

Before enumerating our 16 binary functions, there is a terribly important function of one variable:

Name: NOT

Symbol: $\neg p$, $\sim p$, $\overline{p}$

BRF: $1 - p$

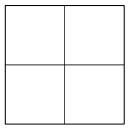


Specifically NOT 1 = 0, and NOT 0 = 1; not true is false and not false is true.

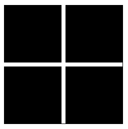
Degeneracy

Of the 16 binary Boolean operators, 6 are degenerate. They are in fact functions of 1 variable or are functions of no variables. They are pretentious and are trying to pass themselves off as binary functions. They are included here for completeness.

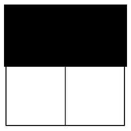
1.
Name: False
Symbol: 0, F
BRF: 0



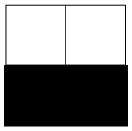
2.
Name: True
Symbol: 1, T
BRF: 1



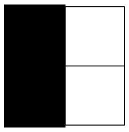
3.
Name: n/a
Symbol: n/a
BRF: p



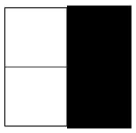
4.
Name: n/a
Symbol: n/a
BRF: 1 − p



5.
Name: n/a
Symbol: n/a
BRF: q



6.
Name: n/a
Symbol: n/a
BRF: 1 − q

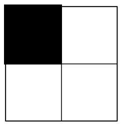


And

Although there is only one AND function, I consider 4 functions in the AND class of functions. They are:

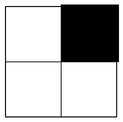
7.

Name:	AND
Symbol:	p·q, p/\q
BRF:	$p \cdot q$



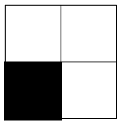
8.

Name:	n/a
Symbol:	n/a
BRF:	$p - p \cdot q$



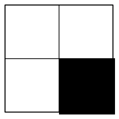
9.

Name:	n/a
Symbol:	n/a
BRF:	$q - p \cdot q$



10.

Name:	NOR
Symbol:	p ↓ q
BRF:	$1 - p - q + p \cdot q$

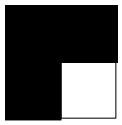


These functions are, respectively, p and q; p and not q; not p and q, and not p and not q.

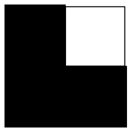
Or

There are also 4 functions in the OR class. However, I regard these same 4 functions as being in the IMPLIES class. These functions are:

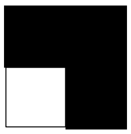
11.
Name: OR
Symbol: p+q, p∨q
BRF: $p+q-p\cdot q$



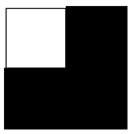
12.
Name: IMPLIES
Symbol: p → q
BRF $1-p\ +p\cdot q$



13.
Name: IMPLIED BY
Symbol: p ← q
BRF: $1-q\ +p\cdot q$



14.
Name: NAND
Symbol: p ↑ q
BRF: $1-p\cdot q$



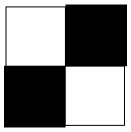
These functions are, respectively, p or q; p or (not q); (not p) or q; and (not p) or (not q). The function (not p) or q is better known as p implies q.. However, each OR is also an implication. For example if we have that “A or B” is true then we know that if A is false, B must be true. That is “A or B” is equivalent to “not A implies B.” By similar reasoning “A or B” is also equivalent to “not B implies A.” More succinctly, I find these two identities very useful:

$$\boxed{A\vee B=\overline{A}\rightarrow B\ ;\quad A\rightarrow B=\overline{A}\vee B}$$

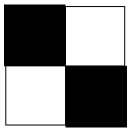
Exclusive Or and Equivalence

The last two functions are a central topic of this essay. They are:

15.
Name: XOR
Symbol: $p \oplus q$, $p \veebar q$
BRF: $p+q-2p\cdot q$



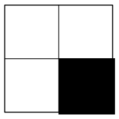
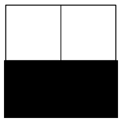
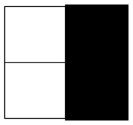
16
Name: EQUIV
Symbol: $p \equiv q$, $p \leftrightarrow q$
BRF: $1-p-q+2p\cdot q$

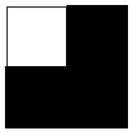
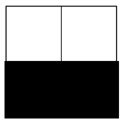
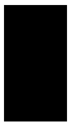


DeMorgan's Laws Revisited

$$\overline{A+B} = \overline{A}\,\overline{B}$$
$$\overline{AB} = \overline{A}+\overline{B}$$

Again, DeMorgan's laws are amongst the most important in logic manipulation. These laws (which I repeat here) are duals of one another. They essentially are laws about complements.

The first law says that  is  AND  .

The second law says that  is  OR  .

Ternary Functions and Beyond

Of course, we can have Boolean functions of three or more variables. Consider, the AND function. As a function of more than two variables it would probably translate to ALL_OF. However, ALL_OF (p,q,r) or if you prefer AND(p,q,r) is equivalent to $p \cdot q \cdot r$. We do not need to group the terms because (binary) AND is associative. Similarly, OR generalized would seem to translate to AT_LEAST_ONE_OF. In this case OR(p,q,r) is equivalent to $p + q + r$. Again we do need to group the terms as (binary) OR is also associative. Note that the functions ALL_OF and AT_LEAST_ONE_OF can be defined without specifying a particular number of variables. However, if we look at XOR (exclusive or, which we denote by ⊕) things are a little more complex. The obvious generalization of it would be to the function EXACTLY_ONE_OF. Note however, that EXACTLY_ONE_OF(p,q,r) is not $p \oplus q \oplus r$ (though XOR is also associative). Rather it is $p \oplus q \oplus r \oplus pqr$. Similar, problems apply to EQUIV (which we denote ≡). Hence, generalizing to more than two variables can be tricky. However, because of disjunctive normal form we can still rely entirely on binary functions (and NOT).

Computationally Complete Sets of Functions

Disjunctive normal form implies that we can do all of our logic functions with NOT, AND, and OR. Hence we say that these three functions are "Computationally complete." However, they are not a minimal computationally complete set. By DeMorgan's laws we have $A + B = \overline{\overline{A} \cdot \overline{B}}$ and $A \cdot B = \overline{\overline{A} + \overline{B}}$. Hence, {NOT and OR} and {NOT and AND} are both computationally complete sets of functions.

It follows that NAND $(\uparrow)$ and NOR $(\downarrow)$ are each also computationally complete. In the case of NAND we have $A \uparrow A = \overline{A}$. Hence $(A \uparrow A) \uparrow (B \uparrow B) = A + B$.

Similarly, for NOR, $A \downarrow A = \overline{A}$, and $\left(A \downarrow A\right) \downarrow \left(B \downarrow B\right) = A \cdot B$. It has been known for more than a century that these are the only functions which are computationally correct by themselves. However, this is not correct. The problem with, say, the implication function $\left(\rightarrow\right)$ is that you cannot use it to indicate NOT X. Except that you can: $X \rightarrow 0 = \overline{X}$. Combining this with the aforementioned fact that $X + Y = \overline{X} \rightarrow Y$, we have that IMPLIES is also computationally complete by itself.

The reason we generally do not rely on NAND or NOR by themselves is that (as truth tables show) neither is associative, e.g. $\left(A \uparrow B\right) \uparrow C \neq A \uparrow \left(B \uparrow C\right)$. As for IMPLIES, it is neither associative nor commutative. Nonetheless it is tempting to do Boolean algebra either with just IMPLIES or with IMPLIES in conjunction with something else. Note that IMPLIES distributes over both AND and OR. In the latter case we also have $\left(A \rightarrow B\right) + C = \left(\overline{A} + B\right) + C = \overline{A} + (B + C) = A \rightarrow (B + C)$. One last aside before going on: DeMorgan's laws can be written as

$$\begin{aligned} A \uparrow B &= \overline{A} + \overline{B} \\ A \downarrow B &= \overline{A} \cdot \overline{B} \end{aligned}.$$

More Duality

Remember duality is based upon interchanging AND and OR and also interchanging 0 and 1. Hence to find the dual of XOR we need to convert it to ANDs and ORs (perhaps by using disjunctive normal form). We get $A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$. Its dual is $\left(A+\overline{B}\right) \cdot \left(\overline{A}+B\right)$.

Multiplying this out we get that the dual of $A \oplus B$ is $A \cdot \overline{B} + \overline{A} \cdot B$. That happens to be equivalent to equivalence itself (EQUIV) or $A \equiv B$. Note that *XOR* and EQUIV are simultaneously duals and complements of one another. This is not generally the case. Again, the complement of $A+B$ is $\overline{A+B} = A \downarrow B$ which by DeMorgan's laws is $\overline{A} \cdot \overline{B}$. However, the dual of $A+B$ is just $A \cdot B$.

Boolean Rings

It cannot be overemphasized that Boolean algebra is usually performed with AND, OR, and NOT. There is an alternative algebra known as a "Boolean ring" (because in abstract algebra it is what is known as a "ring"). Boolean rings have huge advantages over ordinary Boolean algebra and commensurate disadvantages. In a Boolean ring, we rely on XOR $\left(\oplus\right)$ and AND.

NOT X is $1 \oplus X$. However, I find the NOT operator useful as well, but technically we no longer need it. The laws of Boolean rings are significantly simpler than the laws of ordinary Boolean algebra. They are:

The Laws of Boolean Rings

Associativity

$(A \oplus B) \oplus C = A \oplus (B \oplus C)$

$(AB)C = A(BC)$

Commutativity

$A \oplus B = B \oplus A$

$AB = BA$

Distribution

$A(B \oplus C) = AB \oplus AC$

Identity

$X \cdot 1 = X$

$X \oplus 0 = X$

Other Identities

$X \oplus X = 0$

$X \cdot X = X$

$X \cdot 0 = 0$

Using Boolean Rings

It cannot be overstated how easy it is to work in Boolean rings, and this will be demonstrated soon. The cost however, is that the OR function is less tractable and uses exponentially more space. It is easy to show that $A+B=A\oplus A\cdot B\oplus B$ and $A+B+C=A\oplus B\oplus C\oplus AB\oplus AC\oplus BC\oplus ABC$. (Here I rely on concatenation instead of the multiplication dot.) The general rule is that OR(A,B,...,N) is the ringsum (⊕) of all products of the various variables which gives 2^n-1 products in the case of n variables. The proof is easy enough. We can extend one of the DeMorgan laws to get $\overline{A+B+\ldots+N}=\overline{A}\cdot\overline{B}\cdot\ldots\cdot\overline{N}$. This leads immediately to $A+B+\ldots+N=\overline{\overline{A}\cdot\overline{B}\cdot\ldots\cdot\overline{N}}$ which translates to $A+B+C+\ldots+N=1\oplus\left(1\oplus A\right)\left(1\oplus B\right)\left(1\oplus C\right)\ldots\left(1\oplus N\right)$. After multiplying out the right side we merely employ $1\oplus 1=0$.

Before continuing it is interesting to look at one more identity. If we define the Boolean function XOR to mean exactly one of the variables is true then $XOR(A,B)=A\oplus B$ as expected, but $XOR(A,B,C)=A\oplus B\oplus C\oplus ABC$. In general XOR(A,B,...,N) is the ringsum (⊕) of all odd products of the variables. For example, $XOR(A,B,C,D)=A\oplus B\oplus C\oplus D\oplus ABC\oplus ABD\oplus ACD\oplus BCD$. The general rule can be proven fairly easily by induction.

The key to Boolean rings is that there are only two reduction rules: $X\cdot X=X$ and $X\oplus X=0$. Suppose we want to prove one of DeMorgan's laws: $\overline{X+Y}=\overline{X}\cdot\overline{Y}$. The expression on the left can be written as $\overline{X+Y}=\overline{X\oplus XY\oplus Y}=1\oplus X\oplus XY\oplus Y$, but that factors into $\left(1\oplus X\right)\left(1\oplus Y\right)=\overline{X}\cdot\overline{Y}$. Similarly, if we want to prove the Boolean reduction rule $AB + A = A$, we rewrite the left side as $AB \oplus AAB \oplus A$, which then simplifies as follows: $AB \oplus AAB \oplus A = AB \oplus AB \oplus A = A$, since $AB \oplus AB = 0$.

As an aside, albeit an important one, it is easy to forget that there is a more direct way of proving a proposition such as $A + AB = A$. B can only take on two values, namely 0 and 1. In the first case we get $A + 0 = A$, and in the second case we get $A + A = A$. Either way we have A and thus we are finished.

A last illustration of the difference between Boolean algebras and Boolean rings is the following simple observation: The boolean algebra expression $A + X = B$ does not have a unique solution for X. However, the unique solution of X in $A \oplus X = B$ is $X = A \oplus B$.

Z_2

As mentioned before, in a number system with just 0 and 1 where 1 + 1 = 0, we are working with a Boolean ring. This system is usually denoted Z_2 and is surprisingly important. More generally in abstract algebra, a ring for which for all x, $x \cdot x = x$ holds, is a Boolean ring. From this one (additional) assumption we can easily show that multiplication is commutative and that for all x, $x + x = 0$. However, none of this really concerns us here; it is just fodder for mathematicians.

The Other Boolean Ring

It does not seem to be well known that there is another Boolean ring that is dual to the one we have been exploring. In our ordinary Boolean ring AND works as multiplication and XOR works as addition; 0 is the additive identity and 1 is the multiplicative identity. However, if we take the laws of boolean rings as listed above and look at the duals of the statements, we get another set of true statements. Here we replace AND ($\cdot$) by OR (+) and XOR ($\oplus$) by EQUIV ($\equiv$) and we interchange 0 with 1. In this algebra OR is the multiplication, EQUIV is the addition; 0 is the multiplicative identity and 1 is the additive identity. In particular note that OR distributes over EQUIV: $A+(B\equiv C)=(A+B)\equiv(A+C)$. The full set of laws are exactly as before but with the above substitutions.

The Laws of the Dual Boolean Rings

Associativity

$$(A\equiv B)\equiv C \;=\; A\equiv(B\equiv C)$$
$$(A+B)+C \;=\; A+(B+C)$$

Commutativity

$$A\equiv B \;=\; B\equiv A$$
$$A+B \;=\; B+A$$

Distribution

$$A+(B\equiv C) \;=\; (A+B)\equiv(A+C)$$

Identity

$$X+0 \;=\; X$$
$$X\equiv 1 \;=\; X$$

Other Identities

$$X \equiv X = 1$$

$$X + X = X$$

$$X + 1 = 1$$

The Interesting Interrelationship Between XOR and EQUIV

Of the ten non-trivial binary Boolean functions, four are associative and all four are also commutative. These functions are AND $(\cdot)$ OR $(+)$ XOR $(\oplus)$ and EQUIV $(\equiv)$. We are interested in all four of these. It seems redundant to use both XOR and EQUIV as these are complements of one another (as well as duals). Either one provides the NOT function as follows: $\overline{X} = X \oplus 1 = X \equiv 0$. More generally if E is a Boolean expression we have $\overline{E} = E \oplus 1 = E \equiv 0$. Equally useful is $E = E \oplus 0 = E \equiv 1$. Because XOR and EQUIV are complementary we can go between them by using $X \oplus Y = X \equiv Y \equiv 0$ and $X \equiv Y = X \oplus Y \oplus 1$.

Given that XOR and EQUIV are opposites the following derivation is surprising:

$$\begin{aligned} & A \oplus B \oplus C \\ &= A \oplus B \oplus C \oplus 0 \\ &= A \oplus B \oplus C \oplus 1 \oplus 1 \\ &= A \oplus B \oplus 1 \oplus C \oplus 1 \\ &= (A \equiv B) \oplus C \oplus 1 \\ &= A \equiv B \equiv C \end{aligned}$$

$$\boxed{A \oplus B \oplus C \; = \; A \equiv B \equiv C}$$

As a first consequence we get $A + B = A \oplus A \cdot B \oplus B = A \equiv A \cdot B \equiv B$. It is easy to prove by induction that $A \oplus B \oplus \ldots \oplus N$ and $A \equiv B \equiv \ldots \equiv N$ are equal when the number of arguments (variables) is odd and that they are complementary when the number of arguments is even.

Another surprising derivation is:

$$\begin{aligned}
&\boxed{\left(A \oplus B\right) \equiv C} \\
&= \left(A \oplus B\right) \oplus C \oplus 1 \\
&= A \oplus \left(B \oplus C \oplus 1\right) \\
&= \boxed{A \oplus \left(B \equiv C\right)} \\
&= 0 \equiv A \equiv \left(B \equiv C\right) \\
&= \left(A \equiv B\right) \equiv C \equiv 0 \\
&= \boxed{\left(A \equiv B\right) \oplus C} \\
&= A \oplus B \oplus 1 \oplus C \\
&= 1 \oplus A \oplus B \oplus C \\
&= 1 \oplus A \oplus \left(B \oplus C\right) \\
&= \boxed{A \equiv \left(B \oplus C\right)}
\end{aligned}$$

The key observations so far:

$$\boxed{\begin{aligned}
A \equiv B \; &= \; 1 \oplus A \oplus B \\
A \oplus B \; &= \; 0 \equiv A \equiv B \\
A \oplus B \oplus C \; &= \; A \equiv B \equiv C \\
A \oplus B \equiv C \; &= \; A \equiv B \oplus C
\end{aligned}}$$

The apparent lack of parentheses in the last equation is not an accident. It is a consequence of the prior derivations; every implied association works. As far as the second to the last equation, both XOR (⊕) and EQUIV (≡) are associative. My own technique for handling a mixed statement containing just those two operations depends on the parity of the variables. First of all remember that such a statement should not have any repeat terms because $X \oplus X = 0$ and $X \equiv X = 1$ (these statements are duals). Any time we have two ⊕s we can replace them with two ≡s and vice versa. For example $A \oplus B \equiv C \oplus D$ has two ⊕s and can be written $A \equiv B \equiv C \equiv D$. Given an expression (tied together with just ⊕s and ≡s) that has an odd number of terms, I add a term to the expression be appending either ⊕ 0 or ≡ 1. For example, $A \oplus B \equiv C \oplus D \oplus E$ can be extended as follows $A \oplus B \equiv C \oplus D \oplus E \oplus 0$ leading to $A \equiv B \equiv C \equiv D \equiv E \equiv 0$ or $A \equiv B \equiv C \equiv D \oplus E$. Or the statement can be extended as here: $A \oplus B \equiv C \oplus D \oplus E \equiv 1$ leading to $A \oplus B \oplus C \oplus D \oplus E \oplus 1$ in which case I might rewrite it as $A \oplus B \oplus C \oplus D \equiv E$ which I could have gotten immediately from the given statement. So the standard format I use is generally to have all ⊕s , or all ⊕s followed by a single ≡. For example consider the following two expressions: $A \oplus B \oplus C \oplus D$, $A \oplus B \oplus C \equiv D$. These represent the two types of mixed XOR (⊕) and EQUIV (≡) statements. The first expression is true if exactly an odd number of variables are True. The second expression is true if exactly an even number of variables are True.

The Combined Boolean Algebra

The combined Boolean algebra uses NOT, AND, OR, XOR, and EQUIV. Therefore it is highly redundant. For example NOT X can be written $\overline{X}$ or $X \oplus 1$ or $X \equiv 0$. The most important thing to remember is distribution. AND distributes over OR and XOR, and almost over EQUIV.

That is:

$$A(B+C)=AB+AC$$
$$A(B\oplus C)=AB\oplus AC$$
$$A(B\equiv C)=A(B\equiv C\equiv 1)=A(B\oplus C\oplus 1)=AB\oplus AC\oplus A=AB\equiv AC\equiv A$$

Similarly, OR distributes over AND and EQUIV, and almost over XOR. That is:

$$A+BC=(A+B)(A+C)$$
$$A+(B\equiv C)=(A+B)\equiv(A+C)$$
$$A+(B\oplus C)=A+(B\oplus C\oplus 0)=A+(B\equiv C\equiv 0)=$$
$$(A+B)\equiv(A+C)\equiv(A+0)=(A+B)\oplus(A+C)\oplus A$$

Implication

In the combined Boolean algebra, A IMPLIES B has three principal forms: $\overline{A}+B$, $1\oplus A\oplus AB$, and $A\equiv AB$. Implication can be added directly to the algebra in which case it is important to remember IMPLIES distributes over AND and OR. In the latter case the identity $(A\to B)+C=A\to(B+C)$ might be useful. Also, remember $X\to 0=\overline{X}$.

Final Comments

There is much more to say. For example the relationship between EQUIV and EQUAL is fascinating. However to see that they are distinct we observe that $A = B = C = 1$ is true if and only if each variable is equal to one. However, $A \equiv B \equiv C = 1$ is true if each variable is equal to one <u>or</u> exactly one variable is true.

References

Cargal, James M. An Alternative Fault Tree Algebra. 1980. *IEEE tr. Reliability.* R-29, 269-272.

Cargal, James M. Some Notes on Elementary Boolean Algebra. 1988. *International Journal of Mathematical Education in Science and Technology.* 19. 231-241.