Episode 4.06 – Properties of Boolean Algebra

Welcome to the Geek Author series on Computer Organization and Design Fundamentals. I'm David Tarnoff, and in this series we are working our way through the topics of Computer Organization, Computer Architecture, Digital Design, and Embedded System Design. If you're interested in the inner workings of a computer, then you're in the right place. The only background you'll need for this series is an understanding of integer math, and if possible, a little experience with a programming language such as Java. And one more thing. Our topics involve a bit of figuring, so it might help to keep a pencil and paper handy.

In Episode 4.05 – Introduction to Boolean Algebra, we mentioned that the properties and identities of traditional algebra can be applied to Boolean algebraic expressions. The result is simpler expressions that will improve the speed, cost, or power consumption of our circuits. In fact, we get a few new properties when we go from an infinite number of values in traditional algebra to just one and zero that are in Boolean algebra. In this episode, we are going to begin our discussion with some properties that may seem a bit familiar, and one that will be brand new.

Before we begin, however, let's talk about the way that we're going to prove these properties and identities. Unlike traditional algebra where we must use a sequence of previously established identities and properties to prove a theory, the restriction on the values of binary variables makes a comprehensive list of all possible input conditions, well, possible. This means that the truth table is a sufficient means by which we can prove a Boolean algebraic theory.

To see this, let's start with a simple algebraic property: the commutative law. The commutative law of algebra states that the result of an operation is the same regardless of the order of operands. In math for example, $4 + 5 = 5 + 4$. Let's see if this law also holds true for Boolean algebra, beginning with the AND operation. Does $A \cdot B$ equal $B \cdot A$?

Well, the two inputs, A and B, have four possible combinations of ones and zeros: 0-0, 0-1, 1-0, and 1-1. If we write the truth table for $A \cdot B$, we get the following values as we go from the top row to the bottom.

$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$

Now, let's swap the variables in the expression.

$0 \cdot 0 = 0$
$1 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 1 = 1$

The result for both expressions is the same. As you go down the rows, the outputs are 0-0-0-1 for both. This means that $A \cdot B$ equals $B \cdot A$.

We could also visualize this property using the switch analogy presented in Episode 4.01 – Intro to Logic Gates. In that episode, we suggested that the AND operation is a lot like a set of switches that are connected in series. The only way for a logic one to make it from one end of the circuit to the other is if all the switches are closed. Open any switch, and the logic one is stopped from reaching the end.
How does this apply to the commutative law? Well, the commutative law suggests that the result is the same regardless of the order in which the switches are connected in series. Reversing the order of the switches does not change the fact that all the switches must be closed before a logic one can make it from one end of the circuit to the other.

Does the commutative law work with the OR operation. In other words, does A + B equal B + A? If we write the truth table for A + B, we get the following values as we go from the top row to the bottom.

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 1

Now, let's swap the variables in the expression.

0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 1

The result for both expressions is the same. As you go down the rows, the outputs are 0-1-1-1 for both. This means that A + B equals B + A.

We can use the switch analogy to visualize this version of the law too. The OR operation can be represented with a set of switches that are connected in parallel. If any of the switches are closed, a logic one can pass from one side of the circuit to the other. The only way to stop the logic one from reaching the other side is if all the switches are open.

The commutative law suggests that the result is the same regardless of the order in which we connect the switches in parallel. Placing the A switch on top or bottom has no effect on the condition that passes a logic one from one end of the circuit to the other.

The next law we are going to look at is the associative law. This law states that when combining three or more variables across the same operation, the order by which you combine them does not matter. For example, if we are AND-ing together the three variables, A, B, and C, it does not matter if you AND A with B, then AND the result with C or if you AND A with the result of B · C.

The way to prove this is with a truth table across all eight possible combinations of ones and zeros for three variables. Let's summarize that proof here staring with (A · B) · C. In a truth table with inputs A, B, and C, there are two rows where A equals one and B equals one: the seventh row where A=1, B=1, and C=0, and the eighth row where A=1, B=1, and C=1. Both of these rows will have ones for the expression A · B.

Note that C equals zero in one of those two rows and equals one in the other. The AND operation will only result in a one if both A · B equals one and C equals one. Therefore, the truth table for (A · B) · C has only one row with a one in it, the row where all three inputs equal one.

Now let's figure out the truth table for the expression on the right side of the equal sign: A · (B · C). First, there are only two rows in the truth table where both B and C equal one: the fourth row where A=0, B=1, and C=1, and the eighth row where A=1, B=1, and C=1. When you AND A with the result of B · C, that puts a zero in the fourth row. The only row with a one for the output is the eighth row were all three inputs equal one. Since the truth table output for the left expression equals the truth table output for the right expression for all eight rows, the associative law for the AND operation is proven true.
The logical OR operation also obeys the associative law. Using a similar sequence of steps, we see that (A + B) + C is equal to zero only for the top row of our truth table where A=0, B=0, and C=0, and that A + (B + C) also equals zero only for the top row of the truth table where A=0, B=0, and C=0.

Now for the distributive law. In traditional algebra, we see that any number multiplied across a sum is equal to that number multiplied by the first addend added to that number multiplied by the second addend. For example, five times the sum of four plus three equals five times four plus five times three. Will this work in Boolean algebra? Well, let's break out the truth table. What we are looking to prove is if A · (B + C) is equal to A · B + A · C. Let's start with the left side.

The parenthesis tell us to OR B with C first. This results in zeros for the first and fifth rows where both B and C are zero, and ones in all of the other rows. Now we need to AND this column with the values in the A column. A equals zero in the first four rows of our truth table, so all four of those rows are cleared to zero. The zero in the fifth row of the B + C column also places a zero in the final column. The last three rows, A=1, B=0, C=1; A=1, B=1, C=0; and A=1, B=1, C=1, all have ones in the result column. This gives us the values 0-0-0-0-0-1-1-1 as we read down the rows.

Now for the right side, A · B + A · C. Based on the order of operations, we need to perform the two AND operations before OR-ing their results. As we discovered when proving the associative law for AND across three variables, A · B equals one in rows seven and eight. A · C equals one in row six where A=1, B=0, and C=1, and in row eight where A=1, B=1, and C=1. That means for the top five rows, both A · B and A · C equal zero. This means that A · B + A · C also equals zero for these first five rows since 0 + 0 = 0. In the last three rows, either A · B or A · C equal one, which means that A · B + A · C also equals one. This gives us the values 0-0-0-0-0-1-1-1 as we read down the rows, which is the exact same result for the left expression. Therefore, the distributive law works when distributing an AND operation across an OR.
It turns out that in Boolean algebra, the distributive law has a second less familiar form. Because we are limiting ourselves to the binary values of one and zero, the distributive law also works when distributing an OR operation across an AND. In other words, A + B · C is equal to (A + B) · (A + C). That seems a little odd, doesn't it? Well, let's go at it again with the truth table.

The left expression, A + B · C, equals one if A equals one OR if B · C equals one. A equals one for the bottom half of the truth table, in other words, the last four rows where A equals one. So where does B · C equal one? As we saw when proving the associative law, B · C equals one for the two rows where both B and C equal one: the fourth row where A=0, B=1, and C=1, and the eighth row where A=1, B=1, and C=1. By OR-ing these two results, we see that the only rows where both A equals zero and B · C equals zero are the first three rows: A=0, B=0, C=0; A=0, B=1, C=0; and A=0, B=0, C=1. This gives us the sequence of results 0-0-0-1-1-1-1-1 as we read down the rows.

Now for the right expression: (A + B) · (A + C). This expression equals one only when both (A + B) equals one and (A + C) equals one. (A + B) equals one when either A is one or B is one. This happens in all but the first two rows where A=0, B=0, and C=0 and where A=0, B=0, and C=1. (A + C) equals one when either A is one or C is one. This happens in all but the first and third rows where A=0, B=0, and C=0 and where A=0, B=1, and C=0. When you AND the results of these two sums, you get a final column of the truth table where the first three rows have either (A + B) equal to zero or (A + C) equal to zero. The last five rows all have both (A + B) and (A + C) equal to one. This gives us the values 0-0-0-1-1-1-1-1 as we read down the rows, which is exactly the same result we derived for the left expression. Therefore, the distributive law works when distributing an OR operation across an AND.

In our next episode, we're going to dig down into some identities. If you feel comfortable with mathematical axioms such as anything added to zero is itself or anything multiplied by one is itself or anything multiplied by zero is zero, then you'll be fine with the identities of Boolean algebra. For transcripts, links, or other podcast notes, please visit us at intermation.com where you will also find links to our Instagram, Twitter, Facebook, and Pinterest pages. Until the next episode remember that while the scope of what makes a computer is immense, it's all just ones and zeros.