CS388T: ANSWER TO HOMEWORK 3

VISHVAS VASUKI

1. Question

1.1. **A.** Prove that a DNF formula can be switched in polynomial time to a CNF formula (with a possibly different number of variables), preserving satisfiability.

1.2. **B.** Prove that if $P \neq NP$, there is no polynomial time algorithm that switches a CNF formula to a DNF formula preserving satisfiability.

1.3. **Solution.**

Acknowledgement. I benefitted from discussions with Zifei and Alok in solving this problem.

1.4. **A.**

Theorem 1.4.1. *A DNF formula can be switched in polynomial time to a CNF formula (with a possibly different number of variables), preserving satisfiability.*

Proof. A DNF is a disjunction of conjunctions. A DNF is satisfiable if some conjunction is satisfiable. Any conjunction of literals is satisfiable if and only if that conjunction does not contain both a literal and its negation. Hence, there exists a linear time algorithm, DNF-SAT-ALG, to verify whether a DNF is satisfiable.

If DNF-SAT-ALG determines that the DNF is satisfiable, it can be easily modified to produce the following CNF: (x). DNF-SAT-ALG can also be modified to produce the CNF $(x \wedge \ x)$, which is unsatisfiable, if the DNF is unsatisfiable.

Hence, we have proved that every DNF formula can be switched in polynomial time to a CNF formula, preserving satisfiability, in polynomial time. □

1.5. **B.**

Theorem 1.5.1. *If $P \neq NP$, there is no polynomial time algorithm that switches a CNF formula to a DNF formula preserving satisfiability.*

Proof. We produce a proof by contradiction. Let us call the problem of switching a CNF formula to a DNF formula while preserving satisfiability CNF-SAT2DNF-SAT.

3-SAT, which is about checking the satisfiability of a CNF with atmost 3 variables in each clause, is NP Complete. We know that if $P \neq NP$, no deterministic polynomial time algorithm can solve an NP complete problem such as 3-SAT.

Suppose that CNF-SAT2DNF-SAT can be solved in polynomial time even when $P \neq NP$. As we explained in the answer to the previous subquestion, DNF-SAT can be solved in linear time. So, this assumption leads us to a way of solving 3-SAT in deterministic polynomial time! This contradicts our earlier result that there is no deterministic polynomial time algorithm to solve 3-SAT if $P \neq NP$.

Hence, discarding our assumption which led us to an absurdity, we conlcude that if $P \neq NP$, there is no polynomial time algorithm for CNF-SAT2DNF-SAT. □

2. Question

A boolean formula in a conjunctive normal form is said to be a Horn formula if each clause contains at most one non negated variable. Describe a polynomial time algorithm to decide whether a Boolean Horn formula is satisfiable.

2.1. **Solution.**

Acknowledgement. I referred to [1] in solving this problem.

Algorithm 2.1.1. Algorithm HORN for HORNSAT:

- Input: Horn formula F.
- Output: TRUE if F is satisfiable, FALSE otherwise.
- The algorithm:

  Initial assignment A: Initialize all variables to FALSE.
  Repeat until all clauses with a positive literal are satisfied:
    Take some unsatisfied clause with a positive literal, U.
    Alter the assignment A: Assign TRUE to the positive literal in U.
  If the A satisfies F, return TRUE. Otherwise, return FALSE.

Theorem 2.1.2. *HORN correctly solves HORNSAT.*

Proof. HORN returns TRUE only if it has a satisfying assignment, A. We only need to prove that HORN returns FALSE only if F is unsatisfiable. Let A be the assignment used by HORN to decide whether F is satisfiable.

HORN starts off with an assignment where all variables are FALSE. Due to its construction, HORN assigns TRUE only to variables in F only if it is necessary to satisfy some clause in F. Thus, A has the minimum number of variables set to true for satisfying clauses in F with positive literals.

Now, consider a purely negative clause. As explained above, any assignment A' will have a greater number of variables set to true, when compared with A. So, if A cannot satisfy this clause, no other assignment, A' can.

Hence, HORN returns FALSE only if F is unsatisfiable. Thus, we have proved that HORN correctly solves HORNSAT. □

Theorem 2.1.3. *HORN takes polynomial time.*

Proof. Consider executing HORN on a random access machine. The initial assignment is created in linear time. The selection of an unsatisfied clause with a positive literal in F can only take linear time. The number of times we search for an unsatisfied clause with a positive literal is also polynomial. The step where we check if the assignment A satisfies the formula can also be done in polynomial time. Hence, as all steps in HORN take polynomial time, we conclude that HORN takes polynomial time. □

3. Question

Prove that if there is a language L in $NP \cap coNP$, that is NP complete under polynomial time reductions, then NP = coNP.

3.1. **Solution.** Consider a language a language L in $NP \cap coNP$. Let L be NP complete under polynomial time reductions.

Lemma 3.1.1. *If there is a language L in $NP \cap coNP$, that is NP complete under polynomial time reductions, then $NP \subset coNP$*

Proof. As every language L' in NP is polynomial time reducible to L, every L' in NP also belongs to coNP. Thus, $NP \subset coNP$. □

Lemma 3.1.2. *If there is a language L in $NP \cap coNP$, that is NP complete under polynomial time reductions, then $coNP \subset NP$.*

Proof. As L is NP complete under polynomial time reductions, L is coNP complete under polynomial time reductions. So, every language L' in coNP can be reduced to L under polynomial time reductions. But L is in NP. So, every L' in coNP is also in NP. □

Due to the above lemmata, the main result follows: "If there is a language L in $NP \cap coNP$, that is NP complete under polynomial time reductions, then NP=coNP."

4. Question

Show that if $P = NP \cap coNP$, there is a polynomial time algorithm for factoring. (Be sure that your argument maintains a clear distinction between decision problems and function computation problems.)

4.1. **Solution.**

Definition 4.1.1. FACTOR(n,m) := A decision problem which returns true if n has a factor smaller than the number m and greater than one.

Definition 4.1.2. NOFACTOR(n,m) := The complement of FACTOR(n,m).

Definition 4.1.3. FACTORING(n) := A function which returns the list of prime factors of n, if n has a factor greater than one.

Lemma 4.1.4. $FACTOR \in NP$

Proof. FACTOR is in NP as the following algorithm can be executed by a non deterministic turing machine in polynomial time.

Algorithm 4.1.5. Algorithm ALG-FACTOR for FACTOR:

- Input: n,m
- Output: TRUE if there is a prime factor p of n, such that $p \leq m$, FALSE other wise.
- Algorithm:

  Generate a number $x \leq m$ by guessing one bit of x in every nondeterministic step. This can be done in O(log m) nondeterministic steps.

  If x divides n, return TRUE. Otherwise return FALSE.

□

Lemma 4.1.6. $FACTORING \in FNP$

Proof. FACTORING is in NP as the following algorithm can be executed by a non deterministic turing machine in polynomial time.

Algorithm 4.1.7. Algorithm ALG-FACTORING for FACTORING:

- Input: n
- Output: A list of prime factors of n.
- Algorithm:

  Initialize the output string, OUTPUT to an empty string.

  Use binary search the list of numbers from 1 to n-1 to find the smallest prime number x which divides n. The identification of the correct range of numbers to search is identified in each step using calls to ALG-FACTOR. This can be done in $O(\log n)$ calls to the algorithm ALG-FACTOR.

  If no such x is found, add n to OUTPUT, and return OUTPUT.

  If x is found, add x to the string OUTPUT.

  Recursively call ALG-FACTORING with the parameter n/x, and add the output of this call to the string OUTPUT.

  Return the string OUTPUT.

Note that the number of recursive calls to ALG-FACTOR is O(log n). This is due to the fact that 2 being the smallest prime, with each recursive call, the parameter to ALG-FACTOR is reduced by atleast a factor of 2.

□

Acknowledgement. The professor gave me many clues to arrive at ALG-FACTORING.

As noted in the above proof, we have the following corollary:

Corollary 4.1.8. *The number of prime factors of n is* $O(\log n)$.

Lemma 4.1.9. $FACTOR \in coNP$

Proof. Consider the problem NOFACTOR(n,m). It belongs to NP as the following algorithm is in NP.

Algorithm 4.1.10. Algorithm ALG-NOFACTOR for NOFACTOR:

- Input: n, m
- Output: FALSE if there is a prime factor p of n, such that $p \leq m$, TRUE otherwise.
- Algorithm:

  Get the list of prime factors of n using the algorithm ALG-FACTORING. This step can be executed in O(log n) steps by a non-deterministic turing machine.

  Parse the list of prime factors to find any number x less than m. If x exists, return FALSE. Otherwise, return TRUE.

□

Lemma 4.1.11. *If* $P = NP \cap coNP$*, there exists a polynomial time algorithm which decides FACTOR.*

Proof. From the above lemmata, we can conclude that $FACTOR \in NP \cap coNP$. If $P = NP \cap coNP$, then $FACTOR \in P$. By the definition of P, we conclude that FACTOR has a polynomial time algorithm if $P = NP \cap coNP$. □

Lemma 4.1.12. *If there exists a polynomial time algorithm which decides FACTOR, there exists a polynomial time algorithm which computes FACTORING.*

Proof. Consider the algorithm ALG-FACTORING introduced in one of the lemmata above. We had stated earlier that the part of the algorithm which uses binary search to find the smallest prime factor involves $O(\log n)$ calls to ALG-FACTOR. The functioning of ALG-FACTORING will remain unaffected with any other implementation of ALG-FACTOR. If there exists a deterministic polynomial time algorithm for ALG-FACTOR, the total time taken by ALG-FACTORING will be $O(\log^2 n)$. (This can be seen by adding the time required by the various steps of ALG-FACTORING.)

Hence, we conclude that if there exists a polynomial time algorithm which decides FACTOR, there exists a polynomial time algorithm which computes FACTORING. □

From the above lemmata, we have the main result:

Theorem 4.1.13. *If $P = NP \cap coNP$, there is a polynomial time algorithm for factoring.*

References

[1] Christos H. Papadimitriou. *Computational Complexity.* Addison Wesley, November 1993.