Notes on Satisfiability-Based Problem Solving
Conjunctive Normal Form and SAT

David Mitchell
mitchell@cs.sfu.ca
September 19, 2013

This is a preliminary draft of these notes. Please do not 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 In discussions of algorithm complexity, \( n \) denotes the number of distinct atoms in a formula, and \( l \) denotes the length, or size, of the formula, which we usually measure by the total number of atoms.

1 Conjunctive Normal Form

A conjunction of formulas is a formula of the form \((A_1 \land A_2 \land \ldots \land A_m)\), and a disjunction of formulas is a formula of the form \((A_1 \lor A_2 \lor \ldots \lor A_m)\).

To satisfy the standard syntax for formulas, we must consider conjunctions or disjunctions of more than two formulas to be implicitly parenthesized. When necessary, we consider them parenthesized by association to the left, i.e., \((A_1 \land A_2 \land A_3 \land A_4)\) means \(((A_1 \land A_2) \land A_3) \land A_4\). However, the parenthesization does not matter semantically, and for our purposes here we may to ignore it and merely think of \((A_1 \lor A_2 \lor \ldots \lor A_m)\) as a formula for which \(\tau \models (A_1 \lor A_2 \lor \ldots \lor A_m)\) iff \(\tau\) satisfies at least one 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.

For convenience (and following standard practice) 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.

Further, it is often convenient to think of a clause as a set of literals, and a CNF formula as a set of clauses. This allows us to apply standard set operations and notation to CNF formulas. 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. Henceforth, we will 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. For each truth assignment $\tau$ (for exactly the atoms in $A$) with $\tau(A) = \text{false}$, include in $A'$ the clause which is the set of all literals $L$ with $\tau(L) = \text{false}$. An equivalent DNF can be constructed dually: by including a conjunction for each assignment which satisfies $A$.

**Example 1.** Let $A = ((P \land Q) \lor (\neg P \land \neg Q))$. The truth table for $A$, adorned with the clauses to include in the CNF, is:

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$A$</th>
<th>clause to include in $A'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>none</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>$(\neg P \lor Q)$</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>$(P \lor \neg Q)$</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>none</td>
</tr>
</tbody>
</table>

If we use the dual method to construct an equivalent DNF, we have a conjunction for the first and fourth lines of the table, which gives us $A$ 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 many quite difficult 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 $\tau$ 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. $(\neg A \lor B) \land (\neg B \lor C) \land (A \lor \neg C)$.
2. $(P \lor Q) \land (\neg P \lor Q) \land (\neg Q \lor P) \land (\neg P \lor \neg Q \lor R) \land (\neg R \lor \neg Q)$.
3. $(P \lor Q) \land (P \lor Q \lor \neg R \lor \neg Q)$
4. $(\neg P \lor \neg Q) \land (\neg P \lor \neg R) \land (\neg Q \lor \neg R) \land (P \lor Q) \land (P \lor R) \land (Q \lor R)$.

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 will be 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 be used. Normally this is done so that each assignment to the set of atoms corresponds to a combinatorial object which constitutes a potential solution.
2. Produce a set of clauses over those atoms which restricts satisfying assignments to those representing solutions.

We illustrate, using 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 monochromatic — 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 $Col : V \rightarrow C$ such that, for every $(u, v) \in E$, $Col(u) \neq Col(v)$?

In practice we are usually interested in actually finding a proper colouring, rather than merely knowing it exists. For a given $G$ and $C$, we want a CNF formula $\Gamma = \Gamma(G, C)$ such that a satisfying assignment for $\Gamma$ more-or-less directly gives us a proper colouring of $G$ with colours from $C$. Next, we will describe two different ways to do this, based on different choices for the atoms to use.
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

\[ \{ C_{v,c} \mid v \in V \text{ and } c \in C \}. \] (1)

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

1. For each vertex \( v \), a clause requiring that \( v \) gets some colour:

\[ \left( \bigvee_{c \in C} C_{v,c} \right); \] (2)

2. For each edge \( (u, v) \) a conjunction of clauses saying that \( u \) and \( v \) do not have the same colour:

\[ \bigwedge_{c \in C} (\overline{C_{u,c}} \lor \overline{C_{v,c}}). \] (3)

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

\[ \bigwedge_{v \in V} \left( \bigvee_{c \in C} C_{v,c} \right) \land \bigwedge_{(u,v) \in E} \bigwedge_{c \in C} (\overline{C_{u,c}} \lor \overline{C_{v,c}}). \] (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 \( G = \langle \{a, b, c\}, \{(a, b), (a, c)\} \rangle \), and \( C = \{R, B, G\} \) (using letters instead of numbers for readability). Then, \( \Gamma(G, C) \) is

\[
(C_{a,R} \lor C_{a,B} \lor C_{a,G}) \land (C_{b,R} \lor C_{b,B} \lor C_{b,G}) \land (C_{c,R} \lor C_{c,B} \lor C_{c,G}) \\
\land (\overline{C_{a,R}} \lor \overline{C_{b,R}}) \land (\overline{C_{a,B}} \lor \overline{C_{b,B}}) \land (\overline{C_{a,G}} \lor \overline{C_{b,G}}) \\
\land (\overline{C_{c,R}} \lor \overline{C_{b,R}}) \land (\overline{C_{c,B}} \lor \overline{C_{b,B}}) \land (\overline{C_{c,G}} \lor \overline{C_{b,G}})
\]

The truth assignment \( \{C_{a,R}, C_{b,B}, C_{c,G}, \overline{C_{a,R}}, \overline{C_{a,B}}, \overline{C_{a,G}}, \overline{C_{b,R}}, \overline{C_{b,B}}, \overline{C_{b,G}}, \overline{C_{c,R}}, \overline{C_{c,G}}\} \) satisfies \( \Gamma \), 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 post-processing step to generate proper colourings by deleting some “extra” colours, or add clauses to $\Gamma$ requiring at most one colour for each vertex, as in:

$$\bigwedge_{v \in V} \bigwedge_{c,c' \in C} (\overline{C_{v,c}} \lor \overline{C_{v,c'}}).$$

### 3.2 Graph Colouring Formulas: Version 2

In this version, we will represent the colour assigned to a vertex with a set of atoms which its binary encoding. Given $G = (V, E)$ and $C = \{1, \ldots K\}$, and letting $r = \lceil \log_2 K \rceil$, our set of atoms will be

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

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.

If $K > 1$ is a power of two, 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 mono-chromatic. If $K$ is not a power of two, there are assignments to $B_{v,1}, \ldots B_{v,r}$ which do not correspond to colours, and we also need clauses to disallow these “non colours”.

To describe the clauses we will use the notation $k_i$ for the $i^{th}$ bit of the binary encoding of the number $k$, and write $B_{j} = k_i$ for the literal that is made true by truth assignments which give $B_{v,i}$ the value of bit $k_i$. That is:

$$B_{v,i} = k_i \text{ denotes } \begin{cases} B_{v,i} & \text{if } k_i = 1 \\ \overline{B_{v,i}} & \text{if } k_i = 0, \end{cases} \quad (7)$$

and $B_{v,i} \neq k_i$ denotes the complement of $B_{v,i} = k_i$.

The clauses which disallow the assignments which correspond to “non-colours” are, for each vertex $v$ and each number $k \in \{K + 1 \ldots 2^r\}$:

$$(B_{v,1} \neq k_1 \lor \ldots \lor B_{v,r} \neq k_r) \lor \ldots \lor (B_{v,1} \neq k_1 \lor \ldots \lor B_{v,r} \neq k_r). \quad (8)$$
To disallow monochromatic edges, for each edge \((u, v) \in E\), and each colour in \(c \in C\), we have:

\[
(B_{u,1} \neq k_1 \lor \ldots \lor B_{u,r} \neq k_r \lor B_{v,1} \neq k_1 \lor \ldots \lor B_{v,r} \neq k_r).
\]

**Exercise 4.** It is well-known that 2-Col can be solved in polynomial time, and in fact in linear time. Consider the representations of \(K\)-colouring as CNF given in the notes on CNF, for the special case when \(K = 2\). What can we say about the CNF representations of 2-Col?

**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 \(\Gamma_n\) of clauses such that, for any \(n \in N\), the satisfying truth assignments of \(\Gamma_n\) 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 \(k\).

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 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 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, such that, 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

It is sometimes easy to write a set of propositional formulas describing the properties we want, but not convenient to write them as a set of clauses. However, most programs for solving propositional satisfiability are SAT solvers — they take input in CNF only
— 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 the 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, i.e., by applying De Morgan’s rules. This is also often not feasible, because it may 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 which is exponential in the size of $A$.

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

$$(P_1 \land Q_1) \lor (P_2 \land Q_2) \lor \ldots \lor (P_n \land Q_n)$$

has at least $2^n$ clauses.

However, it is possible, by adding extra atoms, to transform any formula $A$ into a CNF formula which is almost 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 which simplifies reasoning about negations.

### 4.1 Transformation of CNF to 3-CNF

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

Let $C = (L_1 \lor L_2 \lor L_3 \lor L_4 \lor \ldots)$ be a clause of $\Gamma$ of length greater than 3. Replace $C$ with the two clauses $(L_1 \lor L_2 \lor T), (T \lor L_3 \lor L_4 \lor \ldots)$, where $T$ is a “new” atom — that is, an atom which does not appear in $\Gamma$.

Any truth assignment that satisfies $\Gamma'$ also satisfies $\Gamma$, and that any truth assignment that satisfies $\Gamma$ can be extended, by selecting suitable values for the new atoms introduced in the re-writing, to an assignment which satisfies $\Gamma'$. Moreover, we can construct $\Gamma'$ from $\Gamma$ in linear time.
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” as far as possible. In particular, if we apply the following rules none are applicable, we will have produced an equivalent formula in NNF.

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

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

1. \( \neg ((\neg Q \land P) \lor (P \lor \neg Q)) \lor (\neg (\neg R \land Q) \land (\neg P \land Q)) \)
2. \( ((\neg (P \lor S) \rightarrow (Q \lor P)) \rightarrow (\neg (P \rightarrow Q) \lor (R \rightarrow S)) \)

4.3 Tseitin’s Polytime Transformation to CNF

We now define the transformation, due to Tseitin, to CNF. We describe it only for formulas in NNF - it easy to extend to arbitrary formulas. For every sub-formula B of A, let \( P_B \) denote B if B is a literal, and a new atom if B is not a literal. Include in CNF(A) the “top clause” \((P_A)\), plus for each sub-formula B of A:

1. if sub-formula B is \((C \lor D)\), include the clauses \((\neg P_B \lor P_C \lor P_D)\), \((\neg P_C \lor P_B)\), and \((\neg P_D \lor P_B)\), which together are equivalent to \((P_B \leftrightarrow (P_C \lor P_D))\);
2. if B is \((C \land D)\), include the clauses \((\neg P_B \lor P_C)\), \((\neg P_B \lor P_D)\), \((\neg P_C \lor \neg P_D \lor P_B)\), which together are equivalent to \((P_B \leftrightarrow (P_C \land P_D))\).

The set of clauses produced for the sub-formulas of A ensure that, for every truth assignment \( \tau \) that satisfies the clauses, the truth value \( \tau \) gives to any one of the “new” atoms is the same as the truth value \( \tau \) gives to the corresponding sub-formula. The “top clause” requires that \( P_A \) be satisfied, and thus that \( \tau \) satisfies A.
Theorem 2. Every assignment that satisfies CNF(A) also satisfies A, and every assignment that satisfies A can be extended to an assignment that satisfies CNF(A).

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

Exercise 8.

1. Let \( A = (\neg P \lor Q) \land (\neg R \lor S) \), and write the set of clauses CNF(A).

2. Give a truth assignment, for the atoms of the formula CNF(A) from Part 1, which demonstrates that A and CNF(A) are not logically equivalent. Justify your choice.

3. Let \( B = (P \lor Q) \land ((R \land S) \lor (S \land Q)) \). Write the set of clauses CNF(B).

4.4 Refinements of the Transformation to CNF

The transformation given in Section 4.3 includes more clauses than are needed to obtain the property of Theorem 2. We illustrate with a simple example. Let \( A = ((Q \lor R) \land (Q \lor S)) \). Denote the subformulas \((Q \lor R)\) and \((Q \lor S)\) by \( X \) and \( Y \), respectively. Then \( \text{CNF}(A) \) is

\[
(P_A), (P_A \rightarrow P_X), (P_A \rightarrow P_Y), (P_X \land P_Y \rightarrow P_A), (P_X \rightarrow P_Q \lor P_R), (P_Q \rightarrow P_X), (P_R \rightarrow P_X).
\]

Now consider the formula \( \text{CNF}'(A) \):

\[
(P_A), (P_A \rightarrow P_X), (P_A \rightarrow P_Y), (P_X \rightarrow P_Q \lor P_R).
\]

\( \text{CNF}(A) \) and \( \text{CNF}'(A) \) are not equivalent, but \( \text{CNF}'(A) \) serves the same purpose as \( \text{CNF}(A) \).

In Section 4.3 we gave the transformation for formulas in NNF only. It is easy to extend to general formulas by adding the rule:

3. if \( B \) is \( \neg C \), include the clauses \((\neg P_B \lor P_C), (P_B \lor \neg P_C)\).

However, this results in new clauses and atoms which are not really necessary: we can delete these two binary clauses, and replace \( P_B \) with \( \neg P_C \) in the remaining clauses, and obtain a formula which accomplishes the same goal but with one less atom and two fewer clauses. One may accomplish the equivalent by adapting the rules for conjunctions and disjunctions to the cases of negated conjunctions and disjunctions:
1a. if sub-formula $B$ is $\neg(C \lor D)$, include the clauses $(P_B \lor P_C \lor P_D)$, $(\neg P_C \lor \neg P_B)$, $(\neg P_D \lor \neg P_B)$;

2a. if $B$ is $\neg(C \land D)$, include the clauses $(P_B \lor P_C)$, $(\neg P_B \lor P_D)$, $(\neg P_C \lor \neg P_D \lor \neg P_B)$.

It is possible to combine both of the ideas just presented, and obtain a reduced size transformation to CNF for arbitrary formulas, but some care is required because of the interaction between negations and the roles of conjunction and disjunction (think of De Morgan’s laws).

It is also possible to reduce the size of the CNF further, for example by “flattening” nested conjunctions and disjunctions. For example, if $X$ denotes the sub-formula $((T \land Q) \land (R \land S))$, we can think of it as $(T \land Q \land R \land S)$, for which we introduce the clauses $(P_X \rightarrow P_T)$, $(P_X \rightarrow P_Q)$, $(P_X \rightarrow P_R)$, and $(P_X \rightarrow P_S)$. This is 2 fewer clauses and 2 fewer new atoms that the standard rules would produce.

---

*Main changes from version of Sept. 6, 2013:*
- SAT moved into new Section 2, and K-SAT added;
- Examples and exercises added;