Computational Complexity Assignment 3

Tyler Tracy

March 7, 2023

Problem 1

Question:

Let $CNF_k = \{\langle \phi \rangle \mid \phi$ is a satisfiable CNF-formula where each variable appear in at most k places $\}$.

(a) Show that $CNF_2 \in P$.

(b) Show that CNF_3 is NP_complete.

To answer part a I would try to find an algorithm to figure it out

Part a

There are V variables and C clauses.

I'll define a algorithm that will decide if some CNF formula is in CNF_k. It will use a recursive algorithm.

Take in a formula ϕ and first check that no variable appears more than 2 times. This should be possible in a couple passes over the formula. If it does then $REJECT$. Next find the smallest clause in ϕ. We set the first variable to be satisfied. I.E True if it is a x and False if it is a $\overline{x}$. Call the clause this variable is it k_0. Find the clause that contains the other occurrence of the variable and call it k_1. Note that the number of vars in k_0 is less than or equal to the number of vars in k_1.

We handle multiple cases to determine what to do

1. If k_0 has a single variable and k_1 has a single variable and they are different signs (i.e. one is $\overline{x}$ and the other is x) then $REJECT$ this is not satisfiable.
2. If the two instance of the variable are the same sign, then remove both clauses from ϕ and recurse on the new formula.
3. If the two instance of the variable are different signs, then remove k_0 from ϕ and remove the other variable from k_1 and recurse on the new formula.
4. If there is not another occurrence of the variable in the formula then remove k_0 from ϕ and recurse on the new formula.

If we follow these rules we will either $REJECT$ or get a formula with no clauses, meaning we $ACCEPT$.

This does some work for each clause in the formula. There can be at most $2V$ clauses in the formula. The work consists of finding the other occurrence and removing other variables from the formula, which would take no more than $O(V^2)$ work. So the total work is at most $O(V^3)$ which is polynomial in the size of the formula, so it is in P.

I give an example of this formula working

$$(x \vee \overline{y} \vee z) \wedge (\overline{z} \vee \overline{w} \vee a) \wedge (\overline{a}) \wedge (w) \wedge (k \vee \overline{x})$$

The $(\overline{a})$ clause is the smallest, so we set a to False and continue.

$$(x \vee \overline{y} \vee z) \wedge (\overline{z} \vee \overline{w}) \wedge (w) \wedge (k \vee \overline{x})$$

The (w) clause is the smallest, so we set w to True and continue.

$$(x \vee \overline{y} \vee z) \wedge (\overline{z}) \wedge (k \vee \overline{x})$$

The $(\overline{z})$ clause is the smallest, so we set z to False and continue.

$$(x \vee \overline{y}) \wedge (k \vee \overline{x})$$

The $(x \vee \overline{y})$ clause is the smallest, so we set x to True and continue.

$$(k)$$

Set k to True and we are done.

This process will always work because if you handle the smallest clause first, then you will have enough control to satisfy the formula.

Part b

To show that something is NP-complete, we need to show that it is in NP and is NP-Hard

CNF_3 is in NP because we can use a certificate of a satisfiable variable assignment to decide if a formula is satisfiable. Having all of the truth values makes it very easy to check in polynomial time. The certificate could be a string of 0s and 1s where each position corresponds to a variable. If the value is 0 then the variable is false and if the value is 1 then the variable is true. Using this to check all the clauses should be easily obtainable in less than $O(n^2)$ time.

To prove that CNF_3 is NP-Hard I will use a reduction from $3SAT$.

We have a formula ϕ in $3SAT$ with n variables and m clauses. For each variable x we will create a new variable x_i for each occurrence of x incrementing i each time. Replace each x in ϕ with the corresponding x_i value. We now need some mechanism to make sure all the x_is have the same value so they act as one variable. We can do this by adding clauses like the following

$(x_0 \vee \overline{x_1}) \wedge (x_i \vee \overline{x_2}) \wedge \cdots \wedge (x_{i-1} \vee \overline{x_i}) \wedge (x_i \vee \overline{x_0})$ This ensures that all variables are the same value, thus acting as a single variable. This gadget uses each variable twice and then each variable is used once in the actual formula so no variable is used more than 3 times.

We need to show that this formula (ϕ_0) is satisfiable if and only if ϕ is satisfiable.

If ϕ is satisfiable, then the main part of ϕ_0 is satisfiable since it is the same as ϕ. The gadget will also be satisfiable since it is satisfiable iff the variables x_i are the same, which they will have to be since they are all assigned the same value as x in the original formula.

If ϕ_0 is satisfiable, then ϕ is satisfiable. The gadget forces each x_i to be the same so we can just remove the gadget and use the main part of ϕ_0 as ϕ.

Since this condition holds both ways CNF_3 is NP-Complete

Problem 2

Question:

Let $L_1 = \{\langle M, c\rangle \mid M$ is a Turing machine which decides its language in c steps $\}$, (i.e. in c steps for all inputs, which is $O(1)$ time). Prove that L_1 is decidable. [HINT: recall that this must hold for all inputs.]

To show that L_1 is decidable I'll define a new machine D that will decide L_1.

D takes as input M and c. It will run M on all inputs of size $c+1$ or less. If M takes more than c steps, then D will reject. If M takes less than c steps, then D will accept. It doesn't need to worry about inputs of size greater than $c+1$ because M can't process those inputs of c or greater in less than c steps. We check up to inputs $c+1$ to ensure that M will not accept any inputs of size $c+1$ or greater.

Problem 3

Question:

Let $L_2 = \{\langle M \rangle \mid M$ is a Turing machine which decides its language in $O(1)$ steps $\}$. Prove that L_2 is undecidable. [Hint: you may want to try using a reduction from HALT.]

L_2 is the set of turing machines that finish in some constant number of steps. I will show this is undecidable by reducing from $HALT$. If there exists some polynomial time reduction from $HALT$ to L_2, that means a machine that can decide L_2 can be used to help solve $HALT$ in polynomial time.

I'll define a machine D that will decide L_2 and use this machine to solve $HATL$. The reduction is rather simple. If D accepts, then $HALT$ accepts. If D rejects, then $HALT$ rejects. This is because D will accept iff the machine it is given finishes in constant time for all inputs, meaning that it will always halt.

Since a solution to L_2 can be used to solve $HALT$ in polynomial time, L_2 is undecidable.