Boolean Satisfiability

There are many alternative ways to define a Boolean expression, but for our discussion, we must fix one of them. We define a string to be a *Boolean expression* if it is generated by the following context-free grammar G, with start symbol S: Let BOOL be the language of all strings generated by G.

$S \to !S$ (logical not)
$S \to S \Rightarrow S$ (implies)
$S \to S \equiv S$ (logical equal)
$S \to S \neq S$ (logical not equal)
$S \to S * S$ (logical and)
$S \to S + S$ (logical or)
$S \to (S)$
$S \to I$ (I generates all identifiers)
$I \to AN$ (The first symbol of an identifier must be a letter)
$A \to a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z$
$N \to AN|0N|1N|2N|3N|4N|5N|6N|7N|8N|9N|\lambda$
$S \to 0$
$S \to 1$

The strings generated by I are called *identifiers*. An *assignment* of a Boolean expression E is an assignment of each identifier in I to a logical value, either 0 (false) or 1 (true). We say that an assignment *satisfies* E the if evaluation of E yields 1, after repacing each identifier by its assigned value. Otherwise, E is not satisfiable, *i.e.,* a *contradiction.* Evaluation uses the rules of precedence of C++.

Definition 1 *A language L is $\mathcal{NP}$–COMPLETE if there is a $\mathcal{P}$–TIME reduction of any given $\mathcal{NP}$–TIME language to L.*

We define an instance of the Boolean satisfiability problem to be a Boolean expression, $E \in$ BOOL, where $E \in$ SAT if E is satisfiable.

Theorem 1 *Every $\mathcal{NP}$–TIME language has a $\mathcal{P}$–TIME reduction to* SAT*.*

Thus, by definition, SAT is $\mathcal{NP}$-complete. You can find the proof of Theorem 1 on the internet.

Conjunctive Normal Form

We say that a Boolean expression E is in *conjunction normal form*, or CNF, if E is the conjunction of clauses, each of which consists of the disjunction of terms, each of which is a variable or the negation of a variable. We say that $E \in$ CNF is in 3CNF if each of its clauses has three terms. That is,

$$E = C_1 * C_2 * \cdots * C_k$$

where $C_i = (t_{i1} + t_{i2} + t_{i3})$, and where each term t_{ij} is a variable or the negation of a variable. 2CNF, 4CNF, *etc.* are defined similarly.

An instance of the 3SAT problem is a Boolean expression in 3CNF form. An expression E

is a member of the language 3SAT if it is satisfiable and in 3CNF form.[1] Thus, 3SAT $=$ 3CNF $\cap$ SAT .

Polynomial Time Reduction of SAT to 3SAT

We define two Boolean expressions E and E' to be *sat-equivalent* if they both have the same satisfiability, *i.e.,* if either E and E' are both satisfiable or E and E' are both contradictions. We will define a $\mathcal{P}$–TIME reduction of SAT to 3SAT, *i.e.,* a $\mathcal{P}$–TIME function

$$R: \text{ BOOL } \rightarrow \text{ 3CNF}$$

such that $E' = R(E)$ is sat-equivalent to E, for any Boolean expression E. We first construct a parse tree for E, using the grammar G. and we simplify the parse tree to combine equivalent nodes. We choose a set of identifiers that are not used for E, such as $e0, e1, \ldots,$ and place one identifier at each internal node of the parse tree, where $e0$ is placed at the root. For each internal node, we write a Boolean expression stating that the variable at that node is equal to the concatenation of its children. Let E'' be the e_0 with the conjunction of those expressions. E'' is sat-equivalent to E. We then use the following table to replace each clause of E'' by a 3CNF expression. The resulting expression is in 3CNF form, and is sat-equivalent to E.

$a \equiv b + c$	equals	$(a+!b) * (!a + b + c) * (a + b+!c)$
$a \equiv b * c$	equals	$(!a + b) * (a+!b+!c) * (!a+!b + c)$
$a \equiv !b$	equals	$(a + b) * (!a+!b)$
$a \equiv b \Rightarrow c$	equals	$(a + b) * (!a+!b + c) * (a+!b+!c)$

Theorem 2 *If* SAT *is* $\mathcal{NP}$–COMPLETE *then* 3SAT *is* $\mathcal{NP}$–COMPLETE.

Example

Let $E = !\,(x + y \Rightarrow z) * z$. We show the parse three and the compressed parse tree of E, and then we replace each internal node by a unique auxiliary variable.

Then

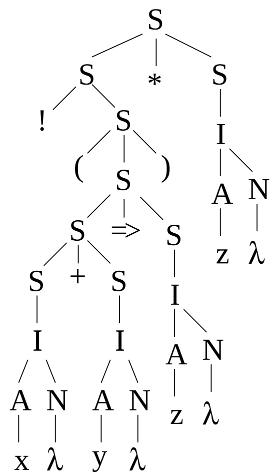
$$E'' = e0 * (e0 \equiv e1 * z) * (e1 \equiv !e2) * (e2 \equiv e3 \Rightarrow z) * (e3 \equiv x + y)$$

Using the equalities given in the table, replace each clause of E'' by an expression in CNF form:

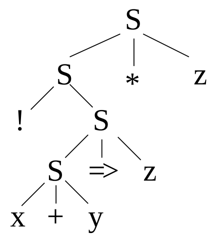
$$E' = e0 * (!e0 + e1) * (e0+!e1+!z) * (e0 + e1+!z) * (e1 + e2) * (!e1+!e2)$$
$$*(e2 + e3) * (!e2+!e3 + z) * (e2+!e3+!z) * (e3+!x) * (!e3 + x + y) * (e3 + x+!y)$$

We can pad with redundant terms to change E' into strict 3CNF form.

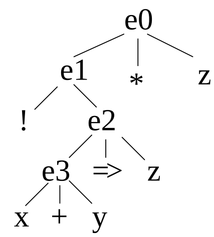
[1] When convenient, We can allow clauses of fewer than three terms, by introducing redundant terms: For example, $(x + y)$ can be replaced by the equivalent $(x + y + y)$.



parse tree of E



compressed
parse tree of E



compressed
parse tree of E
with auxiliary
variables