Other NP-Complete Problems
More terminology for Boolean expressions:

- A *literal* is a variable or the negation of a variable.
- A *clause* is a single literal or the disjunction (OR) of literals.
- A Boolean expression is in *conjunctive normal form* if it is a single clause or the conjunction (AND) of clauses. For example, 
  \((\sim x \lor \sim y \lor z) \land (x \lor \sim y \lor \sim z)\)

CNF-SAT is the language of satisfiable conjunctive normal form expressions.
Theorem: CNF-SAT is NP-Complete.
Proof: We will show that SAT reduces (in polynomial time) to CNF-SAT. In other words we will start with a Boolean expression s and produce expression s' so that s is in SAT if and only if s' is in CNF-SAT.

If we had a truth table for s it would be easy to make s'. For example, suppose we know that the only times s is F is when x=T, y=T, z=F and when x=F, y=T, z=T. We can build clauses that negate these instances:

$$s' = (\neg x \lor \neg y \lor z) \land (x \lor \neg y \lor \neg z)$$

Unfortunately, building a truth table for s takes exponential time.
Rather than building a truth table, given $s$ we will build a CNF expression $s'$ that has additional variables (and so is not equivalent to $s$) but is satisfiable if and only if $s$ is satisfiable.

Step 1: Parse $s$ into a parse tree.
For example, if $s$ is $\sim(x \lor \sim y) \lor \sim z$ the parse tree is

```
   V
  /\  \
 /   \ / \
\sim  \sim
  |  |  |
  x  z
  |  /  |
  \sim y
  |  \
  \  |
  y
```
Step 2: Walk down the tree using DeMorgan's laws to push negations to variables.
Step 3. Start at the leaves and walk up, replacing each node with a CNF expression that is satisfiable if and only if the subtree rooted at the node is satisfiable.

Case 3A: Suppose the tree is

\[ \land 
\quad \land 
\quad \land 
\]

and we have already replaced E1 with CNF expression F1 and E2 with F2. We replace the \(\land\)-node with F1\(\land\)F2.
Case 3B:

Suppose the tree is

\[
\begin{array}{c}
\lor \\
E1 & E2
\end{array}
\]

and we have already replaced E1 with CNF expression
\[
F1 = g_1 \land g_2 \land g_3 \land ... \land g_K \text{ (the } g_i \text{ are the clauses of } F1) \text{ and } E2 \text{ with }
\]
\[
F2 = h_1 \land h_2 \land h_3 \land ... \land h_L.
\]

Let \( y \) be a new variable not used in \( s \) or any of the \( F \)-expressions. We replace the \( \lor \)-node with
\[
F = (y \lor g_1) \land (y \lor g_2) \land ... \land (y \lor g_K) \land (\neg y \lor h_1) \land (\neg y \lor h_1) \land ... \land (\neg y \lor h_L)
\]

If \( y = T \) this requires \( h_1 \land h_2 \land h_3 \land ... \land h_L \) to be \( T \), so \( F2 \) must be \( T \). Similarly, if \( y = F \) then \( F1 \) must be \( T \). \( F \) is satisfiable if and only if \( F1 \lor F2 \) is satisfiable.
By the time we get to the root of the tree this has produced a CNF expression $s'$ that is satisfiable if and only if $s$ is satisfiable. If the length of $s$ is $n$ then $s$ has no more than $n$ literals, each with length no more than $n$, so $|s'| \leq n^2$. 
Example: In an earlier example we parsed \( s = \neg(x \lor \neg y) \lor \neg z \) as

\[
\begin{align*}
\lor & \quad \neg \quad \neg \\
\lor & \quad \neg \quad \neg \\
\land & \\
\lor & \quad \neg \\
\land & \\
\lor & \quad \neg \\
\land & \quad \neg \\
\land & \\
\end{align*}
\]

and converted that to

\[
\begin{align*}
\land & \\
\land & \\
\land & \\
\land & \\
\end{align*}
\]

The corresponding CNF expression is \((w \lor \neg x) \land (w \lor y) \land (\neg w \lor \neg z)\)
Example: Start with \( \sim(x \land (y \lor z)) \lor \sim x \lor (y \land \sim z) \). This parses into

\[
\begin{align*}
& \lor \\
& \sim \\
& \land \\
& x \\
& \lor \\
& \sim \\
& \land \\
& y \\
& \lor \\
& \sim \\
& \land \\
& z
\end{align*}
\]

which converts to

\[
\begin{align*}
& \lor \\
& \sim \\
& \land \\
& \sim \\
& \land \\
& x \\
& \lor \\
& \sim \\
& \land \\
& y \\
& \lor \\
& \sim \\
& \land \\
& z
\end{align*}
\]
Node A becomes \((w \lor \neg x) \land (\neg w \lor \neg y) \land (\neg w \lor \neg z)\)

B becomes \((t \lor \neg x) \land (\neg t \lor \neg y) \land (\neg t \lor \neg z)\)

C becomes
\[(u \lor w \lor \neg x) \land (u \lor \neg w \lor \neg y) \land (u \lor \neg w \lor \neg z) \land (\neg u \lor t \lor \neg x) \land (\neg u \lor \neg t \lor y) \land (\neg u \lor \neg t \lor z)\]
3CNF is the language of conjunctive normal form expressions where each clause has exactly 3 literals. For example, one expression in 3CNF is $(x \lor \sim y \lor z) \land (x \lor y \lor \sim z)$

3CNF-SAT (also called 3SAT) is the language of satisfiable 3CNF expressions.
Theorem: 3CNF-SAT is NP-Complete
Proof: We will reduce CNF-SAT to 3CNF-SAT by converting CNF expressions to 3CNF expressions.

Let \( e = e_1 \land e_2 \land e_3 \land \ldots \land e_k \) be an expression in CNF. Each \( e_i \) must be a disjunction of literals.

a) Suppose \( e_i \) has only one literal, \( x \). Let \( r \) and \( s \) be new variables. Replace \( e_i \) by
\[
(f_i = (x \lor r \lor s) \land (x \lor \lnot r \lor \lnot s) \land (x \lor r \lor \lnot s) \land (x \lor \lnot r \lor \lnot s))
\]
f_i can be satisfied if and only if \( x \) is satisfied.

b) Suppose \( e_i \) has only two literals, such as \( x \lor y \). Let \( r \) be a new variable and replace \( e_i \) by
\[
(f_i = (x \lor y \lor r) \land (x \lor y \lor \lnot r))
\]
c) Suppose \( e_i \) has 4 literals: \( e_i = x_1 \lor x_2 \lor x_3 \lor x_4 \). Let \( r \) be a new variable. Then \( f_i = (x_1 \lor x_2 \lor r) \land (x_3 \lor x_4 \lor \neg r) \)

d) Suppose \( e_i \) has 5 literals: \( e_i = x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5 \). Let \( s_1 \) and \( s_2 \) be new variables. Then

\[
   f_i = (x_1 \lor x_2 \lor s_1) \land (x_3 \lor \neg s_1 \lor s_2) \land (x_4 \lor x_5 \lor \neg s_2)
\]

<table>
<thead>
<tr>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( f_i ) reduces to</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>( x_4 \lor x_5 )</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>( x_3 )</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>( (x_1 \lor x_2) \land (x_4 \lor x_5) )</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>( x_1 \lor x_2 )</td>
</tr>
</tbody>
</table>
We can extend this pattern to any number of literals. If $e_i$ has $n$ literals then $f_i$ has $n-2$ clauses each with 3 literals and uses $n-2$ new variables. $|f_i| \leq 3|e_i|$ so the length of the 3CNF expression this builds is a polynomial function of the length of the original CNF expression.