INTRODUCTION TO BOOLEAN ALGEBRA

Robert P. Webber, Longwood University

Boolean algebra is named for George Boole, an English logician and mathematician in the middle 1800s. He was interested in developing rules of algebra for logical thinking, similar to the rules of algebra for numerical thinking. Boole's genius was realizing that apparently dissimilar systems, such as symbolic logic and set theory, actually worked very much alike. Not much was done with Boole's work until the 1930s, when a graduate student, Claude Shannon, realized that Boole's work also applied to electrical circuits – which hadn't even been invented when Boole was alive, of course!

The basic Boolean operations correspond to the basic gates we have seen. They also correspond to standard operations in set theory and in symbolic logic.

Boolean operation	*Gate*	*Set theory*	*Symbolic logic*
$+$	**or**	$\bigcup$ (union)	$\vee$ (disjunction)
$\bullet$	**and**	$\bigcap$ (intersection)	$\wedge$ (conjunction)
$'$	**not**	^{C} (complement)	$\neg$ (negation)

The Boolean operation $\bullet$ is often abbreviated by writing the operands side by side and leaving out the operation symbol: xy instead of $x \bullet y$, for instance. Also, you may see the $'$ operation written as an overscore: $\overline{x}$ instead of x', for example.

Like arithmetic operations, Boolean operations have a default order in which a series of them are performed. This is called the **precedence** or **hierarchy** of operations.

$'$ has the highest priority;
$\bullet$ has the next highest priority;
$+$ has the lowest priority.

This means that a series of operations is done from left to right, except that

- all $'$ operations are done first;
- after that, all $\bullet$ operations are done next;
- and finally, all $+$ operations are done.

You can change the hierarchy by using parentheses.

For example,

$(xy)'$ means **not** (x **and** y) ;

xy'	means	x **and** (**not** y) ;
$x'y$	means	(**not** x) **and** y ;
$x'y'$	means	(**not** x) **and** (**not** y) .

Notice that these are not equivalent!

Boole realized that there are some arithmetic laws that apply to these operations. The collection of all the laws is called **Boolean algebra**. Here are two of them.

Double negation: $A'' = A$ (that is, **not** (**not** A)) = A).

DeMorgan's Laws: $(a+b)' = a'b'$ and $(ab)' = a'+b'$.

We can prove these by writing truth tables. Here is the proof of the first DeMorgan Law.

a	*b*	*a+b*	*(a+b)'*	*a'*	*b'*	*a'b'*
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Since the columns for $(a+b)'$ and $a'b'$ are the same, the expressions are equivalent.

Here's an example of how a teacher might make use of these laws. Suppose that students were assigned as homework the problem of designing a circuit that is equivalent to a **nand** gate using only **and**, **or**, and **not** gates. One student designed the circuit

(**not** x) **or** (**not** y)

and another designed the circuit

not (x **and** y) .

Are the circuits equivalent? The answer is yes, because the first circuit is $x'+y'$ and the second is $(xy)'$. By the second DeMorgan Law, they are equivalent.

Here are some more important laws of Boolean algebra.

Identity laws: Every Boolean algebra has an identity element for + (called 0) and one for $\bullet$ (called 1). That is,

$1x = x = x1$,

$0 + x = x = x + 0$ for every x.

It is important to realize that these symbols don't literally mean the integers 1 and 0. In set theory, for example, 1 is the universal set and 0 is the empty set. In symbolic logic, 1 is `true` and 0 is `false`.

Null laws: $0x = 0 = x0$ and $1 + x = 1 = x + 1$ for every x.

The second of these certainly doesn't look like an ordinary arithmetic law! Remember that 0 and 1 are not necessarily numbers.

Inverse laws: $xx' = 0 = x'x$ and $x + x' = 1 = x' + x$ for every x.

Here's a really weird one.

Idempotent laws: $xx = x$ and $x + x = x$ for every x.

The operations + and $\bullet$ are also **commutative** (order of the operands doesn't matter: $x + y = y + x$, for instance) and **associative** (you can group by pairs in any order, as long as the same operation is involved: $(xy)z = x(yz)$, for example).

Using these rules, we can simplify Boolean expressions. Here is an example. One way to simplify the expression $(a + a')(bb'')'$ is as follows:

$(a + a')(bb'')' = (a + a')(bb)'$ (Double negation law)

$= (a + a')b'$ (Idempotent law)

$= 1b'$ (Inverse law)

$= b'$ (Identity law)

So the circuits $(a + a')(bb'')'$ and b' are equivalent! This type of manipulation is used constantly in circuit design to simplify circuits or to show two circuits are equivalent. It is quicker and less tedious than constructing truth tables – but it requires more ingenuity!

Here are two more laws that are useful when simplifying circuits.

Distributive laws: $a(b + c) = ab + ac$ ($\bullet$ distributes over +)

and $a + bc = (a + b)(a + c)$ (+ distributes over $\bullet$).

Absorption laws: $a(a + b) = a$

and $a + ab = a$.

Here's another example of simplifying a circuit. Consider the implication function, whose truth table is

A	*B*	*implies*
0	0	1
0	1	1
1	0	0
1	1	1

Here are three circuits representing this function. Show they are equivalent.

Circuit 1:	$A'B'+A'B+AB$
Circuit 2:	$A'B'+B$
Circuit 3:	$A'+B$

Proof that circuits 1 and 2 are equivalent:

$$\begin{aligned} A'B'+A'B+AB &= A'B'+(A'+A)B && \text{(Distributive)} \\ &= A'B'+1B && \text{(Inverse)} \\ &= A'B'+B && \text{(Identity)} \end{aligned}$$

Proof that circuits 2 and 3 are equivalent:

$$\begin{aligned} A'B'+B &= A'B'+A'B+B && \text{(Absorption)} \\ &= A'(B'+B)+B && \text{(Distributive)} \\ &= A'1+B && \text{(Inverse)} \\ &= A'+B && \text{(Identity)} \end{aligned}$$

Therefore, all three circuits are equivalent.

This example begs two questions. First, how did we come up with Circuit 1? Second, how did we know to apply the Boolean algebra laws in those orders to get the other circuits? We will answer the first question here, and the second question in the next section.

There is a standard, cookbook algorithm to get a Boolean algebra expression for a circuit from a truth table. It is guaranteed to work, although it might be tedious and require a lot of gates. It produces what is called the **canonical sum of products form**, also called the **disjunctive normal form**. The resulting Boolean expression will be a sum of products, with each term being the product of variables and their negations. Here is how it works.

- First, focus on the rows that have 1 in the result column. Ignore all other rows.

- Next, for each such row, form the product of all the variables, using the variable itself if its value in the row is 1, and the negation of the variable if its value in the row is 0.
- Finally, form the sum of all these products.

This is exactly how we got Circuit 1 for the implication function.

- The first row has 1 in the result column. Both variables are 0, so the product for the first row will be $A'B'$.
- The second row also has 1 in the result column. The first variable is 0 in this row and the second is 1, so the product for the second row will be $A'B$.
- The third row has 0 in the result column, so ignore it.
- The fourth row has 1 in the result column, and both variables are 1 in the row. The product for the fourth row is AB.

Adding up the products, we get the circuit expression $A'B'+A'B+AB$. We can simplify this circuit to $A'+B$, as shown above.

Here's another example. Consider the function f defined by the truth table

x	y	z	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

The first, third, fourth, fifth, and seventh rows have 1 in the f column. Ignore the other rows. In the first row, all three variables are 0, so the product for the first row will be $x'y'z'$. In the third row, x is 0, y is 1, and z is 0, so the product is $x'yz'$. Similarly, the products for the remaining rows are $x'yz$, $xy'z'$, and xyz', respectively. The circuit is

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

We can probably simplify this circuit a lot!

EXERCISES

In exercises 1 through 8, use **and**, **or**, and **not** and parentheses to show the order of operations in the expression.

1. $(x+y)'$

2. $x+y'$

3. $x'+y$

4. $x'+y'$

5. $x+yz'$

6. $x+(yz)'$

7. $(x+y)z'$

8. $(x+yz)'$

In exercises 9 through 14 , prove the law using truth tables.

9. $(a')'=a$

10. $(ab)'=a'+b'$

11. $a+a=a$

12. $xx=x$

13. $a(a+b)=a$

14. $a+ab=a$

In exercises 15 through 18, show the circuits are equivalent.

15. $(a+b+(a+b)')'$; 0

16. $(a'+b')(a+b)$; $ab'+a'b$

17 $((a'b)')'+((ab')')'$; the *xor* circuit

18. $ab'c + abc$; ac

In problems 19 through 22, a function f is defined by the truth table. Write a Boolean expression for f and simplify the expression using laws of Boolean algebra.

19.

x	y	f
0	0	1
0	1	1
1	0	0
1	1	1

20.

x	y	f
0	0	0
0	1	1
1	0	1
1	1	1

21.

x	y	z	f
0	0	0	1
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	0

22.

x	y	z	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

23. We need to design a circuit for the *majority function*, which has three inputs, and outputs 1 when a majority of its inputs are 1. One expression for the circuit is $abc' + ab'c + a'bc + abc$.

a. Verify that this circuit works by constructing a truth table.

b. Using the laws of Boolean algebra, simplify the circuit by finding an equivalent circuit with fewer gates.

24. We are trying to design a circuit that has five inputs and outputs 1 whenever an odd number of inputs are 1. Write a Boolean expression for the function.