Boolean Normal Forms

A **boolean expression** is an expression involving variables each of which can take on either the value **true** or the value **false**. These variables are combined using boolean operations such as **and** (conjunction), **or** (disjunction), and **not** (negation). Another way of looking at this is that every boolean expression is identified with a **boolean function** which takes a certain number of parameters each of which may be true or false and returns a result of true or false. Thus we can also specify a boolean function by writing out a **truth table**, which is a table giving all possible assignments of true or false to the parameters and the resulting output from the function.

For example, here is a truth table for X = A and B:

A	B	A and B
false	false	false
false	true	false
true	false	false
true	true	true

There is a theorem that says that any boolean function may be written using only two levels of logic and possible negation of variables (called **literals**). There are two special forms, respectively called **disjunctive normal form** and **conjunctive normal form**, that are particularly useful. Let a **term** be the disjunction ("or") of a collection of variables, each optionally negated. Let a **clause** be the conjunction ("and") of a collection of variables, each optionally negated. If a boolean expression is precisely a conjunction of terms then it is said to be in conjunctive normal form, and if it is precisely a disjunction of clauses then it is said to be in disjunctive normal form.

For example, **(((not A) and B) or (A and (not B)))** is in disjunctive normal form (it's an OR of two terms) while **(((not A) or (not B)) and (A or B))** is in conjunctive normal form (it's an AND of two clauses). You'll notice that they represent the same function, **A xor B**. (**xor** is the ``exclusive OR" function, which is true if either one or the other of its arguments is true, *but not both*.)

It's easy to prove that any boolean function can be written in both DNF and CNF. While not a formal proof that it always works, I will provide a method for forming the DNF and CNF expressions of a boolean function. First, write out the truth table for the function. To form a DNF representation of the function, we'll include a term for each row of the truth table in which the value of the function is true. For each of these terms, include all variables that are parameters to the function. If a variable is given a value of false in the truth table row, negate it in the term; otherwise, leave it non-negated. In this way a DNF expression is generated directly from the truth table, and it's fairly intuitive to see that it works and that it always can be done.

To form a CNF expression we work similarly, forming a *clause* this time for every row in which the value of the function is *false*. Negate variables when they're given the value *true* in the truth table, and don't negate them when they're false (opposite of for DNF).

A	b	A xor B	terms	clauses
false	false	false		(A or B)
false	true	true	((not A) and B)	
true	false	true	(A and (not B))	
true	true	false		((not A) or (not B))

As a practical matter, we usually associate conjunction with multiplication and disjunction with addition. Indeed, if we identify true with 1 and false with 0, then {0,1} coupled with the usual definitions of addition and multiplication over the Galois field of size 2 (eg, arithmetic modulo 2), then addition (+) and disjunction (or) really are the same, as are multiplication and conjunction (and). This convention makes boolean expressions more concise to write. Negation is often written as a bar over a variable (or larger subexpression), or a "/" when an overbar is not available.

Thus, we write: `A xor B = (/A)B+A(/B) = (/A+/B)(A+B)`
