CSE547 Class 20

Jeremy Buhler

April 5, 2017

1 A Simpler SAT

We showed last time that SAT is NP-complete.

- Maybe if we restrict the problem, it will become easier?
- Will consider two restrictions on form of input formula.
- Let ϕ be a Boolean formula over a set X of variables
- **Defn**: ϕ is a *CNF formula* (conjunctive normal form) if it has the form

$$\bigwedge_{j=1}^{m} \bigvee_{k=1}^{n_j} \ell_k^j$$

  where each ℓ_k^j is either x or $\neg x$ for some $x \in X$.

- The expressions ℓ_k^j are called *literals*.
- The m disjunctions are called *clauses*.
- **Example**:

$$(\neg x \vee y \vee z) \wedge (x \vee y) \wedge (x \vee \neg w).$$

- (This formula has 3 vars and 3 clauses.)
- **Defn**: if each clause of ϕ contains *exactly* 3 literals, then ϕ is said to be in *3-CNF form*.

How hard is SAT if we restrict its inputs to only 3-CNF formulas?

2 3-SAT is Still Hard

- Will consider the **3-CNF-SAT**, or just **3-SAT**, problem.
- Given a 3-CNF formula ϕ, is there a satisfying assignment to ϕ?
- is 3-CNF-SAT NP-complete?
- If we can solve SAT, we can trivially solve 3-CNF-SAT.

- But this only shows 3-CNF-SAT $\leq_p$ SAT.
- "Soln to a hard problem can be used to solve an easy problem" – duh.
- We need to prove the *other* direction.

Claim: 3-CNF-SAT is NP-complete.

- **Pf**: first, must show that 3-CNF-SAT is in NP.
- I'll just reuse NP-ness proof for SAT; same certificate and verification scheme works.
- Still must show NP-hardness.
- Will prove that SAT $\leq_p$ 3-CNF-SAT.
- Given a formula ϕ, we will construct a 3-CNF formula $\psi = f(\phi)$.
- Will show that ψ is satisfiable iff ϕ is satisfiable.

OK, here's the reduction, given ϕ.

- **Step 1**: push all $\neg$s to inside to create an equivalent formula ϕ_1 using only ands, ors, and literals.
- **Example**: if $\phi = (x \wedge y) \vee \neg(x \wedge z)$, then

$$\phi_1 = (x \wedge y) \vee (\neg x \vee \neg z).$$

- (Takes $O(|\phi|)$ transformations, each in time $O(|\phi|)$)
- **Step 2**: construct a *parse tree* for ϕ_1.
- **Example**:

- Each leaf of parse tree contains a literal.
- Each internal node contains a binary connective.
- At most as many internal nodes as leaves, so tree has $O(|\phi|)$ nodes.
- Number all nodes of tree; for each node j, assign a variable y_j.
- **Draw y's on above parse tree**
- **Step 3**: construct a formula ψ_0 from parse tree as follows.

- If node j is a leaf with literal ℓ, set

$$C_j = (y_j \iff \ell).$$

- If node j connects nodes p and q with connective $\oplus$, set

$$C_j = (y_j \iff y_p \oplus y_q).$$

- Finally, if y_1 labels the root of parse tree T, set

$$\psi_0(T) = y_1 \wedge \bigwedge_j C_j.$$

- **Example**:

- (Note that $\psi_0(T)$ has size linear in $|T|$, and hence linear in $|\phi_1|$.)
- **Claim**: ψ_0 is satisfiable iff ϕ_1 is satisfiable.
- **Pf** (sketch): suppose assignment A satisfies ψ_0.
- Can show inductively on structure of T that A makes y_j true iff A's assignment to literals satisfies subformula of ϕ_1 corresponding to subtree rooted at node j.
- Since A makes y_1 true, the whole formula ϕ_1 must be satisfiable.
- Conversely, if assignment A satisfies ϕ_1, construct assignment A' for ψ_0 from A by setting each y_j in bottom-up fashion to make its C_j true. QED

We now want to turn ψ_0 into an equivalent 3-CNF formula.

- Each subformula C_j has *at most* 3 variables but is not necessarily CNF.
- **Step 4**: turn each C_j into equivalent CNF formula α_j of at most constant size.
- First, write down the truth table for C_j.
- Then, build an equivalent *disjunctive* formula giving all the 0s of C_j.
- Finally, negate the result to achieve a CNF formula for C_j.

- **Example**: suppose we have formula $\neg(x \otimes y)$ ($\otimes$ means "exclusive-or").

- Transformation turns C_j into at most 2^3 clauses, each with at most 3 variables.
- Hence, $|\alpha_j|$ is $O(1)$.
- let $\psi_1 = \bigwedge_j \alpha_j$.
- Observe that $|\psi_1| = O(|\psi_0|)$, and that ψ_1 is satisfiable iff ψ_0 is.

Almost done.

- Formally, a 3-CNF formula must have *exactly* three literals per clause.
- Our ψ_1 might have only 1 or 2 literals.
- **Step 5**: transform each α_j into a valid 3-CNF formula.
- Let p and q be dummy variables.
- Replace each 2-literal clause $\ell_1 \vee \ell_2$ by

$$(p \vee \ell_1 \vee \ell_2) \wedge (\neg p \vee \ell_1 \vee \ell_2)$$

- Above is logically eqv to $(p \wedge \neg p) \vee (\ell_1 \vee \ell_2)$, so is satisfied precisely when $\ell_1 \vee \ell_2$ is satisfied (for any p).
- Replace each 1-literal clause ℓ by

$$(\neg p \vee \neg q \vee \ell) \wedge (\neg p \vee q \vee \ell) \wedge (p \vee \neg q \vee \ell) \wedge (p \vee q \vee \ell)$$

- Above is logically eqv to $(p \wedge \neg p) \vee (q \wedge \neg q) \vee \ell$, so is satisfied precisely when ℓ is satisfied (for any p, q).
- Let ψ be formula obtained by ψ_1 by above transformation.
- Each clause of ψ_1 expands to at most 4 clauses, so $|\psi|$ is $O(|\psi_1|)$.
- Moreover, ψ is logically equivalent to ψ_1.
- Hence, ψ is satisfied iff ϕ is satisfied!
- Moreover, $|\psi|$ is $O(|\phi|)$, and ψ can be built from ϕ in time $O(|\phi|^2)$.
- Conclude that SAT $\leq_p$ 3-CNF-SAT! QED