NP – Completeness Continued

Richard Cole

December 3, 2007

Example 8 General Satisfiability

Input Boolean Formula $F$.

$F$ is an expression over Boolean variables using the operators $\land, \lor, \neg$ (not). e.g. $F = [\neg(x_1 \lor \overline{x}_2) \land x_3] \lor (x_3 \land \overline{x}_2)$.

Question Is $F$ satisfiable?

Example 9 3-SAT

Input 3-CNF formula $F$.

A 3-CNF formula is one in which each clause contains at most 3 literals.

Question Is $F$ satisfiable?

Claim 5 Given a polynomial time algorithm for 3-SAT there is a polynomial time algorithm for general satisfiability.

Proof First rewrite $F$ as the equivalent Boolean formula $F'$ where all the “nots” in $F'$ are applied directly to the literals. So for the above example, $F' = [(\overline{x}_1 \land x_2) \land x_3] \lor (x_3 \land \overline{x}_2)$. This is done by applying the rules $\neg(x \lor y) = \overline{x} \land \overline{y}$ and $\neg(w \land z) = \overline{w} \lor \overline{z}$.

Next we give a 3-CNF formula $G$ such that $G$ is satisfiable if and only if $F'$ is satisfiable. To this end, view $F'$ as an expression tree. For the above example, as

```
\( \neg x_1 \lor \neg x_2 \) \land (x_3 \lor \overline{x}_2)
```

In $G$, for each node in the tree, we introduce a new variable. Let $r$ be the variable at the root. For the configuration

```
\( x \lor y \lor z \)
```

we introduce the clauses $C \land D \land E = (\overline{x} \lor y \lor z) \land (x \lor \overline{y}) \land (x \lor \overline{z})$. We observe that these 3 clauses are satisfiable if and only if $x$ and $y \lor z$ have the same truth value. For if
$x = \text{True}$ then $y \lor z$ must be true to cause $C$ to evaluate to True ($D \land E$ evaluates to True with $x = \text{True}$). While if $x = \text{False}$, then both $\overline{y}$ and $\overline{z}$ must be True or $y \lor z$ must be False to cause $C \land D \land E$ to evaluate to True. Thus if $C \land D \land E$ is True then $x$ and $y \lor z$ have the same truth value.

Similarly, for the configuration

we introduce the clauses $A \land B \land C = (u \lor \overline{v} \lor \overline{w}) \land (\overline{u} \lor v) \land (\overline{u} \lor w)$.

Again, if $A \land B \land C$ evaluates to True then $u$ and $v \land w$ have the same truth value. For if $u$ is True then both $v$ and $w$ are True. While if $u$ is False then $\overline{v} \lor \overline{w}$ is True and $\overline{u} \lor w = \neg(v \land w)$, that is $v \land w$ is False.

So $G$ consists of the "and" of the triples of clauses, one triple per internal node in the expression tree, plus the clause $(r)$. This forces $r$ to be set to True; the remaining triples of clauses state that $G$ evaluates to the same truth value as the expression tree, namely as formula $F$. Thus $F$ is satisfiable by a truth setting to its variables if and only if $G$ is satisfiable by the same truth setting to these variables.

So the algorithm for testing $F$'s satisfiability is as follows.

1. Build $G$ (this takes time linear in $|F|$).
2. Run the 3-SAT algorithm on input $G$.
3. Report the result from Step 2.

Clearly, this algorithm runs in polynomial time.

\[\square\]

**Example 10** Undirected Hamiltonian Circuit (UHC).

**Input:** An undirected graph $G$.

**Question** Does $G$ have a Hamiltonian Circuit, i.e. a circuit that goes through each node exactly once?

**Claim 6** Given a polynomial time for SAT there is a polynomial time algorithm for UHC.

**Proof** Given an input graph $G = (V, E)$, we construct a CNF formula $F$ s.t. $F \in \text{SAT} \iff G \in \text{UHC}$.

Let $|V| = n$. $F$ has $E \cdot V$ variables $y_{u,v}^i, 1 \leq i \leq n$, and $(u,v) \in E$. The meaning of a variable is that $y_{u,v}^i = \text{True}$ if and only if $(u,v)$ is the $i$th edge in the Hamiltonian Circuit (on choosing an arbitrary first edge and traversal direction). We write $F$ as an "and" of subformulas which ensure that $F$ can evaluate to True if and only if $G$ has a Hamiltonian Circuit.

First, $F_{v}^{\overline{f}}$ ensures node $v$ has at least one incoming edge.

$$F_{v}^{\overline{f}} = \bigvee_{u : (u,v) \in E} y_{u,v}^i$$

$$1 \leq i \leq n$$
$F^{I_2}_v$ ensures node $v$ has at most one incoming edge.

$$F^{I_2}_v = \bigwedge_{i \neq u} (\overline{y}_{u,v}^i \lor \overline{y}_{t,v}^i) \land \bigwedge_{u : (u,v) \in E} \bigwedge_{1 \leq i, j \leq n} (\overline{y}_{u,v}^i \lor \overline{y}_{u,v}^j).$$

Second, there are analogous constraints to ensure $v$ has exactly one outgoing edge:

$$F^{O_1}_v = \bigvee_{w : (v,w) \in E} y_{v,w}^i,$$

$$F^{O_2}_v = \bigwedge_{w \neq x} (\overline{y}_{v,w}^i \lor \overline{y}_{v,w}^j) \land \bigwedge_{x : (v,x) \in E} \bigwedge_{1 \leq i, j \leq n} (\overline{y}_{v,w}^i \lor \overline{y}_{v,w}^j).$$

The constraint $\land_{u} (F^{I_1}_v \land F^{I_2}_v \land F^{O_1}_v \land F^{O_2}_v)$ is True ensures that $G$ has one or more cycles that go through every vertex exactly once. To ensure that there is one cycle, we use subformula $F^A$ to ensure the $i$th edge is followed by the $[(i + 1) \mod n]$th edge, for each $i, 1 \leq i \leq n$.

$$F^A = \bigvee_{1 \leq i \leq n} [y_{u,v}^i \Rightarrow \bigvee_{w : (v,w) \in E} y_{w,u}^{[(i+1) \mod n]}]$$

$$(u,v) \in E$$

Note that a formula $C \Rightarrow D$ can be rewritten as $\overline{C} \lor D$, so

$$F^A = \bigvee_{1 \leq i \leq n} [\overline{y}_{u,v}^i \lor \bigvee_{w : (v,w) \in E} y_{w,u}^{[(i+1) \mod n]}]$$

$$(u,v) \in E$$

Finally,

$$F = F^A \land \bigwedge_{v} (F^{I_1}_v \land F^{I_2}_v \land F^{O_1}_v \land F^{O_2}_v).$$

Clearly, given $G$, $F$ can be built in polynomial time. It remains to argue that $F \in SAT \Rightarrow G \in UHC$.

Suppose that $G \in UHC$. Let $v_1, v_2 \cdots v_n$ be a Hamiltonian Circuit of $G$. Then set $y_{v_1,v_{i+1}}^i = True$ for $1 \leq i \leq n - 1$, $y_{v_n,v_1}^n = True$, and all other variables to False. It is readily checked that $F$ evaluates to True.

Conversely, suppose that $F \in SAT$. Consider a satisfying assignment. For each $v$, as $F^{I_1}_v$ is True at least one of $y_{u,v}^i$ is True for $(u,v) \in E$ and $1 \leq i \leq n$. As $F^{I_2}_v$ is True, at most one
of these variables is True. Let $y_{vw}^i$ be the True variable. Similarly, exactly one of the variables $y_{vw}^j, 1 \leq j \leq n, (v, w) \in E$ is True. As $F^A$ is True, $j = i + 1$. Let $y_{v_{2}v_{3}}, y_{v_{3}v_{4}}, \ldots, y_{v_{n-1}v_{n}}, y_{v_{n}v_{1}}$ be the True variables in superscript order. Then it is easy to check that $v_{1}v_{2}\cdots v_{n}$ is a Hamiltonian Circuit for $G$.

Thus $F \in \text{SAT} \iff G \in \text{UHC}$.

This leads to the following algorithm:

1. Build $F$.
2. Run $A_{\text{SAT}}(F)$ and report its answer.

As $F$ is polynomial sized (as a function of $|G|$), if $A_{\text{SAT}}$ runs in polynomial time then so does this algorithm.

□

**Definition** A language $L$ is NP-complete ($L \in \text{NPC}$) if

1. $L \in \text{NP}$, and
2. For every $J \in \text{NP}$, $J \leq_{P} L$.

**Reminder** $J \leq_{P} L$ means that given a polynomial time algorithm $A_J$ for $L$ there is a polynomial time algorithm for $J$ of the following form:

1. On input $x$, in polynomial time compute $y = f(x)$
2. Run $A_J(f(x))$ and report its answer. For this to be correct, it must be that

   $$x \in J \iff f(x) \in L.$$ 

**Lemma 7** If $J \leq_{P} K \leq_{P} L$ then $J \leq_{P} L$.

**Proof** As $J \leq_{P} K$ there is a polynomial time computable function $f$ such that $x \in J \iff f(x) \in K$. And as $K \leq_{P} L$ there is a polynomial time computable function $g$ such that $y \in K \iff g(y) \in L$. So $x \in J \iff g(f(x)) \in L$, and $g \circ f$ is also polynomial time computable. That is, $J \leq_{P} L$.

□

**Lemma 8** If $k$ is NP-complete and we show that

(i) $L \in \text{NP}$, and

(ii) $K \leq_{P} L$
then we can conclude $L$ is NP-Complete also.

**Proof** It suffices to show that for all $J \in \text{NP}$, $J \leq_p L$. But as $K$ is NP-complete, $J \leq_p K$, and we know $K \leq_p L$. By Lemma 7, we conclude $J \leq_p L$.

Thus once one problem is shown to be NP-Complete it becomes much easier to show additional problems are also NP-Complete.

We now sketch a proof of Cook’s Theorem.

**Cook’s Theorem (1972)** General SAT is NP Complete.

**Proof** Ultimately, the proof amounts to viewing the computation of a computer as nothing more than the evaluation of a giant, albeit polynomial sized circuit, which can be viewed as a Boolean formula.

Let $L \in \text{NP}$. Let $V_L$ be the polynomial time verifier algorithm for $L$. Let $x$ be an input string to be tested for membership in $L$. Let $V_L$ run in time bounded by polynomial $p(x)$. Then we built the following polynomial sized circuit $D(x)$ to compute the output of $V_L$.

The circuit will have $p(x)$ levels. Each level simulates one step of the computer running $V_L$. Surely, there is a circuit for each level, for in the end what does a computer do in one step? It can move a datum from one location to another, or perform a simple operation (addition, comparison, etc.). The nodes of the circuit, which can also be thought of as Boolean variables, hold all the different bits stored in the memory and registers of the computer. The input to the $i$th level of the circuit is just the state or memory contents of the computer at the start of its $i$th step; the output of the $i$th step are the memory contents at the end of the $i$th step; this is also input to the $(i+1)$th step.

The output value we are concerned with is the variable corresponding to the Boolean output of $V_L$, at the $p(x)$th level. (One small detail is that if the algorithm terminates before $p(x)$ steps, then in subsequent steps the state of the computer is unchanged so that the output variable has the same value after $p(x)$ steps.) The question of whether there is a certificate for $x$, i.e. where $x \in L$, amounts to asking whether the circuit can have output True (the free variables in the circuit are the inputs corresponding to the certificate).

But the circuit just employs ‘and’, ‘or’ and ‘not’ operations, so it amounts to a Boolean formula $F$ of polynomial size and consequently, the question becomes whether $F$ can evaluate to True, i.e. whether $F \in \text{General Set}$.

**Definition** Language $L$ is NP-hard if for all $K \in \text{NP}$, $K \leq_p L$.

**Lemma 9** $J$ is NP-hard and $J \leq_p L$, then $L$ is NP-hard. This is proved in the same way as Lemma 8.

**Lemma 10** 3-SAT $\in \text{NPC}$, SAT $\in \text{NPC}$, VC $\in \text{NPC}$, HC $\in \text{NPC}$.

(VC is vertex cover, HC is Hamiltonian Circuit).
Proof We have shown that each of these is in NP. Cook's Theorem shows General SAT ∈ NPC. We have shown the reductions General Set ≤_P SAT, SAT ≤_P 3-SAT, 3-SAT ≤_P VC, 3-SAT ≤_P HC. The result follows by Lemma 8.

□