Propositional Logic

Propositions

A proposition is a statement which can either true or false, but not both. Some example of propositions:

> Ron works here.
> The Bears play football in Chicago.
> It is raining outside.
> $2 + 3 = 5$

In many cases we can replace statements like those above with letters or symbols, such as p, q, or r.

Connectives

Connectives are binary operators which link two or more propositions to create more complex logical expressions. They can also be called compound propositions. The use of connectives is similar to other binary operators used in arithmetic or algebra. There is one unary operator, which means that it operates on only one proposition.

Conjunction and Disjunction

A conjunction is commonly denoted by the caret symbol as the operator representing a ***logical and***. This means that if two propositions, p and q, are both true, then the result of the logical *and* is true, otherwise it is false.

$$p \wedge q$$

Conjunction can also be represented by the dot, and even just implied.

$$p \cdot q$$
$$p\,q$$

A disjunction is commonly denoted by a symbol looking like a lower case 'v' representing a ***logical or***. This means that if either of two propositions, p or q, are true, then the result of the logical *or* is true, otherwise it is false.

$$p \vee q$$

Disjunction can also be represented by the + symbol.

$$p + q$$

Negation

A negation is a unary operator which inverts the logical value of a proposition. Thus, if p is true, then *not p* is false. Similarly, if p is false, *not p* is true.

The unary negative, or *not*, is complicated by the fact that there are many ways that have been used to represent it. The most correct is to use a symbol similar to a minus sign, '¬', just before the proposition, as in this example:

$\neg p$

A close second is a bar over the proposition, as in this example:

Other symbols you may well see include the tilde, '~', especially on web pages as well as an apostrophe, such as *p'*.

Truth Tables

The truth value (T or F) of a compound proposition can be determined by using a truth table, displaying its constituent propositions and then solving for each line or row in the table. Let's look at a truth table for logical *and*:

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Notice that the table lists ever combination of true and false for the variables p and q. The number of rows is determined by the number of variables. Each added variable doubles the number of rows.

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

The truth table for logical *not* looks like this:

p	$\neg p$
T	F
F	T

Complicated Expressions

Propositions can be composed of other expressions, in the same way that math expression can be composed of others, like this:

$(2 + 3) / 6$

In the same way, propositions can be grouped using parentheses, like this:

$(p \wedge q) \vee r$

This entire expression can be treated as a single proposition.

Operator Precedence

When working with complicated or compound expressions, it is important to understand how to interpret them. For example, we know in arithmetic that in the case of (2 + 3) * 4 we add the 2 and the 3 together first because they are in parentheses, and then that sum gets multiplied by 4. One the other hand, 2+3*4 would be processed differently. We would multiply the 3 and the 4 because multiplication has a higher *precedent* than addition. There are similar rules in propositional logic.

> Expressions in parentheses are processed from the inside outward, and the entire expression treated as a single propositions
> $\neg$ (or *negation*) is applied next
> $\wedge$ (logical *and*) is applied next
> $\vee$ (logical *or*) after that

Let's look at an example:

$(p \wedge q) \vee r$

We would evaluate the expression using the logical *and* first because it is within parentheses. The result from that would be used in evaluating the logical *or*. A truth table below may clarify that.

p	q	r	$p \wedge q$	$(p \wedge q) \vee r$
T	T	T	T	T
T	T	F	T	T
T	F	T	F	T
T	F	F	F	F
F	T	T	F	T
F	T	F	F	F
F	F	T	F	T
F	F	F	F	F

Notice that we need to determine the truth value for $p \wedge q$ first, then apply that value in solving for $(p \wedge q) \vdash r$. For example, in the fifth row, p is F, q is T and r is T. Thus, $p \wedge$

q is F. We then use that in the next column to solve for $(p \wedge q) \vee r$, where F $\vee$ r (r is T, remember) results in T.

Commutative Property (Commutativity)

An operator that is said to be commutative if the order of the arguments doesn't matter. For example:

$$p \wedge q \equiv q \wedge p$$

or

$$p \vee q \equiv q \vee p$$

Associative Property (Associativity)

An operator is said to be associative if the sequence of evaluation doesn't matter. For example:

$$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$$

or

$$(p \vee q) \vee r \equiv p \vee (q \vee r)$$

Notice in each case that the operators are the same. You cannot "mix-and-match" operators and still apply the associative rule with any validity.

Conditional Propositions

A conditional proposition can be described as one proposition implying another:

if p then q

or

p implies q

This can be expressed with a symbol like this:

$p \Rightarrow q$ or often $p \rightarrow q$

This can be expressed in natural language like

If the Bears win on Sunday then they will be in the playoffs

Another way to think of this statement is

The Bears winning on Sunday implies that they will be in the playoffs.

While we tend to think of conditionals as operations, the way they are used here suggests a truth value. A better way to think of this last statement so that it can represent a truth value would be

Is it true that if the Bears win on Sunday then they will be in the playoffs?

The truth value then calls to question the validity of the conditional. It could be that nothing the Bears do on Sunday will get them into the playoffs (sound familiar?).

The truth table for the conditional is shown here:

p	q	$p \Rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

The last two rows are said to be *vacuously true*[1].

[1] From Wikipedia: Informally, a logical statement is **vacuously true** if it is true but doesn't say anything; examples are statements of the form "everything with property A also has property B", where there is nothing with property A. The statement

All elephants inside a loaf of bread are pink.

is vacuously true since there are no elephants inside a loaf of bread; here property A is "being an elephant inside a loaf of bread", and property B is "being pink". The suggestion is that *if you could* get an elephant into a loaf of bread, then it would be pink.

Biconditional Proposition

The biconditional operator is the conjunction of $p \Rightarrow q$ and $q \Rightarrow p$, or

$$p \Rightarrow q \wedge q \Rightarrow p \quad \equiv \quad (p \Leftrightarrow q)$$

Using a truth table, we get the following:

p	q	$p \Rightarrow q$	$q \Rightarrow p$	$(p \Rightarrow q) \wedge (q \Rightarrow p)$	$p \Leftrightarrow q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	T	F	F	F
F	F	T	T	T	T

This operator is used commonly enough that there is a separate symbol for it; $\Leftrightarrow$. Thus, the biconditional for p and q would look like this:

$$p \Leftrightarrow q \qquad \text{or} \qquad p \leftrightarrow q$$

Operator Precedence Part Two

With the addition of conditional operators, we have to modify the rules for precedent somewhat.

> Expressions in parentheses are processed from the inside outward, and the entire expression treated as a single propositions
> $\neg$ (or *negation*) is applied next
> $\wedge$ (logical *and*) is applied next
> $\vee$ (logical *or*) after that
> $\Rightarrow$ (conditionals)
> $\Leftrightarrow$ (biconditionals) last

When there are multiple uses of a connective, such as

$$p \wedge q \wedge r$$

then we assume that they are commutative and evaluate them from left to right. (By the way, logical operations in a compiler *may be* processed from right to left. This may have an affect on your program.)

DeMorgan's Law

DeMorgan's Theorem or Law provides a way to distribute (or collect) NOT operators.

$$\neg p \wedge \neg q = \neg (p \vee q)$$

$$\neg p \vee \neg q = \neg (p \wedge q)$$

Notice that the internal operator *inverts*. That is, logical *and*'s become logical *or*'s, and vice versa.

A common purpose for applying this rule is to reduce the number of operators. This can be helpful in database operations where each operation – including *not*'s – can be particularly time consuming.

Below is a truth table which can be used to verify the validity of one application of DeMorgan's Law.

p	q	$\neg p$	$\neg q$	$p \wedge q$	$\neg (p \wedge q)$	$\neg p \vee \neg q$
T	T	F	F	T	F	F
T	F	F	T	F	T	T
F	T	T	F	F	T	T
F	F	T	T	F	T	T

Note:
When simplifying expressions it's common to apply a negation to a term which has already been negated. Like negative symbols used in arithmetic, such negations cancel each other.

$$\neg (\neg r) = r$$