Boolean Satisfiability

BOOL is the language consisting of all Boolean expressions, where we allow the Boolean operators in Table 1 below.

1. ! not
2. + or
3. $*$ and
4. $\Rightarrow$ implies
5. $=$ equal
6. $\neq$ not equal

Table 1: Boolean Operators

An *assignment* of a Boolean expression E is an assignment of a truth value to every variable of E. An assignment is *satisfying* if the resulting value of E is 1 (true). E is *satisfiable* if it has a satisfying assignment. SAT $\subseteq$ BOOL is the subset consisting of all satisfiable expressions.

SAT is a very important problem in a practical sense. Solving instances of SAT has applications to software verification, program analysis, constraint solving, artificial intelligence, electronic design automation, and operations research. No known deterministic algorithm solves SAT in less than exponential time. However, due to the importance of the problem, a great deal of effort has been expended trying to improve SAT-solvers, some of which use randomization. Read the Wikipedia article: `https://en.wikipedia.org/wiki/SAT_solver`

Reductions

A *reduction* of a language $L_1 \subseteq \Sigma_1^*$ to a language $L_2 \subseteq \Sigma_2^*$ is a function $R : \Sigma_1^* \to \Sigma_2^*$ such that $w \in L_1$ if and only if $R(w) \in L_2$. A language L is defined to be $\mathcal{NP}$*–complete* if there is a $\mathcal{P}$–TIME reduction of every $\mathcal{NP}$ language to L.

Theorem 1 (Cook-Levin) *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.

Theorem 2 *If* L_1 *is* $\mathcal{NP}$*–complete,* L_2 *is* $\mathcal{NP}$*, and there is a* $\mathcal{P}$*–*TIME *reduction of* L_1 *to* L_2*, then* L_2 *is* $\mathcal{NP}$*–complete.*

Starting with SAT, many languages have been shown to be $\mathcal{NP}$–complete, using Theorem 2, and the list continues to grow.

Conjunctive Normal Form (CNF)

A Boolean expression E is in CNF if it is the conjunction of *clauses*, each of which is the disjunction of *literals*, each of which is either a variable or the negation of a variable. 3-CNF is the subset of CNF where each clause consists of at most three literals. 3-SAT = 3-CNF $\cap$ SAT, the set of satifiable CNF expressions where each clause has at most three literals.

Linear Time Reduction

We use Theorem 2 to show that 3-SAT is also $\mathcal{NP}$–complete, by giving a polynomial time (actually linear time) reduction R of SAT to 3-SAT.

Our reduction algorithm uses the method pioneered by Tseytin in 1966, who gave a linear time reduction of the Boolean circuit problem to 3-SAT.

Let $E \in$ BOOL. The variables of E we call *primary* variables. For clarity, we require each primary variable to be of the form of the form xN, where N is a positive numeral, *i.e.,* $x1$, $x2$, *etc.*. Let T be a parse tree for E. Each subtree of T corresponds to a subexpression of E, and each internal node of T contains a distinct *secondary* variable, written as yN, where N is a positive numeral. Each leaf is a primary variable.

Subtree Equations. For each non-leaf subtree of T gives rise to an equation setting the root variable of the subtree equal to the concatenation of its children. Let $Q(E)$ be the conjunction of the root of T and all its subtree equations. $Q(E)$ and E are equisatisfiable. Finally, we obtain $R(E)$ by replacing each subtree equation in $Q(E)$ by an equivalent 3-CNF expression, as given in Table 2 below. $R(E)$ and E are then equisatisfiable, and we are done.

subtree equation	equivalent 3-CNF expression
$q = a$	$(q+!a) * (!q + a)$
$q = !a$	$(q + a) * (!q+!a)$
$q = a + b$	$(q+!a) * (!q + a + b) * (q+!b)$
$q = a * b$	$(!q + a) * (q+!a+!b) * (!q + b)$
$q = (a \Rightarrow b)$	$(q + a) * (!q+!a + b) * (q+!b)$
$q = (a = b)$	$(q + a + b) * (q+!a+!b) * (!q + a+!b) * (!q+!a + b)$
$q = (a \neq b)$	$(q + a+!b) * (q+!a + b) * (!q + a + b) * (!q+!a+!b)$

Table 2: Mapping subtree equations to 3-CNF expressions.

Example

Let $E =!(x1 + x2 \Rightarrow x3)+!x3 \neq x1 * x2$. The parse tree of E is shown in Figure 1. Then $Q(E)$ is the conjunction of $y1$ and all subtree equations of T:

$y1*(y1 = (y2 \neq y3))*(y2 = y4+y5)*(y3 = x1*x2)*(y4 =!y6)*(y5 =!x3)*(y6 = (y7 \Rightarrow x3))*(y7 = x1+x2)$

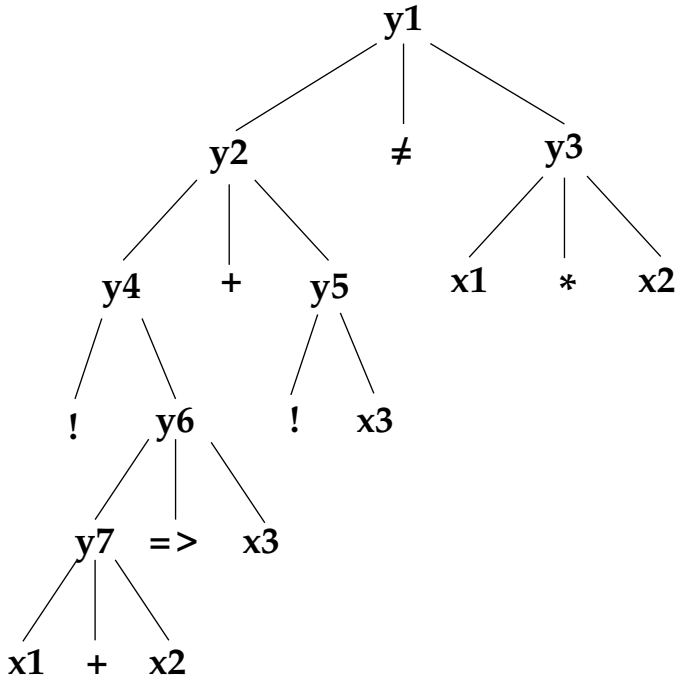

Figure 1: Parse tree T of $!(x1 + x2 \Rightarrow x3)+!x3 \neq x1 * x2$

$y1 = (y2 \neq y3)$	$(y1 + y2+!y3) * (y1+!y2 + y3) * (!y1 + y2 + y3) * (!y1+!y2+!y3)$
$y2 = y4 + y5$	$(y2+!y4) * (!y2 + y4 + y5) * (y2+!y5)$
$y3 = x1 * x2$	$(!y3 + x1) * (y3+!x1+!x2) * (!y3 + x2)$
$y4 = !y6$	$(y4 + y6) * (!y4+!y6)$
$y5 = !x3$	$(y5 + x3) * (!y5+!x3)$
$y6 = (y7 \Rightarrow x3)$	$(y6 + y7) * (!y6+!y7 + x3) * (y6+!x3)$
$y7 = x1 + x2$	$(y7+!x1) * (!y7 + x1 + x2) * (y7+!x2)$

Table 3: Mapping each subtree equation of T to a 3-CNF expression

Finally, R(E) is the conjunction of (y1) and the right sides of Table 3.

$R(E) = (y1) *$
$(y1 + y2+!y3) * (y1+!y2 + y3) * (!y1 + y2 + y3) * (!y1+!y2+!y3) *$
$(y2+!y4) * (!y2 + y4 + y5) * (y2+!y5) *$
$(!y3 + x1) * (y3+!x1+!x2) * (!y3 + x2) * (y4 + y6) * (!y4+!y6) *$
$(y5 + x3) * (!y5+!x3) *$
$(y6 + x7) * (!y6+!x7 + x3) * (y6+!x3) *$
$(y7+!x1) * (!y7 + x1 + x2) * (y7+!x2)$

Exercise 1: Let $E = !((x1 + x2) \Rightarrow x3) * x3$. Using the Tseytin Transformation method given above, construct a 3-CNF expression equisatisfiable with E.