Com S 311 Section B
Introduction to the Design and Analysis of Algorithms

Lecture Two for Week 12

Xiaoqiu (See-ow-chew) Huang

Iowa State University

April 15, 2021

NP-Completeness Proofs

We show how to prove that languages are NP-complete by reducing a known NP-complete language to them.

Lemma 34.8
A language L is NP-hard if there exists an NP-complete language L' such that $L' \leq_P L$. In addition, if $L \in \mathrm{NP}$, then $L \in \mathrm{NPC}$.
Proof
Since L' is NP-complete, for all $L'' \in \mathrm{NP}$, we have $L'' \leq_P L'$.

By the assumption that $L' \leq_P L$ and by transitivity, we have $L'' \leq_P L$. This shows that L is NP-hard.

If $L \in \mathrm{NP}$, then we also have $L \in \mathrm{NPC}$.

Method for Showing that L is NP-Complete

1. Prove that $L \in \mathrm{NP}$.
2. Select a known NP-complete language L'.
3. Describe an algorithm computing a function f.
4. Show that $x \in L'$ if and only if $f(x) \in L$ for all $x \in \{0,1\}^*$.
5. Prove that the algorithm runs in polynomial time.

Formula Satisfiability

We use the method to show that the formula satisfiability problem is NP-complete.

An instance of the **formula satisfiability** problem is a boolean formula θ composed of

1. n boolean variables: $x_1, x_2, ..., x_n$;

2. m boolean connectives: $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT), $\rightarrow$ (implication), $\leftrightarrow$ (if and only if); and

3. parentheses: one pair of parentheses per boolean connective.

Formula Satisfiability

A **truth assignment** for a boolean formula θ is a set of values for its variables.

A **satisfying assignment** for a boolean formula is a truth assignment on which it evaluates to 1.

A formula is **satisfiable** if it has a satisfying assignment.

The formula satisfiability problem is defined as a formal language

SAT $= \{ < \theta >: \theta \text{ is a satisfiable boolean formula } \}$.

Example 1

Consider
$\theta(x_1, x_2, x_3) = (x_1 \leftrightarrow x_2) \wedge \neg x_3$.

Evaluate
$\theta(x_1 = 0, x_2 = 0, x_3 = 0) = (0 \leftrightarrow 0) \wedge \neg 0$
$= 1 \wedge 1 = 1$.

Thus, $\theta(x_1, x_2, x_3)$ has a satisfying assignment and is in SAT.

Example 2

Consider
$\delta(x_1, x_2) = ((x_1 \to x_2) \land (x_1 \to \neg x_2)) \land x_1$.

Evaluate
$\delta(x_1 = 1, x_2 = 1) = ((1 \to 1) \land (1 \to \neg 1)) \land 1$
$= (1 \land (1 \to 0)) \land 1 = (1 \land 0) \land 1 = 0 \land 1 = 0$.

Evaluate
$\delta(x_1 = 1, x_2 = 0) = ((1 \to 0) \land (1 \to \neg 0)) \land 1$
$= (0 \land (1 \to 1)) \land 1 = (0 \land 1) \land 1 = 0 \land 1 = 0$.

Note that $\delta(x_1 = 0, x_2) = 0$.

Thus, $\delta(x_1, x_2)$ has no satisfying assignment and does not belong to SAT.

Showing that SAT is NP-Complete

Theorem 34.9

SAT is NP-Complete.

Proof

First we show that SAT is in NP, and then we show that $\text{CIRCUIT-SAT} \leq_P \text{SAT}$.

We describe an algorithm for verifying a given boolean formula θ with a certificate in polynomial time.

The certificate consists of a truth assignment to the variables in θ.

The algorithm replaces each variable in θ with its corresponding value, and evaluates the expression.

If the expression evaluates to 1, then the algorithm reports 1. Otherwise, it reports 0.

Prove that SAT is NP-Hard

Let C be a boolean circuit with n input wires $x_1, x_2, ..., x_n$ and m logic gates. For $j = 1, 2, ..., m$, let y_j denote the output wire of logic gate j. Assume that y_m is the output of C.

Let $\theta(j)$ be a boolean formula constucted for logic gate j.

If logic gate j is AND with $h(j)$ input wires $z_1, z_2, ..., z_{h(j)}$, where for $k = 1, 2, ..., h(j)$, z_k is one of the input wires $x_1, x_2, ..., x_n$ or one of the output wires $y_1, y_2, ..., y_{j-1}$, then $\theta(j)$ is

$\theta(j) = (y_j \leftrightarrow (z_1 \wedge z_2 \wedge ... \wedge z_{h(j)}))$.

Prove that SAT is NP-Hard

If logic gate j is OR with $h(j)$ input wires $z_1, z_2, ..., z_{h(j)}$, then $\theta(j)$ is

$\theta(j) = (y_j \leftrightarrow (z_1 \vee z_2 \vee ... \vee z_{h(j)}))$.

If logic gate j is NOT with one input wire z_1, then $\theta(j)$ is

$\theta(j) = (y_j \leftrightarrow (\neg z_1))$.

Then, the whole formula $\theta(x_1, x_2, ..., x_n, y_1, y_2, ..., y_m)$ is

$= y_m \wedge \theta(1) \wedge \theta(2) \wedge ... \wedge \theta(m)$.

The whole formula can be constructed in polynomial time.

Correctness

If C has a satisfying assignment, then each output wire of C has a well-defined value, and the output of C is 1.

Thus, when those values are assigned to the variables in $\theta(x_1, x_2, ..., x_n, y_1, y_2, ..., y_m)$, each $\theta(j)$ evaluates to 1, and so the whole formula evaluates to 1.

Conversely, if the whole formula has a satisfying assignment, then C evaluates to 1 under the part of this assignment to its variables.

Thus, $\text{CIRCUIT-SAT} \leq_P \text{SAT}$.

3-CNF Satisfiability

We show that 3-CNF-SAT, a restricted language of boolean formulas, is NP-complete. This problem is useful in proving other problems NP-complete.

A **literal** in a boolean formula is an occurrence of a variable or its negation.

A boolean formula is in **conjunctive normal form**, or **CNF**, if it is expressed as an AND of clauses, each of which is the OR of one or more literals.

A boolean formula is in **3-conjunctive normal form** or **3-CNF**, if each clause has exactly three literals.

The following example formula is in 3-CNF:

$$\theta(x_1, x_2, x_3, x_4) = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4).$$

CIRCUIT-SAT is in NP

The 3-CNF-SAT is defined as follows:

3-CNF-SAT
$= \{ < C >$: C is a satisfiable boolean formula in 3-CNF $\}$.

Theorem 34.10
3-CNF-SAT is NP-complete.

Proof

The algorithm for SAT can be used to verify 3-CNF-SAT, so 3-CNF-SAT is in NP.

Next, we show that $\mathrm{SAT} \leq_P \mathrm{3\text{-}CNF\ SAT}$.

Let $\theta(x_1, x_2, ..., x_n)$ be a boolean formula with n variables.

If the formula contains a clause such as the OR of several literals, we use associativity to parenthesize the expression fully so that each operator in the reulsting formula has 1 or 2 operands.

For example,
$\theta(x_1, x_2, x_3, x_4) = (\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4) \wedge ((x_1 \leftrightarrow x_2) \vee x_3)$
$= (\neg x_1 \vee (x_2 \vee (\neg x_3 \vee x_4))) \wedge ((x_1 \leftrightarrow x_2) \vee x_3)$

Proof

As in the proof for Theorem 34.9, we introduce a variable y_i for the output of each operation.

Then we rewrite the formula as the AND of the output variable for the last operation and a conjuction of clauses for each operation in the formula.

Proof

The resulting expression for the above example is

$$\delta(x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4, y_5, y_6)$$

$$y_6 \wedge (y_1 \leftrightarrow (\neg x_3 \vee x_4))$$

$$\wedge (y_2 \leftrightarrow (x_2 \vee y_1))$$

$$\wedge (y_3 \leftrightarrow (\neg x_1 \vee y_2))$$

$$\wedge (y_4 \leftrightarrow (x_1 \leftrightarrow x_2))$$

$$\wedge (y_5 \leftrightarrow (y_4 \vee x_3))$$

$$\wedge (y_6 \leftrightarrow (y_3 \wedge y_5)).$$

Proof

Each clause has at most three lierals.

y_1	x_3	x_4	$(y_1 \leftrightarrow (\neg x_3 \vee x_4))$
1	1	1	1
1	1	0	0
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	0