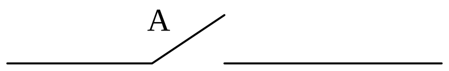
Section 1.8: Logic Circuits and Boolean Algebra

Logic Circuits

A logic circuit or digital circuit is an electrical circuit based on a discrete number of voltage levels, usually two. Two-level circuits usually have one voltage set at zero volts, and the circuit then behaves like a switch, being either **on** or **off**. A nice diagram for a switch looks like this:



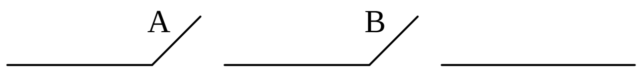
so that when the switch is open, as if the diagram, no current flows and the switch is **off**. When the switch closes and there's a clear path from the left side to the right side, the switch is **on**.

A digital circuit then makes logical decisions, based on the input to the circuit. The simplest logic circuits are called **gates**. Physically, a gate is a transistor circuit which takes one or more voltage inputs and gives a single voltage output.

One way to represent the action of a gate is by using a truth table. As usual in a truth table, all possible combinations of the input voltages are given, as well as the output of the gate for each set of inputs. Each input voltage is given a symbol, such as A. When the input signal is off, the value of A is given as 0, and when it's on, the value of A is 1. This then looks exactly like the truth tables we studied with logical propositions.

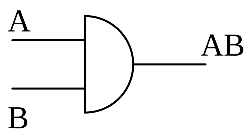
"and" gate

The **switch representation** of an "and" gate looks like this:



It is a series circuit, and both switches must be closed (on) for the circuit to be complete. You can see, then, that this is the same as "A and B", since "A and B" is true when both A is true and B is true.

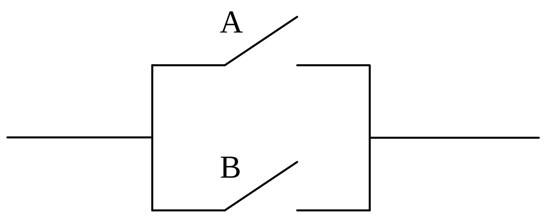
Another common representation is the **gate representation**, which looks like this:



In symbols, we write "A and B" as A·B or just AB.

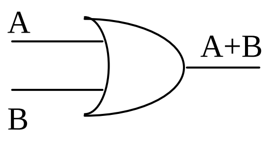
“or” gate

The **switch representation** of an “or” gate looks like this:



It is a parallel circuit, and at least one switch must be closed (on) for the circuit to be complete from left to right. You can see, then, that this is the same as “A or B”, since “A or B” is true when either A is true or B is true or both.

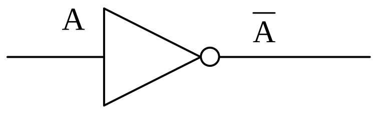
Another common representation is the **gate representation**, which looks like this:



In symbols, we write “A or B” as A+B.

“not” gate

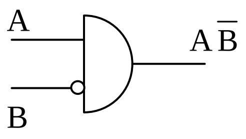
The “not” gate, or inverter, has the diagram



and we write (not-a) as $\overline{A}$, just as in set notation. If the negation happens in combination with another gate, we usually omit the triangle and just have a little circle to show the negation, as in the next example.

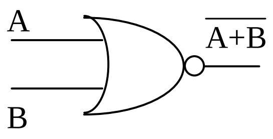
Gate Representations of Logic Circuits

The gate representation of the logic circuit for $A\overline{B}$ is then



with the round circle on the input B negating it, so that the two inputs to the “and” gate (the semicircle) are then A and $\overline{B}$.

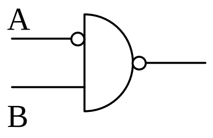
The gate representation for $\overline{A+B}$ is then



with the "or" gate giving A+B, which the little round circle then negates.

Example:

What is the logic circuit expression for the following gate diagram?



Answer: $\overline{\overline{A}B}$

Boolean Algebra

The symbols used for circuits, AB, A+B, and $\overline{A}$, are the same symbols as used in Boolean algebra. In this type of algebra, each variable (A, B, etc.) can only have two values, 0 and 1.

Truth tables in Boolean algebra then look very similar to the truth tables that we've studied in logic. For example, the truth table showing AB and A+B is:

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

If you have more than one operation happening in a Boolean expression, the order of operations is very similar to the order of operations in arithmetic. For example, if you have the logical expression AB+C, in arithmetic you multiply before you add. In Boolean algebra, you "and" before you "or". And, as before, the negation sign behaves in the same way as brackets do.

For example, here is the order in which you would evaluate the following Boolean expressions.

AB+C	First evaluate AB, then do the "+C".
A+BC	First evaluate BC, then "or" with A.
$\overline{A}B$	First evaluate $\overline{A}$, then "and" with B.
$\overline{A+B}$	First evaluate A+B, then negate the result
A(B+C)	First evaluate the expression in the brackets, then "and" with A.

Truth tables can then be used to demonstrate logical equivalence between Boolean expressions.

Example

Is AB+C logically equivalent to A(B+C)?

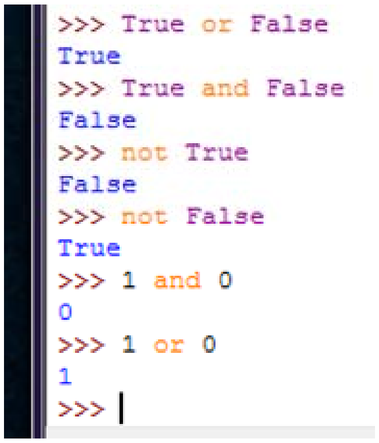
Answer:

A	B	C	AB	AB+C	B+C	A(B+C)
0	0	0	0	0	0	0
0	0	1	0	1	1	0
0	1	0	0	0	1	0
0	1	1	0	1	1	0
1	0	0	0	0	0	0
1	0	1	0	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

And as the 5th and 7th columns aren't identical, these two expressions aren't logically equivalent. Once again, order of operations matters.

Boolean Syntax in Python

Python allows you to perform logical operations on Boolean variables in the way that you would expect, as you can see in the accompanying figure.



```
>>> True or False
True
>>> True and False
False
>>> not True
False
>>> not False
True
>>> 1 and 0
0
>>> 1 or 0
1
>>> 
```

Figure 1: Boolean Syntax in Python