Computational Complexity - HW3

Ali Arabyarmohammadi

May 2022

Exercise 1

Reduction to #3-SAT. Verify that $\#\text{CNF} - \text{SAT} \leq_T^p \#3 - \text{SAT}$

We will verify that CNF-SAT is reducible to 3-SAT.

Let the collection C which has m clauses over the n number of a variable set L be contained in the given instance of CNF-SAT.

$$\text{Set of clause: } C = \{c_1, c_2, \ldots, c_m\}$$
$$\text{Set of literals: } L = \{l_1, l_2, \ldots, l_n\}$$

We can show how to reduce CNF-SAT to 3-SAT, which has the same number of satisfied solutions. The oracle could then be queried with this as the input, and we'll obtain our answer.

We will create a collection C that has no more than three literal (At most 3 literals) over the variables L. This new set of variables includes the original variables as well as the following extra variables:

Let the clause c_i, belongs to the set C that each clause c_i has k variables $x_1, \ldots, x_k$ that contains new literals $\boldsymbol{x}$ over the set L.

We will define the following rules:

- If $k \leq 3$, then the clause requires no change.
- If $k > 3$, we will convert the original clause by introducing additional variables recursively $c_i = \{x_1, x_2, \ldots, x_k\}$.

Let f be this recursive function.

$$f\{c_i\} = \{\{x_1, x_2, y\}, \{\bar{y}, \overline{x_1}\}, \{\bar{y}, \overline{x_2}\}, \{f\{\bar{y}, x_3, \ldots, x_k\}\}\}$$

and form the set:

$$C_i' = \{\{x_1, x_2, y\}, \{\bar{y}, \overline{x_1}\}, \{\bar{y}, \overline{x_2}\}, \{f\{\bar{y}, x_3, \ldots, x_k\}\}\}$$

This process takes a polynomial amount of time. Furthermore, if C is satisfiable, the C' clauses created in situations $k \leq 3$, for arbitrary assignment of any extra variables are satisfied automatically.
While $k > 3$, the number of satisfiable assignments is maintained.

Furthermore, the clause's satisfiability remains unaffected because we have the following logic:

- The additional variable y is set to "True" if x_1 and x_2 are both "False".
- The additional variable y is assigned to "False" if either x_1 or x_2 is "True".

When x_j satisfies C, there is a unique $y = y(x)$ that will satisfy the transformed clause. In fact, The value of y is determined by the x_j input value. That is a bijection between the certificates. Therefore, the number of satisfying assignments between C and C' is conserved.

Using the following expressions, we maintain satisfiability:

- If x_1 or x_2 are "True" then y is forced to be "False" and the transformed clause is satisfied.
- If x_1 and x_2 are "False" then y is forced to be "True", and to satisfy the original clause at least one of $x_3, \ldots, x_k$ must be "True" by induction. As a result, our changed clause will be satisfied as well.

We will query the oracle with this C' and get the number of satisfying assignments, which is the solution we require.

Exercise 2

Reduction to 3-E-SAT, Verify that # 3-SAT $\leq_T^p$ $\#3-\mathrm{E}-\mathrm{SAT}$

We show that counting the number of all not-all-equal assignments is hard even for the promise problem in which we only have inputs with at least one such assignment. A truth assignment is a not-all-equal assignment if all constraints $\{a, b, c\} \in \varphi$ contain a true and a false truth value. Formally, we use the following promise version of 3-E-SAT.

Lemma. There is a polynomial-time reduction from 3-SAT to 3-E-SAT$^+$ that maps formulas with m clauses to formulas with $O(m)$ clauses.

Proof. Let ψ be a 3-CNF formula with n variables and m clauses. To fulfil the promise, we first plant a satisfying assignment.
We obtain a 3-CNF formula φ with $O(m)$ variables and clauses such that $\#SAT(\varphi) = \#SAT(\psi) + 1$.

To construct the instance φ' to 3-E-SAT, we introduce a new variable $\boldsymbol{x}$ for every trivariate clause $(a \vee b \vee c)$ of φ, and we replace that clause with:

$$(x \vee \bar{a}) \wedge (x \vee \bar{b}) \wedge (\bar{x} \vee a \vee b) \wedge (x \vee c).$$

These clauses force $\boldsymbol{x}$ to have the same value as $a \vee b$ in any satisfying assignment.
It can be checked that these clauses are satisfied exactly if the original clause was satisfied and moreover that the trivariate clause is never all-false or all-true.

In total, we increased the number of clauses four-fold without changing the number of satisfying assignments.
Finally, introduce a single fresh variable $\boldsymbol{z}$ and add this variable (positively) to every mono- and bivariate clause.

It is well-known that this modification turns φ' into an instance φ'' of 3-E-SAT: The not-all-equal assignments of φ'' are exactly the satisfying assignments of φ' (if z is set to false) or their complements (if z is set to true).

The reduction computes φ'' from ψ in polynomial time, φ'' has at most $O(m)$ clauses, and we have

$$\#3-E-SAT\left(\varphi''\right) = 2 \cdot (\#SAT(\psi) + 1)$$

Exercise 3

Arithmetic in counting satisfying assignments

We know $\#F$ represents the number of satisfying assignments of F for a Boolean formula F on n variables.

We will construct in polynomial time an $n+m$ variables using two formulas F, G on variables $x \in \{0,1\}^n, y \in \{0,1\}^m$ for:

$$F * G : \; MULT(F,G) = F(x) \wedge G(y)$$

This is the case since,

$$\textstyle\sum_{x,y} F(x) \wedge G(y) = \sum_{x,y} F(x) \cdot G(y) = (\sum_x F(x)) \cdot \left(\sum_y G(y)\right)$$

Thus,

$$\text{MULT } (F,G) = \#F * \#G$$

For $F + G$ We will construct in polynomial-time a $(\max\{n,m\}+1\}$ variable formula :

$$SUM(F,G)(z) = ((z_0 = 0) \wedge F\,(z_1, \ldots, Z_n)) \vee$$
$$((z_0 = 1) \wedge (z_{m+1} = 0) \wedge \ldots \wedge (z_n = 0) \wedge G\,(z_1, \ldots, z_m)) \text{ where } m < n.$$

Therefore,

$$\text{SUM(F, G)} = \#\text{F} + \#\text{G}$$

We can alternatively suppose that they both have the same number of variables (n) without losing generality.
This can always be enforced by "padding" with additional variables that are forced to be 0 in any satisfactory assignment, without affecting the number of satisfying assignments.
We can define the formula $F + G$ on $n + 1$ variables as follows:

$$(F+G)(z,x) = ((z=0) \wedge F(x)) \vee ((z=1) \wedge G(x)).$$

The notation F + 1 is used to represent the formula F + G, where G is a canonical formula with a single satisfying assignment. As previously stated, this is a special case of the *SUM(F, G)*.

Therefore,

$$\#INC(F) = \#F + 1$$

***Reference:** *Notes on Complexity Theory by Jonathan Katz, University Of Maryland, Lecture 24, November, 2011.*