# 3-CNF Satisfiability

Vassos Hadzilacos

Recall that a propositional formula is in conjunctive normal form (CNF) if it is a conjunction of one or more clauses, where a clause is a disjunction of one or more literals (and a literal is a propositional variable or the negation of a propositional variable). You know from CSCB36 that for any propositional formula $F$ there is a CNF propositional formula $F'$ that is logically equivalent to $F$. There are cases, however, where going from $F$ to $F'$ ***requires*** an exponential explosion in the size of the formula. For example, if $F = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$, which has size $\Theta(n)$, the shortest CNF formula that is equivalent to $F$ has size $\Theta(2^n)$.[1]

There is a special case of CNF formulas, called 3-CNF, where every clause is required to have at most three literals. The problem of deciding whether a 3-CNF formula is satisfiable is called ***3-CNF satisfiability***, abbreviated 3SAT. 3SAT plays an important role in the theory of **NP**-completeness.

In this document we will prove that SAT, the satisfiability problem for arbitrary propositional formulas, is polytime mapping-reducible to 3SAT, and therefore 3SAT is **NP**-complete. (The fact that 3SAT is in **NP** is obvious, since it is a special case of SAT, which we have seen is in **NP**.) We will do so by showing how, given any propositional formula $F$, we can construct in polytime a 3-CNF formula $\hat{F}$ that is satisfiable if and only if $F$ is satisfiable.

## Some preliminaries

Recall that a propositional formula is defined by induction as an expression $F$ that is either (base case) a propositional variable, or (induction step) an expression of one of the following three forms: $\neg F_1$, $(F_1 \vee F_2)$, and $(F_1 \wedge F_2)$, where $F_1$ and $F_2$ are themselves propositional formulas. Although we often take advantage of the commutativity of $\wedge$ and $\vee$ to avoid writing too many parentheses, we will assume that a propositional formula is given to us in the above fully-parenthesized form.

We will also assume that in the given formula $F$ negations are applied *only* to propositional variables and not to more complex subformulas. We can make this assumption without loss of generality because of DeMorgan's laws: $\neg(F_1 \wedge F_2)$ is logically equivalent to $(\neg F_1 \vee \neg F_2)$ and $\neg(F_1 \vee F_2)$ is logically equivalent to $(\neg F_1 \wedge \neg F_2)$. Thus, we can "push" negations deeper into the formula until they apply only to propositional variables. Doing so increases the size of the formula by only a constant factor; so this transformation, which preserves the truth value of the formula, can be applied in polynomial time.

We will use expressions such as $z \leftrightarrow \ell_1 \wedge \ell_2$ and $z \leftrightarrow \ell_1 \vee \ell_2$, where $z$ is a propositional variable and $\ell_1, \ell_2$ are literals; intuitively these mean, respectively, that $z$ has the same truth value as $\ell_1 \wedge \ell_2$ and $\ell_1 \vee \ell_2$. For this reason we will call such expressions ***definition clauses***, since they define the truth value of the variable $z$, in terms of the truth values of the literals $\ell_1$ and $\ell_2$.

Consider the first of these definition clauses, $z \leftrightarrow \ell_1 \wedge \ell_2$. This double implication is logically equivalent to the formula

$$\big(z \to (\ell_1 \wedge \ell_2)\big) \wedge \big((\ell_1 \wedge \ell_2) \to z\big) \tag{1}$$

By recalling that $(F_1 \to F_2)$ is logically equivalent to $(\neg F_1 \vee F_2)$ we get that (1) is logically equivalent to $\big(\neg z \vee (\ell_1 \wedge \ell_2)\big) \wedge \big(\neg(\ell_1 \wedge \ell_2) \vee z\big)$. By applying the distributive laws and DeMorgan'a laws, we get that

[1] This is an interesting but nontrivial exercise. Note that $F$ is in ***disjunctive*** normal form (DNF), i.e., it is a disjunction of one or more terms, where a term is a conjunction of one or more literals.

(1) is logically equivalent to

$$(\neg z \vee \ell_1) \wedge (\neg z \vee \ell_2) \wedge (\neg \ell_1 \vee \neg \ell_2 \vee z), \tag{2}$$

which is in 3-CNF. So we will view the definition clause $z \leftrightarrow \ell_1 \wedge \ell_2$ as an abbreviation for the logically equivalent 3-CNF formula (2).

By similar reasoning, we will view the definition clause $z \leftrightarrow \ell_1 \vee \ell_2$ as an abbreviation for the logically equivalent 3-CNF formula

$$(\neg z \vee \ell_1 \vee \ell_2) \wedge (\neg \ell_1 \vee z) \wedge (\neg \ell_2 \vee z\big). \tag{3}$$

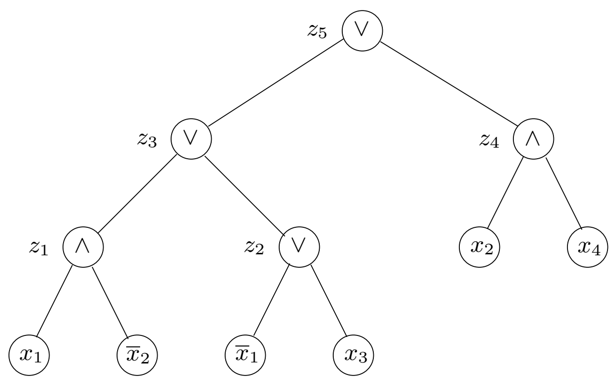
Note that the full expressions (2) and (3) are only a constant factor longer than the corresponding abbreviations that use the $\leftrightarrow$ connective.

## An example

We start with an example to illustrate the reduction and gain some intuition, before proceeding with the complete description. For convenience we will write the literal $\neg x$ as $\overline{x}$. Consider the propositional formula

$$F = \big((x_1 \wedge \overline{x}_2) \vee (\overline{x}_1 \vee x_3)\big) \vee (x_2 \wedge x_4). \tag{4}$$

It will be useful to think of the formula as a full binary tree, as illustrated below.




The leaves of this tree correspond to the literals of $F$, and every internal node to a subformula of $F$, excluding the literals. We associate a new variable with each internal node of the tree, i.e., with every subformula that is not a literal. For example, we associate $z_3$ with the subformula $\big((x_1 \wedge \overline{x}_2) \vee (\overline{x}_1 \vee x_3)\big)$. The intention is that each of these new variables (which we will call "$z$-variables") represents the truth value of the corresponding subformula, given a truth assignment to the variables of $F$ (the "$x$-variables").

For instance, $z_1$ is intended to represent the truth value of the subformula $x_1 \wedge \overline{x}_2$, $z_3$ the truth value of the subformula $(x_1 \wedge \overline{x}_2) \vee (\overline{x}_1 \vee x_3)$, and $z_5$ the truth value of the entire formula $F$. We can achieve this by using definition clauses of the kind described above, each of which involves only three literals. For example, for $z_1$ we have the definition clause $z_1 \leftrightarrow x_1 \wedge \overline{x}_2$: this states that $z_1$ is true exactly when both $x_1$ is true and $x_2$ is false, which is what the subformula $x_1 \wedge \overline{x}_2$ asserts. As noted earlier, the formula $z_1 \leftrightarrow x_1 \wedge \overline{x}_2$ is shorthand for a 3-CNF formula (see (2) above). Similarly, for $z_5$ we have the definition clause $z_5 \leftrightarrow z_3 \vee z_4$: $z_5$ is true exactly when at least one of $z_3$ and $z_4$ is true; again, this is shorthand for a 3-CNF formula.

We will now create a 3-CNF formula $\hat{F}$ that is satisfiable if and only if the given formula $F$ is satisfiable by taking the conjunction of (a) the $z$-variable that corresponds to the entire formula $F$ — $z_5$ in our example — and (b) all the definition clauses of the $z$-variables. Specifically, in our example we have:

$$\hat{F} = z_5 \wedge (z_1 \leftrightarrow x_1 \wedge \overline{x}_2) \wedge (z_2 \leftrightarrow \overline{x}_1 \vee x_3) \wedge (z_3 \leftrightarrow z_1 \vee z_2) \wedge (z_4 \leftrightarrow x_2 \wedge x_4) \wedge (z_5 \leftrightarrow z_3 \vee z_4).$$

Intuitively, this formula asserts that $z_5$ is true, and that each of the $z$-variables is true exactly when the corresponding subformula is true under a truth assignment to the variables of $F$. If some truth assignment to the $x$-variables satisfies $F$, then by assigning truth values to the $z$-variables based on their definitions, the "root" $z$-variable, in our case $z_5$, is true; and this truth assignment to all the variables of $\hat{F}$ (the $x$-variables as well as the $z$-variables) satisfies $\hat{F}$. If, on the other hand, $F$ is unsatisfiable, there is no truth assignment that satisfies $\hat{F}$: either the "root" $z$-variable is false, or one (or more) of the $z$-variable definition clauses is false, and therefore the entire conjunction that constitutes $\hat{F}$ is false.

## The reduction in detail

Let $\mathcal{F}$ be the set of propositional formulas in which negations are applied only to variables, and $\mathcal{F}'$ be the set of propositional formulas that are either (1) a single literal or (2) a conjunction of one or more definition clauses.

We now define by induction a function that maps each $F \in \mathcal{F}$ to an $F' \in \mathcal{F}'$ and, at the same time, a subset $\mathit{zvar}(F')$ of the variables of $F'$, called the ***z-variables*** of $F'$, and $\mathit{root}(F')$, a literal of $F$ or an element of $\mathit{zvar}(F')$, called the ***root*** of $F'$.

**Definition 1**

*Case 1:* $F$ *is a literal. Then* $F' = F$; $\mathit{zvar}(F') = \varnothing$, *and* $\mathit{root}(F')$ *is the literal* $F$ *itself.*

*Case 2a:* $F = \ell_1 \wedge \ell_2$, *where* $\ell_1$ *and* $\ell_2$ *are literals. Then* $F' = (z \leftrightarrow \ell_1 \wedge \ell_2)$, *where* $z$ *is a new variable that does not appear in* $\ell_1$ *or* $\ell_2$; $\mathit{zvar}(F) = \{z\}$ *and* $\mathit{root}(F') = z$.

*Case 2b:* $F = F_1 \wedge \ell_2$, *where* $F_1$ *is a formula that is not a literal and* $\ell_2$ *is a literal. Suppose, by induction, that* $F_1$ *is mapped to* $F_1'$; *and* $z_1 = \mathit{root}(F_1')$. *Then* $F' = (z \leftrightarrow z_1 \wedge \ell_2) \wedge F_1'$, *where* $z$ *is a new variable that does not appear in* $F_1'$ *or* $\ell_2$; $\mathit{zvar}(F') = \{z\} \cup \mathit{zvar}(F_1')$ *and* $\mathit{root}(F') = z$.

*Case 2c:* $F = \ell_1 \wedge F_2$, *where* $\ell_1$ *is a literal and* $F_2$ *is a formula that is not a literal. Suppose, by induction, that* $F_2$ *is mapped to* $F_2'$; *and* $z_2 = \mathit{root}(F_2')$. *Then* $F' = (z \leftrightarrow \ell_1 \wedge z_2) \wedge F_2'$, *where* $z$ *is a new variable that does not appear in* $\ell_1$ *or* $F_2'$; $\mathit{zvar}(F') = \{z\} \cup \mathit{zvar}(F_2')$ *and* $\mathit{root}(F') = z$.

*Case 2d:* $F = F_1 \wedge F_2$, *where* $F_1$ *and* $F_2$ *are formulas that are not literals. Suppose, by induction, that* $F_1$ *is mapped to* $F_1'$, $F_2$ *is mapped to* $F_2'$, $z_1 = \mathit{root}(F_1')$, *and* $z_2 = \mathit{root}(F_2')$; *we assume, without loss of generality, that* $\mathit{zvar}(F_1') \cap \mathit{zvar}(F_2') = \varnothing$ *— we can enforce this by renaming common* $z$*-variables, if necessary. Then* $F' = (z \leftrightarrow z_1 \wedge z_2) \wedge F_1' \wedge F_2'$, *where* $z$ *is a new variable that does not appear in* $F_1'$ *or* $F_2'$; $\mathit{zvar}(F') = \{z\} \cup \mathit{zvar}(F_1') \cup \mathit{zvar}(F_2')$ *and* $\mathit{root}(F') = z$.

*Case 3a:* $F = \ell_1 \vee \ell_2$, *where* $\ell_1$ *and* $\ell_2$ *are literals. Then* $F' = (z \leftrightarrow \ell_1 \vee \ell_2)$, *where* $z$ *is a new variable that does not appear in* $\ell_1$ *or* $\ell_2$; $\mathit{zvar}(F) = \{z\}$ *and* $\mathit{root}(F') = z$.

*Case 3b:* $F = F_1 \vee \ell_2$, *where* $F_1$ *is a formula that is not a literal and* $\ell_2$ *is a literal. Suppose, by induction, that* $F_1$ *is mapped to* $F_1'$; *and* $z_1 = \mathit{root}(F_1')$. *Then* $F' = (z \leftrightarrow z_1 \vee \ell_2) \wedge F_1'$, *where* $z$ *is a new variable that does not appear in* $F_1'$ *or* $\ell_2$; $\mathit{zvar}(F') = \{z\} \cup \mathit{zvar}(F_1')$ *and* $\mathit{root}(F') = z$.

*Case 3c:* $F = \ell_1 \vee F_2$, *where* $\ell_1$ *is a literal and* $F_2$ *is a formula that is not a literal. Suppose, by induction, that* $F_2$ *is mapped to* $F_2'$; *and* $z_2 = \mathit{root}(F_2')$. *Then* $F' = (z \leftrightarrow \ell_1 \vee z_2) \wedge F_2'$, *where* $z$ *is a new variable that does not appear in* $\ell_1$ *or* $F_2'$; $\mathit{zvar}(F') = \{z\} \cup \mathit{zvar}(F_2')$ *and* $\mathit{root}(F') = z$.

*Case 3d:* $F = F_1 \vee F_2$, *where* $F_1$ *and* $F_2$ *are formulas that are not literals. Suppose, by induction, that* $F_1$ *is mapped to* $F_1'$, $F_2$ *is mapped to* $F_2'$, $z_1 = \mathit{root}(F_1')$, *and* $z_2 = \mathit{root}(F_2')$; *we assume, without loss of generality, that* $\mathit{zvar}(F_1') \cap \mathit{zvar}(F_2') = \varnothing$. *Then* $F' = (z \leftrightarrow z_1 \vee z_2) \wedge F_1' \wedge F_2'$, *where* $z$ *is a new variable that does not appear in* $F_1'$ *or* $F_2'$; $\mathit{zvar}(F') = \{z\} \cup \mathit{zvar}(F_1') \cup \mathit{zvar}(F_2')$ *and* $\mathit{root}(F') = z$.

If $F \in \mathcal{F}$ is a formula, we denote with $var(F)$ the set of variables of $F$, and with $F'$ the formula to which $F$ is mapped under Definition 1. It is easy to see (and prove by a straightforward structural induction) that $var(F') = var(F) \cup zvar(F')$. We say that a truth assignment $\tau'$ to $var(F')$ ***extends*** a truth assignment $\tau$ to $var(F)$, written $\tau' \succeq \tau$, if for every variable $x \in var(F)$, $\tau'(x) = \tau(x)$; in other words, $\tau'$ agrees with $\tau$ on the variables common to both truth assignments.

**Lemma 2** *For every $F \in \mathcal{F}$ and every truth assignment $\tau$ to $var(F)$:*
*(a) There is a truth assignment $\tau' \succeq \tau$ that satisfies $F'$; furthermore, if $\tau$ satisfies $F$ then $\tau'\big(root(F')\big) = 1$.*
*(b) If $\tau' \succeq \tau$ is a truth assignment to $var(F')$ that satisfies $F'$ and $\tau'\big(root(F')\big) = 1$ then $\tau$ satisfies $F$.*

PROOF. (a) By structural induction on $F$. The base case (Case 1) is trivial by taking $\tau' = \tau$. For the induction step (Cases 2a-3d), we show only Case 3d; the other cases are similar.

Suppose $F = F_1 \vee F_2$, where $F_1$ and $F_2$ are formulas in $\mathcal{F}$ that are not literals. Let $\tau_1$ and $\tau_2$ be the restrictions of $\tau$ to the variables of $F_1$ and $F_2$, respectively. By Definition 1, $F' = (z \leftrightarrow z_1 \vee z_2) \wedge F_1' \wedge F_2'$, where $z$ is a new $z$-variable (not in $var(F_1')$ or $var(F_2')$), $z_1$ and $z_2$ are the roots of $F_1'$ and $F_2'$, and the $z$-variables of $F_1'$ and $F_2'$ are disjoint. By the induction hypothesis, there are truth assignments $\tau_1' \succeq \tau_1$ and $\tau_2' \succeq \tau_2$ that satisfy $F_1'$ and $F_2'$, respectively. Furthermore, if $\tau_1$ satisfies $F_1$ then $\tau_1'(z_1) = 1$; and if $\tau_2$ satisfies $F_2$ then $\tau_2'(z_2) = 1$.

Define the truth assignment $\tau'$ to $var(F')$ as follows:

$$\tau'(y) = \begin{cases} \tau(y), & \text{if } y \in var(F) \text{ — i.e., } y \notin zvar(F') \\ \tau_1'(y), & \text{if } y \in zvar(F_1') \\ \tau_2'(y), & \text{if } y \in zvar(F_2') \\ 1, & \text{if } y = z \text{ and at least one of } \tau_1'(z_1),\ \tau_2'(z_2) \text{ is } 1 \\ 0, & \text{otherwise — i.e., } y = z \text{ and both } \tau_1'(z_1),\ \tau_2'(z_2) \text{ are } 0 \end{cases}$$

By this definition, $\tau'$ extends both $\tau_1'$ and $\tau_2'$, and so, by the induction hypothesis, it satisfies both $F_1'$ and $F_2'$, and by the last two clauses in the definition it also satisfies $z \leftrightarrow z_1 \vee z_2$; so it satisfies $F' = (z \leftrightarrow z_1 \vee z_2) \wedge F_1' \wedge F_2'$.

Finally, suppose that $\tau$ satisfies $F$. Since $F = F_1 \vee F_2$, this implies that $\tau_1$ satisfies $F_1$ or $\tau_2$ satisfies $F_2$. Then, by the induction hypothesis, $\tau_1'(z_1) = 1$ or $\tau_2'(z_2) = 1$ and therefore $\tau'(z) = 1$ by the penultimate clause in the definition of $\tau'$. This completes the induction step (for Case 3d).

(b) This is also proved using structural induction on $F$. The base case (Case 1) is again trivial since in this case $\tau' = \tau$. For the induction step (Cases 2a-3d), we show only Case 3d; the other cases are similar.

Let $F = F_1 \vee F_2$, where $F_1$ and $F_2$ are formulas in $\mathcal{F}$ that are not literals. Since $F = F_1 \vee F_2$, $F' = (z \leftrightarrow z_1 \vee z_2) \wedge F_1' \wedge F_2'$, where $z = root(F')$, $z_1 = root(F_1')$, and $z_2 = root(F_2')$. Suppose $\tau' \succeq \tau$ satisfies $F'$ and $\tau'(z) = 1$; we must show that $\tau$ satisfies $F$. Indeed, since $\tau'$ satisfies $F'$, it must satisfy all of $z \leftrightarrow z_1 \vee z_2$, $F_1'$, and $F_2'$. Since it satisfies $z \leftrightarrow z_1 \vee z_2$ and $\tau'(z) = 1$, it follows that $\tau'(z_1) = 1$ or $\tau'(z_2) = 1$. Since $\tau'$ satisfies $F_1'$ and $F_2'$, it follows that either $\tau'$ satisfies $F_1'$ and $\tau'(z_1) = 1$ or $\tau'$ satisfies $F_2'$ and $\tau'(z_2) = 1$. Thus, by the induction hypothesis, $\tau$ satisfies one of $F_1$ or $F_2$ and therefore it satisfies $F_1 \vee F_2 = F$, as wanted. □

For any $F \in \mathcal{F}$, let $\hat{F} = F' \wedge root(F')$. Intuitively, $\hat{F}$ asserts that the definitions of all the $z$-variables of $F'$ are true and, furthermore, the root of $F'$, which intuitively represents the truth value of $F$, is also true. Note that $\hat{F}$, like $F'$, is in 3-CNF.

**Theorem 3** *Let $F \in \mathcal{F}$. A truth assignment $\tau$ to $var(F)$ satisfies $F$ if and only if there is a truth assignment $\tau' \succeq \tau$ that satisfies $\hat{F}$.*

PROOF. Since $\hat{F} = F' \wedge root(F')$, the only-if direction follows immediately from Lemma 2(a), and the if direction from Lemma 2(b). $\square$

**Corollary 4** *$F \in \mathcal{F}$ is satisfiable if and only if $\hat{F}$ is satisfiable.*

Since $\hat{F}$ is in 3-CNF and can be constructed from $F$ in polytime, we have:

**Corollary 5** 3SAT *is* ***NP****-complete.*

What if we further restrict CNF formulas so that each clause has at most two literals? The satisfiability problem for such formulas, called 2SAT, turns out to be solvable in polynomial time!

CNF-SAT is the problem of determining whether a propositional formula in CNF is satisfiable. Since 3SAT is a special case of CNF-SAT, by Corollary 5 we also have:

**Corollary 6** CNF-SAT *is* ***NP****-complete.*