Notes on Satisfiability-Based Problem Solving

Conjunctive Normal Form and SAT

David Mitchell
mitchell@cs.sfu.ca
September 18, 2019

These notes are a preliminary draft. Please use freely, but do not re-distribute without permission. Corrections and suggestions are welcome.

In this section, we define conjunctive normal form (CNF) formulas, introduce the problems SAT and K-SAT, give examples of representing problems in CNF, and discuss transformation of general propositional formulas to CNF.

Terms and Conventions By n we denote the number of distinct atoms in a formula, and l denotes the length, or size, of the formula, typically measured by the total number of occurrences of atoms (which is also the number of leaves in the formula tree).

1 Conjunctive Normal Form

A *conjunction* of formulas is a formula of the form $(A_1 \wedge A_2 \wedge ... \wedge A_m)$, and a *disjunction* of formulas is a formula of the form $(A_1 \vee A_2 \vee ... \vee A_m)$.

To satisfy the standard syntax for formulas, we may consider conjunctions or disjunctions of more than two formulas to be implicitly parenthesized, for example by association to the left, so that $(A_1 \wedge A_2 \wedge A_3 \wedge A_4)$ means $(((A_1 \wedge A_2) \wedge A_3) \wedge A_4)$. However, the choice of parenthesization does not matter semantically, and we tend to just think of $(A_1 \wedge \ldots \wedge A_m)$ as a formula for which $\tau \models (A_1 \wedge \ldots \wedge A_m)$ iff τ satisfies all of $A_1, \ldots, A_m$.

Definition 1. A *literal* is an atom P or a negated atom $\neg P$. The *complement* of a literal L, denoted $\overline{L}$, is $\neg P$ if L is P and P if L is $\neg P$. A *clause* is a disjunction of literals. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses.

We consider the empty disjunction to be a clause, and the empty conjunction to be a CNF formula, even though they are not formulas by the usual definition. Semantically, the empty clause is unsatisfiable, and the empty conjunction is valid.

It is often convenient to treat a clause as a *set* of literals, and a CNF formula as a *set* of clauses. Then we may apply standard set operations and notation to CNF formulas. Viewed this way, τ satisfies a clause if it satisfies at least one literal in the clause, and satisfies a clause set if it satisfies every clause in the set. Sets differ from formulas in that they are un-ordered and cannot contain duplicate elements. In the present context neither of these distinctions is very important, and henceforth we treat a CNF formula as a conjunction or a set as is convenient to the context.

We can also define a dual form to CNF.

Definition 2. A formula is in *disjunctive normal form* (DNF) if it is a disjunction of conjunctions of literals.

Theorem 1. *For every formula A, there exists a CNF formula which is equivalent to A and a DNF formula which is equivalent to A.*

We may construct a CNF formula A' which is equivalent to A by means of the following truth table method. Let S be the set of atoms in A. For each truth assignment τ for S with $\tau(A) = false$, include in A' the clause which is the set of all literals L with $\tau(L) = false$. An equivalent DNF can be constructed dually, by including a conjunction for each assignment which satisfies A.

Example 1. *Let $A = ((P \wedge Q) \vee (\neg P \wedge \neg Q))$. The truth table for A, adorned with the clauses to include in equivalent CNF and DNF formulas, is:*

P	Q	A	*clause to include in the CNF*	*to include in the DNF*
$true$	$true$	$true$	*none*	$(P \wedge Q)$
$true$	$false$	$false$	$(\neg P \vee Q)$	*none*
$false$	$true$	$false$	$(P \vee \neg Q)$	*none*
$false$	$false$	$true$	*none*	$(\neg P \wedge \neg Q)$

We obtain $(\neg P \vee Q) \wedge (P \vee \neg Q)$ as a CNF formula equivalent to A. For the DNF, in this case, we get A itself.

2 SAT and K-SAT

Definition 3. The CNF Satisfiability Problem (SAT) is:

Given: A propositional CNF formula A;
Question: Is A satisfiable?

SAT was the first problem shown to be NP-complete. One implication of NP-completeness is that all other problems in NP can be efficiently transformed to SAT. A piece of software for solving SAT is called a SAT solver. NP-completeness of SAT tells us that, in principle at least, every problem in NP can be solved by an efficient transformation to SAT plus a SAT solver. In spite of very bad worst-case lower bounds on the algorithms used, modern SAT solvers are capable of solving challenging instances of many problems.

Proposition 1. *Let S be a set of atoms. The truth assignments to S are one-to-one with the sets of literals over S which, for each $P \in S$, contain exactly one P or $\neg P$.*

Proposition 1 justifies a convenient notation for writing truth assignments. For example, rather than write $\tau(P_1) = true, \tau(P_2) = false, \tau(P_3) = true, \ldots$, we may write that τ is $\{P_1, \overline{P_2}, P_3, \ldots\}$.

Exercise 1. *Identify each of the following CNF formulas as being unsatisfiable, valid, or satisfiable but not valid. For each formula which is satisfiable, give a satisfying assignment, and for each formula which is not valid, give a truth assignment under which it is false.*

1. $(\neg A \vee B) \wedge (\neg B \vee C) \wedge (A \vee \neg C)$.
2. $(P \vee Q) \wedge (\neg P \vee Q) \wedge (\neg Q \vee P) \wedge (\neg P \vee \neg Q \vee R) \wedge (\neg R \vee \neg Q)$.
3. $(P \vee Q \vee R \vee \neg P) \wedge (P \vee Q \vee \neg R \vee \neg Q)$
4. $(\neg P \vee \neg Q) \wedge (\neg P \vee \neg R) \wedge (\neg Q \vee \neg R) \wedge (P \vee Q) \wedge (P \vee R) \wedge (Q \vee R)$.

Exercise 2. *Describe an easy linear-time algorithm for deciding if a CNF formula is a tautology.*

Definition 4. For any integer $k > 0$, we say a formula is in k-CNF if it is in CNF and every clause has size at most k. The k-SAT problem is:

Given: A propositional k-CNF formula A.
Question: Is A satisfiable?

For each $k > 2$, k-SAT is NP-complete. In a later section of these notes, we will see linear-time algorithms for 2-SAT.

Exercise 3. *Describe an easy linear-time algorithm for* 1*-SAT.*

3 Representing Problems with CNF Formulas

We are interested in solving problems by representing their sets of solutions with CNF formulas, such that each satisfying assignment represents (ideally, in a fairly transparent way) a solution. Doing this involves two general steps:

1. Choose the set of atoms to use. Normally this is done so that each assignment to the atoms corresponds to a combinatorial object which is a candidate solution.
2. Produce a set of clauses over the chosen atoms which restricts the satisfying assignments to those representing solutions.

We illustrate with graph colouring as a simple example. Graph colouring is the problem of assigning "colours", from some given set, to the vertices of a graph so that no edge of the graph is monochrome — that is, the colours of any two adjacent vertices are different. Such a colouring is called "proper".

Definition 5. K-Col

- **Given:** Graph G=$\langle V, E\rangle$, set C of k of colours;
- **Question:** Does G have a proper k-colouring? More precisely, is there a function $Col : V \to C$ such that, for every $(u, v) \in E$, $Col(u) \neq Col(v)$?

In practice we are usually interested in finding a proper colouring, not just knowing one exists. For a given G and C, we want a CNF formula $\Gamma = \Gamma(G, C)$ such that a satisfying assignment for Γ more-or-less directly gives us a proper colouring of G with colours from C. Next we describe two ways to do this, based on different choices of atoms.

In describing formulas, we will often use a short-hand notation similar to Σ notation for summation. For example, if $S = \{1, 2, 3\}$, then

$$\bigwedge_{i \in S} P_i$$

denotes $(P_1 \wedge P_1 \wedge P_3)$.

3.1 Graph Colouring Formulas: Version 1

One natural way to represent graph colouring with a propositional formula is to have the set of atoms

$$\{C_{v,c} \mid v \in V \text{ and } c \in C\}. \tag{1}$$

The intuitive interpretation of these atoms is that assigning $C_{v,c}$ to true means vertex v is coloured c, or $Col(v) = c$. We need to write clauses over these atoms which restrict assignments to those which correspond to proper colourings. We will do this by writing two sets of clauses:

1. For each vertex v, a write clause requiring that v gets some colour:
$$\left(\bigvee_{c \in C} C_{v,c} \right); \tag{2}$$
2. For each edge (u, v), write a conjunction of clauses saying that u and v do not have the same colour:
$$\bigwedge_{c \in C} (\overline{C_{u,c}} \vee \overline{C_{v,c}}). \tag{3}$$

So the overall formula $\Gamma(G, C)$, for $G = \langle V, E \rangle$, is

$$\bigwedge_{v \in V} \left(\bigvee_{c \in C} C_{v,c} \right) \ \wedge \ \bigwedge_{(u,v) \in E} \bigwedge_{c \in C} (\overline{C_{u,c}} \vee \overline{C_{v,c}}). \tag{4}$$

It is not hard to see that G is k-colourable if and only if $\Gamma(G, \{1, \ldots k\})$ is satisfiable. Also, from each satisfying assignment, we can easily obtain a proper k-colouring of G.

Example 2. *Let* $G = \langle \{a, b, c\}, \{(a, b), (c, b)\} \rangle$, *and* $C = \{R, G, B\}$. *Then,* $\Gamma(G, C)$ *is*

$$\begin{aligned} &(C_{a,R} \vee C_{a,B} \vee C_{a,G}) \ \wedge \ (C_{b,R} \vee C_{b,B} \vee C_{b,G}) \ \wedge \ (C_{c,R} \vee C_{c,B} \vee C_{c,G}) \\ &\wedge (\overline{C_{a,R}} \vee \overline{C_{b,R}}) \ \wedge \ (\overline{C_{a,B}} \vee \overline{C_{b,B}}) \ \wedge \ (\overline{C_{a,G}} \vee \overline{C_{b,G}}) \\ &\wedge (\overline{C_{c,R}} \vee \overline{C_{b,R}}) \ \wedge \ (\overline{C_{c,B}} \vee \overline{C_{b,B}}) \ \wedge \ (\overline{C_{c,G}} \vee \overline{C_{b,G}}) \end{aligned}$$

The truth assignment $\{C_{a,R}, C_{b,B}, C_{c,B}, \overline{C_{a,B}}, \overline{C_{a,G}}, \overline{C_{b,R}}, \overline{C_{b,G}}, \overline{C_{c,R}}, \overline{C_{c,G}}\}$ *satisfies* Γ, *and corresponds to colouring a Red and b and c both Blue.*

The satisfying assignments of $\Gamma(G, \{1, \ldots K\})$ do not correspond one-to-one with proper K-colourings of G, because the formula allows assigning multiple colours to each vertex. These "multi-colourings" are still proper - they don't have any monochrome edges. Depending upon application, we may choose to ignore them, perform a post-processing step to generate proper colourings by deleting some "extra" colours, or add clauses to Γ requiring at most one colour for each vertex, as in:

$$\bigwedge_{v \in V} \bigwedge_{\substack{c, c' \in C \\ c \neq c'}} (\overline{C_{v,c}} \vee \overline{C_{v,c'}}). \tag{5}$$

3.2 Graph Colouring Formulas: Version 2

In this version, we take colours to be integers and for each vertex we have a set of atoms which represent the binary encoding of its colour. Given $G = \langle V, E \rangle$ and $C = \{1, \ldots k\}$, and letting $r = \lceil \log_2 k \rceil$, our set of atoms will be

$$\{B_{v,i} \mid v \in V \text{ and } i \in \{1, \ldots r\}\}. \tag{6}$$

The intuitive meaning of $B_{v,i}$ is that the i^{th} bit of the binary encoding of the colour assigned to vertex v is 1.

To describe the clauses we will use the notation c_i for the i^{th} bit of the binary encoding of the number (colour) c, and write $B_{v,i,c}$ for the literal that is made true by assignments which give $B_{v,i}$ the value of bit c_i. That is:

$$B_{v,i,c} \text{ denotes } \begin{cases} B_{v,i} & \text{if } c_i = 1 \\ \neg B_{v,i} & \text{if } c_i = 0. \end{cases} \tag{7}$$

So, $B_{v,i,c}$ is the literal that says the colour on vertex v has the same i^{th} bit as colour c.

To disallow monochrome edges, for each edge $(u, v) \in E$, and each colour in $c \in C$, we have a clause that says that either u does not have colour c or v does not. To do this, we say that at least one of the colours on u and v differs from c on at least one bit:

$$\left(\overline{B_{u,1,c}} \vee \ldots \vee \overline{B_{u,r,c}} \vee \overline{B_{v,1,c}} \vee \ldots \vee \overline{B_{v,r,c}}\right).$$

If k is a power of two (and $k > 0$), then for each $v \in V$, every truth assignment to the literals $B_{v,1}, \ldots B_{v,r}$ gives vertex v a unique colour. Thus, every truth assignment gives a colouring to V, so we have no need of clauses which assert that every vertex must get a colour, or clauses that assert that vertices are not multi-coloured. So, the only clauses we need are those asserting that no edge is monochrome.

If k is not a power of two, there are assignments to $B_{v,1}, \ldots B_{v,r}$ which do not correspond to any colour, and we also need clauses to disallow these "non colours". These are, for each vertex v and each number $c \in \{k + 1 \ldots 2^r\}$:

$$\left(\overline{B_{v,1,c}} \vee \ldots \vee \overline{B_{v,r,c}}\right). \tag{8}$$

Exercise 4. *2-Col can be solved in linear time. Can 2-Col be solved in linear time by transforming instances to SAT and then running a SAT algorithm? Consider the two representations of k-colouring as CNF given above, for the special case when $K = 2$.*

Exercise 5. *An n-by-n Latin Square is an n by n matrix with entries in $[n] = \{1, \ldots, n\}$, such that no entry appears twice in any row or column. (It follows that each row and each column contains every number in $[n]$.)*

1. *Give the general form of a set Γ_n of clauses such that, for any $n \in N$, the satisfying truth assignments of Γ_n are in 1-1 correspondence with the $n \times n$ Latin squares. Use atoms $C_{i,j,k}$, intuitively saying that the entry (i, j) is k. (Your solution should look something like the formula (4) for graph colouring, above.)*

2. *Write out the exact set of clauses for n=3, and give one satisfying assignment for them (you can just write the atoms which are true, and state that the remainder are false).*

Constructing Latin squares is easy. However, given a table with numbers in some cells and other cells free, it is NP-complete to decide if the free cells can be filled to produce a Latin square. A quasigroup is a set with a binary operation $\cdot$ defined by a Latin square.

Definition 6. The Quasigroup Completion Problem (QCP), is:

> Given: A finite set S and a collection $C = \{C_1, \ldots\}$ of triples from $S \times S \times S$.
> Question: Is there a quasigroup with $\cdot$ consistent with C, that is, for each $(a, b, c) \in C$, $a \cdot b = c$?

QCP is NP-complete.

Exercise 6. *Describe a scheme for solving instances of QCP using a SAT solver.*

4 CNF Transformations

For a variety of reasons, almost all programs for solving propositional satisfiability are SAT solvers — they require input in CNF. It is not always easy to directly write a CNF formula describing the properties we want, and in applications often the formula we need to satisfy comes from some other piece of software which does not produce CNF. So we have a need to efficiently transform general formulas to CNF. The truth-table method, described in Section 1, is feasible only for very small formulas - indeed only for formulas small enough that the truth table method of checking satisfiability would be practical. This includes very few formulas describing interesting problems.

We can also transform a formula into a logically equivalent CNF formula by suitable re-writing of sub-formulas, for example by applying De Morgan's laws to eliminate conjunctions that are nested within disjunctions. This is also often not feasible, because it

often will produce a formula which is much larger than the original. And this is not avoidable: there are formulas A for which even the smallest equivalent CNF formula has size exponentially larger than the size of A.

Example 3. *Every CNF formula which is logically equivalent to*

$$(P_1 \wedge Q_1) \vee (P_2 \wedge Q_2) \vee \ldots \vee (P_n \wedge Q_n)$$

has at least 2^n *clauses.*

However, by using extra atoms, is it possible to transform any formula A into a CNF formula which can almost be treated as being equivalent, in linear time.

Before giving the method, we present two other transformations. The first is a widely used linear-time transformation from CNF to 3-CNF, which illustrates the main idea. The second is the transformation to a normal form with limited use of negation, which is often convenient to work with in practice.

4.1 Transformation of CNF to 3-CNF

Let Γ be a set of clauses. Produce a new set of clauses Γ' from Γ by applying the following rule, until there are no clauses of size greater than three:

> Let $C = (L_1 \vee L_2 \vee L_3 \vee L_4 \vee \ldots)$ be a clause of Γ of length greater than 3. Replace C with the two clauses $(L_1 \vee L_2 \vee T), (\overline{T} \vee L_3 \vee L_4 \vee \ldots)$, where T is a "new" atom - that is, an atom which does not appear in Γ.

Any truth assignment that satisfies Γ' also satisfies Γ, and any truth assignment that satisfies Γ can be extended, by selecting suitable values for the new atoms introduced in the re-writing, to an assignment which satisfies Γ'. Moreover, we can construct Γ' from Γ in linear time, and the total size (number of literal occurrences) of Γ' is no more than 3 times that of Γ.

4.2 Negation Normal Form

Definition 7. Formula A is in negation normal form (NNF) if the only negated subformulas of A are atoms.

We may transform a formula to NNF by repeated application of equivalences to "move negations inward". In particular, if we apply the following rules until none are applicable, we will have produced an equivalent formula in NNF.

1. If there is a sub-formula of the form $\neg(A \wedge B)$, re-write it as $(\neg A \vee \neg B)$.
2. If there is a sub-formula of the form $\neg(A \vee B)$, re-write it as $(\neg A \wedge \neg B)$.
3. If there is a sub-formula of the form $\neg\neg A$ re-write it as A.

This transformation to NNF can be carried out in linear time, by a recursive procedure which begins at the top of the abstract syntax tree for the formula, and applies the rules to move negations down toward leaves.

Exercise 7. *Transform each of the following formulas to a logically equivalent formula in NNF.*

1. $\neg(((\neg Q \wedge P) \vee \neg(P \vee \neg Q)) \vee \neg(\neg(\neg R \wedge Q) \wedge \neg(\neg P \wedge Q)))$
2. $((\neg(P \vee S)) \rightarrow (Q \vee P)) \rightarrow \neg((P \rightarrow Q) \vee (R \rightarrow S))$

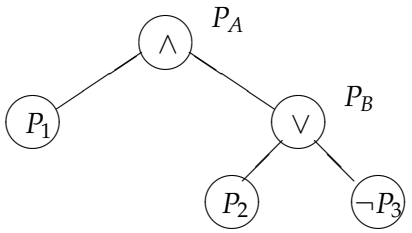
4.3 Tseitin's Polytime Transformation to CNF

We now define the efficient transformation to CNF. We describe a version only for formulas in NNF, but it is easy to extend to arbitrary formulas. Given a formula A, associate with each sub-formula B a literal that we denote P_B. If B is a literal, then P_B is just B; otherwise P_B denotes a new atom uniquely associated with sub-formula B. Now, let ToCNF(A) be the new formula defined as follows. For each sub-formula B of A, include in ToCNF(A) the following clauses:

1. Corresponding to the entire formula A, include the clause (P_A);
2. For each sub-formula B of the form $(C \vee D)$, include in ToCNF(A) the clause $(\neg P_B \vee P_C \vee P_D)$, which is equivalent to $(P_B \rightarrow (P_C \vee P_D))$;
3. For each sub-formula B of the form $(C \wedge D)$, include in ToCNF(A) the clauses $(\neg P_B \vee P_C)$ and $(\neg P_B \vee P_D)$, which together are equivalent to $(P_B \rightarrow (P_C \wedge P_D))$.

In the formula ToCNF(A), we have an atom for each sub-formula of A (including A itself). The clauses ensure that, for each sub-formula B with main connective c, any satisfying assignment for ToCNF(A) gives values to B and it's immediate sub-formulas that are consistent with the semantics of the connective c. For example, if sub-formula B is a disjunction, the clauses ensure that τ can only make B true if it also makes at least one of the sub-formulas of true. The "top clause" requires that P_A be true, and thus that τ satisfies A.

Example 4. *Let A be the formula $(P_1 \wedge (P_2 \vee \neg P_3))$. Here is the formula tree T_A. Each subformula of A that is not a literal is labelled with a distinct atom (we have chosen the atoms P_A and P_B).*



The formula ToCNF(A) is

$$(P_A), (\overline{P_A} \vee P_1), (\overline{P_A} \vee P_B), (\overline{P_B} \vee P_2 \vee \overline{P_3})$$

Clearly, if an assignment τ satisfies this formula, it must make P_A true, so must also make P_1 and P_B true, and therefore must make either P_2 or $\overline{P_3}$ true. Such an assignment also satisfies A.

Exercise 8.

1. *Let $A = ((\neg P \wedge Q) \vee (\neg R \wedge S))$, and write the set of clauses ToCNF(A).*
2. *Let $B = ((P \vee Q) \wedge ((R \wedge S) \vee (S \wedge Q)))$. Write the set of clauses ToCNF(B).*

Theorem 2. *Let τ be an assignment for the atoms of A, and α be an assignment for the atoms of CNF(A). Then,*

1. *If $\alpha \models$ ToCNF(A), then $\alpha \models A$.*
2. *If $\tau \not\models A$, then there is some extension τ' of τ to the atoms of ToCNF(A) such that $\tau' \not\models$ToCNF(A).*

Further, and importantly for practical use, CNF(A) can be produced from A by means of a simple linear-time algorithm.

Exercise 9.

1. *Show that, in general, A and ToCNF(A) are not equivalent, by giving a suitable truth assignment for the atoms of the formula ToCNF(A) where A is the formula from Part 1 of Exercise 8. Justify your choice of assignment.*
2. *Prove Theorem 2.*

Remark 1. The transformation originally given by Tseitin has more clauses than the one described above. For example, for a sub-formula B of the form $(C \vee D)$, in addition to the clause $(\overline{P_B} \vee P_C \vee P_D)$, it included the two clauses $(\overline{P_C} \vee P_B)$ and $(\overline{P_D} \vee P_B)$. The three clauses together are equivalent to the formula $(P_B \leftrightarrow (P_C \vee P_C))$ expressing the exact semantics of $\vee$. It also was defined for general formulas, not just formulas in NNF. (This version of the transformation also does not produce a logically equivalent formula.)

Exercise 10.

1. *Give the clauses that Tseitin would have included for a sub-formula B of the form $(C \wedge D)$.*
2. *Our two rules, 1 and 2, give a method for formulas in NNF only. Extend the method to work for arbitrary formulas by adding a rule to apply to negated sub-formulas which are not literals, and possibly altering the rules for conjunctions and disjunctions.*
3. *The obvious rule for handling negated sub-formulas adds one new atom for each occurrence of $\neg$ (other than those applied to atoms). Devise a method that works for arbitrary formulas, but does* not *add new atoms for negated sub-formulas, and does not require more clauses than our rules 1 and 2 for conjunctions of disjunctions.*

Remark 2. Other polytime transformations from general formulas to CNF are possible. There is some evidence that other transformations may result in better SAT solver performance in practice.
