Boolean Logic

Boolean Logic, Operators, and Truth Tables

Similar to previously examined propositions, Boolean algebra has only three operators: *and*, *or*, and *not*. (Notice that this is very similar to propositional logic except that there are no conditional or biconditional operators.)

In Boolean logic, variables hold one of two values: 0 and 1. This differs from propositional logic's T and F only semantically. A truth table with two variables is commonly set up starting with 0's, and then advancing to 1's. It is also common in Boolean logic to use *x*, *y* and *z* rather than *p*, *q* and *r*.

x	*y*	
0	0	
0	1	
1	0	
1	1	

A three variable truth table would look like this:

x	*y*	*z*	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Like propositional logic, Boolean logic uses $\wedge$ for logical *and*, and $\vee$ for logical *or*. However, for logical *not*, Boolean algebra uses a bar over the affected terms.

Boolean truth table for logical *and*:

x	*y*	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

Boolean truth table for logical *or*:

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

Boolean truth table for logical *not*:

x	$\bar{x}$
0	1
1	0

Deriving Boolean Expressions from Truth Tables

In design and programming applications it is sometimes necessary to create a truth table to describe when events or operations occur. This is especially true when there appears no particular logic. This allows design to proceed in an *ad hoc* manner.

Let's consider the following example:

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

Operation z occurs only when input variables x and y are in the states described in the table. We look at only those rows where z is 1. For each of those rows, we create a short expression describing z in terms of x and y.

x	y	z	
0	0	0	
0	1	1	x is 0 and y is 1
1	0	1	x is 1 and y is 0
1	1	0	

This kind of terminology is awkward at best, so we include the negations of x and y.

x	y	$\bar{x}$	$\bar{y}$	z	
0	0	1	1	0	
0	1	1	0	1	$\bar{x}$ is 1 and y is 1
1	0	0	1	1	x is 1 and $\bar{y}$ is 1
1	1	0	0	0	

This is more convenient because all of the terms describing z have a value of 1. Thus, we could say that z is true (1) when $\bar{x}$ is 1 and y is 1, or x is 1 and $\bar{y}$ is 1. We can put this in symbolic form:

$$z = (\bar{x} \wedge y) \vee (x \wedge \bar{y})$$

Logical *and*'s are commonly joined so that the expression above might look like this:

$$z = (\bar{x}\, y) \vee (x\, \bar{y})$$

Boolean Algebra

Logical expressions in this form are commonly manipulated using Boolean algebra. In order to modify algebraic expressions the equivalence rules that were used in proofs can be used here.

Complement Laws	$\neg 1 = 0$ $\neg 0 = 1$
Excluded Middle Law	$x \vee \bar{x} = 1$
Contradiction Law	$x \wedge \bar{x} = 0$
Identity Laws	$x \vee 0 = x$ $x \wedge 1 = x$
Domination Laws	$x \vee 1 = 1$ $x \wedge 0 = 0$
Idempotent Laws	$x \vee x = x$ $x \wedge x = x$
Double Negation Law	$\bar{\bar{x}} = x$
Commutative Laws	$x \vee y = y \vee x$ $x \wedge y = y \wedge x$
Associative Laws	$(x \wedge y) \wedge z = x \wedge (y \wedge z)$ $(x \vee y) \vee z = x \vee (y \vee z)$
Distributive Laws	$(x \vee y) \wedge (x \vee z) = x \vee (y \wedge z)$ $(x \wedge y) \vee (x \wedge z) = x \wedge (y \vee z)$
DeMorgan's Laws	$\bar{x} \vee \bar{y} = \overline{(x \wedge y)}$ $\bar{x} \wedge \bar{y} = \overline{(x \vee y)}$

When working with Boolean algebra a primary goal is to reduce the number of operators. This means the total number of AND's, OR's, and NOT's. These are sometimes called 'gates' because they may represent actual hardware as we will see. In some cases it isn't possible to reduce the number of gates. (If you go on to study computer science or computer engineering, there are techniques that can seem to change the rules.) Otherwise, that's the way it is.

Example:

x	y	z	
0	0	1	x is 0 and y is 0
0	1	1	x is 0 and y is 1
1	0	1	x is 1 and y is 0
1	1	0	

Because values of zero are used, we can change the problem slightly, and still have the same meaning:

x	y	$\bar{x}$	$\bar{y}$	z	
0	0	1	1	1	$\bar{x}$ is 1 and $\bar{y}$ is 1
0	1	1	0	1	$\bar{x}$ is 1 and y is 1
1	0	0	1	1	x is 1 and $\bar{y}$ is 1
1	1	0	0	0	

At its most basic we could now say:

$$z = (\bar{x} \wedge \bar{y}) \vee (\bar{x} \wedge y) \vee (x \wedge \bar{y})$$

By counting all of the operators, we have a total of nine (9) operators. Let's try to reduce that number. Using the distributive law on the first two sub-expression, we join on $\bar{x}$:

$$z = (\bar{x} \wedge (\bar{y} \vee y)) \vee (x \wedge \bar{y})$$

Using the excluded middle law, we can simplify this.

$$z = (\bar{x} \wedge 1) \vee (x \wedge \bar{y})$$

Applying the identity law, we remove an unnecessary term.

$$z = (\bar{x}) \vee (x \wedge \bar{y})$$

Applying the distributive law in the other direction, we expand.

$$z = (\bar{x} \vee x) \wedge (\bar{x} \vee \bar{y})$$

This looks like we went in the wrong direction, but using the excluded middle law again gets us this:

$$z = 1 \wedge (\bar{x} \vee \bar{y})$$

The identity law eliminates the "1" term.

$$z = (\bar{x} \vee \bar{y})$$

We are now down to three (3) operators. This is significantly easier to implement than the nine we saw before.

But there can be an easier way. In this case, notice that the number of cases where z is 0 is significantly fewer than the number of cases where z is 1.

x	y	z	
0	0	1	x is 0 and y is 0
0	1	1	x is 0 and y is 1
1	0	1	x is 1 and y is 0
1	1	0	

We can add a column for $\bar{z}$ and identify when $\bar{z}$ is true.

x	y	z	$\bar{z}$	
0	0	1	0	
0	1	1	0	
1	0	1	0	
1	1	0	1	x is 1 and y is 1

From this we can derive the expression:

$$\bar{z} = (x \wedge y)$$

But notice that z is negative. We can't use that directly, so we apply a negative to each side of the equation.

$$\bar{\bar{z}} = \overline{(x \wedge y)}$$

Remembering the double negative rule:

$$z = \overline{(x \wedge y)}$$

This looks really nice. We are down to two operators, but it doesn't seem to match the previous result. Are the two the same?

If we apply DeMorgan's law, we see that they *are* the same.

$$z = (\bar{x} \vee \bar{y})$$

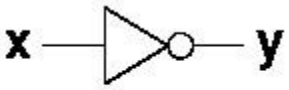
But better still, we found a way to describe this truth table with only two operators. The moral of the story is that there may well be more than one way to solve a problem, and the non-obvious way may be better in the long term.

Combinatorial Logic

In combinatorial logic symbols, representing electronic components, can be arranged and connected to represent manifestations of circuitry. This is much more graphical rather than text based as previous methods have been. There are several benefits of combinatorial logic among them the ability to describe a circuit that alters its own inputs, as well as being able to represent similarly operating hardware in multiple discipline.

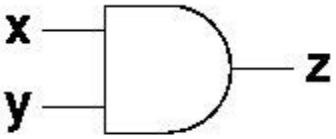
In this course of study, combinatorial logic will use the three basic Boolean operators, *and*, *or*, and *not*, and represent them symbolically. They are shown below.

NOT



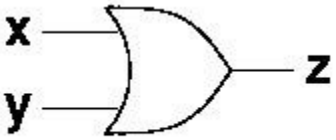
x	*y*
0	1
1	0

AND



x	*y*	*z*
0	0	0
0	1	0
1	0	0
1	1	1

OR



x	*y*	*z*
0	0	0
0	1	1
1	0	1
1	1	1

We will limit our examination of combinatorial logic to mean "single expression". This means that we will not define or create any circuits that can alter their own input. Those have special applications, and serve essential functions in computer engineering. For us a combinatorial circuit has one or more inputs, and one or more outputs. The circuitry inside the "box" can be described in a single Boolean expression.

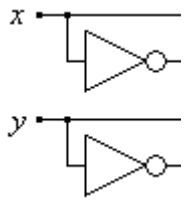
Let's take a look at an example we saw earlier:

x	y	$\bar{x}$	$\bar{y}$	z	
0	0	1	1	1	$\bar{x}$ is 1 and $\bar{y}$ is 1
0	1	1	0	1	$\bar{x}$ is 1 and y is 1
1	0	0	1	1	x is 1 and $\bar{y}$ is 1
1	1	0	0	0	

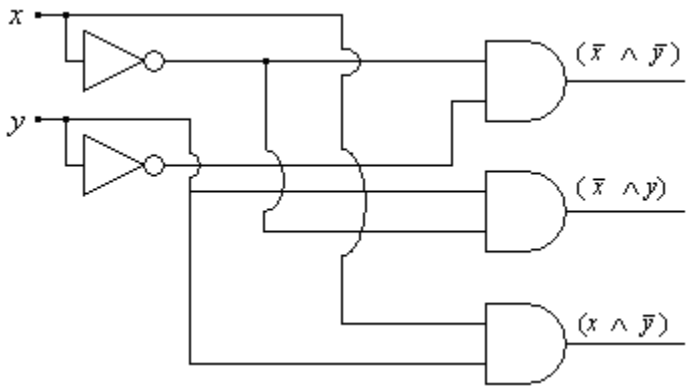
$$z = (\bar{x} \wedge \bar{y}) \vee (\bar{x} \wedge y) \vee (x \wedge \bar{y})$$

We know that this can be simplified, but let's look at this raw version.

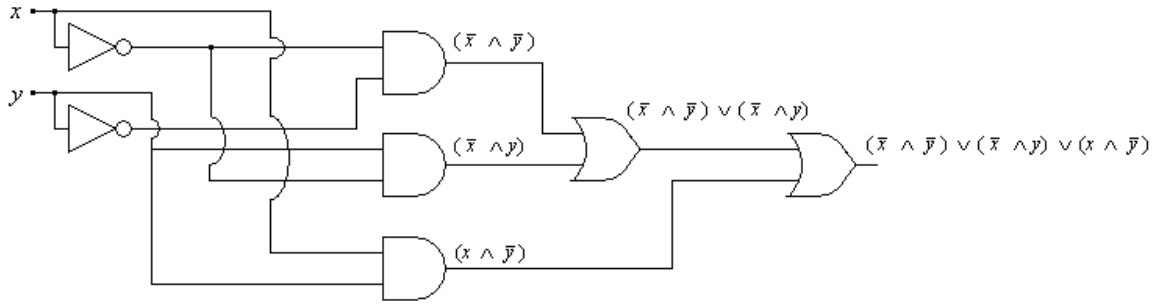
The first thing we do is identify the inputs. If the input value's 'negative' value will be needed or used this is a good time to include it.



We then start with the first expression before the OR operator, and work our way across creating little circuits for each nested expression.



We can then connect the intermediate terms with the OR gates.

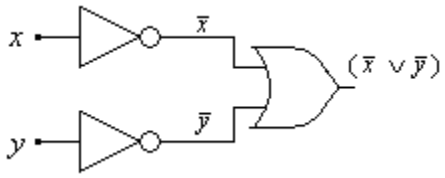


Notice that we didn't actually use 11 gates as we initially predicted. Why is that?

We can take a look at the condensed version of this expression:

$$z = (\bar{x} \vee \bar{y})$$

Here is the combinatorial logic that is equivalent:



As you can see, there are significantly fewer gates, less work in setting them up and connecting them, and fewer opportunities to make a mistake.

Notice that in each case the "values" of the inputs, *x* and *y*, "flow" to the right to produce the result. At no time does a result "flow" to the left.

Helpful hint: write the intermediate terms for each gate at each output. This will make it much easier to verify that the result is what you are looking for. This is also a useful method for determining the output of an existing circuit.